スケジューリング労力低減のための知的エージェント実装の基礎検討

三宅佑磨*　　岡田　正**

Basic Study of an Intellectual Agent Implementation for Scheduling Labor Reduction

Yuuma MIYAKE* and Tadashi OKADA**

In order to improve a quality of communication, we study a system to automate a personal schedule. The central part of the scheduling system is implemented as an intelligent agent build by a hybrid system with production and frame systems. The system is built as a Web system using the open source software such as Ruby, PostgreSQL, FreeBSD. We confirm that the basic operation of a scheduling generation is confirmed for sample tasks.

Key Words: Intellectual Agent, Scheduling Problem, Production and Frame Systems, Web system

１．はじめに

インターネットが普及することにより，人と人とのコミュニケーションは場所・時間を超えて活発に行われるようになった．しかし，コミニュケーションにおいて，実際に会って話すということの価値は変わっておらず，その価値は連絡手段の発達によりむしろ上がっているといえる．直接会うことは最も密にコミニュケーションを取ることができる方法で，インターネットが広まった現在でも変わることはない．

しかし直接会うには互いの予定を調整する必要があり，それは人数が増えるほど煩雑である．このことが複数人で予定を組み合わせて会うことの敷居を上げる原因となっている．この問題を解決するためには,複数人の予定調整にう伴う労力を低減できて,コミュニケーション自体の質を高めるためのスケジューリングシステムが必要であると考えた．

本報告では，スケジューリングを行うためにエキスパートシステム [1-3] を使用した知的エージェントを実装するための基礎検討を行ったので，その内容を述べる．エキスパートシステムの知識表現には，プロダクションとフレームのハイブリッドシステムを用いる．また，スケジューラとして外出先やモバイル端末での使用が想定されるため，ウェブシステムとして実装する．

実装にあたっては，全面的にオープンソースソフトウェアを採用し，Ruby[4] とウェブアプリケーションフレームワークである Ruby on Rails[5] を使い，運用には FreeBSD[6] 上の Apache HTTP Server[7] を用いる．さらに，エキスパートシステムで使用する知識ベースをリレーショナルデータベースで管理しているので，この方法も合わせて報告する．なお，本報告ではシステム開発の中間報告として，個人のスケジューリングまでを扱う．

本報告では，スケジューリング問題とエキスパートシステムとの関係を２章で取り上げる．３章でシステムの要素と実装方法について，４章では実際のシステム実装と動作確認のシミュレーションについて述べる．

２．スケジューリングとエキスパートシステム

２．１　スケジューリングとコミュニケーション

多くの人はスケジュールに則って行動している．現状の多くのスケジューラは，決まった日付にタスクを挿入する型式であるため，新たに会合などを計画する場合,あらかじめ参加者全員の予定を調整し,日付を決める必要がある．これは労力と時間を多大に要する作業で，結果としてコミュニケーションの

原稿受付　平成２４年８月３１日

*専攻科電子・情報システム工学専攻平成２３年度修了生

**情報工学科

敷居を上げる原因となっている．

人の行うスケジュールとは，幾つかのタスクの集まりであるといえる．タスクには，「朝会」「会議」「営業」のように具体的なタスクの開始時刻が決まっているものと，「報告書作成」「読書」「掃除」などのように開始時刻は決まっておらず完了期限が決められているものや，それすらも決まっていない「趣味」のようなものがある．

現状の多くのスケジューラは，具体的な開始時刻が決まっているものしか扱うことができない．開始時刻が決まっていないタスクをスケジュールに組み込むためには，人が考えて入力するしかなく，これに人が費やす労力は多大なものとなる．この開始する時刻が決まっていないタスクを，スケジューリングシステムで扱うことができるようにすることによって，タスクからスケジュールを組むことができる．組まれたスケジュールを複数人分照らし合わせ，空いている時間や重要度の低いタスクを移動させるなどの調整を行い，複数人で共有できる時間を生み出すことになる．

本報告のスケジュール生成には，エキスパートシステム [1-3] を用いた知的エージェントを使用する．ここでいう知的エージェントとは，ユーザに代わって知識を使い，ユーザに最適化されたスケジュールを，ユーザに代わって生成するシステムである．事前に入力したルール知識と事実知識に加え，ユーザが追加した情報に基づき，ユーザが入力したタスクからエキスパートシステムがタスクの優先度を求め，エージェントがスケジュールを作成する．

2.2　エキスパートシステム

エキスパートシステムとは，専門家の知識を取り出し，一定の形式で表現し利用することで問題解決を行うコンピュータシステムのことである [1-3]．特定分野の問題について情報を解析するルール群（知識ベース）と，その知識を使って答えを導きだすための推論エンジンから構成される．特徴として

＊知識ベースを書き換えることで様々な問題に対応可能な汎用性
＊対話式の推論実行
＊推論により得られた結論の根拠を説明可能
＊知識が多少不完全でも処理部分が正しければ動作可能

などを挙げることができる．

エキスパートシステムでは，知識をどのように表現するかが重要で，今回のスケジューリング問題では，プロダクションとフレームによる知識表現を用いる [8]．これらエキスパートシステムの構成要素とその特徴は，次のようになっている．

（1）プロダクションシステム

プロダクションシステムとは，IF-THEN 型のプロダクションルールを知識表現単位としたシステムで，ルール型システムとも呼ばれる．プロダクションシステムを構成するには，

＊ルールの集合であるプロダクション集合
＊事実知識や途中結果を記憶する作業記憶（ワーキングメモリ）
＊推論を制御する推論部

が必要となる．

（2）フレームシステム

フレームシステムとは，フレームを知識表現として使っているシステムのことである．フレームは，知識を表現し格納するための枠組みで，人間の認知活動の研究から生まれた．フレーム内の知識は，属性と値ならびに手続きの組として扱われ，フレーム同士を階層的につなぐことで，あらゆる知識を柔軟に表現・管理できる．

知識の単位がフレームであり，これを構成する多くの属性をスロット，その値をスロット値と呼ぶ．各スロットは，別のフレームへと結合することができ，複雑な知識であっても階層的に表現できる．フレーム内の各スロットの属性や条件を記述したものをファセットと呼ぶ．例えば，他フレームと結合しているかどうか，どのような制約内容があるかなどである．一般的に上位のフレームの値は，下位のフレームに種々の方法で継承される．

（3）ハイブリッドシステム

フレームシステムの長所は，使うフレームを選択することにより柔軟な推論を行えることである．一方，プロダクションシステムの長所は，厳密にルールを記述できることである．この両方の長所を場面に応じて使うため，フレームシステムを大枠に，フレームのスロットにルールを記述するなどして，ハイブリッドシステムとして使うことが多い．

本研究においても，フレームシステムを大枠とし，一部のスロットにプロダクションルールを記述する手法により，柔軟性の高いシステムを目指した．

（4）知識ベース

エキスパートシステムの知識ベースに格納される情報には，『事実知識』『ルール知識』『メタ知識』の３種類がある．事実知識とは事実を単に述べるもの，ルール知識とは事実や処理の規則を表す知識である．事実知識に加えルール知識を持つことがエキスパートシステムの特徴である．一方，メタ知識は，探索条件や適用条件を定めた知識のように，知識を使うための知識である．「一般的なルールより具体的なルールを選択する」「危険なルールより安全なルールを選択する」「費用が少ないルールを優先する」などが，メタ知識の例である．

（5）推論エンジン

推論とは，知識を使って結論を求めていく一連の過程のことをいう．

プロダクションシステムにおける推論の基本は，『事実知識』『ルール知識』『結論』からなる三段論法である．三段論法は無限に組み合わせることができ，この組み合わせの順序と方向により，前向き推論と後ろ向き推論に分けられる．前向き推論は事実（もしくは仮説）からスタートし，ルールから結論へ辿っていき証明を行っていく推論である．これに対して，後ろ向き推論は，証明したい結論からスタートし事実に辿り着いたとき証明を終了する．

一方，フレームシステムでは，知識はファセットに記述されており，直接知識（値）が参照されるか，もしくはデーモン（関数）が呼び出されることで推論が進む．デーモンは特定の値を求める関数のほか，フレームの追加や削除を行う場合がある．フレームの継承機能により，参照したい知識がファセットにない場合，結合されているフレームをたどることで，すべての知識を与えなくても推論が可能になる．

本研究のハイブリッドシステムの場合，オブジェクトとしてフレームを使い，ファセットにプロダクションを記述する．推論の方法はプロダクションシステムと同じ三段論法を使い，同一フレームに結合されているフレームのルールを使う．さらに，状態に応じたフレームを用意し，フレームを使い分けることで，ユーザや場面に最適化した推論を行う．

（６）知識エディタと利用者インタフェース

知識エディタとは，知識ベースが知識を獲得するための道具で，知識ベースの内容をエディタに問い合わせることにより対話的に知識を入力していく．知識の入力には，他にもナレッジエンジニアと呼ばれる人が専門家にインタビューを行い知識を引き出し，知識表現のルールに置き換えていくというものがある．

利用者インタフェースは，利用者が推論を行うためのインタフェースである．対話式の質問に対して利用者が回答していく．回答の方法に，メニュー形式での回答選択，形式言語での回答，自然言語での回答，グラフィック表示とマウスなどでの選択などの方法がある．推論により結論に至った過程を説明する機能も，利用者インタフェースに含まれる．

３．知的エージェントの実装

３.１　スケジューリング問題の定式化

今回扱うスケジューリング問題の内容に近いものの一つに，ナース・スケジューリング問題（NSP）[8,9]がある．NSP とは，ある期間におけるナースの勤務シフトを，各ナースの休み，時間帯ごとの必要なナースの人数など，さまざまな制約の下で求める問題である．

これを参考にし，人の日常におけるスケジュールの生成方法を考える．タスクの分類・生成や割り当て時の制約等は NSP を参考にしながら，エキスパートシステムの手法を取り入れることで，柔軟性の高いスケジューリング生成を工夫している．本システムのスケジュール生成方法は，次の通りである．

(1) ユーザ情報とタスク情報の設定
(2) タスクを予定に追加
(3) 予定に追加されたタスクのタスク情報と拘束条件から重要度の算出
(4) スケジュールの生成
(5) スケジュールの調整と決定

複数人でのタスクが追加された場合，既にスケジュールが組まれていたら空いている時間帯や重要度の低いタスクを移動させそこに追加，スケジュールが組まれていない場合は個人のタスクに追加する．これらの処理を，エキスパートシステムを用いた知的エージェントを使って処理する．エージェントはスケジュールを生成するために，ルール知識（スケジュール拘束条件）と事実知識（タスク情報・ユーザ情報・スケジュール情報）を使う．この二つの知識を照らし合せスケジュールを生成する．

スケジュールの拘束条件を大きく３つに分ける．

(1) 決定済みタスク
(2) タスク拘束条件
(3) スケジュール拘束条件

決定済みタスクは，既にスケジュール開始時刻が確定されているという拘束条件である．タスク拘束条件は，タスクの質を保つための要素であり，続けて行うことが望ましくないタスクの連続割り振りを禁止にするなどの条件である．スケジュール拘束条件とは，タスクの質を保つための要素であり，例えばスケジュール全体から規定の休み時間を下回らないなどの条件となる．

拘束条件には個別に優先度を設け，競合した場合は優先度が高い順にスケジュールを生成していく．優先度が高いもの同士の競合についてはユーザに判断を委ねる．原則として禁止パターンの優先度は高く，次いで好ましくないパターン，望ましいパターンのような優先度を設定する．

事実知識には以下の３つを用いる．

(1) タスク情報
(2) ユーザ情報
(3) スケジュール情報

タスク情報は，タスクに付随する情報で静的なものと動的なものがある．ユーザ情報は，ユーザに関する情報で，静的なものである．これに対して，スケジュール情報は，そのときのスケジュールに左右される動的なものとなる．

ルール知識と事実知識の照らし合わせには，有能な秘書のような専門知識を想定したエキスパートシステムとして実装していく．

3．2　ルール知識の実装

本システムをウェブ上で動作させるため，知識ベースをリレーショナルデータベース(RDB)を使って管理し，プログラミング言語には Ruby[4] を用いて実装している．

（1）フレームの実装

フレームはスロットを集めたものでできており，スロットには別の下層のフレームとなるものもある．スロット中の属性を表すものはファセットに入る．これをまとめると図1のようになる．これを Flame テーブルと Facet テーブルに分け 1 対 n で関連づけることにより，RDB でフレームシステムを表すことができる．この ER 図を図2に示す．

フレームの継承は，Flame テーブルに，継承元 ID (acced_id)カラムを外部キーとし，ID を主キーとして，継承元フレームと継承先フレームを 1 対 n で関連づけることにより実装する．継承元 ID を参照することで親フレームを見つけ出し，親フレームもそれを繰り返す．継承元 ID が Nil であればトップフレームである．

ファセットは Facet テーブルにより表す．Flame テーブルの ID を主キーとして，Facet テーブルのフレーム ID カラムを外部キーとして，1 対 n の関連により実装する．Flame テーブルには，継承元 ID とフレーム名が入る．Facet テーブルには，ファセットが入っているフレームの ID とファセット名・説明・プロダクションルールが入る．

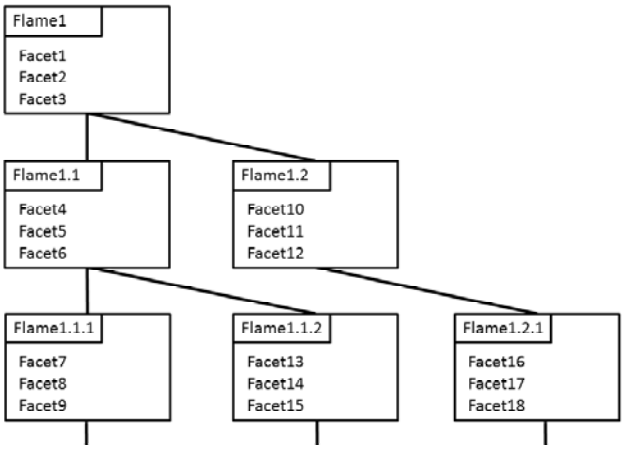

図1　各フレームとファセットの関係

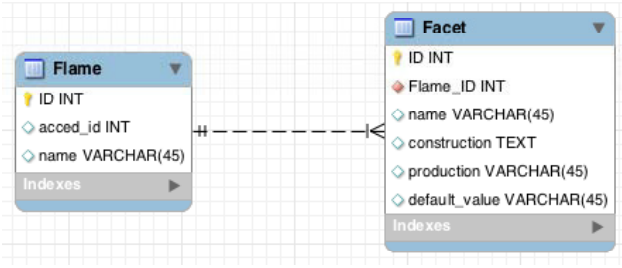

図2　フレームとファセットの ER 図

以上より，継承が行えるフレームと，その内部にある複数のファセットを実装することができた．

（2）ワーキングメモリの実装

ワーキングメモリは，ハッシュテーブル（連想配列）を用いて実現する．状態名をハッシュ名ととし，内容は真偽を持つ場合と数値を用いる場合が考えられるため文字列とする．実際には次のようになる．

```
WorkingMemory = Hash.new
```

ワーキングメモリを追加する場合は，次のように記述する．

```
WorkingMemory[‘状態名’] = (‘内容’)
```

スケジュール生成には t 時におけるタスクの情報を使う．しかし，過去との関連（例えば肉体的労働を続けて行わないなど）を必要とするルールも存在しうるため，過去のワーキングメモリを残す必要がある．過去のワーキングメモリ(PastMemory)を次のように宣言する．

```
PastMomory = Array.new
PastMemory[n] = Hash.new
```

ここで n は t 時のタスクである．スケジュールとして採用された優先度が一番高いタスクは，ワーキングメモリ PastMemory に記録する．

（3）プロダクションルールの実装

プロダクションルールは，Facet テーブル内の production カラムに記述する．既存エキスパートシステムでは，一般に Prolog や Lisp など言語で記述することが多い．しかし，これらの言語は関数型プログラミング言語であり，近年主流である C 言語系の記述ではないため可読性が低い．開発を迅速に行い可読性を向上させるため，今回はシステム開発言語である Ruby で直接記述することにした．

例えば，下記のプロダクションルールは，

```
IF 咳が出る THEN 風邪である
IF 風邪 THEN 薬を飲む（結論）
```

次のように表現する．

```
if WorkingMemory[‘have_a_cough’]
  WorkingMemory[‘have_a_cold’] = true
end
if WorkingMemory[‘have_a_cold’]
  conclusion = ‘take medicine’
end
```

if の条件とワーキングメモリの内容が一致すると，ワーキングメモリに新たな内容を追加したり変更したりする．conclusion は結論であり，本システムではタスクの優先度となる．

（4）事実知識の実装

本システムでは3種類の事実知識（タスク情報，ユーザ情報，スケジュール情報）を使う．それをル

ール知識と同様に RDB で表す方法を考える．

タスク情報はタスクに関する情報である．実際のタスクは「数学の講義」「英語の講義」のように扱う内容は同じでも表現は別であったり，また繰り返しを行うタスクもあり，スケジュールに入力するタスク毎にタスクの全情報を決めるのは好ましくない．そこで，タスク情報を入れる Task テーブルと行動の種類を入れる ActionType テーブルとを別に設け，ActionType テーブルの ID を主キーとし，Task テーブルの ActionType_ID カラムを外部キーとして 1 対 n の関連を作る．

事実知識はルール知識の内容により必要な種類や数が変化するので，これも行動の種類と行動の事実知識とを分け，ActionFact テーブルを設ける．ActionType テーブルの ID を主キーとし，ActionFact テーブルの ActionType_ID カラムを外部キーとして 1 対 n の関連を作る．Task テーブルには，使用する ActionType・名前・必要経過時間（必要ならば）・開始時間（必要ならば）・人数が入る．ActionType には行動名が入る．ActionFact には，事実の種類(FactName)と事実(Fact)が入る．

ActionType と ActionFact は事前に入力しておき，タスク自体(Task)はスケジュール追加したいタスクが発生した時に追加する（３．３で後述）．

（５）ユーザ情報の実装

ユーザ情報もルール知識により内容や種類が変化するので別テーブルとし，User テーブルの ID を主キーとし，UserFact テーブルの User_ID を外部キーとして 1 対 n の関連を作る．User テーブルにはユーザの名前などが入り，UserFact テーブルには事実の種類(FactName)と事実(Fact)が入る．

（６）スケジュール情報

スケジュール情報には，スケジュール生成に使う生成日時や期間などが入る．これはスケジュールを生成するたびに変わるため，HTTP の POST メソッドを使いスケジュール生成毎に入力する．

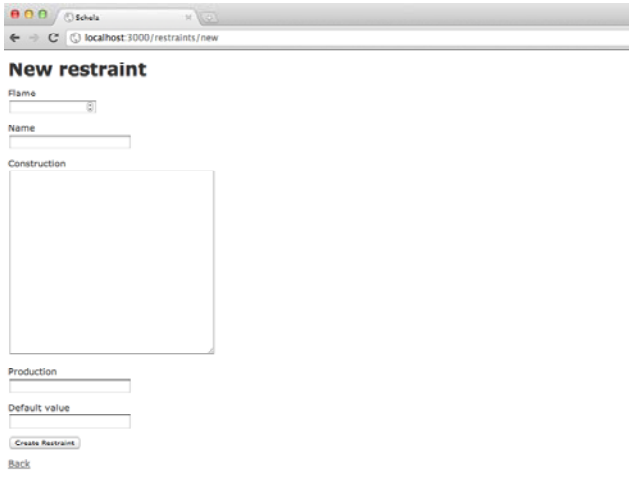

図３　知識入力フォーム

（７）知識エディタ

本来ならば対話的に知識を入力していくことが望ましい．今回は開発を迅速化させるため今後実装するとし，図３のような一般的な入力フォームを用いる．今回，知識の獲得は報告者自身の知識を用いる．

３．３　スケジュール生成の方法

スケジュールに追加したいタスクが生まれたらタスクを追加する．タスク作成に必要な情報は３．２で述べた内容である．名前(Name)はタスクの名前を，ActionType はこのタスクの種類を示す．必要経過時間(Elapsed time)は，ActionType で経過時間が決まっておらず，それを必要とする場合に入力する．人数(People)は ActionType が参加人数を必要とする場合に入力する．開始時刻(Start time)はあらかじめ決まっているタスクのみ入力する．これらにより，ルール知識・事実知識・タスクを獲得し管理することができるので，これらを使ってスケジュール生成が可能となる．

スケジュールは，生成したい期間を時間で区切って考える．今回は 1 日を 1 時間ごとで区切って処理したので，1 日間のスケジュールで 24 個のタスクが入る．挿入するタスクは優先度が一番高かったものであり，優先度はプロダクションルールで次のように書かれている．

```
#開始時刻が決まっているタスクを決定する
if WorkingMemory[‘StartTime’]
  Priority[n] = 1
end
#正午に近い時間帯で昼食を取る
if t<11 && t>14
  Priority[n] += 0.4
end
#連続する肉体労働を避ける
if PastMemory[t-1][‘labor’]
  Priority[n] -= 0.3
end
```

このとき，Priority[n]は t 時におけるタスク n の優先度となり，この最も高いタスクが t 時のタスクとなる．スケジュールは，テーブルにタスク ID と DATE で保存される．0 時におけるタスク ID の Task ID0 から 23 時の Task ID23 まである．DATE はスケジュールの日時である．

４．システム構成とシミュレーション

４．１　システム構成

スケジューリングシステムは，外出先からの閲覧やモバイル環境からの使用も想定されるため，ウェブ対応が必要である．そこで，サーバサイドスクリ

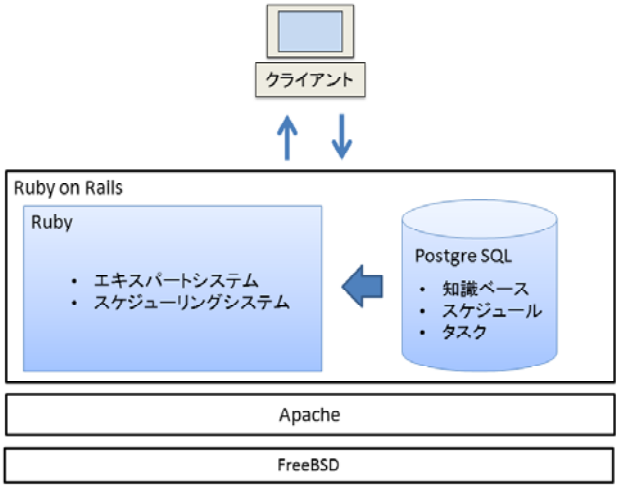

図 4　システム構成

プト言語やデータベース，それらを一括で扱うためのウェブアプリケーションフレームワークを使って実装を行う．システム構成は図 4 のようになる．

サーバサイドスクリプト言語には，可読性を高めるためオブジェクト指向言語の Ruby を，ウェブアプリケーションフレームワークには Ruby で書かれている Ruby on Rails[5] を使用する．データベースには PostgreSQL[10] を採用した．これらを動かすサーバプログラムの開発には，Ruby on Rails に標準搭載されている WEBrick[11] を使った．サーバはすべてオープンソースソフトウェアで構成されており，HTTP サーバの Apache[6] を OS FreeBSD[12] のもとで運用している．

システムの動作は，パソコン・モバイル端末などから，タスクを必要情報とともにデータベースに追加する．タスクが貯まったらスケジュール生成を選択し，知的エージェントを実行してスケジュールを生成する．生成されたスケジュールはデータベースに保存され，いつでも閲覧することができる．

具体的な内部動作は，ブラウザから要求された HTTP リクエストを Apache が受け取り Ruby on Rails に渡す．Ruby on Rails では経路に従い呼び出されたコントローラに情報を渡し，必要ならばモデルを返して PostgreSQL にアクセスし，揃った情報をビューに渡し動的なサイトを生成する．これらの開発では，MacOSX，AptamaStudio3，Vim を用いた．

4．2　シミュレーション

本研究で開発したシステムを使って，実際にタスクを与えスケジュールの獲得を行って，動作シミュレーションを行った．シミュレーションでは，実際的な架空のルール知識と事実知識を使いスケジュールを生成している．スケジュール生成期間を 9 時から 20 時の 12 時間分とし，タスクを追加したり変更したりして，スケジュールを生成させた．その結果，利用に支障のいないスケジュールが生成できており，基本的な動作確認ができた．

ただ，場面に応じたフレームに実装すべきルールや，競合の処理を行うプロダクションルールの高度化など，実用化に向けての課題が残されている．

5．あとがき

本報告では，多人数でのスケジュール管理の前段階として，個人のスケジュール管理を行うシステムを提案し検討した．本システムは，曖昧さを扱うことができ，さらにプロダクションシステムを場合により使い分けることで，個別ユーザや状況に応じたスケジュールを生成することができるものである．

基本部分の実装を行い，シミュレーションを通して，提案したシステムでスケジュールを獲得できるということが示された．しかしながら現時点ではエキスパートシステムの基本部分が完成したのみであり，フレームの整理や知識の追加，ユーザインタフェースの改善などを整備し，多人数のタスク管理機能の実現が今後の課題である．

参　考　文　献

1）戸内順一：図解 エキスパートシステム入門［新版］，日本理工出版会（1997）．
2）人工知能覚書：http://fry.no.coocan.jp/lecture/AI/aimemo.pdf.
3）北橋忠宏：知識情報処理，森北出版（1998）．
4）オブジェクト指向スクリプト言語 Ruby：http://www.ruby-lang.org/ja/
5）Ruby on Rails：http://rubyonrails.org/.
6）The FreeBSD Project：http://www.freebsd.org/.
7）The Apache HTTP Server Project：http://httpd.apache.org/.
8）現行の知識表現の比較：http://www.sap-jp.net/mmi/sg001014.htm
9）池上敦子：“ナース・スケジューリング 調査・モデル化・アルゴリズム-”，数理統計，Vol.153，No.2，pp.231-259（2005）．
10）芝垣ゆかり，高木繁則，加藤康記，林和代：“制約指向型言語による看護婦スケジューリングシステム”，情報処理学会全国大会論文集，第 51 回平成 7 年後期（1），pp.379-380 （1995）．
11）PostgreSQL:The world's most advanced open source database：http://www.postgresql.org/
12）Gnome's Guide to WEBrick：http://microjet.ath.cx/WebWiki/WEBrick.html