TU

Technische Universität Wien

DIPLOMARBEIT

Entwicklung einer Ausführeinheit für Asbru Light'

Anhand der Kontrollierten Beatmung für Früh- und Neugeborene

ausgeführt am Institut für

Softwaretechnik & Interaktive Systeme
der Technischen Universität Wien

unter Anleitung von
ao.Univ.Prof. Mag. Dr. Silvia Miksch

durch

Christian Fuchsberger
Strudlhofgasse 5
Matr.Nr. 9825710

Wien, 19. Mai 2003 _____

Zusammenfassung

In dieser Arbeit wird eine Ansatz präsentiert, welcher das medizinische Personal bei der Umsetzung von komplexen Therapieplänen unterstützt. Die Wissensbasis für diese Applikation bilden Medizinische Leitlinien, welche das umfangreiche Wissen für spezielle Therapieprozesse in Form von Empfehlungen bereitstellen. Mittels der Planrepräsentationssprache Asbru werden diese Leitlinien formalisiert und somit computerverarbeitbar gemacht. Zu Evalutationszwecken wurde als medizinische Thema die Kontrollierte Beatmung von Früh- und Neugeborenen gewählt, da deren hohe Anforderungen an die Beatmung durch computergestützte Prozesse besser erfüllt werden können. Weiters wurde der Umfang der Sprache Asbru aufgabenspezifisch auf Asbru Light' eingeschränkt, um sie im Rahmen dieser Arbeit verarbeiten zu können.

Das Ergebnis der Arbeit ist eine Applikation, die die in Asbru Light' geschriebenen Medizinischen Leitlinien ausführen kann. Die Applikation wurde dann mittels eines Fallbeispiels evaluiert. Die überaus positiven Ergebnisse werden kurz angeführt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Je länger man vor der Tür zögert, desto fremder wird man.
(Franz Kafka)

Jährlich wird eine große Anzahl von Wissenschaftlichen Arbeiten auf dem Gebiet der Medizin veröffentlicht. Wissen über mögliche therapeutische Vorgehensweisen wird in eine für den Einzelnen nicht überschaubare Anzahl von Medizinischen Richtlinien gepackt. Dieser Wissenspool kann schwerlich aufgrund der Menge vom medizischem Personal bei der konkreten Therapieplanung einer Patientin genutzt werden. Die elektronische Datenverarbeitung könnte diesen Wissenspool erschließen, aber in der Medizin, wo nur absolut zuverlässige Ergebnisse zählen, wo jeder Fehler katastrophale Folgen haben kann, ist die Skepsis bezüglich solcher Systeme sehr hoch.
Aus diesen Gründen etablieren sich immer mehr Systeme, die nicht „closed loop“ die Therapieplanung übernehmen, sondern den Arzt bei Planung und Umsetzung der Therapie unterstützen und ihm das umfangreiche medizinische Wissen nutzbar machen.
Das Asgaard Projekt ist ein solches Planungssystem. Es stellt mit seiner Planrepräsentationssprache Asbru ein mächtiges Werkzeug zur Verfügung, um Medizinische Leitlinien zu formalisieren. Das Ziel der Arbeit ist es, mit diesem System eine konkrete Applikation zu erstellen. Als medizinische Thema wurde hierfür die Kontrollierte Beatmung für Früh- und Neugeborene ausgewählt, da sie ein neues Forschungsgebiet im Bereich der computerunterstützten Therapieplanung darstellt und aufgrund der hohen Anforderungen von Früh- und Neugeborenen an die Beatmung, die durch den Einsatz von computergestützen Prozessen besser erfüllt werden können. Somit wird zuerst aus einer Medizinischen Leitlinie ein maschinenverarbeitbarer

Asbru-Plan erstellt. Für welchen daraufhin ein Interpreter entwickelt wird, der Empfehlungen für die Ausführung von Leitlinien generiert.

1.1 Aufbau dieser Diplomarbeit

Diese Arbeit ist folgendermaßen aufgebaut:
In Kapitel 2 wird die Problemanalyse durchgeführt. Es werden die grundlegenden Begriffe eingeführt, die verschiedenen Ansätze der Formalisierung von Medizinischen Leitlinien besprochen, die Planrepräsentationssprache Asbru und deren reduzierte Form Asbru Light ausführlich dargestellt, die Grundlagen der Beatmung besprochen und abschließend der Asbru-Interpreter von Tibor Bosse vorgestellt.
Kapitel 3 beschäftigt sich mit dem Design der Applikation. Es werden mögliche Lösungsstrategien aufgezeigt, die Module der Applikation, genannt AsbruRTM, näher besprochen und das System- bzw. Klassenmodell in UML erarbeitet.
In Kapitel 4 wird der Prozess der Implementierung angeführt. Es wird die allgemeine Vorgehensweise aufgezeigt, worauf dann die Teilbereiche, sprich AsbruRTM, Asbru, die Domainspezifikation, die Überwachungseinheint, die Pläne und die Benutzeroberfläche im Einzelnen besprochen werden.
Kapitel 5 schließt die Arbeit mit der Evaluierung der Applikation anhand eines Fallbeispiels und der daraus gewonnen Erkenntnisse ab.

1.2 Konventionen

Entsprechend dem Grundsatz der Gleichberechtigung von Mann und Frau gelten alle Personen- und Funktionsbezeichnungen dieser Diplomarbeit, ungeachtet der männlichen oder weiblichen Sprachform, für beide Geschlechter.

Kapitel 2

Problemanalyse

Es ist weniger schwierig, Probleme zu lösen, als mit ihnen zu leben.
(Pierre Teilhard de Chardin, Palantologe)

2.1 Medizinische Leitlinien

Ziel des Asgaard/Asbru Projektes[1] ist es die immer komplexer werdende Therapieplanung und deren Umsetzung, durch die Entwicklung von Werkzeugen zu unterstützen [22, 23]. Den Ausgangspunkt dieses Prozesses bildet die Ausarbeitung von Medizinischen Leitlinien, die im nachfolgendem Abschnitt näher untersucht wird.
Medizinische Leitlinien können nach einem evidenz- beziehungsweise nach einem consensbasiertem Ansatz erstellt werden. In dieser Arbeit werden evidenzbasierte Leitlinien behandelt. Evidenzbasierte Medizin ist:

Definition 2.1 (Evidenzbasierte Medizin) *Evidenzbasierte Medizin (EBM) bezeichnet die medizinische Versorgung nach aktuellem Wissensstand. EBM ist der gewissenhafte, ausdrückliche und vernünftige Gebrauch der gegenwärtig besten externen, wissenschaftlichen Evidenz für Entscheidungen in der medizinischen Versorgung individueller Patienten [38].*

Eine mögliche Definition des Begriffs Medizinische Leitlinien ist:

Definition 2.2 (Medizinische Leitlinien) *Medizinische Leitlinien sind systematisch entwickelte Darstellungen und Empfehlungen mit dem Zweck,*

[1] Im Asgaard Projekt werden vielfach Namen aus der Nordischen Mythologie verwendet. Asgaard ist die Heimat der Götter. Die einzige Verbindung zu Asgaard ist die Regenbogenbrücke, genannt Asbru (oder Bifrost).

Ärzte und Patienten bei der Entscheidung über angemessene Maßnahmen der Krankenversorgung (Prävention, Diagnostik, Therapie und Nachsorge) unter spezifischen medizinischen Umständen zu unterstützen.[...]
Leitlinien geben den Stand des Wissens (Ergebnisse von Kontrollierten Klinischen Studien und Wissen von Experten) über effektive und angemessene Krankenversorgung zum Zeitpunkt der „Drucklegung"wieder. In Anbetracht der unausbleiblichen Fortschritte wissenschaftlicher Erkenntnisse und der Technik müssen periodische Überarbeitungen, Erneuerungen und Korrekturen unternommen werden.[...]
Die Empfehlungen der Leitlinien können nicht unter allen Umständen angemessen genutzt werden. Die Entscheidung darüber, ob einer bestimmten Empfehlung gefolgt werden soll, muß vom Arzt unter Berücksichtigung der beim individuellen Patienten vorliegenden Gegebenheiten und der verfügbaren Ressourcen getroffen werden.[11, 1]

Somit lassen sich folgende allgemeine Ziele für Medizinische Leitlinien festlegen [2]:

- Verbesserung der Qualität ärztlicher und nichtärztlicher Leistungen.
- Verbesserung der klinischen Forschung.
- Verhaltensänderung von Ärzten, medizinischem Personal und Patienten durch Empfehlungen, nicht Richtlinien.[2]
- Zielgerichtetere Versorgung bei Erhalt der ärztlichen Entscheidungsfreiheit.
- Kosten-Ersparnis durch Vermeidung unnötiger diagnostischer und therapeutischer Verfahren.
- Verbesserung der Wissensvermittlung für alle im Gesundheitssystem Tätigen und Patienten.

Diese Ziele können aber nur selten erreicht werden, da die Umsetzung in die Praxis durch etliche Probleme behindert wird [32, 22, 23, 27, 37]:

[2] *Richtlinien sind Handlungsregeln einer gesetzlich, berufsrechtlich, standesrechtlich oder satzungsrechtlich legitimierten Institution, die für den Rechtsraum dieser Institution verbindlich sind und deren Nichtbeachtung definierte Sanktionen nach sich ziehen kann. Somit unterscheiden sie sich im Hinblick auf diese Verbindlichkeit deutlich von Leitlinien. Diese Unterscheidung ist spezifisch für den deutschen und europäischen Sprachraum*[2].

- **Anzahl:** Es gibt eine fast unüberschaubare Anzahl von Leitlinien, da sich viele Ärzte, Gesellschaften und Verbände verpflichtet fühlen ihre eigene Leitlinien zu erstellen.
- **Form:** Leitlinien sind in den verschiedensten Formaten verfügbar: verbal, Flussdiagramm, Tabellen, Graphiken und Listen. Selten können diese computergestützt verarbeitet werden.
- **Implizites Hintergrundwissen:** Oft fließt bei der Therapieplanung implizites Hintergrundwissen ein. Diese Wissen muss bei der Erstellung von Leitlinien explizit gemacht werden.
- **Interpretation:** Die Aussagen von Leitlinien sind nicht immer eindeutig, da es oftmals schwierig ist eine komplexe Therapieplanung darzustellen.
- **Qualität:** Die von den Autoren angewandten Methoden zur Erstellung sind oft sehr unterschiedlich, somit ist die resultierende Qualität der Leitlinien sehr schwankend.
- **Vollständigkeit:** Leitlinien sind oft unvollständig und decken nicht den gesamten Bereich ab, für den sie entwickelt worden sind.
- **Ziele:** Zu gleichen Themen erstellte Leitlinien stimmen in ihren Empfehlungen nicht immer überein. Oft sind sogar keine expliziten Ziele enthalten.
- **Variablität:** Gesundheitszustände von Patienten und die entsprechenden therapeutischen Interventionen sind so mannigfaltig, dass diese oft schwer modellierbar sind.

Eine Lösung dieser Schwierigkeiten stellt die im nächsten Abschnitt beschriebene Formalisierung dar.

2.2 Formalisierung

Aufgrund der im Abschnitt 2.1 beschriebenen Probleme und Unzulänglichkeiten von Medizinischen Leitlinien, wurden Repräsentationsformen gesucht, die computerverarbeitbar sind. Diese Entwicklung ging einher mit der Erstellung von automatischen Planungssystemen, die die medizinische Therapieplanung erleichtern sollten [22, 23]. Die Ziele dieser Systeme, auch Elek-

tronische Medizinischen Leitlinien (EML) genannt, lassen sich in folgende vier Punkte fassen [8]:

- **Wiederverwendbarkeit:** EML als Hilfe für ähnliche klinische Situationen.
- **Aktualisierung:** Schnelle Aktualisierung und Änderung von Leitlinien.
- **Wissensakquirierung:** Spezifisches Wissen, welches bei der Ausführung von Leitlinien an Patienten erworben wird, soll nutzbar gemacht werden.
- **Autorenunterstützung:** Autoren von Leitlinien sollen bei der Entwicklung von Techniken, die die syntaktische, logische und medizinische Gültigkeit von Leitlinien gewähren unterstützt werden.

In den nachfolgenden Abschnitten werden sechs verschiedene Planungsssysteme für Leitlinien vorgestellt. Wie aus der Graphik 2.1 hervorgeht wurden die meisten Systeme in den letzten zehn Jahre entwickelt.

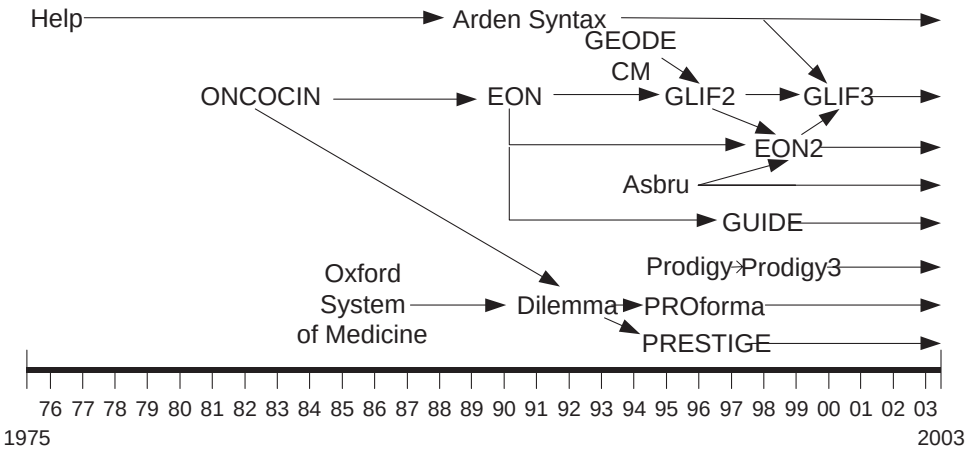

Abbildung 2.1: Historische Entwicklung der Planungssysteme basierend auf [8]. Die Position in der Zeitachse gibt an, wann mit der Entwicklung begonnen worden ist. Ein Pfeil zwischen zwei Systemen gibt an, dass das System auf welches der Pfeil zeigt, vom System, von welchem der Pfeil ausgeht, beeinflußt worden ist.

2.2.1 Asbru

Asbru / Asgaard ist ein Projekt von Forschungsgruppen der „Ben Gurion University“ und der TU Wien. Asbru dient dabei zur Repräsentation von zeitbezogenen, skeletalen Plänen mit entsprechenden medizinischen Absichten. Skeletale Pläne sind Planschemata mit unterschiedlichen Spezifikationsniveaus, die die primären Abläufe und Verfahren spezifizieren, jedoch während der Ausführung genug Freiraum lassen, um die vorgegebenen Absichten zu erreichen.
Asbru erweitert Skeletale Pläne um Planattribute, wie z.B. Absichten, Bedingungen, Effekte, Zeitrepräsentation, Plankorpus und um Operatoren für die ungeordnete, sequentielle, parallele und zyklische Ausführung von Plänen und Aktionen. Die Zeitrepräsentation ermöglicht auch die Darstellung von Unsicherheit in den Ereignissen [22, 23, 21, 39].

2.2.2 EON

EON wird an der „Stanford University“ entwickelt. Das Ziel von EON ist die Bereitstellung von leitlinien- und protokollbasierten Applikationen. Dafür stellt es ein umfangreiches Modellierungsgerüst bereit, welches aus einem Patientendaten-Informationssytem, einem medizinischen Konzeptmodell und einem taskbasierten Leitlinienmodell besteht. Das Leitlinienmodell, genannt Dharma-Modell, definiert die Leitlinien Wissensstruktur wie etwa Leitlinienalgorithmen, Entscheidungsmodelle, empfohlene Aktionen, Abstraktionsdefinitionen und Eignungskriterien.
Die Ausführeinheit EON's wird mit Patientendaten, welche aus einer spezifischen zeitlichen Datenbank stammen, und Benutzereingaben gespeist. Daraus werden dann, basierend auf die im System enthaltenen Leitlinien, Empfehlungen erzeugt. Für den Benutzer sind Erläuterungen verfügbar, mit welchen er die Generierung der Empfehlungen nachvollziehen kann [25, 24, 28].

2.2.3 GLIF

GLIF (Guideline Interchange Format) wird von Forschungsgruppen in Columbia, Stanford und dem „Brigham & Women's Hospital (Harvard)“ entwickelt. Das Ziel von GLIF ist das Design und die Darstellung von Leitlinien in einem computerfähigen Format. Die Sprache selbst basierte ursprünglich

auf den Arden Syntax[3] [26, 29]. In GLIF werden drei Abstraktionsebenen unterschieden:

1. **Konzeptebene:** Leitlinien werden visuell lesbar als Flussdiagramme dargestellt.

2. **Maschinenebene:** Leitlinien werden als berechenbares Modell bestehend aus Ausdrücken, Klinischen Handlungen und Leitlinienkontrollfluss dargestellt.

3. **Implementierungsebene (in Entwicklung):** Unterstützt die Abbildung von Leitlinien auf Elektronische Patientenakten und Prozeduren der Informationssysteme.

Die Erstellung einer „standalone“ Ausführeinheit ist bei GLIF kein erstrangiges Ziel. Vielmehr werden Module entwickelt, die in ein bestehendes Krankenhausinformationssystem (KIS) integriert werden und somit die Therapieplanung implizit durch Empfehlungen unterstützen können [40].

2.2.4 GUIDE

GUIDE wird an der Universität von Pavia, als Teil eines Leitlinienmodellierungs- und Ausführgerüstes entwickelt. Es unterstützt die Integration von Leitlinien in die Organisationsstruktur mittels Entscheidungsbäumen. Mit Petri-Netzen[4] können dann die implementierten Leitlinien simuliert werden.
GUIDE besitzt keine eigentliche Ausführeinheit, vielmehr wird die Ausführbarkeit implizit durch die Darstellung der Leitlinien mittels Petri-Netzen und Arbeitsabläufen in Form von Flussdiagrammen erreicht. Die Ausführung selbst wird von der Software Oracle Workflow 11i^{TM}, einem „Workflow-Managementsystem“, übernommmen [6, 35, 28].

2.2.5 PRODIGY

PRODIGY wird an der „University of Newcastle Upon Tyne“ entwickelt. Das Ziel von PRODIGY ist das Erleichtern der Wissensverarbeitung durch

[3] Der Arden Syntax ist eine Sprache zur Darstellung von Medizinischem Wissen. Er wurde von der Arden Syntax Group entwickelt, die seit 1998 zur Health Level Seven (HL7) Organisation gehört.[3]

[4] Petri-Netze wurden in den fünfziger Jahren von Carl Adam Petri zur Simulation und Darstellung von nebenläufigen diskreten Prozessen vorgeschlagen. Sie basieren auf der Automatentheorie.

die Bereitstellung eines einfachen, leicht verständlichen Modells. PRODIGY beschränkt sich ausschließlich auf chronische Krankheiten[34, 15].
PRODIGY wird in etlichen kommerziellen Produkten [42, 9, 28] eingesetzt, deshalb sind keine detaillierten Informationen zu den jeweiligen Ausführeinheiten verfügbar.

2.2.6 PROforma

PROforma wurde an dem „Advanced Computational Laboratory of Cancer Research (U.K.)“ entwickelt. PROforma modelliert Leitlinien als Netzwerk von Tasks, bestehend aus gekapselten Prozeduren, die bestimmte Ziele verfolgen. Es gibt vier verschiedene Tasks: Aktionen, Pläne, Entscheidungen und Prüfungen. Für jeden Task können Attribute spezifziert werden, wie z.B. Vor- und Nachbedingungen. PROforma selbst kombiniert Logische Programmierung mit Objektorientierten(OO)-Design [5].
PROforma wird in etlichen kommerziellen Produkten [41, 18, 28] eingesetzt, deshalb sind keine detaillierten Informationen zu den jeweiligen Ausführeinheiten verfügbar.

2.2.7 Vergleich

Aus der Tabelle 2.1 ist ersichtlich, dass alle sechs Sprachen viele Ähnlichkeiten aufweisen, die Großteils in ihrer historischen Entwicklung (siehe Abbildung 2.1) begründet liegen. Somit können die einzelnen Sprachen vielmehr durch ihre Besonderheiten unterschieden werden. Asbru weist umfangreiche Möglichkeiten zur Wissensmodellierung auf. EON hingegen konzentriert sich vermehrt auf die Wiederverwendbarkeit dieses Wissens. GLIF erweitert diese Konzepte um die Möglichkeit didaktisches und zusätzliches Material in die Modelle zu integrieren. Bei GUIDE wird vor allem versucht mittels Petri-Netze parallele Tasks zu simulieren. Das sehr einfache Erstellen von Leitlinien für chronische Krankheiten ist das Hauptziel von PRODIGY. Das Highlight von PROforma ist hingegen der logikbasierte Formalismus. Somit muss bei der Selektion der Sprache besonders der jeweilige Anwendungsbereich berücksichtigt werden. Um dann, nach abwägen der Vor- und Nachteile, die geeignetste Sprache auszuwählen [30].

Modelle	Asbru	GLIF	GUIDE	EON	PRODIGY	PROforma
Algorithmisch	+	+	+	+	+	-
Subpläne	+	+	+	+	+	+
Entscheidungskriterien	+ (temporär)	+	+	+	+	+
Absichten und Ziele	+	+ (mittels Subplänen)	+	+	+	+
Rangliste für Optionen	-	+	+	+	+	+
Temporäre Abstraktion	+	-	+	+	-	-

Legende: + Modell vorhanden; - Modell nicht vorhanden

Tabelle 2.1: Vergleich von sechs Leitlinienmodellen aus [8]

2.3 Asbru

Diese Arbeit wird mittels der Sprache Asbru ausgeführt. Deshalb wird in diesem Abschnitt ausführlich auf die in 2.2.1 angeführten Grundkonzepte und Basiselemente Asbru's eingegangen.

2.3.1 Grundkonzepte

2.3.1.1 Hierarchischer Aufbau und Vererbung

Jede Prozedur in einer Medizinischen Leitlinie stellt in Asbru ein oder mehrere Pläne dar. Diese Pläne werden in Hauptpläne, sowie in Subpläne unterteilt, welche wieder Subpläne enthalten können. Aus Abbildung 2.2 kann die hierarchische Struktur für die Kontrollierte Beatmung entnommen werden.

Um alle Pläne über eingetretene Ereignisse zu informieren, werden diese propagiert, das heißt weitergeleitet. Ist z.B. eine Abbruchbedingungen in einem Plan erfüllt so werden alle seine Subpläne mit Signalen/Ereignissen darüber informiert.
Die hierarchische Strukturierung gewährleistet auch eine Wiederverwendbarkeit der Pläne, einmal entwickelte Pläne werden in einer Planbibliothek abgelegt und stehen dann für weitere Pläne zur Verfügung.

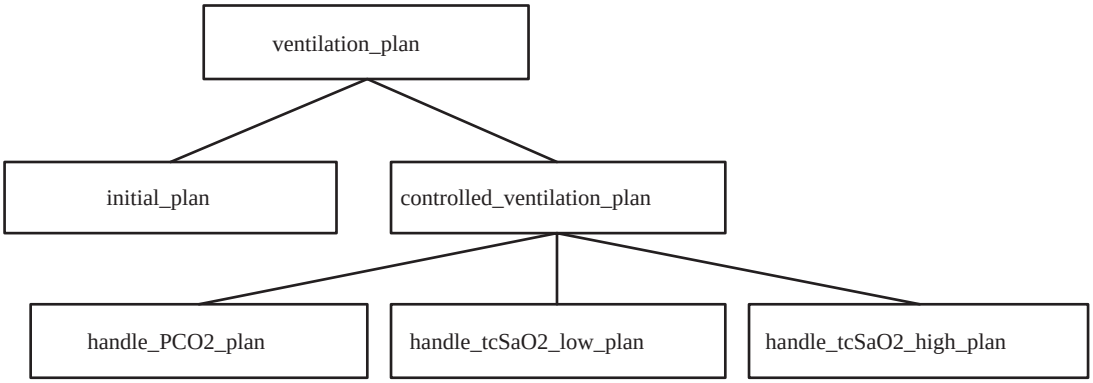

Abbildung 2.2: Planhierarchie Kontrollierte Beatmung

2.3.1.2 Absichten

An dieser Stelle wird schon näher auf das Element Absicht, gesehen als unterschiedliche Arten von Zielen, eingegangen, da es sich nicht nur um ein Sprachelement Asbru's handelt, sondern auch um ein wichtiges Grundkonzept. Asbru erlaubt es nämlich Absichten als Teil einer Leitlinie zu definieren. Somit kann sichergestellt beziehungsweise überwacht werden, ob während und nach der Ausführung von Plänen die beabsichtigten Ziele erreicht worden sind. Gegebenenfalls kann schon frühzeitig gewarnt werden, um dann Alternativaktionen einzuleiten. [28, 21]

2.3.1.3 Zeitrepräsentation

Asbru verfügt über umfangreiche Möglichkeiten zur Zeitrepräsentation. So können nicht nur präzise Zeitangaben gemacht werden, sondern es ist auch möglich Unsicherheiten bzgl. der Anfangszeit, Endzeit und Dauer des Ereignisses anzugeben. Unterschiedliche Referenzzeitpunkte ermöglichen weiters die Darstellung von verschiedenen Zeitachsen[22, 23]. Eine Zeitrepräsentation ist wie folgt aufgebaut:

```
([ESS,LSS],[EFS,LFS],[MinDu,MaxDu],REFERENCE)
```

Ausgehend von einen Referenzzeitpunkt(`REFERENCE`) wird der frühestmögliche Startzeitpunkt (`ESS`), der spätestmögliche Startzeitpunkt (`LSS`), der frühestmögliche Endzeitpunkt (`EFS`), der spätestmögliche Endzeitpunkt (`LFS`) und die minimale (`MinDu`) beziehungsweise maximale Dauer (`MaxDu`)

angegeben. In Abbildung 2.3 wird dieser Zusammenhang nochmals graphisch veranschaulicht.

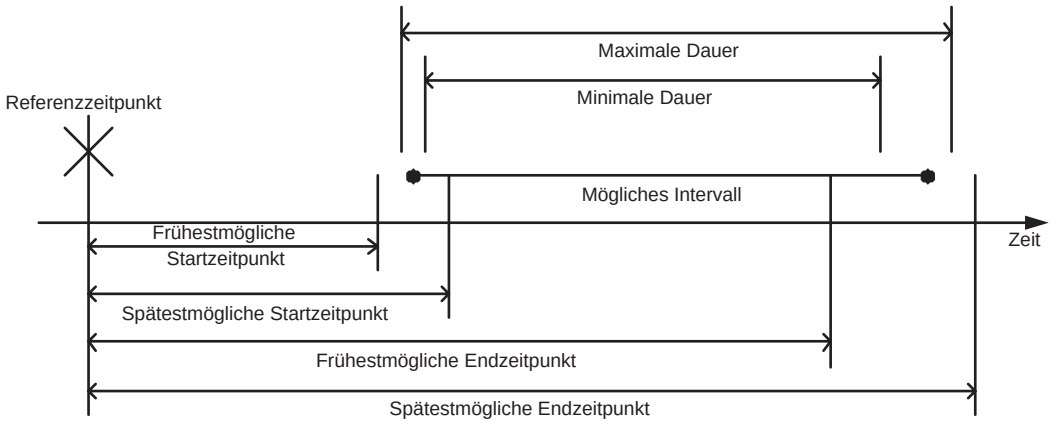

Abbildung 2.3: Zeitrepräsentation in Asbru aus [39]

2.3.1.4 Zustände

Zustände sind ein integraler Bestandteil von Asbru. Jeder Plan in Ausführung befindet sich in einem definierten Zustand (siehe Abbildung 2.4). Der Übergang eines Planes von einem Zustand in den nächsten ist durch Bedingungen (siehe 2.3.2.3) definiert. Die eigentlichen Auslöser einer Zustandsänderung sind aber immer „parameter prositions“, welche bei Erfüllung von der Überwachungseinheit (siehe 3.2.3) zur Ausführeinheit (siehe 3.2.3) weitergeleitet werden, die dann die Zustandsänderung durchführt.

2.3.2 Sprachelemente

2.3.2.1 Präferenzen

Präferenzen beeinflussen und beschränken die Auswahl der auszuführenden Plänen.
Es wird unterschieden zwischen [21]:

- **Strategie:** Allgemeine Strategie zur Behandlung des Problems (z.B. aggressiv, normal)
- **Zweck:** Menge von Zweckangaben (z.B. minimiere die Kosten)
- **Auswahlmethode:** Auswahlheuristik für die Anwendbarkeit des Planes (z.B. ”exact-fit”)

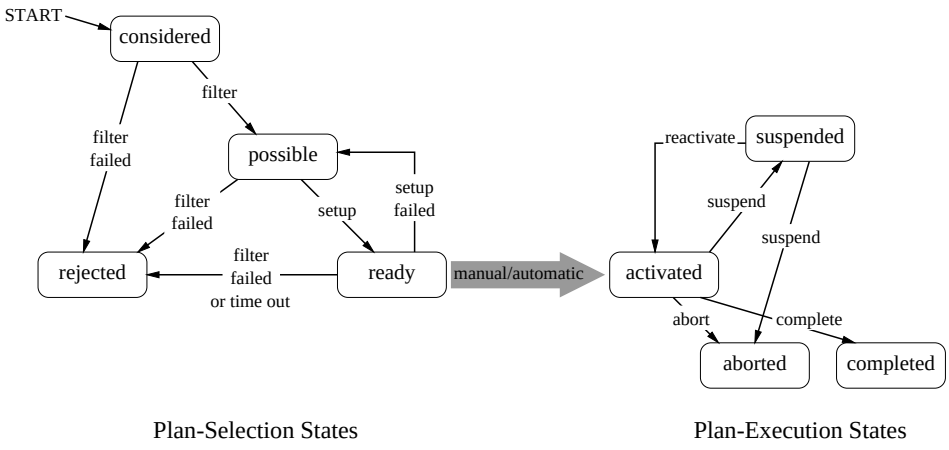

Abbildung 2.4: Planzustände in Asbru aus [39]

- **Ressourcen:** Spezifikation von verbotenen beziehungsweise vorgeschrieben Ressourcen (z.B. Ausschluss von bestimmten Medikamenten)
- **Startbedingungen:** Gibt an, ob ein generischer Plan automatisch instanziert werden kann (nach Erfüllung der Filter- und Setup-Bedingungen), oder ob es dazu die Zustimmung des Benutzer benötigt.

2.3.2.2 Absichten

Wie schon unter 2.3.1.2 angedeutet, beschreiben Absichten die Ziele eines Planes beziehungsweise die anzustrebenden Gesundheitszustände und Aktionen während und nach der Ausführung des Planes. [22, 23]
Folgende vier Kategorien können unterschieden werden [21]:

- **„Intermediate state":** Der Zustand, welcher während der Ausführung des Planes erhalten, erreicht oder vermieden werden soll.
- **„Intermediate action":** Die Aktionen, die während der Ausführung des Planes durchgeführt werden sollen.
- **„Overall state pattern":** Das Muster von Zuständen, welches nach der Ausführung des Planes erfüllt sein soll.
- **„Overall action pattern":** Das Muster von Aktionen, welches nach der Ausführung des Planes beendet sein soll.

2.3.2.3 Bedingungen

Bedingungen sind durative Muster und definieren die Übergänge von Planzuständen (siehe 2.3.1.4). Bedingungen sind optional.
Folgende sechs Typen stehen zur Verfügung[21]:

- **„Filter-preconditions":** Müssen erfüllt sein, damit der Plan gestartet werden kann, können ihrerseits aber nicht erreicht werden (z.B. der Patient muss ein Neugeborenes sein). Sie sind notwendig für den Planzustand *möglich*.
- **„Setup-preconditions":** Müssen erreicht werden, damit der Plan gestartet werden kann (z.B. der Gesundheitszustand des Patienten ist schlecht bis sehr schlecht). Sie ermöglichen den Übergang vom Planzustand *möglich* zu *ausführbar*.
- **„Suspend-conditions":** Geben an, wenn ein gestarteter Plan ausgesetzt werden muss.
- **„Abort-conditions":** Geben an, wenn ein gestarteter, ausgesetzter, oder wieder aufgenommener Plan abgebrochen werden muss (z.B. die Blutgaswerte können mittels Kontrollierter Beatmung nicht mehr stabilisiert werden).
- **„Complete-conditions":** Geben an, wenn ein gestarteter oder wieder aufgenommener Plan erfolgreich abgeschlossen ist. (z.B. wenn die Blutgaswerte über einen gewissen Zeitraum innerhalb bestimmter Grenzen liegen).
- **„Restart-conditions":** Geben an, wenn ein ausgesetzter Plan wieder aufgenommen werden kann.

2.3.2.4 Effekte

Effekte beschreiben die Auswirkungen von Plänen sowohl auf Argumentebene, als auch auf einer qualitativen Trendebene, unabhängig von Bedingungen und Absichten. Für Effekte kann weiteres die Wahrscheinlichkeit ihres Auftretens spezifiziert werden.

2.3.2.5 Plankorpus

Der Plankorpus beinhaltet entweder die zu verfeinernden Subpläne oder die auszuführenden Aktionen. Operatoren ermöglichen die Spezifikation von un-

geordneten, sequentiellen, parallelen und zyklischen Subplänen oder Aktionen.
Folgende Elemente können in einem Plankorpus auftreten [39]:

- `subplans`: Eine Menge von parallel oder sequentiell ausgeführten Planschritten.
- `cyclical-plan`: Ein periodisch wiederholter Plan.
- `single-step`: Ein einzelner Schritt der Planausführung (Variablenzuweisung, Planaktivierung oder Kontextdefinition).
- `refer-to`: Bezug auf den Plankorpus eines anderen Planes.
- `to-be-definied`: Gibt an, dass dieser Plan nicht ausführbar ist, sondern abstrakt (bezüglich OO).
- `user-performed`: Gibt an, dass dieser Plan nicht vom System ausgeführt wird, sondern vom Benutzer (z.B. Anamnese, manuelles Einstellen eines Beatmungsgerätes).

Nachfolgend wird auf drei wichtige Elemente des Plankorpuses, nämlich `subplans, cyclical plan` und `single-step`, näher eingegangen.

Subplans Das Element `subplans` dient zur Gruppierung von einzelnen Planschritten in einer angegebenen zeitlichen Ordnung.
Folgende zeitliche Ordnungen können spezifiziert werden[39]:

- **„sequential":** Die Schritte werden in strikter Ordnung ausgeführt. Jeder weitere Schritt wird erst ausgeführt, nachdem sein Vorgänger abgeschlossen hat (siehe Abbildung 2.5).

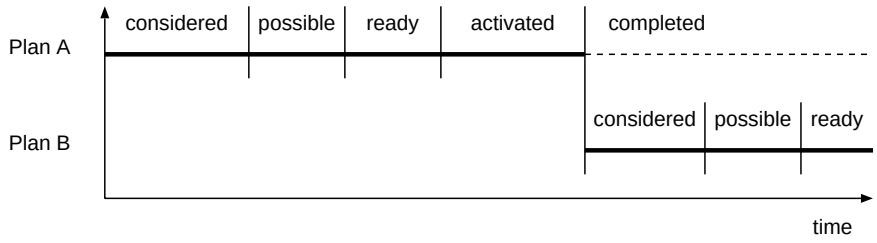

Abbildung 2.5: Sequentielle Ausführung von Subplänen in Asbru aus [39]

- **„parallel":** Alle Schritte beziehungsweise Subplans werden zur gleichen Zeit gestartet. Sie beenden unabhängig, es gibt keinen Sychronisationspunkt (siehe Abbildung 2.6).

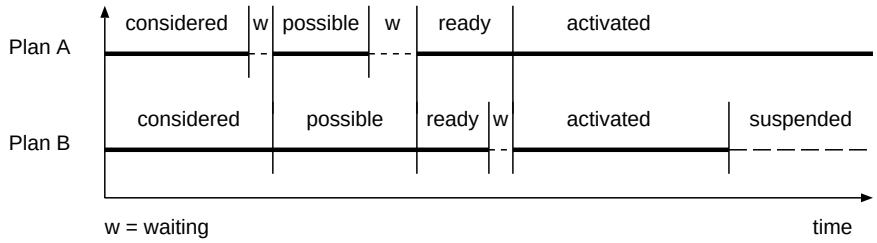

Abbildung 2.6: Parallele Ausführung von Subplänen in Asbru aus [39]

- **„Any-order“:** Die Subpläne werden hintereinander ausgeführt, es ist aber keine Ordnung festgelegt. So kann z.B. der Subplan 2 vor dem Subplan 1 mit der Ausführung beginnen. Zu jedem Zeitpunkt befindet sich aber immer nur ein Subplan in Ausführung (siehe Abbildung 2.7).

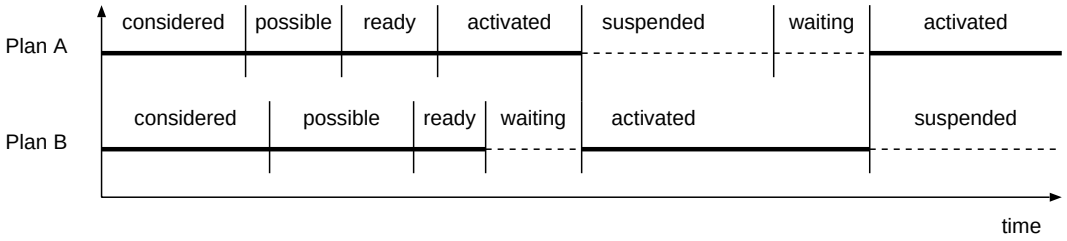

Abbildung 2.7: „Any-Order“ Ausführung von Subplänen in Asbru aus [39]

- **„Ordered":** Es gibt keine Ordnung zwischen den Schritten. Sie können sich überlappen oder auch nicht, jeder Schritt kann dann ausgeführt werden, wenn es passend ist (siehe Abbildung 2.8).

Cyclical Plan Der Zyklische Plan ist für die Modellierung von Medizinischen Leitlinien ein notwendiges Element. Viele Medizinische Prozeduren werden periodisch durchgeführt, so z.B. die Messung der Blutgaswerte alle 10 Sekunden, die Gabe von Medikamenten einmal wöchentlich.
Der Zyklische Plan besteht aus den folgenden Teilen [39]:

- `cyclical-plan-body:` Der zyklisch auszuführende Plankorpus, meistens eine Planaktivierung.

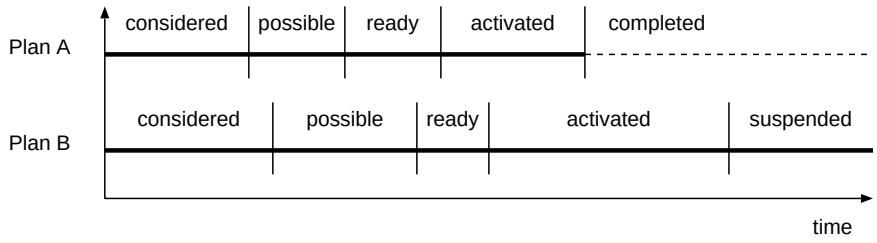

Abbildung 2.8: „Unordered“ Ausführung von Subplänen in Asbru aus [39]

- `start-time:` Zeitpunkt wenn der Zyklische Plankorpus das erste Mal ausgeführt wird.
- `any-repeat-specification:` Spezifiziert die Zeitpunkte an denen der Plan wiederholt wird.
- `max-attempts:` Anzahl der Versuche, um die Aktionen des Zyklischen Plankorpuses erfolgreich abzuschliessen, vor der Plan abbricht.

Single Step Das Planelement `single-step` ist abstrakt und steht stellvertretend für folgende Elemente[39]:

- `plan-activation:` Ein anderer Plan wird aktiviert.
- `variable-assigment:` Der Wert eines arithmetischen Ausdruckes wird einer Variable zugewiesen.
- `set-context:` Eine Kontextvariable wird auf einen bestimmten Wert gesetzt.
- `ask:` Der Benutzer wird angewiesen einen neuen Wert für einen bestimmten Parameter einzugeben.
- `list-manipulations:` Durchführung von verschiedenen Listenmanipulationen, welche keinen Wert zurück liefern.
- `if-then-else:` Ist der gegebene Ausdruck wahr, so wird der erste Teil ausgeführt, andernfalls der zweite.

2.3.2.6 Domainspezifikation

Die Domainspezifikation beschreibt die Schnittstelle der Planbibliothek zur Umgebung, in welcher der Plan ausgeführt wird. Sie spezifiziert somit

Außenwelt-Entitäten, auf welche in der Planbibliothek zugegriffen wird. Daher kann oftmals eine Domainspezifikation für mehrere Planbibliotheken verwendet werden.
Die Domainspezifikation beschreibt:

- Typen von Variablen und Parametern
- Kontext in dem die Abstraktion der Parameter durchgeführt wird
- Parameter
- Variablen
- Konstanten
- Iteratoren, zur Verwendung in Listen und Mengen
- externe Funktionen (z.B. in Java)
- externe Primitive Pläne (z.B. in Java)
- Komplexe Zeitdefinitionen (z.B. periodische Zeitrepräsentationen)

2.4 Asbru Light

Asbru Light wurde von Tibor Bosse im Rahmen seiner Diplomarbeit [4] verwendet, da Asbru eine zu mächtige und umfangreiche Sprache darstellt, als dass man im Rahmen einer Diplomarbeit einen Interpreter für sie implementieren könnte. Asbru Light ist daher eine Untermenge von Asbru, die alle notwendigen Sprachelemente für die Modellierung der Protokolle Jaundice und Diabetes [33] enthält. In dieser Arbeit wurde Asbru Light um einige Aspekte erweitert (Absichten, Zeitrepräsentation), da diese im Plan Kontrollierte Beatmung benötigt werden. Somit ist das hier vorgestellt Asbru Light eine Weiterentwicklung und wird nachfolgend mit Asbru Light' bezeichnet. Diese Version basiert nun auf Asbru 7.3 im Gegensatz zu Asbru 7.2 der ursprünglichen Version, daher waren einige Anpassungen notwendig, so werden z.B. keine `Activate Modes` mehr unterstützt.
Die nachfolgenden Unterabschnitte weisen kurz auf die Unterschiede zu Asbru 7.3 und eventuell zu Asbru Light hin.

2.4.1 Präferenzen

Präferenzen werden in Asbru Light' nicht unterstützt, da sie nicht Teil der unter 2.4 angeführten Protokolle sind.

2.4.2 Absichten

Absichten werden nicht zwingend für die Ausführung der behandelten Protokolle benötigt, da sie aber ein integraler Bestandteil der Sprache Asbru darstellen, wurden sie Asbru Light' hinzugefügt.

2.4.3 Bedingungen

Asbru Light' unterstützt `Filter-preconditions`, `Complete-conditions` und `Abort-conditions`. Daraus folgt, dass nur mehr die Planzustände *considered, ready, rejected, activated, aborted* und *completed* erreicht werden können. Bedingungen können aus folgenden atomaren Ausdrücken zusammengesetzt werden:

- x equal y
- x not-equal y
- x greater-than y
- x less-than y
- x completed

2.4.4 Effekte

Effekte werden in Asbru Light' nicht unterstützt, da sie nicht Teil der unter 2.4 angeführten Protokolle sind.

2.4.5 Plankörper

Der Plankörper kann in Asbru Light' aus Subplänen, Zyklischen Plänen und Einzelschritten bestehen. Bei den Subplänen werden alle vier in Abschnitt 2.3.2.5 angeführten Typen formal unterstützt. Es wurden nur jene Einzelschritte in Asbru Light' aufgenommen, die in den zugrunde liegenden Protokollen (siehe 2.4) verwendet werden. Dies wären `ask`, `variable-assignment`, `plan-activation` und `if-then-else`.

2.4.6 Zeitrepräsentation

Da das Protokoll Kontrollierte Beatmung im Gegensatz zum Jaundice beziehungsweise Diabetes Protokoll eine Echtzeitanwendung (siehe 3.2.5) ist, wurde die im Abschnitt 2.3.1.3 beschriebene Zeitrepräsentation eingeschränkt Asbru Light' hinzugefügt. Als Referenzzeitpunkt ist nur „now" zulässig. Weitere getroffene Einschränkungen werden im Kapitel Implementierung (siehe Abschnitt 4.3.5) angeführt.

2.5 Kontrollierte Beatmung

Ein wichtiger Teil dieser Diplomarbeit ist die Erstellung des Planes für die Kontrollierte Beatmung von Früh- und Neugeborenen in Asbru. In der klinischen Praxis konnte gezeigt werden, dass die Künstliche Beatmung von Früh- und Neugeborenen, deren Überlebenschancen entscheidend vergrößert [13]. Die Künstliche Beatmung ist ein großes Forschungsgebiet und so wurde aus Gründen der Realisierbarkeit nur der Teilbereich Kontrollierte Beatmung herausgegriffen. Dieser wurde weiters auf Früh- und Neugeborenen beschränkt, da diese sehr hohe Anforderungen an die Beatmung stellen und somit eine computerunterstützte Therapie sehr sinnvoll ist.
In den folgenden Abschnitten werden die physiologischen Grundbegriffe der Atmung und Beatmung erläutert, es wird auf die Besonderheiten bei Früh- und Neugeborenen eingegangen und die eigentliche Entwicklung des Planes durchgeführt.

2.5.1 Physiologische Grundbegriffe

2.5.1.1 Atmung

Definition 2.3 (Atmung) *Den Gasaustausch im menschlichem Organismus nennt man Atmung. Sie wird in äußere Atmung (Lungenatmung) und innere Atmung (Zellatmung) eingeteilt [45].*

Für unsere Anwendung ist primär die Lungenatmung relevant. Sie ist gekennzeichnet durch:

- **Ventilation:** Belüftung der Lungenalveolen im Wechsel von Einatmung (Inspiration) und Ausatmung (Expiration)
- **Perfusion:** der Ventilation angepaßte Durchblutung der Lungenkapillaren

- **Diffusion:** Sauerstoffaufnahme und Kohlendioxidabgabe über die alveolokapilläre Membran mit anschließender Bindung von Sauerstoff an Hämoglobin im Blut der Lungenkapillaren beziehungsweise mit Abgabe von Kohlendioxid (CO_2) in die Alveolarluft
- **Konvektion:** Gastransport im Blut

2.5.1.2 Beatmung

Beatmung ermöglicht den pulmonalen Gastransport, wenn die Atemmuskulatur nicht in der Lage ist, die notwendige Atemarbeit zu leisten. Der Gastransport in die Lunge erfolgt durch Aufbau einer Druckdifferenz zwischen Lunge (intrapulmonaler Druck, Palv) und Umgebung (Umgebungsdruck, Pamb, beziehungsweise Druck am oberen Ende des Respirationstraktes, Pawo). Der Gastransport aus der Lunge erfolgt im allgemeinen passiv entlang der Druckdifferenz Palv - Pawo bei entspannter Inspirationsmuskulatur.
Ziel der Beatmung ist es, die Atemarbeit des Patienten zu ersetzen oder zu ergänzen und ausreichende Ventilation und Oxygenierung zu gewährleisten. Bei der Beatmung wird während der Inspiration ein Druckgradient durch eine externe Quelle (Druckgenerator), die mit dem Patienten direkt verbunden ist, erzeugt. Diese externe Quelle z.B. ein Beatmungsbeutel oder ein Respirator(Apparat zur maschinellen, künstlichen Beatmung) sein, die mit dem Patienten über eine Gesichtsmaske, einen Endotrachealtubus usw. verbunden ist. Früh- und Neugeborene stellen durch die bei der Beatmung notwendigen hohen Atemfrequenzen und -widerstände, die geringe Toleranz ihrer Lungen gegenüber mechanischer Belastung und die enorme Kreislaufinstabilität insbesondere in den ersten Lebenstagen besondere Anforderungen an Geräte und Betreuer[31].
Abbildung 2.9 veranschaulicht die Unterschiede im Druckverlauf in den Atemwegen zwischen der Spontanatmung und der maschinellen Beatmung.
Bei der Beatmung sind folgende Parameter von Bedeutung:

- **SO_2:** Sauerstoffsättigung des Blutes
- **$tcSaO_2$:** Transcutan gemessene Sauerstoffsättigung des Blutes
- **pCO_2:** Partieller CO_2 Druck im Blut
- **$tcpCO_2$/$PtcCO_2$:** Transcutan gemessener partieller CO_2 Druck im Blut

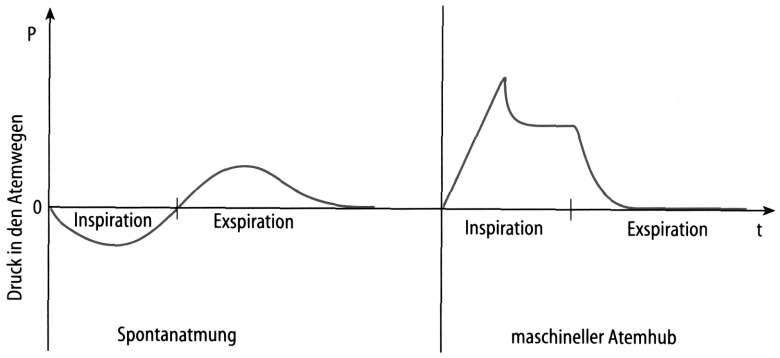

Abbildung 2.9: Druckverlauf in den Atemwegen bei der Spontanatmung und bei maschineller Beatmung aus [19]

- **PEEP:** Positiver endexpiratorischer Druck; Aufrechterhaltung eines Überdrucks auch während der endexpiratorischen Pause der Beatmung [43]
- **PIP:** Inspiratorischer Spitzendruck
- **FIO_2:** Inspiratorische Sauerstofffraktion
- **f:** Atemfrequenz (Atmungen pro Minute)

2.5.2 Plan: Kontrollierte Beatmung von Früh- und Neugeborenen

Definition 2.4 (Kontrollierte Beatmung) *Vollständig künstliche äußere Atmung bei Atemstillstand oder als weitgehend gestützte Lungenbelüftung bei Atemdepression, wobei sich die Intensität (Frequenz, Druck, Atemvolumen) nach Herz-Kreislauf- und Gasanalyse-Parametern richtet[45].*

Die Kontrollierte Beatmung von Früh- und Neugeborenen läuft in folgenden 4 Schritten ab:

1. **Standardwerte setzen**
   Je nach Status des Patienten werden die entsprechenden Standardwerte gemäß Tabelle 2.2 gesetzt.
2. **Optimieren**
   Die Absicht der Optimierungsphase ist es die Werte für FiO_2 und PIP

Status	Grad	O_2	PIP	f	PEEP
gesund		1	10	40	3
krank	leicht	30-40	15	40	3-4
	mittel	40-60	20	40-60	5
	schwer	100	20-25	60	5

Tabelle 2.2: Startwerte Kontrollierte Beatmung

möglichst gering zu halten. Dabei muss aber versucht werden die Vitalparameter in den Zielbereich laut Tabelle 2.3 zu bringen. Dazu werden die Veränderungen der beiden Parameter SO_2 und pCO_2 beobachtet. Da pCO_2 offline gemessen wird und dies in längeren Intervallen als erforderlich, wird mittels der online Messung des ungenaueren $tcPCO_2$ Wertes und der Formel 2.1, ein Zwischenwert approximiert, der für die Berechnungen herangezogen wird. Der genaue Ablauf ist aus Tabelle 2.4 ersichtlich.

$$pCO_{2calc} = \frac{tcpCO_2}{pCO_2} * pCO_2 \tag{2.1}$$

$$f_{soll} = f_{ist} * \frac{pCO_2ist}{pCO_2soll} \tag{2.2}$$

3. **Beobachten**

   (a) Wenn sich die Parameter SO_2 und pCO_2 innerhalb der Grenzwerte befinden, aber außerhalb der Zielwerte, befinden, dann zurück zur Optimierung.

   (b) Wenn sich die Parameter SO_2 und pCO_2 außerhalb der Grenzwerte befinden, dann in ein anderes Beatmungs-Programm wechseln. Die Kontrollierte Beatmung ist beendet.

   (c) Wenn die Parameter SO_2 und pCO_2 die Zielwerte erreicht haben, dann kann zur Entwöhnung übergegangen werden.

Parameter	min	max
SO_2	90	92
pCO_2	40	60

Tabelle 2.3: Zielwerte Kontrollierte Beatmung

Parameter	Zustand	Aktion	Zyklus
SO_2	erhöht	FiO2 in 10er Schritten erhöhen	alle 10 Sekunden
	erniedrigt	FiO2 in 5er Schritten reduzieren	alle 2-10 Minuten
pCO_{2calc}	erhöht	f laut Formel 2.2 erhöhen	alle 5-10 Minuten
	erniedrigt	f laut Formel 2.2 reduzieren	alle 5-10 Minuten

Tabelle 2.4: Optimierung Kontrollierte Beatmung

4. **Entwöhnen**
   Die Kontrollierte Beatmung ist beendet. Es wird nun das Programm Entwöhnung gestartet.

2.5.2.1 Beispiele Kontrollierte Beatmung in Asbru

Nachfolgend werden einige Ausschnitte aus dem Plan Kontrollierte Beatmung in Asbru wiedergeben. Die verbale Beschreibung der Beispiele *Standardwerte setzen* und *Optimieren* findet man im Abschnitt 2.5.2. Das Beispiel für die Domainspezifikation zeigt auf, wie der Parameter $tcSaO_2$ definiert werden muss, bevor er in den eigentlichen Plänen verwendet werden kann.

- **Domainspezifikation**

```
[...]
<parameter-group title="raw data blood gase online">
   <parameter-def name="tcSaO2" type="parts">
     <raw-data-def mode="automatic" unit="%" channel-name="tcSaO2"/>
       <sampling-frequency>
         <numerical-constant value="10"/>
       </sampling-frequency>
   </parameter-def>
 [...]
</parameter-group>
[...]
```

- **Standardwerte setzen**

```
[...]
<if-then-else>
  <simple-condition>
```

```
    <comparison type="equal">
      <left-hand-side>
        <parameter-ref name="patient-state"/>
      </left-hand-side>
      <right-hand-side>
        <qualitative-constant value="healthy"/>
      </right-hand-side>
    </comparison>
  </simple-condition>
<then-branch>
  <variable-assignment variable="FiO2">
    <numerical-constant value="1"/>
  </variable-assignment>
  <variable-assignment variable="PIP">
    <numerical-constant value="10"/>
  </variable-assignment>
  [...]
</then-branch>
</if-then-else>
[...]
```

- **Optimieren**

```
[...]
<plan name="controlled_ventilation_plan">
  <plan-body>
    <subplans type="parallel">
      <wait-for>
        <static-plan-pointer plan-name=""/>
      </wait-for>
      <plan-activation>
         <plan-schema name="handle_PCO2_plan"/>
      </plan-activation>
         <plan-activation>
      <plan-schema name="handle_tcSaO2_low_plan"/>
      </plan-activation>
    </subplans>
  </plan-body>
```

```
</plan>
[...]
```

2.6 Interpreter

Der erste Interpreter für die Sprache Asbru wurde von Tibor Bosse im Rahmen seiner Diplomarbeit [4] implementiert. Dieser besteht aus zwei Programmteilen:

- einen Parser in Prolog, welcher die in XML [12] geschriebenen Asbru Pläne in Clips Fakten übersetzt
- dem eigentliche Hauptprogramm in Clips, das mit dem vom Parser erzeugten Fakten die Ausführung simuliert.

Als Eingabesprache wird nicht der gesamte Sprachumfang von Asbru unterstützt, sondern nur eine reduzierte Form, genannt Asbru Light (siehe 2.4). Vor- und Nachteile dieses Interpreters werden im Abschnitt 3.1 behandelt.

Nachdem nun das Problemumfeld näher beleuchtet worden ist:
Es wurde das Konzept der Medizinische Leitlinie vorgestellt und deren Vor- und Nachteile aufgezeigt. Weiters wurde näher auf Planmodellierungsmodelle eingegangen, die einige Probleme von Leitlinien lösen und eine computerunterstützte Verarbeitung von Leitlinien ermöglichen. Ausführlicher wurde daraufhin auf die Sprache Asbru eingegangen, die sich besonders gut für die Domäne Medizinische Intensivmedizin eignet und deshalb in dieser Arbeit verwendet wird. Um die in dieser Arbeit vorliegende Domäne besser zu verstehen, wurde auch kurz auf die Beatmung eingegangen.
Wird im nächsten Kapitel mit dem Design der eigentlichen Applikation begonnen.

Kapitel 3

Design

Aus der Nähe betrachtet, löste sich das Geschehen in einem Kosmos von Möglichkeiten auf, aber wenn man es aus der Ferne sah, war alles ganz deutlich und sogar einfach

(Andreas Maier, Schriftsteller)

Das Design gliedert sich in drei Abschnitte. Im ersten Abschnitt werden mögliche Lösungsstrategien diskutiert und dann eine geeignete ausgewählt. Im zweiten Abschnitt werden verbal die Erfordernisse an eine Ausführeinheit für Asbru erläutert und vertieft. Im dritten Abschnitt wird mittels UML[12] die theoretische Analyse durchgeführt und die Grundlagen für die Implementierung erarbeitet.

3.1 Lösungsstrategien

Folgende zwei Lösungsstrategien stehen zur Auswahl:

1. Erweiterung des Asbru Light Interpreters von Tibor Bosse

   Vorteile:

   - Die Grundfunktionalität ist schon vorhanden und getestet.
   - Das Parsen der Pläne ist relativ einfach.
   - Asbru-Light könnte weiter vervollständigt werden, um den Sprachumfang von Asbru 7.3 zu erreichen.

   Nachteile:

   - Die Weiterentwicklung von Clips ist nicht gesichert, vorhandene OO-Versionen von Clips befinden sich noch im Beta-Statium

- Zeitliche Aspekte können nur simuliert werden.
- Das Datenabstraktionsmodul fehlt.
- Netzwerkfähigkeit und 3-Schichten-Modell können nicht mit Clips realisiert werden.
- Mit Clips ist keine Echtzeitverarbeitung möglich.

2. Neuentwicklung einer Interpreters in Java

   Vorteile:

   - Ein OO-Ansatz könnte umgesetzt werden.
   - Die Portabilität wäre gewährleistet.
   - Zeitliche Aspekte und Absichten könnten integriert werden.
   - Die starke Verbreitung von Java begünstigt eine Weiterentwicklung.

   Nachteile:

   - Objektstruktur umfangreich und kompliziert.
   - Interpreter müßte ein zweites Mal entwickelt werden.
   - Echtzeitverarbeitung in Java eingeschränkt möglich.
   - Teilweise sind die Anwendungen und Tools für Java sehr versionsabhängig.

Nach Abwägung aller Vorteile und Nachteile wurde die Entscheidung getroffen den Interpreter in Java zu entwickeln. Der Mehraufwand durch die wiederholte Entwicklung rechtfertigt sich dadurch, dass ein OO-Design umgesetzt werden kann, das bezüglich Weiterentwicklungen mehr Zukunft hat, als eine Clips-Lösung.

3.2 AsbruRTM

In diesem Abschnitt werden die einzelnen Module der Ausführeinheit AsbruRTM[1], analysiert [20]. Zu Beginn wird ihre Funktionalität beschrieben und im Systemkontext betrachtet. Daraus wird dann mittels OO-Methoden das Klassenmodell entwickelt.
Da aber der Plan Kontrollierte Beatmung eine Echtzeitanwendung (siehe

[1] Asbru Run-Time Modules

3.2.5) ist, wird auch kurz auf die Problematik von Echtzeitsystemen eingegangen, da dies auch im Design und bei der Implementierung berücksichtigt werden muss.

3.2.1 Datenabstraktion

Die Datenabstraktionseinheit ist das Bindeglied zwischen dem Krankenhausinformationssystem, den Sensordaten, den Benutzereingaben und der Überwachungseinheit. Bei den Sensordaten wird nochmals unterschieden zwischen nieder- und hochfrequenten Daten. Hochfrequente Datenquellen liefern in der Regel alle Sekunden beziehungsweise Minuten Daten und bedürfen daher einer Echtzeitverarbeitung. Niederfrequente Datenquellen liefern hingegen sehr selten Daten (einige Male am Tag oder in der Woche) beziehungsweise die Daten werden manuell eingegeben (z.B. der aktuelle Zustand des Patienten). Somit ist eine Echtzeitverarbeitung für sie nicht notwendig, aber die Behandlung von fehlenden Daten stellt hier das Hauptproblem dar.
Die Datenabstraktion kann weiters in drei Abschnitte unterteilt werden: Datenvalidierung, Berechnung von abgeleiteten Werten und Ableitung von qualitativer Information.

3.2.1.1 Datenvalidierung

Da nicht alle erhaltenen Daten andauernd die gleiche geforderte Qualität aufweisen, müssen diese erkannt und selektiert werden. Dazu werden in Plänen Validierungsrichtlinien festgelegt, nach welchen selektiert wird. So kann z.B. bei der Kontrollierten Beatmung eine Richtlinie festlegen, dass Herzfrequenzdaten nur gültig sind, wenn sie zwischen 0-220 Schlägen liegen und sich innerhalb von 2 Sekunden nicht mehr als um 30 Schläge ändern.

3.2.1.2 Berechnung von abgeleiteten Werten

In vielen Medizinischen Leitlinien werden nicht nur die von den Sensoren gelieferten Daten weiterverarbeitet, sondern auch daraus abgeleitete Werte. Diese Funktionalität wird in diesem Teil der Datenabstraktionseinheit bereitgestellt. In der Medizinischen Leitlinie für die Kontrollierte Beatmung werden z.B. die Blutgaswerte aus den Laborwerten und der Online-Messung berechnet. Dies deshalb da Labormessungen nicht in den benötigten kurzen Abständen durchgeführt werden können und da die Online-Messungen zu ungenau sind.

3.2.1.3 Ableitung von qualitativen Informationen

Qualitative Informationen / Werte können selten direkt von Sensoren bezogen werden. Asbru ermöglicht es deshalb qualitative Werte in den Plänen zu definieren (z.B. eine Herzfrequenz zwischen 170 und 220 ist qualitativ gesehen hoch). Dabei muss aber beachtet werden, dass keine semantische Überprüfung stattfindet, d.h. wenn Limits sich überlappen, werden alle beiden qualitativen Werte zurückgeliefert.

3.2.2 Überwachung

Der Informationsaustausch zwischen der Datenabstraktionseinheit und der Ausführeinheit erfolgt mittels der Überwachungseinheit. Die Datenabstraktion übergibt der Überwachungseinheit eine Liste mit allen Parametern (OPP[2]), die verfügbar sind. Die Ausführeinheit übergibt ihrerseits eine Liste zeitlicher Muster von Parametern (MPP[3]) die für zukünftige Zustandsübergänge von Bedeutung sind. Wenn nun ein OPP die Vorgaben eines MPP entspricht (z.B. der Parameter A ist 10 Minuten über Wert 30), so wird die Ausführeinheit von der Überwachungseinheit darüber informiert.

3.2.3 Ausführeinheit

Diese Einheit übernimmt die eigentliche Ausführung der Pläne. Sie bildet die Pläne auf die aktuelle Situation der Umgebung ab.

3.2.4 Benutzeroberfläche

AsbruRTM soll einerseits mit einer einfachen Benutzeroberfläche ausgestattet sein, die es ermöglicht die Ausführung der Pläne zu überwachen und nachzuvollziehen. Andererseits sollen auch leistungsstarke Schnittstellen zur Verfügung gestellt werden, mit Hilfe derer andere Visualisierungsprogramme die Ausführung der Pläne darstellen können.
Die Benutzeroberfläche selbst ist zweigeteilt: eine Oberfläche wird zur Steuerung der AsbruRTM, sprich starten von Plänen, Eingabe von `ask`, vorort benötigt, die andere dient nur zur Überwachung und Auswertung und soll auch extern verfügbar sein. Damit entfernte Experten in schwierigen Situationen zurate gezogen werden können (sprich Telemedizin siehe [46])

[2] Observed parameter propositions[20]
[3] Monitored parameter propositions[20]

3.2.5 Echtzeitsystem

Definition 3.1 (Echtzeitsystem nach DIN 443000) *Ein Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind.*

Somit muss laut Definition ein Echtzeitsystem korrekte Ergebnisse innerhalb eines definierten Zeitintervalls liefern. Da bei der Kontrollierten Beatmung hochfrequente Daten (siehe 3.2.1) verarbeitet werden und Entscheidungen innerhalb von vorgegebenen Zeitintervallen (Sekundenbereich) getroffen werden müssen, handelt es sich hier um eine Echtzeitanwendung. Genauer betrachtet um ein „Hartes Echtzeitsystem", denn die möglichen Konsequenzen bei Nichteinhaltung der Zeitlimits können katastrophal sein. Der Patient könnte sterben oder schwer geschädigt werden.
Für Echtzeitsysteme gelten folgende Kriterien [16, 44]:

- **Rechtzeitigkeit:**
  - **Antwortzeitverhalten:** Die Antwortzeit einer Aktion ist das Zeitintervall zwischen dem Stimulus der Aktion, der die Aktion auslöst und der Bereitstellung des Resultats durch den Rechner.
  - **Zeitliche Gültigkeit der Echtzeitdaten:** Damit wird das maximal zulässige Zeitintervall zwischen dem Zeitpunkt der Beobachtung des Prozeßobjektes im Rechner und dem Zeitpunkt der Verwendung dieser Beobachtung im Rechner bezeichnet.
  - **Hochlastfähigkeit:** Jeder Rechner hat eine endliche Verarbeitungskapazität. Das Antwortzeitverhalten eines Rechnersystems kann nur garantiert werden, wenn die Lastanforderungen an das Rechnersystem abgegrenzt werden können.
- **Verläßlichkeit:** Da medizinische Anwendungen oft sicherheitskritische Anwendungen darstellen, darf es im System keine Komponente geben, deren Ausfall zu einem katastrophalen Fehler des Gesamtsystems führen kann. Diese Anforderungen an die Verläßlichkeit erzwingen den Einsatz von fehlertoleranten Systemen in sicherheitskritischen Anwendungen.

Ausgrund der im Abschnitt 4 angeführten Gründen wurde zur Implementierung die Sprache Java verwendet. Java ist aber nicht uneingeschränkt

echtzeitfähig [4][47]. Für unsere Anwendung ist aber diese eingeschränkte Fähigkeit ausreichend. Zukünftige Asbru-Anwendungen werden aber eventuell auf eine der voll-echtzeitfähigen Versionen von Java ausweichen müssen, die momentan entwickelt werden (z.B. Real-Time for Java siehe [36]).

3.3 Systemmodell

Aus Abbildung 3.1 ist die Systemarchitektur ersichtlich. Das Kernstück dieser Architektur bilden die AsbruRTM. Sie bestehen aus der Datenabstraktions- , Überwachungs- und Ausführeinheit die in Abschnitt 3.2 beschrieben sind. Ein zusätzliches Modul stellt die Systemzeit dar, die nicht nur vom Betriebssystem übernommen wird, sondern eine eigene Logik enthält. Somit können Pläne im Zeitraffer simuliert werden, die Granularität der Zeit kann den jeweiligen Bedürfnissen angepasst werden und es wird somit eine einheitliche Zeitbasis für alle anderen Module zur Verfügung gestellt.

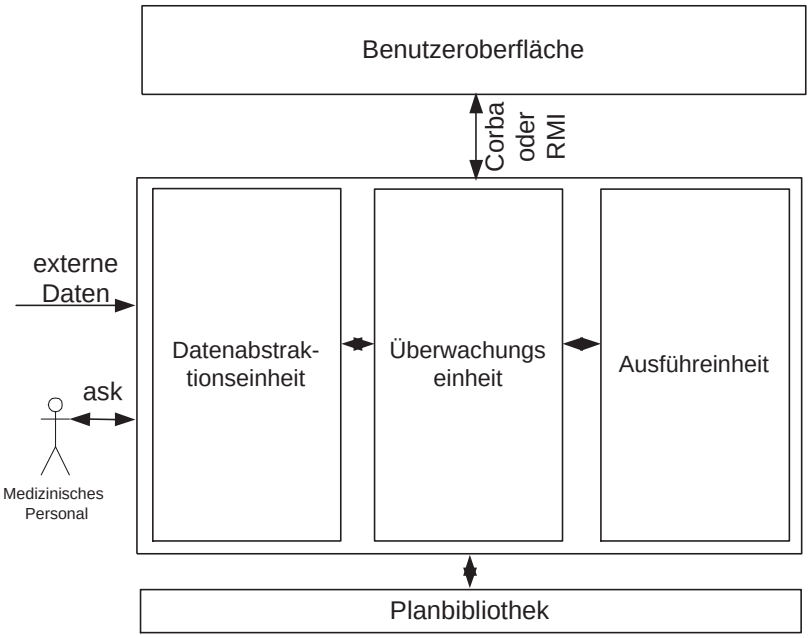

Abbildung 3.1: Vereinfachte Systemarchitektur

[4] Java ist primär nicht echtzeitfähig, da keine Reaktions- bzw. Antwortzeiten garantiert werden können, da das Verhalten des Systems nicht vorhersehbar ist, da der Zugriff auf EA-Signale (z.B. Echtzeit Vitalparameter) weder ausreichend schnell noch zeitlich deterministisch ist

3.3.1 Anwendungsfall

Das Anwendungsfalldiagramm in Abbildung 3.2 repräsentiert einen Teil der Funktionalität des Systems.

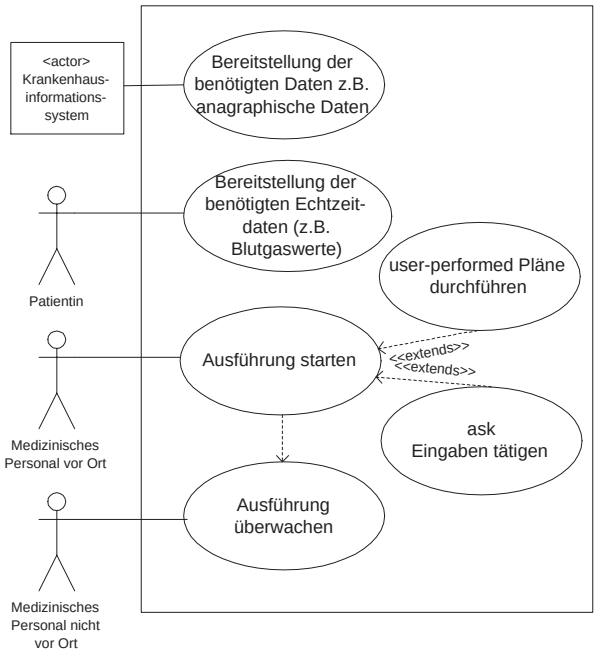

Abbildung 3.2: Anwendungsfalldiagramm Asbru RTM

Tabelle 3.1 beschreibt näher den Hauptanwendungsfall *Ausführung starten*.

3.4 Klassenmodell

Da das Klassenmodell sehr umfangreich ist, wird an dieser Stelle nur auf gewählte Klassen eingegangen. Aus Abbildung 3.3 sind alle Pakete der AsbruRTM's ersichtlich. Sie entsprechen den im Abschnitt 3.2 vorgestellten Einheiten. Hinzugekommen ist nur das Paket *asbru_7_3*, dass die Klassen für Asbru 7.3 enthält. Dieses Paket unterstützt zurzeit nur Asbru Light', kann aber auf den Sprachumfang von Asbru 7.3 erweitert werden.
Die öffentlichen Methoden einiger Klassen aus den AsbruRTM werden in der Abbildung 3.4 angeführt. Abbildung 3.5 stellt den grundsätzlichen Aufbau des Paketes *asbru_7_3* dar. Diese Strukturierung entspricht genau dem Aufbau eines Asbru Planes.

Titel:	Ausführung starten
Kurzbeschreibung:	Der Benutzer wählt einen für den Patienten geeigneten Plan aus und startet dessen Ausführung.
Vorbedingungen:	Genügend passende Pläne in der Planbibliothek, es sind genügend Patientendaten verfügbar.
Beschreibung des Ablaufs: (E ... Benutzerereignis, A ... Systemantwort, AE ... Alternativ Ereignis [48])	E1) Der Benutzer wählt den gewünschten Plan aus A1) Das System überprüft, ob genügend Daten für die Planausführung verfügbar sind. A2) Das System überprüft, ob der Plan gestartet werden kann. A3) Das System startet den Plan
Auswirkungen:	Veränderung des Patientenzustandes

Tabelle 3.1: Anwendungsfall Ausführung starten

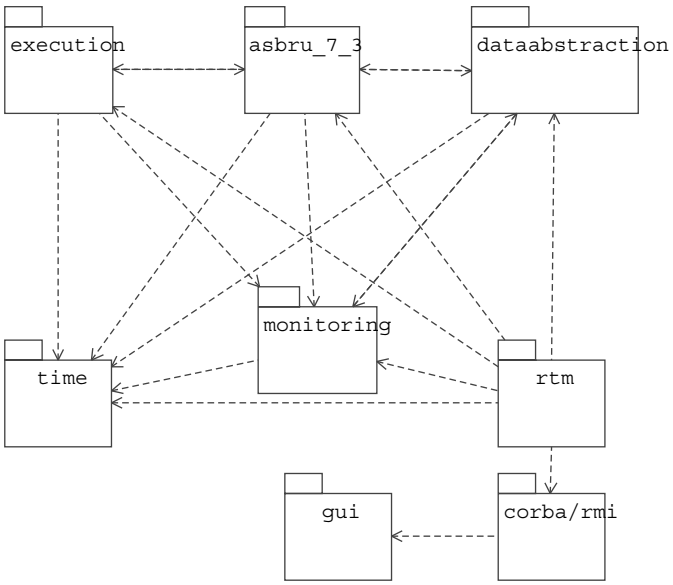

Abbildung 3.3: Paketabhängigkeiten in AsbruRTM

Abbildung 3.4: Klassen aus den AsbruRTM

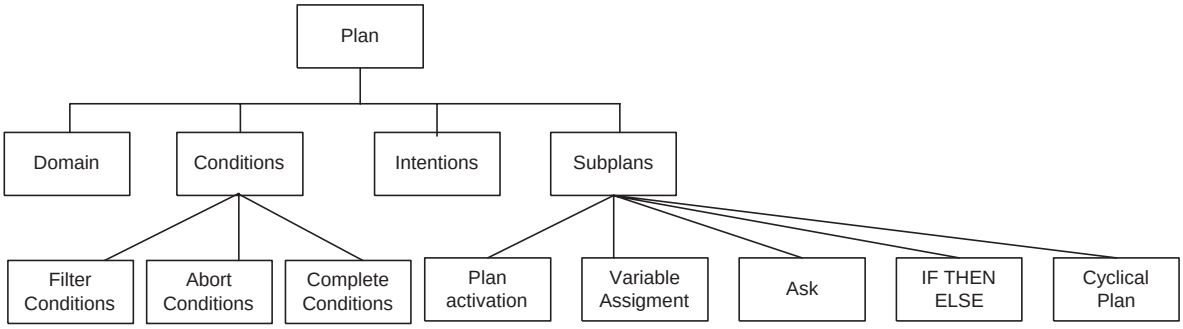

Abbildung 3.5: Schematisches Klassenmodell mit Assoziationen für Asbru

Kapitel 4

Implementierung

If there is a possibility of several things going wrong, the one that will cause the most damage will be the one to go wrong. Corollary: If there is a worse time for something to go wrong, it will happen then.

(Murphy's laws)

4.1 Vorgehensweise

Die Implementierung der Klassen vollzieht sich in zwei Stufen.

1. **Auswahl der Entwicklungswerkzeuge**
   Nachdem in der Designphase Java für die Entwicklung ausgewählt wurde (siehe 3.1), standen schon eine Reihe von Entwicklungswerkzeugen fest. Das Hauptproblem lag aber darin, ein geeignetes Tool zu finden, dass aus der Definition der Sprache Asbru eine entsprechende Java Klassenstruktur erzeugt. Pontifex [17], ein Tool, welches diese Funktionalität besaß, schied aber nach einigen Tests aus. Denn etliche Pakete, welche Pontifex benötigte, hatten sich verändert und so wäre eine sehr aufwändige Überarbeitung von Pontifex nötig gewesen. Nach langer Suche und vielen Tests wurde das Tool Castor[10] ausgewählt. Es ist OpenSource und imstande die mächtige Sprache Asbru zu verarbeiten. Der Nachteil ist, dass es eine Unmenge von Klassen erzeugt (bei Asbru über 2700 Klassen bei nur über 130 Schlüsselelementen). Der Umgang mit dieser großen Anzahl von Klassen ist aber sehr intuitiv.

   Für die Entwicklung werden daher folgende Werkzeuge verwendet:

   - **Java Entwicklungskit:** jdk 1.4.1
   - **Java Editor:** JCreator LE

- **XML-to-Java Klassengenerator:** Castor 0.9.4.1
- **XML-Parser:** Xerces 2.2.0 (wird von Castor benötigt)

2. **Phasenintegration**
   Die Klassen der AsbruRTM und der Sprache Asbru werden jeweils als Funktionskette implementiert. Jede Funktionskette stellt eine Reihe von kollaborierenden Klassen dar. Somit wird schrittweise die Funktionalität des Systems erhöht und man erhält frühzeitig ein testbares System. Problematisch ist es nur dann, wenn Klassen in mehreren Funktionsketten auftreten, dann müssen sie vollständig implementiert werden [48].

4.2 AsbruRTM

Der Schwerpunkt bei der Implementierung der AsbruRTM bildet die parallele Verarbeitung. Denn die einzelnen Module müssen ihre Aufgaben gleichzeitig erfüllen. So muss die Dataabstraktionseinheit andauernd die von den Sensoren erhaltenen Werte auswerten und der Überwachungseinheit mitteilen, die ihrerseits andauernd überprüft, ob die Werte den Vorgaben der Ausführeinheit entsprechen und diese eventuell darüber informiert werden muss.
Die Hauptlogik für die Ausführung steckt in den Klassen von Asbru, die Ausführeinheit liefert die aufbereiteten Daten, stellt das Grundgerüst für die Ausführung bereit und interagiert mit der Umgebung.

4.3 Asbru

Asbru-Pläne enthalten die gesamte Information bezüglich der Ausführung und Initialisierung von Medizinischen Leitlinien. Deshalb steckt die eigentliche Logik in den Asbru-Klassen. Diese Klassen wurden, wie schon in 4.1 beschrieben, mit dem Tool Castor erzeugt. Die Grundfunktionalität dieser Klassen besteht darin, dass mit ihnen ein Asbru Plan von XML nach Java-Objekten abgebildet werden kann. Weiters werden alle notwendigen Methoden bereitgestellt, um auf die Klassen und deren Inhalt zugreifen zu können. Diese Klassenstruktur diente als Ausgangspunkt für alle weiteren Implementierungsstufen.
In den nachfolgenden Unterabschnitten werden einige Teilbereiche angeführt und ihre Besonderheiten beschrieben.

4.3.1 Domainspezifikation

Die in den Asbru-Plänen enthaltene Domainspezifikation wird mithilfe der Datenabstraktionseinheit verarbeitet. Wobei die einzelnen Elemente unterschiedlich behandelt werden:

- **Variablen:** Werden von der Dataabstraktionseinheit aus den Asbru-Plänen herausgefiltern und in einer Hashtabelle abgelegt.
- **Parameter:** Werden von der Dataabstraktionseinheit aus den Asbru-Plänen herausgefiltert, in eine Hashtabelle abgelegt und das entsprechende Parameterobjekt der Asbru-Klasse wird als Thread bzw. wenn es als „automatic“ deklariert worden ist als Timerthread gestartet. Diese Objekt trägt dann je nach Parametertyp und Einstellung die entsprechenden Werte in die Hashtabelle ein (z.B. frägt es alle 10 Sekunden den aktuellen $tcSaO_2$ Wert ab und übermittelt ihn der Dataabstraktionseinheit).
- **Berechnete Parameter:** Wenn ein berechneter Parameter von der Überwachungseinheit verwendet wird, so wird er bei Änderung einer seiner abhängigen Parameter neu berechnet.

Da das System nicht online arbeitet, werden alle Parameterdaten aus Dateien gelesen. Der Dateiname besteht aus dem Parameternamen erweitert um das Suffix „dat“. Tabelle 4.1 verdeutlicht diese Verarbeitung anhand eines Beispiels aus dem Plan Kontrollierte Beatmung.

4.3.2 Überwachungseinheit

Die Klassen der Überwachungseinheit stellen die Verbindung zwischen der Ausführeinheit (der Planlogik) und der Datenabstraktionseinheit (den Parametern) her. Sie lösen somit mittels „parameter propositons“ Änderungen von Planzuständen aus (siehe 2.3.1.4) und stellen weiters eine Abstraktionsschicht zwischen Planlogik und Daten dar. Die Überwachungseinheit selbst ist eine zentrale Stelle die Botschaften aufnimmt, verarbeitet und weiterleitet. Sie wird von der Datenabstraktionseinheit ständig darüber informiert, ob sich Parameter geändert haben. Diese Information wird den entsprechenden Plan-Bedingungen weitergeleitet, wenn deren zuvor eingetragenen „parameter propositions“ erfüllt sind.

Planausschnitt

```
  [...]
2 <parameter-def name="tcSaO2" type="parts" >
   <raw-data-def mode="automatic" channel-name="tcSaO2"/>
4    <sampling-frequency>
       <numerical-constant value="10"/>
6    </sampling-frequency>
    [...]
8 </parameter-def>
  [...]
```

Programmaktionen

Zeile 3: Da der Parameter „automatic“ ist, wird ein neuer Timerthread erzeugt. In die Parameter-Hashtabelle wird der Schlüssel „tcSaO2“ eingetragen. Die Daten werden aus der Datei tcSaO2.dat gelesen.

Zeile 5: Der Parameter soll alle 10 Sekunden neu ausgelesen

Tabelle 4.1: Implementierung Domainspezifikation. Die Elemente im Planausschnitt stellen hier implizit die entsprechenden Java-Objekte dar.

4.3.3 Pläne

Das primäre Planobjekt wird von der Ausführeinheit gestartet. Daraufhin werden Absichten und Bedingungen in eigenen Tasks initialisiert. Subpläne werden dann sequentiell, für die Typen „sequential“ und „any-order“, oder parallel für die Typen „parallel“ und „unordered“ ausgeführt(siehe 2.3.2.5). Diese Vereinfachung des Ausführverhalten von Subpläne ist legitim, da hier nur eine Untermenge der Möglichkeiten verwendet wird. Bei der parallelen Ausführung wird für jeden Subplan ein eigener Thread generiert, der die Verarbeitung übernimmt. Dem Hauptplan werden nur Ereignisse bezüglich Absichten und Bedingungen beziehungsweise die Abarbeitung aller parallelen Subpläne mitgeteilt. Umgekehrt löst der Hauptplan nur Ereignisse für Absichten und Bedingungen in den Subplänen aus. Diese Signalweiterleitung wird im Asgaard/Asbru Projekt „Propagation“ genannt.
Den Zustand in dem sich ein Plan befindet, lässt sich aus dem aktuellem Verarbeitungsschritt und aus den erhaltenen Ereignissen ableiten. Einige Beispiele werden in Tabelle 4.2 angeführt. Tabelle 4.3 hingegen zeigt exemplarisch die Zusammenhänge zwischen einem Plan und der Ausführeinheit auf. Für dieses Planfragment wird in Abbildung 4.1 die entsprechende parallele Abarbeitung dargestellt.

4.3.4 Zyklische Pläne

Die periodische Ausführung von Arbeitsschritten wird mittels Zyklischen Plänen modelliert. Für Zyklische Pläne gibt es in Java die Klasse Timerthreads, welche diese Grundfunktionalität bereitstellt. Somit werden Zyklische Plänen von dieser Klasse abgeleitet und die entsprechenden Objekte während der Ausführung mit der Periode initialisiert. Variable Periodendau-

Verarbeitungsschritt(V), Ereignis(E)	**Planzustand**
V: Planthread wird gestartet	*activated*
V: Fehler z.B. keine Überwachungseinheit	*aborted*
E: ”Abort”—Bedingung erfüllt	*aborted*
E: ”Completed”—Bedingung erfüllt	*completed*
E: Subpläne abgebrochen	*aborted*

Tabelle 4.2: Bestimmung des aktuellen Zustandes eines Planes

Planausschnitt

```
 [...]
2<plans>
 <plan-group>
4 <plan name="ventilation_plan">
   <conditions>
6   <complete-condition>
     <constraint-combination type="and">
8      <parameter-proposition parameter-name="FiO2">
        <value-description type="less-or-equal">
10        <numerical-constant value="40"/>
          [...]
12  <plan-body>
     <subplans type="sequentially">
14    [...]
     <plan-activation>
16    <plan-schema name="initial_plan"/>
     </plan-activation>
18   <plan-activation>
      <plan-schema name="controlled_ventilation_plan"/>
20    [...]
```

Programmaktionen

Zeile 4: „ventilation_plan“ ist der primäre Plan und wird somit von der Ausführeinheit gestartet

Zeile 6: Die „Complete Conditon“ wird in einen eigenen Thread gestartet

Zeile 7-8: Es wird eine Threadgruppe erzeugt, in die die Threads der einzelnen „parameter propositions“ eingefügt werden.

Zeile 8-10: Der Thread dieser „parameter proposition“ ist nun so lange aktiv bis FiO_2 <= 40 ist, oder die Anwendung abgebrochen wird. Er kümmert sich weiters um die Kommunikation mit der Überwachungseinheit und nimmt deren Botschaften entgegen.

Zeile 18: Die nachfolgenden Plänen werden sequentiell abgearbeitet, d.h. es werden hierfür keine zusätzlichen Threads generiert.

Zeile 18: Die nachfolgenden Plänen werden sequentiell abgearbeitet, d.h. es werden hierfür keine zusätzlichen Threads generiert.

Zeile 21: Der Plan „initial_plan“ wird gestartet.

Zeile 24: Der Plan „controlled_ventilation_plan“ wird gestartet.

Tabelle 4.3: Implementierung Plan. Die Elemente im Planausschnitt stellen hier implizit die entsprechenden Java-Objekte dar.

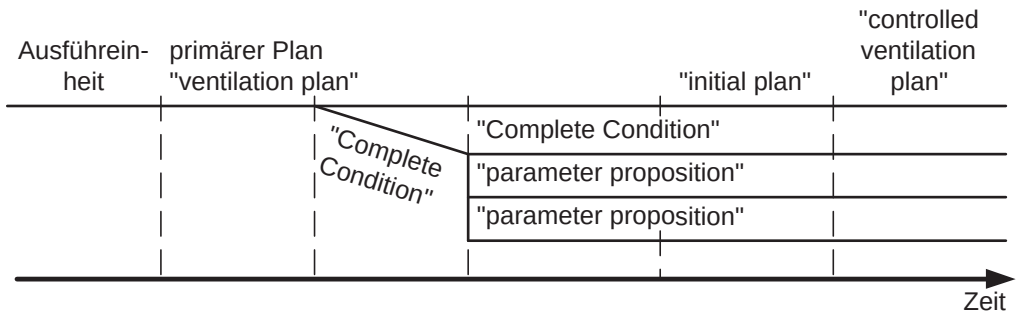

Abbildung 4.1: Parallele Abarbeitung eines Planfragments. Die Linien stellen Threads dar. Die einzelnen Abzweigungen stellen neu generierte Threads dar. Die Beschriftungen der Threads geben an, mit welchen Aufgaben sich zurzeit der Thread beschäftig.

ern und variable Startzeiten werden nicht unterstützt, da es keine Regeln gibt, mit welchen konkrete Werte dafür bestimmt werden könnten.

4.3.5 Zeitrepräsentation

Bei der Implementierung der Zeitrepräsentation wurde bezüglich ihrer Definition in Abschnitt 2.3.1.3 einige Vereinfachungen angewandt, da in der Sprachbeschreibung Asbrus keine Algorithmen definiert sind, nach welchen die Zeitrepräsenation implementiert werden könnte.

- **Startzeitpunkt:** Als Startzeitpunkt wird immer der frühestmögliche Startzeitpunkt (`ESS`) verwendet.
- **Endzeitpunkt:** Als Endzeitpunkt wird immer der spätestmögliche Endzeitpunkt (`LFS`) verwendet.
- **Dauer:** Als Dauer wird immer die maximale Dauer (`MaxDu`) ausgewählt.

4.4 Benutzeroberfläche und CORBA

Die Benutzeroberfläche stellt in ihrer Implementierung eine Ausnahme dar. Sie ist im Gegensatz zu den anderen Teilen des Systems nicht in Java, sondern in Delphi codiert. Diese Entscheidung wurde aufgrund mehrerer Überlegungen getroffen:

- Die primäre Aufgabe dieser Arbeit liegt in der Entwicklung einer Ausführeinheit, die Benutzeroberfläche wird nur zu Testzwecken benötigt. Daher soll sie schnell und einfach entwickelt werden können. Hierzu ist Delphi besonders gut geeignet.

- Benutzeroberflächen verschiedenster Entwicklungssprachen sollen an die Ausführeinheit angebunden werden können. Mittels dem CORBA[7] wurde dies ermöglicht.

Aus diesen Gründen ist die Funktionalität der Benutzeroberfläche auf die Anzeige von Planzuständen, Fehlern, Warnungen, und Informationen in Listenform beschränkt. Vielmehr ist die CORBA-Schnittstelle interessant. Sie ermöglicht es Benutzeroberflächen in den verschiedensten Implementierungssprachen (C++, Java, Delphi) entfernt an die Ausführeinheit anzukoppeln.

Kapitel 5

Evaluierung

Es irrt der Mensch, solang' er strebt.
(Johann Wolfgang von Goethe, Dichter)

In diesem Kapitel werden die Ergebnisse der Evaluierung präsentiert. Zuerst wurde eine klinische Evaluierung in Erwägung gezogen, da es aber zum jetzigen Zeitpunkt der Entwicklung nicht möglich ist das Programm online zu testen, wurde mithilfe von realen Daten ein Fallbeispiel simuliert.

5.1 Fallbeispiel

Daten
Die Daten für das Fallbespiel, also die Parameter $tcSaO_2$ und $PtcCO_2$, werden mittels realen Daten einer Patientin (siehe Diagramm 5.1), simuliert. Da die Daten in Minutenabständen aufgezeichnet wurden, aber die Anwendung 10-Sekunden-Abstände benötigt, werden die dazwischenliegenden Werte mittels Interpolation generiert. Weiters wird der Verlauf der beiden Parameter dem Verhalten des Protokolls Kontrollierte Beatmung angepaßt.
1. Phase: Vorbereitung
Die Patientin ist an den Respirator (siehe Abbildung 5.2) angeschlossen. Die Voreinstellungen des Respirators können aus der Tabelle 5.1 entnommen werden.
Weiters sind folgende Informationen zur Patientin bekannt:
Gewicht: ca. 1000g
Grad der Erkrankung: schwer

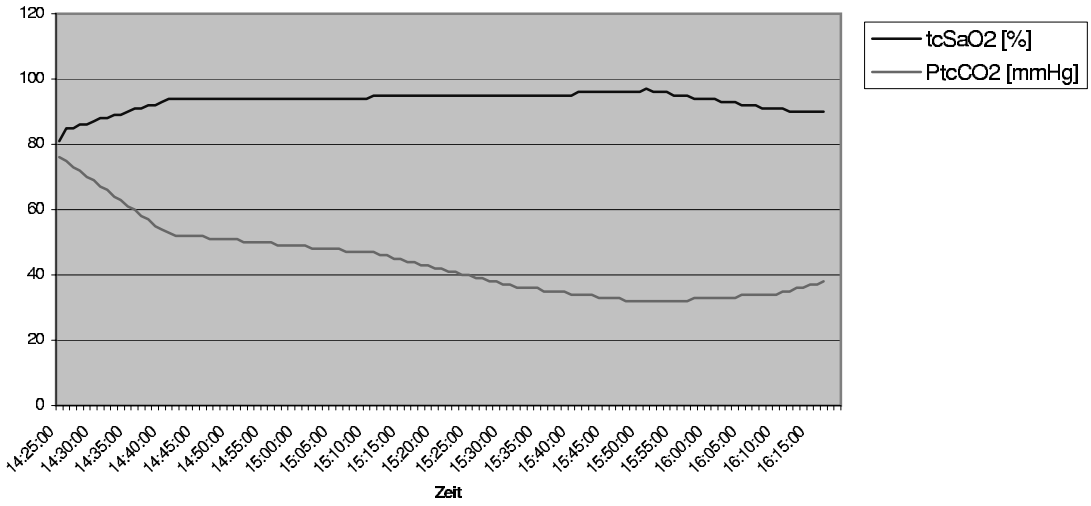

Abbildung 5.1: Patientendaten

Abbildung 5.2: Patientin angeschlossen am Respirator aus [14]

2. Phase: Starten

Das Programm wird nun in drei Stufen gestartet:

1. **CORBA:** Der für CORBA benötigte Namensdienst wird gestartet.
2. **Visserver:** Die Benutzeroberfläche wird gestartet und macht ihre Visualisierungsdienste über CORBA verfügbar.
3. **AsbruRTM:** AsbruRTM wird mit der Angabe des gewünschten Planes gestartet. Der Plan wird zuerst geparst und auf Validität hin überprüft. Danach werden aus dem XML-Plan die entsprechenden Java-Objekte generiert. AsbruRTM beginnt daraufhin mit der Verarbeitung der Domainspezifikation und startet nach Einrichtung der entsprechenden Domäne die eigentliche Planausführung.

Parameter	Wert
PIP	20mbar
PEEP	3-5mbar
f	40-60
FIO_2	0.4 - 1.0
Atemgastemperatur	37°

Tabelle 5.1: Voreinstellungen Respirator

Der Plan für die Kontrollierte Beatmung benötigt in diesem ersten Schritt nun zwei Grundeinstellungen, um mit der weiteren Verarbeitung fortfahren zu können: Der Gesundheitszustand und das Gewicht der Patientin. Hierzu werden in der Benutzeroberfläche Dialoge angezeigt, in welche die entsprechenden Werte eingegeben werden. (siehe Abbildungen 5.3 und 5.4). In diesem Fallbeispiel ist das „ill“ für den Gesundheitszustand und 1 kg für das Gewicht.

Abbildung 5.3: Eingabeaufforderung: Status der Patientin

Abbildung 5.4: Eingabeaufforderung: Gewicht der Patientin

3. Phase: Kontrollierte Beatmung
In dieser Phase beginnt das Programm mit der Ausführung der Pläne. Abbildung 5.5 zeigt die Textausgabe der Ausführeinheit auf Kommandozeile-

nebene. Sie dient hauptsächlich zur Kontrolle des Ablaufverhaltens der Ausführeinheit und zur spezifischen Fehlersuche in den Plänen.

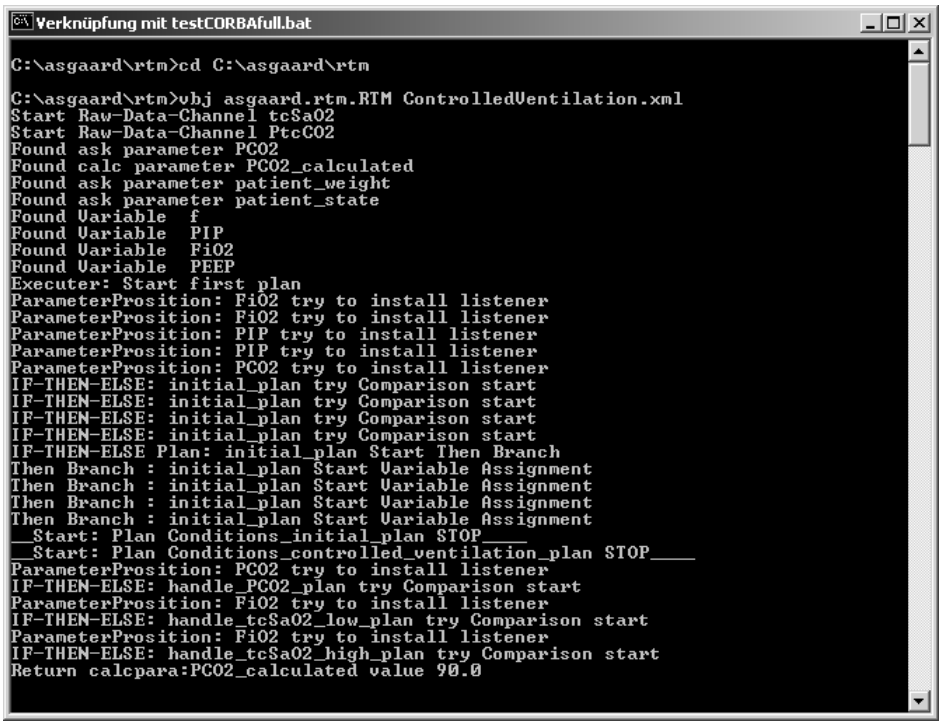

Abbildung 5.5: AsbruRTM Statusinformationen

In Abbildung 5.6 wird der Bereich der Planmeldungen der Benutzeroberfläche dargestellt. Die Meldungen werden chronologisch nach ihrem Auftreten angeordnet, wobei die aktuellsten Meldungen immer ganz oben angezeigt werden. Prinzipiell wird immer ein Zeitstempel, der Elternplan, der Planname und die eigentliche Planmeldung angezeigt. Folgende Planmeldungen werden unterstützt:

- Zustand / Status des Planes (siehe 2.3.1.4)
- Zustand / Status von Bedingungen (siehe 2.3.2.3)
- Planaktivierungen
- Informationen über Verzweigungen

Die aktuellen Werte von Parametern und Variablen versehen mit einem Zeitstempel werden in einem eigenen Karteiblatt angezeigt (siehe Abbildung 5.7).

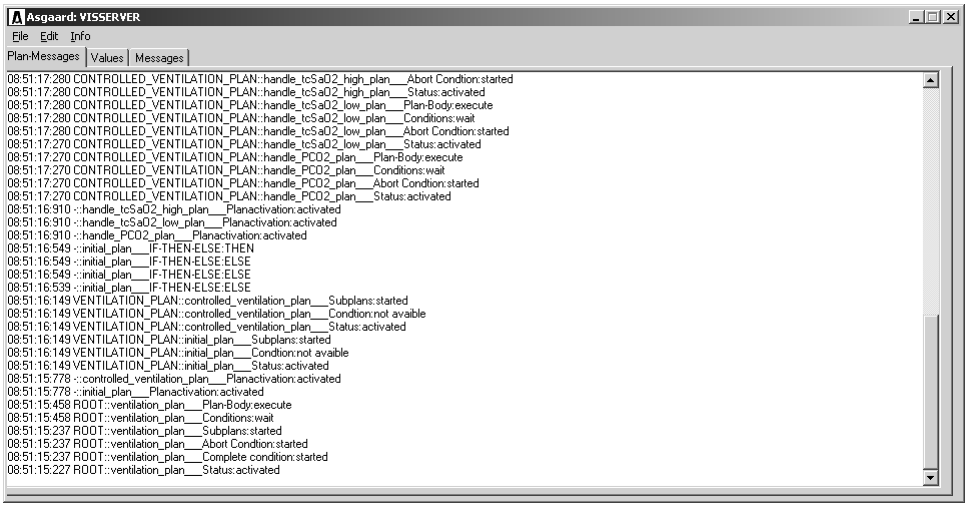

Abbildung 5.6: Benutzeroberfläche: Planmeldungen

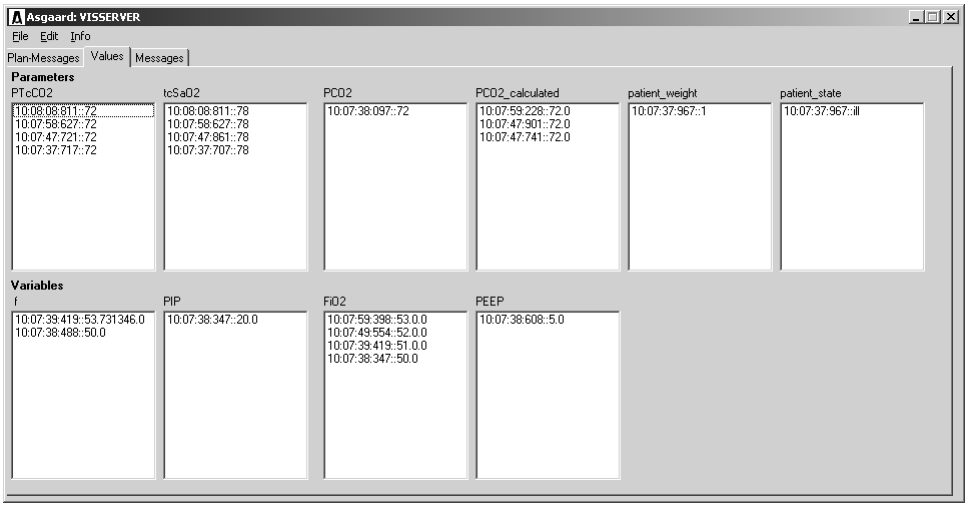

Abbildung 5.7: Benutzeroberfläche: Werte von Parametern und Variablen

Einen Ausschnitt aus dem Ablauf der Kontrollierten Beatmung der simulierten Patientin ist aus der Abbildung 5.8 ersichtlich. Die Parameter $tcSaO_2$ und $PtcCO_2$ wurden aus den Realdaten simuliert, die Variablen FiO_2, f, PEEP und PIP werden von der Ausführeinheit basierend auf dem Asbru-Plan gesetzt.

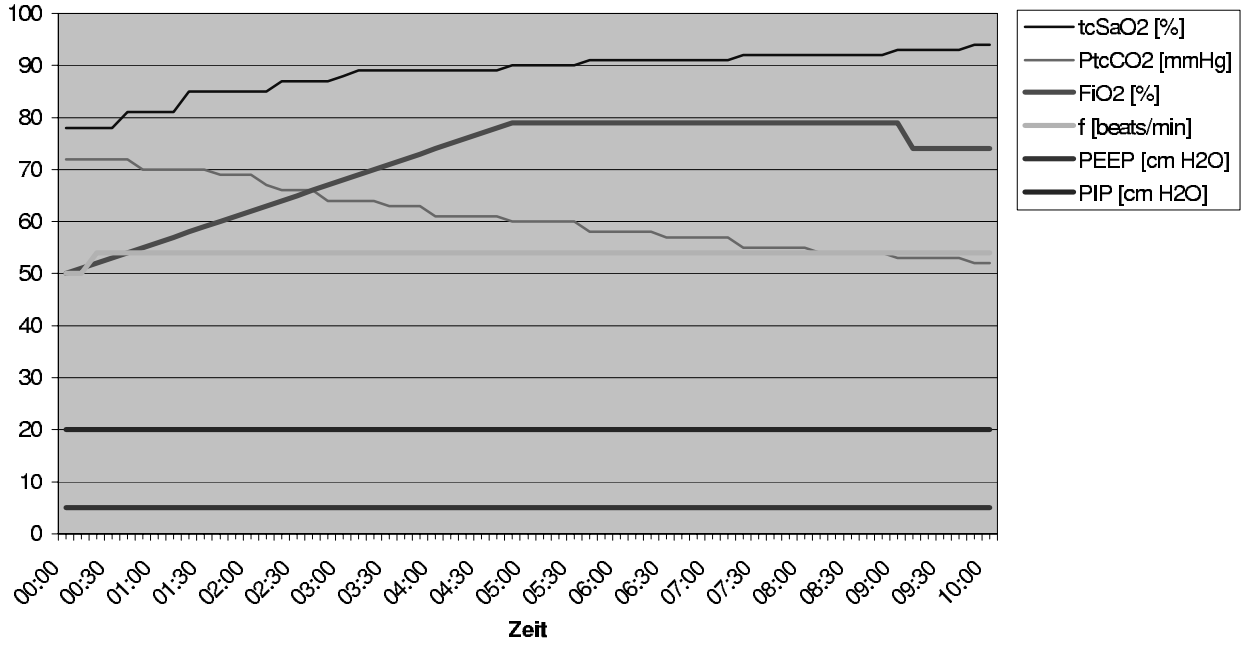

Abbildung 5.8: Diagramm Simulation Fallbeispiel

Folgende Abläufe sind aus der Simulation ersichtlich:

- **Grundeinstellungen:** Nachdem der Zustand und das Gewicht der Patientin bekannt ist, werden die Variablen auf Standardwerte laut Tabelle 2.2 gesetzt.
- **Stabilisierung von $tcSaO_2$:** Da die Sauerstoffsättigung des Blutes viel zu niedrig ist, wird alle 5 Sekunden FiO_2 erhöht. Nach ca. 5 Minuten befindet sich die Sauerstoffsättigung im Normalbereich (siehe Tabelle 2.3) und FiO_2 muss nicht mehr erhöht werden. Nach ca. 8:40 Minuten überschreitet die Sättigung den Normbereich und somit muss FiO_2 wiederum vermindert werden.
- **Stabilisierung von $PtcCO_2$:** Der partieller CO_2 Druck im Blut ist viel zu hoch (siehe Tabelle 2.3) , deshalb wird die Beatmungsfrequenz laut Formel 2.2 angepaßt

- **PEEP und PIP:** Werden konstant gehalten, da sie erst in zweiter Instanz (wenn FiO_2 und f allein nicht mehr stabilisieren können) geändert werden.

5.2 Erkenntnisse

Die Erkenntnisse die im Rahmen dieser Arbeit, insbesondere bei der Erstellung des Asbru Planes für die Kontrollierte Beatmung und in der Evaluierungsphase, gewonnen wurden, werden in diesem Abschnitt angeführt.

Erstellung von Medizinischen Leitlinien ist komplex
Schon bei der Sichtung des verfügbaren Wissens für die Kontrollierte Beatmung von Früh- und Neugeborenen ist aufgefallen, dass die Informationen zum Großteil unvollständig bezüglich einer algorithmischen Verarbeitung sind. Werte sind nur qualitativ angegeben (bei zu hohem PEEP abbrechen), Werte von Intervallen überschneiden sich, zeitliche Vorgaben sind nicht computerverarbeitbar: Alle 10 Sekunden wird eine Messung durchgeführt und alle 10 Sekunden soll mit diesen Werten eine Entscheidung getroffen werden. Dies führt unweigerlich dazu, dass die Entscheidung teilweise mit „alten“ Messergebnisse getroffen wird.

Asbru ist sehr gut für die Formalisierung geeignet
Nachdem eine Medizinische Leitlinie erarbeitet worden ist, wurde mit der Formalisierung begonnen. Dabei hat sich Asbru als sehr gutes Werkzeug für die Formalisierung herausgestellt. Durch die vielfältigen Möglichkeiten der Sprache, konnte die Leitlinie uneingeschränkt formalisiert werden und alle Abläufe konnten entsprechend ihrere (zeitlichen) Definition eins zu eins in den Asbru Plan übernommen werden.

AsbruRTM unterstützt das Testen von Plänen
Bei der Simulation des Fallbeispiels konnte man einige kritische Stellen in der Medizinischen Leitlinie erkennen und ausbessern. So etwa war es möglich, dass ein Plan sofort nach dem Starten beendet wird, da eine „Complete Condition“ schon durch das Setzen von Startwerten erfüllt wird. Daher mußte diese „Complete Condition“ entsprechend angepaßt werden.
Somit ist AsbruRTM ein nützliches Werkzeug zum Testen und Erstellen

von Plänen und kann diesen Prozess positiv unterstützen.

AsbruRTM umfasst viele informatische Teilbereiche
Bei dem Design und bei der Implementierung der AsbruRTM hat sich immer mehr gezeigt, dass nicht nur die Funktionalität von Asbru Light' berücksichtigt werden muss, sondern auch eine Reihe von weiteren informatischen Problemfeldern: Inwieweit sollen beziehungsweise müssen die AsbruRTM echtzeitfähig sein? Wie kann die gewünschte parallele Verarbeitung erreicht werden? Welche zeitlichen Limits ergeben sich daraus? Welche Schnittstellen werden benötigt, um eine breite Anbindung von anderen Applikationen an die AsbruRTM zu gewährleisten?

Anhang A

Glossar

Absichten Intentionen, engl. Intentions.

Ausführeinheit engl. Executionmodule.

Bedingungen engl. Conditions.

closed loop control Ist eine geschlossene Prozesskoppelung. Die Aufgabe des Menschen beschränkt sich auf die Beobachtung des Prozesses.

CORBA Common Object Request Broker Architecture wurde von der Object Management Group (OMG), einem Standardisierungsgremium mit mehr als 700 Mitgliedern, 1991 in der ersten Version definiert. CORBA war eine Antwort auf die starke Zunahme von Hardware- und Software-Produkten und Ziel war es, eine Middleware zu schaffen, welche eine orts-, plattform- und implementations-unabhängige Kommunikation zwischen Applikationen erlaubt [7].

Datenabstraktionseinheit engl. Dataabstractionsmodule.

Effekte engl. Effects.

Hashtabelle Assoziativer Speicher, der Schlüssel auf Werte abbildet.

HL7 *Health Level Seven* Standard für die Kommunikation auf der Ebene 7 des ISO/OSI-Refernzmodells für die Kommunikation. HL7 beschreibt, zu welchen Ereignissen Nachrichten mit welchem Aufbau zwischen Anwendungsbausteinen im Gesundheitswesen und speziell im Krankenhaus ausgetauscht werden [46].

Interpreter Eine Software, die ein ausführbares Programm liest (hier Asbru-Pläne), ausführt und dann ein Ergebnis liefert.

Klasse Abstrakte Beschreibung der Eigenschaften und Fähigkeiten vieler ähnlicher Objekte.

Objekt Eine Einheit mit Eigenschaften und Fähigkeiten. Eigenschaften können ihren Wert während der Lebensdauer dieser Einheit ändern. Das Ergebnis der Durchführung von Fähigkeiten wird durch den jeweils aktuellen Zustand der Einheit beeinflußt.

Parser Ist ein Programm, das einen Eingabetext bezüglich einer gegebenen Grammatik auf die syntaktische Struktur hin analysiert und zerlegt.

Plankorpus engl. Planbody.

Prozeß Ein Prozeß ist der Ablauf eines sequentiellen Programms in einer Rechenanlage. Der Unterschied zum Programm besteht in seiner zeitlichen Dimension [44]. Siehe auch Thread.

3-Schichten-Modell Bezeichnet die Aufteilung einer Applikation in drei Schichten: Präsentation, Anwendungslogik und Datenbank.

Thread Einem Prozeß, dem nur die minimale Menge an Kontextinformationen zugeordnet ist, nennt man leichtgewichtigen Prozeß oder Thread [44]. Siehe auch Prozeß.

Timerthread Thread dessen Hauptroutine in regelmäßigen Abschnitten neu ausgeführt wird. Siehe auch Thread.

Überwachungseinheit engl. Monitoringmodule.

UML *Unified Modeling Language* Ist eine graphische Sprache, die sich im gesamten Software-Entwicklungsprozeß, von der konzeptionellen Analyse bis zur Beschreibung der Implementierung einsetzen läßt. UML enthält aber kein Vorgehensmodell, welches die methodische Entwicklung von Software angibt [12].

XML *eXtensible Markup Language* Beschreibt eine Klasse von computergespeicherten Datenobjekten und beschreibt das Verhalten von Programmen die diese Objekte ausführen. Diese Objekte werden XML Dokumente genannt. XML ist SGML (Standard Generalized Markup)-Anwendung [12].

Zyklischer Plan engl. Cyclical Plan.

Anhang B

Plan Kontrollierte Beatmung

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Christian Fuchsberger Version 1-->
<!DOCTYPE plan-library SYSTEM "asbru_7_3.dtd">
<plan-library>
 <domain-defs>
  <domain name="controlled_ventilation_domain">
   <unit-def name="mgdl" default-unit="mg/dl">
    <compound-def>
     <numerator>
      <unit-class name="numerator"/>
     </numerator>
     <denominator>
      <unit-class name="volume"/>
     </denominator>
    </compound-def>
   </unit-def>
   <qualitative-scale-def name="patient_state_typ">
    <qualitative-entry entry="healthy"/>
    <qualitative-entry entry="slight_ill"/>
    <qualitative-entry entry="ill"/>
    <qualitative-entry entry="very_ill"/>
   </qualitative-scale-def>
   <!-############################################################-->
   <parameter-group title="raw_data_blood_gase_online">
    <parameter-def name="tcSaO2" type="parts">
     <raw-data-def mode="automatic" unit="%" channel-name="tcSaO2"/>
     <sampling-frequency>
      <numerical-constant value="10"/>
     </sampling-frequency>
```

```
30      </parameter-def>
        <parameter-def name="PtcCO2" type="pressure">
32       <raw-data-def mode="automatic" unit="%" channel-name="PtcCO2"/>
         <sampling-frequency>
34        <numerical-constant value="10"/>
         </sampling-frequency>
36      </parameter-def>
       </parameter-group>
38     <parameter-group title="raw data blood gase offline">
        <parameter-def name="PCO2" type="pressure">
40       <raw-data-def mode="manual" user-text="PCO2"/>
        </parameter-def>
42     </parameter-group>
       <parameter-group title="parameters calculated">
44      <parameter-def name="PCO2_calculated" type="pressure">
         <calculation-def operator="multiply">
46        <parameter-ref name="PtcCO2"/>
          <calculation-def operator="divide">
48         <parameter-ref name="PtcCO2"/>
           <parameter-ref name="PCO2"/>
50        </calculation-def>
         </calculation-def>
52      </parameter-def>
       </parameter-group>
54     <parameter-group title="parameters ask">
        <parameter-def name="patient_weight" type="kg">
56       <raw-data-def mode="manual" user-text="Weight of the patient [kg]?"/>
        </parameter-def>
58      <parameter-def name="patient_state" type="patient_state_typ">
         <raw-data-def mode="manual" user-text="State of the patient?"/>
60      </parameter-def>
       </parameter-group>
62     <!--#################################################################-->
       <variable-def name="f">
64      <scalar-def type="beats"/>
       </variable-def>
66     <variable-def name="PIP">
        <scalar-def type="pressure"/>
68     </variable-def>
       <variable-def name="FiO2">
70      <scalar-def type="pressure"/>
       </variable-def>
```

```
72      <variable-def name="PEEP">
          <scalar-def type="pressure"/>
74      </variable-def>
      </domain>
76  </domain-defs>
    <plans>
78    <plan-group>
        <plan name="ventilation_plan">
80        <intentions>
            <intention type="overall-state" verb="achieve">
82            <parameter-proposition parameter-name="PEEP">
                <value-description type="less-or-equal">
84                <numerical-constant value="3"/>
                </value-description>
86              <context>
                  <any/>
88              </context>
                <time-annotation>
90                <any/>
                </time-annotation>
92            </parameter-proposition>
            </intention>
94        </intentions>
          <conditions>
96          <complete-condition>
              <constraint-combination type="and">
98              <parameter-proposition parameter-name="FiO2">
                  <value-description type="less-or-equal">
100                 <numerical-constant value="40"/>
                  </value-description>
102               <context>
                    <any/>
104               </context>
                  <time-annotation>
106                 <any/>
                  </time-annotation>
108             </parameter-proposition>
                <parameter-proposition parameter-name="PIP">
110               <value-description type="less-or-equal">
                    <numerical-constant value="20"/>
112               </value-description>
                  <context>
```

```
114           <any/>
              </context>
116           <time-annotation>
               <any/>
118           </time-annotation>
             </parameter-proposition>
120          <parameter-proposition parameter-name="PCO2_calculated">
              <value-description type="less-or-equal">
122            <numerical-constant value="60"/>
              </value-description>
124           <context>
               <any/>
126           </context>
              <time-annotation>
128            <any/>
              </time-annotation>
130          </parameter-proposition>
             <parameter-proposition parameter-name="PCO2_calculated">
132           <value-description type="greater-than">
               <numerical-constant value="40"/>
134           </value-description>
              <context>
136            <any/>
              </context>
138           <time-annotation>
               <any/>
140           </time-annotation>
             </parameter-proposition>
142          <parameter-proposition parameter-name="tcSaO2">
              <value-description type="less-or-equal">
144            <numerical-constant value="92"/>
              </value-description>
146           <context>
               <any/>
148           </context>
              <time-annotation>
150            <any/>
              </time-annotation>
152          </parameter-proposition>
             <parameter-proposition parameter-name="tcSaO2">
154           <value-description type="greater-than">
               <numerical-constant value="90"/>
```

```
156           </value-description>
              <context>
158            <any/>
              </context>
160           <time-annotation>
               <any/>
162           </time-annotation>
             </parameter-proposition>
164         </constraint-combination>
           </complete-condition>
166        <abort-condition>
            <constraint-combination type="or">
168          <parameter-proposition parameter-name="FiO2">
              <value-description type="greater-than">
170            <numerical-constant value="90"/>
              </value-description>
172           <context>
               <any/>
174           </context>
              <time-annotation>
176            <any/>
              </time-annotation>
178          </parameter-proposition>
             <parameter-proposition parameter-name="PIP">
180           <value-description type="greater-than">
               <numerical-constant value="25"/>
182           </value-description>
              <context>
184            <any/>
              </context>
186           <time-annotation>
               <any/>
188           </time-annotation>
             </parameter-proposition>
190          <parameter-proposition parameter-name="PCO2_calculated">
              <value-description type="greater-than">
192            <numerical-constant value="100"/>
              </value-description>
194           <context>
               <any/>
196           </context>
              <time-annotation>
```

```
198             <any/>
                </time-annotation>
200           </parameter-proposition>
            </constraint-combination>
202       </abort-condition>
        </conditions>
204     <plan-body>
        <subplans type="sequentially">
206       <wait-for>
          <static-plan-pointer plan-name=""/>
208       </wait-for>
          <plan-activation>
210         <plan-schema name="initial_plan">
            <comment text="initial_plan"/>
212         </plan-schema>
          </plan-activation>
214       <plan-activation>
          <plan-schema name="controlled_ventilation_plan">
216         <comment text="controlled_ventilation_plan"/>
          </plan-schema>
218       </plan-activation>
        </subplans>
220   </plan-body>
    </plan>
222 <!--#################################################################-->
    <plan name="initial_plan">
224   <plan-body>
        <subplans type="sequentially">
226       <wait-for>
            <none/>
228       </wait-for>
          <ask>
230         <parameter-ref name="patient_weight"/>
            <time-out>
232           <now/>
            </time-out>
234       </ask>
          <ask>
236         <parameter-ref name="patient_state"/>
            <time-out>
238           <now/>
            </time-out>
```

```
240         </ask>
            <ask>
242          <parameter-ref name="PCO2"/>
             <time-out>
244           <now/>
             </time-out>
246         </ask>
            <if-then-else>
248          <simple-condition>
              <comparison type="equal">
250            <left-hand-side>
                <parameter-ref name="patient_state"/>
252            </left-hand-side>
               <right-hand-side>
254             <qualitative-constant value="healthy"/>
               </right-hand-side>
256           </comparison>
             </simple-condition>
258          <then-branch>
              <variable-assignment variable="FiO2">
260            <numerical-constant value="1"/>
              </variable-assignment>
262           <variable-assignment variable="PIP">
               <numerical-constant value="10"/>
264           </variable-assignment>
              <variable-assignment variable="f">
266            <numerical-constant value="40"/>
              </variable-assignment>
268           <variable-assignment variable="PEEP">
               <numerical-constant value="3"/>
270           </variable-assignment>
             </then-branch>
272         </if-then-else>
            <if-then-else>
274          <simple-condition>
              <comparison type="equal">
276            <left-hand-side>
                <parameter-ref name="patient_state"/>
278            </left-hand-side>
               <right-hand-side>
280             <qualitative-constant value="slight_ill"/>
               </right-hand-side>
```

```
282           </comparison>
             </simple-condition>
284          <then-branch>
              <variable-assignment variable="FiO2">
286            <numerical-constant value="35"/>
              </variable-assignment>
288           <variable-assignment variable="PIP">
               <numerical-constant value="15"/>
290           </variable-assignment>
              <variable-assignment variable="f">
292            <numerical-constant value="40"/>
              </variable-assignment>
294           <variable-assignment variable="PEEP">
               <numerical-constant value="4"/>
296           </variable-assignment>
             </then-branch>
298         </if-then-else>
            <if-then-else>
300          <simple-condition>
              <comparison type="equal">
302            <left-hand-side>
                <parameter-ref name="patient_state"/>
304            </left-hand-side>
               <right-hand-side>
306             <qualitative-constant value="ill"/>
               </right-hand-side>
308           </comparison>
             </simple-condition>
310          <then-branch>
              <variable-assignment variable="FiO2">
312            <numerical-constant value="50"/>
              </variable-assignment>
314           <variable-assignment variable="PIP">
               <numerical-constant value="20"/>
316           </variable-assignment>
              <variable-assignment variable="f">
318            <numerical-constant value="50"/>
              </variable-assignment>
320           <variable-assignment variable="PEEP">
               <numerical-constant value="5"/>
322           </variable-assignment>
             </then-branch>
```

```
      </if-then-else>
      <if-then-else>
       <simple-condition>
        <comparison type="equal">
         <left-hand-side>
          <parameter-ref name="patient_state"/>
         </left-hand-side>
         <right-hand-side>
          <qualitative-constant value="very_ill"/>
         </right-hand-side>
        </comparison>
       </simple-condition>
       <then-branch>
        <variable-assignment variable="FiO2">
         <numerical-constant value="100"/>
        </variable-assignment>
        <variable-assignment variable="PIP">
         <numerical-constant value="23"/>
        </variable-assignment>
        <variable-assignment variable="f">
         <numerical-constant value="60"/>
        </variable-assignment>
        <variable-assignment variable="PEEP">
         <numerical-constant value="5"/>
        </variable-assignment>
       </then-branch>
      </if-then-else>
     </subplans>
    </plan-body>
   </plan>
   <!--#########################################################################-->
   <plan name="controlled_ventilation_plan">
    <plan-body>
     <subplans type="parallel">
      <wait-for>
       <static-plan-pointer plan-name=""/>
      </wait-for>
      <plan-activation>
       <plan-schema name="handle_PCO2_plan">
        <comment text="handle_PCO2_plan"/>
       </plan-schema>
      </plan-activation>
```

```
366         <plan-activation>
             <plan-schema name="handle_tcSaO2_low_plan">
368           <comment text="handle_tcSaO2_low_plan"/>
             </plan-schema>
370         </plan-activation>
            <plan-activation>
372           <plan-schema name="handle_tcSaO2_high_plan">
               <comment text="handle_tcSaO2_high_plan"/>
374           </plan-schema>
            </plan-activation>
376       </subplans>
        </plan-body>
378   </plan>
      <!--#################################################################-->
380   <plan name="handle_PCO2_plan">
       <conditions>
382      <abort-condition>
          <parameter-proposition parameter-name="PtcCO2">
384         <value-description type="greater-than">
             <numerical-constant value="100"/>
386         </value-description>
            <context>
388           <any/>
            </context>
390         <time-annotation>
             <any/>
392         </time-annotation>
          </parameter-proposition>
394      </abort-condition>
       </conditions>
396    <plan-body>
        <cyclical-plan label="handle_PCO2_plan">
398       <start-time>
           <cyclical-time-annotation>
400          <time-range/>
             <set-of-cyclical-time-points>
402            <time-point>
                <numerical-constant scale="time" unit="min" value="0"/>
404            </time-point>
               <offset>
406              <numerical-constant scale="time" unit="min" value="0"/>
               </offset>
```

```
408           <frequency>
               <numerical-constant scale="time" unit="min" value="5"/>
410           </frequency>
             </set-of-cyclical-time-points>
412         </cyclical-time-annotation>
           </start-time>
414        <cyclical-plan-body>
            <subplans type="sequentially">
416          <wait-for>
              <none/>
418          </wait-for>
             <if-then-else>
420           <simple-condition>
              <comparison type="greater-than">
422            <left-hand-side>
                <parameter-ref name="PtcCO2"/>
424            </left-hand-side>
               <right-hand-side>
426             <numerical-constant value="60"/>
               </right-hand-side>
428           </comparison>
             </simple-condition>
430          <then-branch>
              <variable-assignment variable="f">
432            <operation operator="multiply">
                <operation operator="divide">
434              <parameter-ref name="PtcCO2"/>
                 <operation operator="add">
436                <parameter-ref name="PtcCO2"/>
                   <numerical-constant value="5"/>
438              </operation>
                </operation>
440             <variable-ref name="f"/>
               </operation>
442           </variable-assignment>
             </then-branch>
444         </if-then-else>
            <if-then-else>
446          <simple-condition>
              <comparison type="less-than">
448            <left-hand-side>
                <parameter-ref name="PtcCO2"/>
```

```
450             </left-hand-side>
                <right-hand-side>
452              <numerical-constant value="40"/>
                </right-hand-side>
454            </comparison>
             </simple-condition>
456          <then-branch>
              <variable-assignment variable="f">
458            <operation operator="multiply">
                <operation operator="divide">
460              <parameter-ref name="PtcCO2"/>
                 <operation operator="subtract">
462                <parameter-ref name="PtcCO2"/>
                   <numerical-constant value="5"/>
464              </operation>
                </operation>
466             <variable-ref name="f"/>
               </operation>
468           </variable-assignment>
            </then-branch>
470       </if-then-else>
        </subplans>
472     </cyclical-plan-body>
      <repeat-specification>
474     <retry-delay>
         <minimum>
476        <numerical-constant scale="time" unit="min" value="5"/>
         </minimum>
478      <maximum>
          <numerical-constant scale="time" unit="min" value="10"/>
480      </maximum>
        </retry-delay>
482   </repeat-specification>
    </cyclical-plan>
484  </plan-body>
  </plan>
486 <!--#################################################################-->
  <plan name="handle_tcSaO2_low_plan">
488  <conditions>
      <abort-condition>
490     <parameter-proposition parameter-name="FiO2">
         <value-description type="greater-than">
```

```
492           <numerical-constant value="100"/>
              </value-description>
494           <context>
               <any/>
496           </context>
              <time-annotation>
498            <any/>
              </time-annotation>
500          </parameter-proposition>
            </abort-condition>
502        </conditions>
           <plan-body>
504         <cyclical-plan label="handle_tcSaO2_low_plan">
             <start-time>
506           <cyclical-time-annotation>
               <time-range/>
508            <set-of-cyclical-time-points>
                <time-point>
510              <numerical-constant scale="time" unit="min" value="0"/>
                </time-point>
512             <offset>
                 <numerical-constant scale="time" unit="min" value="0"/>
514             </offset>
                <frequency>
516              <numerical-constant scale="time" unit="sec" value="10"/>
                </frequency>
518            </set-of-cyclical-time-points>
              </cyclical-time-annotation>
520         </start-time>
            <cyclical-plan-body>
522          <subplans type="sequentially">
              <wait-for>
524            <none/>
              </wait-for>
526           <if-then-else>
               <simple-condition>
528             <comparison type="less-than">
                 <left-hand-side>
530                <parameter-ref name="tcSaO2"/>
                 </left-hand-side>
532              <right-hand-side>
                  <numerical-constant value="90"/>
```

```
534             </right-hand-side>
               </comparison>
536           </simple-condition>
              <then-branch>
538            <variable-assignment variable="FiO2">
                <operation operator="add">
540              <variable-ref name="FiO2"/>
                 <numerical-constant value="10"/>
542              </operation>
               </variable-assignment>
544           </then-branch>
             </if-then-else>
546         </subplans>
          </cyclical-plan-body>
548       <repeat-specification>
           <retry-delay>
550          <minimum>
              <numerical-constant scale="time" unit="sec" value="10"/>
552          </minimum>
             <maximum>
554            <numerical-constant scale="time" unit="sec" value="10"/>
             </maximum>
556          </retry-delay>
          </repeat-specification>
558     </cyclical-plan>
      </plan-body>
560   </plan>
      <!--#################################################################-->
562   <plan name="handle_tcSaO2_high_plan">
       <conditions>
564      <abort-condition>
          <parameter-proposition parameter-name="FiO2">
566         <value-description type="greater-than">
              <numerical-constant value="100"/>
568         </value-description>
            <context>
570           <any/>
            </context>
572         <time-annotation>
              <any/>
574         </time-annotation>
          </parameter-proposition>
```

```
576       </abort-condition>
        </conditions>
578     <plan-body>
         <cyclical-plan label="handle_tcSaO2_high_plan">
580       <start-time>
           <cyclical-time-annotation>
582          <time-range/>
             <set-of-cyclical-time-points>
584            <time-point>
                 <numerical-constant scale="time" unit="min" value="0"/>
586            </time-point>
               <offset>
588              <numerical-constant scale="time" unit="min" value="0"/>
               </offset>
590            <frequency>
                 <numerical-constant scale="time" unit="min" value="5"/>
592            </frequency>
             </set-of-cyclical-time-points>
594        </cyclical-time-annotation>
         </start-time>
596      <cyclical-plan-body>
          <subplans type="sequentially">
598         <wait-for>
              <none/>
600         </wait-for>
            <if-then-else>
602           <simple-condition>
                <comparison type="greater-than">
604               <left-hand-side>
                    <parameter-ref name="tcSaO2"/>
606               </left-hand-side>
                  <right-hand-side>
608                 <numerical-constant value="92"/>
                  </right-hand-side>
610           </comparison>
            </simple-condition>
612         <then-branch>
              <variable-assignment variable="FiO2">
614             <operation operator="subtract">
                  <variable-ref name="FiO2"/>
616               <numerical-constant value="5"/>
                </operation>
```

```
618             </variable-assignment>
              </then-branch>
620           </if-then-else>
            </subplans>
622         </cyclical-plan-body>
          <repeat-specification>
624         <retry-delay>
            <minimum>
626           <numerical-constant scale="time" unit="min" value="2"/>
            </minimum>
628         <maximum>
              <numerical-constant scale="time" unit="min" value="10"/>
630         </maximum>
          </retry-delay>
632       </repeat-specification>
        </cyclical-plan>
634     </plan-body>
      </plan>
636   </plan-group>
    </plans>
638 </plan-library>
```

Literaturverzeichnis

[1] Agency for Healthcare Research and Quality. *Quality Research for Quality Healthcare*. http://www.ahcpr.gov

[2] Arbeitsgemeinschaft der Wissenschaftlichen Medizinischen Fachgesellschaften (AWMF). *Wissenschaftlich begründete Leitlinien für Diagnostik und Therapie*. http://www.leitlinien.net

[3] Arden Syntax Group. *The Arden Syntax for Medical Logic Systems*. http://cslcinfmtcs.csmc.edu/hl7/arden

[4] Bosse: *An interpreter for clinical guidelines in Asbru*. De Boelelaan 1081-83, 1081 HV Amsterdam, Vrije Universiteit Amsterdem, Diplomarbeit, 2001

[5] Bury ; Fox ; Sutton: The PROforma guideline specification language: progress and prospects. In: *Proceedings of the First European Workshop, Computer-based Support for Clinical Guidelines and Protocols (EWGLP 2000)* (2000)

[6] Dazzi ; Fassino ; Saracco ; Quaglini ; Stefanelli: A Patient Workflow Management System Built on Guidelines. In: *AMIA Annual Fall Symposium* (1997)

[7] Distributed Technologies GmbH. *Corba*. http://www.corba.ch/corba.html

[8] Elkin ; Peleg ; Lacson [u. a.]: Toward Standardization of Electronic Guidelines. In: *MD Computing* 17 (2000), Nr. 6, S. 39–44

[9] EMIS. *Egton Medical Information Systems*. http://www.emis-online.com/

[10] ExoLab Group. *Castor*. http://castor.exolab.org

[11] Field ; Lohr: *Clinical Practice Guidelines: Directions for a New Program.* Washington, DC : National Academy Press, 1990

[12] GI Gesellschaft für Informatik e.V. *Informatik Lexikon.* http://www.gi-ev.de/informatik/lexikon/inf-lex-uml.shtml

[13] Goldsmith ; Karotkin: *Assisted Ventialtion of the Neonates.* 3. Philadelphia : Saunders, 1993

[14] Helios Klinikum Berlin. *Neonatologie.* http://www.kinderklinik-buch.de

[15] Johnson ; Tu ; Booth [u. a.]: Using Scenarios in Chronic Disease Management Guidelines for Primary Care. In: *Proc. AMIA Annual Symposium* (2000)

[16] Kopetz ; Zainlinger ; Fohler ; Kantz ; Puschner ; Schütz: An Engineering Approach to Hard Real-Time System Design. In: *Proc. 3rd European Software Engineering Conference*, 1991, S. 166–188

[17] Kosara ; Miksch. *Pontifex.* http://www.asgaard.tuwien.ac.at/tools/pontifex.html

[18] Laboratory, Advanced C. *Early Referrals Applications.* http://www.acl.icnet.uk/lab/era

[19] Larsen ; Ziegenfuss: *Beatmung.* 2. Springer, 1999

[20] Miksch ; Seyfang: Continual Planning with Time-Oriented, Skeletal Plans. In: *ECAI 2000 Proceedings of the 14th European Conference on Artificial Intelligence* (2000), S. 511–514

[21] Miksch ; Shahar ; Johnson: Asbru: A Task-Specific, Intention-Based, and Time-Oriented Language for Representing Skeletal Plans. In: *7th Workshop on Knowledge Engineering: Methods & Languages (KEML-97)* (1997)

[22] Miksch ; Shahar ; Johnson: Medizinische Leitlinien und Protokolle: das Asgaard/Asbru Projekt. In: *KI-Journal* (1997), Nr. 3, S. 34–37

[23] Miksch ; Shahar ; Johnson: The Asgaard project: A task-specific framework for the application and critiquing of timeoriented clinical guidelines. In: *Artificial Intelligenced in Medicine* (1998), Nr. 14, S. 29–51

[24] Musen ; Tu: From Guideline Modeling to Guideline Execution: Defining Guideline-Based Decision-Support Services. In: *Proc. AMIA Symposium* (2000), S. 863–867

[25] Musen ; Tu ; Das ; Shahar: EON: A component-based approach to automation of protocol-directed therapy. In: *Journal of the American Medical Information Association* 6 (1996), S. 367–388

[26] Ohno-Machado ; Gennari ; Murphy [u. a.]: The guideline interchange format: a model for representing guidelines. In: *Journal of the American Medical Informatics Association* 4 (1998), S. 357–372

[27] Ollenschläger ; Kirchner ; Fiene: Leitlinien in der Medizin - scheitern sie an der praktischen Umsetzung? In: *Internist* (2001), Nr. 42, S. 473–483

[28] Openclinical. *Knowledge management for medical care.* `http://www.openclinical.org`

[29] Peleg ; Ogunyemi ; Tu ; Boxwala ; Zeng ; Greenes ; Shortliffe: Using features of Arden Syntax with object-oriented medical data models for guideline modeling. In: *Proc AMIA Symp* 7 (2001), S. 523

[30] Peleg ; Tu ; Bury ; Ciccarese ; Fox ; Greenes ; Hall ; Johnson ; Jones ; Kumar ; Miksch ; Quaglini ; Seyfang ; Shortliffe ; Stefanelli: Comparing Models of Decision and Action for Guideline-Based Decision Support: a Case-Study. In: *SMI Technical Report* (2001)

[31] Popow. *Beatmung von Früh- und Neugeborenen.* `http://www.akh-wien.ac.at/kikli/neonat/popow/beat-skr.htm`. 1996

[32] Porzsolt. *Klinische Ökonomik: Anforderungen an Leitlinien und deren Erfüllbarkeit.* `http://www.ebm-net.de`. 1999

[33] Protocure Consortium. *Reference Protocols.* `http://www.protocure.org/resources-publications.html`

[34] Purves ; Sugden ; Booth ; Sowerby: The PRODIGY project–the iterative development of the release one model. In: *Proc AMIA Symp.* (1999), S. 359–63

[35] Quaglini ; Stefanelli ; Lanzola ; Caporusso ; Panzarasai: Flexible guideline-based patient careflow systems. In: *AI Med* 22 (2001), S. 65–80

[36] Real-Time for Java Expert Group. *Real-Time for Java.* http://www.rtj.org

[37] Roomans: *Formalisation of a medical guideline.* De Boelelaan 1081-83, 1081 HV Amsterdam, Vrije Universiteit Amsterdem, Diplomarbeit, 2001

[38] Sackett ; Rosenberg ; Gray ; Haynes ; Richardson: Evidence based medicine: what it is and what it isn't. In: *British Medical Journal* (1996), Nr. 312, S. 71–72

[39] Seyfang ; Kosara ; Miksch. *Asbru Reference Manual 7.3 Rev. 2.* http://www.asgaard.tuwien.ac.at. 2002

[40] Stanford University School of Medicine. *The InterMed Collaboratory.* http://www.smi.stanford.edu/projects/intermed-web

[41] Taylor ; Alberdi ; Lee. *Computer Aided Decision Making in Image Understanding in Medicine.* http://www.chime.ucl.ac.uk/work-areas/dst/CADMIUM/

[42] Torex. *Providing innovative and leading edge IT solutions to healthcare, government and commercial organisations.* http://www.torexhealth.co.uk/

[43] Verschiedene Autoren: *Pschyrembel Klinisches Wörterbuch.* 258. Berlin : de Gruyter, 1998

[44] Verschiedene Autoren: *Informatik-Handbuch.* 2. München, Wien : HANSER, 1999

[45] Verschiedene Autoren: *Roche-Lexikon Medizin.* 4. München : Urban & Schwarzenberg, 1999

[46] Verschiedene Autoren: *Handbuch der Medizinischen Informatik.* 1. München, Wien : HANSER, 2002

[47] Weiskirchner: *JAVA in Echtzeitsystemen mit Bezug auf die WCET und Analysierbarkeit des Java Bytecodes.* Treitlstr. 3/3/182-1, 1040 Vienna, Austria, Technische Universität Wien, Institut für Technische Informatik, Diplomarbeit, 2001

[48] Zuser ; Biffl ; Grechenig ; Köhle: *Software Engineering mit UML und dem Unified Process.* 1. München : Pearson Studium, 2001