多値議論の論理LMAのソーシャルウェブ：SNS,Wikipediaへの応用とその実装

栗原 秀輔
新潟大学大学院自然科学研究科博士前期課程
数理・情報電子工学専攻

目 次

第1章　序論

概要

多値議論の論理（LMA）は不確実な知識下で不確実な問題に対して議論による推論を可能とする議論フレームワークである．本論文では，LMA のソーシャルウェブ：SNS, Wikipedia への応用とその実装を述べる．これらは，現在および未来の情報社会に最も影響を与えるであろうソーシャルウェブアプリケーションだと言われている．SNS については，mixi の Mymixi への登録承認を LMA をもとに判断するエージェントとその実装方法を提案する．Wikipedia については，Wikipedia の削除問題に着目し，LMA をもとにその問題の原因となった記事を削除するべきか否かという問題について議論し，削除依頼による議論の分析を行うエージェントを提案する．

1.1　背景と動機

現在，人と人とをつなぐソーシャルなウェブサービスが目立っている．代表的なサービスとして，SNS(Social Networking Service) の Myspace[10] や mixi[8]，情報発信サービスの Amazon[2]，情報共有サービスの Youtube[19] や Flickr[5]，パーソナル情報検索・処理サービスの Google[6] や Yahoo![18] や MSN[9]，ユーザー発信型情報提供サービスの Wikipedia[17] などが挙げられる．

本論文では上述したサービスの中でも，これからの情報社会の社会形態やコミュニケーション方法に影響を与えるであろう SNS とフリーな情報の利用と人類共有情報の形成に影響を与えるであろう Wikipedia に着目する．

SNS については，本論文では数ある SNS の中でも日本で最大の会員数を持つ mixi に着目する．mixi では Mymixi と呼ばれる機能によってユーザの友達リストを管理することができる．Mymixi への追加は，ユーザが登録を依頼し，それを受け取ったユーザが承認するか否かによって行われ，また登録の削除は一方的に可能である．この登録依頼は全く面識の無いユーザからも来ることがあるが，中には悪質なユーザもいるために，様々な人と接点を持ちたいと思うユーザであっても，何の疑いも無く他人を承認することはためらわれ，何らかの意思決定の支援が望まれる．

本論文では議論を Mymixi の登録依頼に応用し，我々が提案した LMA をもとに Mymixi への登録承認を判断するエージェントを提案する．また，その実装方法も提案することで本研究の潜在的有効性だけでなく，実際的有効性についても言及する． エージェントは Mymixi への追加の審査，Mymixi からの削除の審査，Mymixi への追加の提案という 3 つの機能を持つ．本論文の提案により，Mymixi に登録するか否かというユーザの意思決定の支援と SNS におけるコミュニケーションの輪を広げられる事が期待できる．

一方で Wikipedia については，本論文では Wikipedia における削除の問題に着目する．Wikipedia には全ての利用者が従うべきだと考えられている基本方針とガイドラインがあるのだが，これに従わない利用者もいるため，Wikipedia で公開すべきでないページや画像が存在するという問題がある．この問題に対処するために，ページ・画像の削除に関する方針をまとめた「削除の方針 [17]」がある．削除の方針に合致するページや画像を削除することを依頼する「削除依頼 [17]」が任意のユーザによって行われると，それらを削除するか存続するか議論を行う．削除依頼による議論の記録はあるのだが，文章だけからなるものでユーザにとって議論の分析は容易ではない．また，最終的な判断は管理者によって行われているので論理的な議論による判断がなされているとは限らない．したがって，何らかの議論分析・意思決定の支援が望まれる．

本論文では削除問題に議論を応用し，LMA をもとに削除依頼による議論の分析を支援するエージェントを提案する．エージェントによる議論の結果を木表示することによって議論が可視化され，ユーザに対して議論分析の支援が期待できる．また，削除依頼による議論の中でも， 長期にわたり対処が決定していない議論については，本論文の提案によって削除するか存続するかという意思決定の支援が期待できる．

本論文では，エージェントによる問題解決の手段として議論に注目する．議論は分散した知識下において，矛盾が生じた場合に，その矛盾を解決する手段として有効である．[3] また，エージェントの知識表現として拡張注釈付き論理プログラム (*Extended Annotated Logic Program: EALP*)[15] を利用する．EALP は，様々な完備束を注釈として利用することで，情報や知識のあいまい性，不確実性を表現できるという高い表現能力を有している．

本論文の構成を説明する．2 節では，本論文で提案するエージェントの知識表現と議論について述べる．3 節では，LMA の SNS への応用を述べる．4 節では，LMA の Wikipedia への応用を述べる．5 章では，SNS への応用の実装方法を説明し，6 節では，本論文のまとめと今後の課題を述べる．

第2章　知識表現

知識 (knowledge) とは問題を解決するために必要な情報であり, 知識表現 (knowledge representation) とはそうした情報をどのような形式で表現し, どのように利用するかということである.

本研究では, エージェントが保持する知識をどのような形式で表現し, どのように利用するかということで, この知識表現の考えが必要となる. よって本章では, 本研究で扱う知識表現について説明する. 知識表現には, 拡張注釈付論理プログラム（Extended Annotated Logic Program:EALP) を用いる.

2.1　拡張注釈付論理プログラム [15]

2.1.1　言語

定義 1 (**注釈と注釈付き原子式**) 真理値の完備束 $(\mathcal{T}, \leq)$ を仮定し，その最小要素と最大要素を $\bot$ と $\top$ によって表す. また最小上界オペレータを $\sqcup$ によって表す. 注釈とは $\mathcal{T}$ の要素 (定数注釈), $\mathcal{T}$ 上の注釈変数, および注釈項のいずれかである. 注釈項は次のように再帰的に定義される. $\mathcal{T}$ の要素と注釈変数は注釈項である. 加えて，$x_1, \ldots, x_n$ が注釈項ならば, $f(x_1, \ldots, x_n)$ は注釈項である. ただし f は $\mathcal{T}^n \to \mathcal{T}$ の形の全域で連続な注釈関数である.

A を原子式, μ を注釈としたとき $A{:}\mu$ を注釈付き原子式という.

$\neg : \mathcal{T} \to \mathcal{T}$ の形の注釈関数を仮定し，$\neg A{:}\mu = A{:}\neg\mu$ とする. このとき，$\neg A{:}\mu$ を $A : \mu$ の認識論的な明示否定という.

注釈付き原子式の認識論的な明示否定 $\neg A : \mu$ は注釈付き原子式 $A : \neg\mu$ の形で記述するものとし，以下では暗に扱う.

定義 2 (**注釈付きリテラル**) $A{:}\mu$ を注釈付き原子式とする. このとき，$\sim A{:}\mu$ を $A{:}\mu$ の存在論的な明示否定という. 注釈付きオブジェクティブリテラルとは $\sim A{:}\mu$ か $A{:}\mu$ のいずれかである. 存在論的な明示否定の記号 $\sim$ は注釈付きオブジェクティブリテラルの補リテラルを表す場合にも用いる. すなわち，$\sim\sim A{:}\mu = A{:}\mu$ とする. L を注釈付きオブジェクティブリテラルとしたとき，**not** L を L のデフォルト否定といい，その形を注釈付きデフォルトリテラルという. また，注釈付きリテラルとは **not** L か L のいずれかである.

例 1 注釈はエージェントの信念の度合いを示す. 拡張注釈付きリテラル $credible(kenji){:}8$ は「$kenji$ はとても信頼できる」と読む. $credible(kenji)$ の部分が原子式, 8 が注釈である. また $\sim credible(kenji){:}6$ は「$kenji$ はそこそこ信頼できるということは認められない」,

not $credible(kenji):6$ は「$kenji$ は今のところ, そこそこ信頼できることが証明されていない」と読む.

定義 3 (**拡張注釈付き論理プログラム**) 拡張注釈付き論理プログラム (*Extended Annotated Logic Program: EALP*) は次の形式の注釈付き規則の集合である.

$$H \leftarrow L_1 \,\&\, \ldots \,\&\, L_n.$$

ここで, H は注釈付きオブジェクティブリテラルであり, L_i $(1 \leq i \leq n)$ は注釈付きリテラルである. ただし, 各 L_i に現れる注釈は定数注釈か注釈変数である.

以下, 本論文では拡張注釈付き論理プログラムを知識ベース (Knowledge Base:KB) と呼ぶ. また, 特に断らない限り注釈付き規則のことを単に規則, 本体のない規則を事実節と呼ぶ.

規則の頭部 H は, 規則の結論という. また, 本体に含まれる注釈付きオブジェクティブリテラルは規則の前提, 注釈付きデフォルトリテラルは規則の仮定という.

例 2 $\sim register(kenji):5 \leftarrow\sim message(kenji):9$ は, 「$kenji$ からのメッセージがないならば, $kenji$ を登録するということは認められない. 」ということを表す規則である. 規則の本体にある $\sim message(kenji):9$ は前提を表し, これが知識ベースから導ければ, 規則の結論 $\sim register(kenji):5$ を導ける.

2.2 多値議論 [15]

本節では, EALP における注釈付き論証と攻撃関係の定義を導入する.

2.2.1 注釈付き論証

定義 4 (**注釈付き論証**) P を $EALP$ とする. P における注釈付き論証とは次の条件を満たす規則の有限列 $Arg = [r_1, \ldots, r_n]$ である. すべての i $(1 \leq i \leq n)$ について,

1. r_i は P の規則, もしくは, P の極小還元である.
2. r_i の本体にあるすべての注釈付き原子式 $A{:}\mu$ に対し, 頭部が $A{:}\rho$ $(\rho \geq \mu)$ となるような r_k $(i < k \leq n)$ が存在する.
3. r_i の本体にあるすべての注釈付き原子式の存在論的な明示否定 $\sim A{:}\mu$ に対し, 頭部が $\sim A{:}\rho$ $(\rho \leq \mu)$ となるような r_k $(i < k \leq n)$ が存在する.
4. 条件 *1* から *3* を満たし r_1 を含んでいるような $[r_1, \ldots, r_n]$ の真部分列が存在しない.

上記において, r_1 の結論を論証 $Arg = [r_1, \ldots, r_n]$ の主張と呼ぶ.

本論文では, 特に断りがない限り注釈付き論証のことを単に論証と呼ぶ. *Arg* の部分論証とはそれ自身が論証となる *Arg* の部分列である. *Arg* を構成する規則の結論を *Arg* の結論と呼び, その仮定を *Arg* の仮定と呼ぶ. また, *Arg* の結論の集合を *concl(Arg)*, 仮定の集合を *assm(Arg)* , *EALP P* から作られるすべての論証の集合を $Args_P$ と書く.

また, *[15]* では還元を定義して, 還元を用いた例題を示しているが, 本研究では, 還元の定義を用いる必要がないので, 還元の定義は省く.

例 ***3*** 以下のような知識ベース KB を仮定する.

$$
\begin{aligned}
KB = \{\quad & info(kenji, japan, clerk, 38) : 9 \leftarrow, \\
& hobby(kenji, music) : 9 \leftarrow, \\
& my\_mixi(kenji, 35) : 9 \leftarrow, \\
& write\_diary(kenji) : 9 \leftarrow, \\
& register(P) : 7 \leftarrow profile\_ok(P) : 8 \,\&\, comment(P) : 7, \\
& profile\_ok(P) : 8 \leftarrow correspond\_hobby(P) : 7, \\
& correspond\_hobby(P) : 7 \leftarrow hobby(P, music) : 9, \\
& \sim correspond\_hobby(P) : 6 \leftarrow hobby(P, basketball) : 9, \\
& comment(P) : 8 \leftarrow my\_mixi(P, Sum) : 9 \,\&\, (Sum < 70) : 9, \\
& \sim comment(P) : 5 \leftarrow \mathbf{not} \ \sim negative(P) : 7, \\
& \sim negative(kenji) : 7 \leftarrow write\_diary(P) : 9 \quad \}
\end{aligned}
$$

上述の知識ベースから作られる論証 Arg を以下に示す.

$$
\begin{aligned}
Arg_1 = [\quad & register(kenji) : 7 \leftarrow profile\_ok(kenji) : 8 \,\&\, comment(kenji) : 7, \\
& profile\_ok(kenji) : 8 \leftarrow correspond\_hobby(kenji) : 7, \\
& correspond\_hobby(kenji) : 7 \leftarrow hobby(kenji, music) : 9, \\
& hobby(kenji, music) : 9 \leftarrow, \\
& comment(kenji) : 7 \leftarrow my\_mixi(kenji, 35) : 9 \,\&\, (35 < 70) : 9, \\
& my\_mixi(kenji, 35) : 9 \leftarrow \quad] \\
Arg_2 = [\quad & profile\_ok(kenji) : 8 \leftarrow correspond\_hobby(kenji) : 7, \\
& correspond\_hobby(kenji) : 7 \leftarrow hobby(kenji, music) : 9, \\
& hobby(kenji, music) : 9 \leftarrow \quad] \\
Arg_3 = [\quad & correspond\_hobby(kenji) : 7 \leftarrow hobby(kenji, music) : 9, \\
& hobby(kenji) : 9 \leftarrow \quad] \\
Arg_4 = [\quad & comment(kenji) : 8 \leftarrow my\_mixi(kenji, 35) : 9 \,\&\, (35 < 70) : 9, \\
& my\_mixi(kenji, 35) : 9 \leftarrow \quad] \\
Arg_5 = [\quad & \sim comment(kenji) : 5 \leftarrow \mathbf{not} \ \sim negative(kenij) : 7 \quad] \\
Arg_6 = [\quad & \sim negative(kenji) : 7 \leftarrow write\_diary(kenji) : 9, \\
& write\_diary(kenji) : 9 \leftarrow \quad] \\
Arg_7 = [\quad & info(kenji, japan, clerk, 38) : 9 \leftarrow \quad] \\
Arg_8 = [\quad & hobby(kenji, music) : 9 \leftarrow \quad] \\
Arg_9 = [\quad & my\_mixi(kenji, 35) : 9 \leftarrow \quad]
\end{aligned}
$$

$Arg_{10} = [\quad write\_diary(kenji) : 9 \leftarrow \quad]$

2.2.2 攻撃関係

ここでは, 論証同士の攻撃関係を示すために, 最初に存在論的な矛盾に関する攻撃関係である反論 (rebut) の定義を導入する.

定義 5 (**反論**) Arg_1 が Arg_2 を反論する $\Leftrightarrow$ $\mu_1 \geq \mu_2$ となる $A : \mu_1 \in concl(Arg_1)$ と $\sim A : \mu_2 \in concl(Arg_2)$ が存在するか, $\mu_1 \leq \mu_2$ となる $\sim A : \mu_1 \in concl(Arg_1)$ と $A : \mu_2 \in concl(Arg_2)$ が存在する.

次に, デフォルトに関する矛盾と対応する攻撃関係として無効化 (undercut) の定義を紹介する.

定義 6 (**無効化**) Arg_1 が Arg_2 を無効化する $\Leftrightarrow$ $\mu_1 \geq \mu_2$ となる $A : \mu_1 \in concl(Arg_1)$ と **not** $A : \mu_2 \in assm(Arg_2)$ が存在するか, $\mu_1 \leq \mu_2$ となる $\sim A : \mu_1 \in concl(Arg_1)$ と **not** $\sim A : \mu_2 \in assm(Arg_2)$ が存在する.

定義 7 (**完全な無効化**) Arg_1 が Arg_2 を完全に無効化する $\Leftrightarrow$ Arg_1 が Arg_2 を無効化し, Arg_2 が Arg_1 を無効化しない.

さらに, 存在論的な矛盾とデフォルトに関する矛盾の両方と対応する攻撃関係として, 反論と無効化を組み合わせたものを定義する.

定義 8 (**攻撃**) Arg_1 が Arg_2 を攻撃 (attack) する $\Leftrightarrow$ Arg_1 が Arg_2 を反論するか, Arg_1 が Arg_2 を無効化する.

以下に, 無効化を反論より強い攻撃関係として扱う論破（defeat）の定義を紹介する.

定義 9 (**論破**) Arg_1 が Arg_2 を論破する $\Leftrightarrow$ Arg_1 が Arg_2 を無効化するか, Arg_1 が Arg_2 を反論し Arg_2 が Arg_1 を無効化しない.

なお, 本研究では論破をエージェントの攻撃関係として採用する.

2.3 対話的証明論における正当化 [15]

本論文で行う議論は対話的証明論によって正当化されるものである. そこで, この節では多値議論の論理より, 対話的証明論における正当化の定義を紹介する.

定義 10 (d/d-**対話**) d/d 対話とは, 次の条件を満たす提議 $move_i = (Player_i, Arg_i)$ $(i \geq 1)$ の空でない有限列である.

1. $Player_i = P$ (提案者) $\Leftrightarrow$ i は奇数である; また $Player_i = O$ (反対者) $\Leftrightarrow$ i は偶数である.

2. $Player_i = Player_j = P$ $(i \neq j)$ ならば $Arg_i \neq Arg_j$.

3. $Player_i = P$ $(i \geq 3)$ ならば $(Arg_i, Arg_{i-1}) \in d$; また, $Player_i = O$ $(i \geq 2)$ ならば $(Arg_i, Arg_{i-1}) \in d$.

条件 1 は，d/d-対話が提案者から始まり，提案者と反対者が交互に提議することを表している．条件 2 は d/d-対話が循環するのを防ぐためのものである．本論文では d/d-対話が提案者の提議で終了するかどうかにだけ注目する．提案者による同じ論証の再提議を禁止したので，d/d-対話が循環するような場合は必ず反対者の提議で停止するようになる．条件 3 は，提案者と反対者が d（defeat: 論破）攻撃を行うことを表している．

定義 11 (d/d**-対話木**) d/d-対話木とは，すべての枝 ($branch$) が d/d-対話となる木である．ただし，すべての提議 $move_i = (P, Arg_i)$ に対し, その子は $(Arg_{i+1,j}, Arg_i) \in d$ となるようなすべての $(O, Arg_{i+1,j})$ $(j \geq 1)$ である.

定義 12 (**証明論的正当化**) d/d-対話 D が d/d-勝利対話である $\Leftrightarrow$ D の終端が提案者の提議である．d/d-対話木 T が d/d-勝利対話木である $\Leftrightarrow$ T のすべての枝が d/d-勝利対話である．論証 Arg が証明論的に d/d-正当化された論証である $\Leftrightarrow$ Arg を根とした d/d-勝利対話木が存在する.

定理 1 $Args$ を論証の集合とする． このとき, $Arg \in Args$ が証明論的に正当化される $\Leftrightarrow$ Arg が正当化される.

定理 1 により, 議論の意味において正当化されるすべての論証は対話的に判定でき，また，対話的に正当化された論証は議論の意味においても正当なものであることが保証される.

第3章　SNSへの応用

本節で提案するエージェントは，ユーザの入力した評価基準と，登録依頼者のプロフィールをもとに議論し登録依頼者をMymixiへ追加するか， Mymixiから削除するか審査する．表3.2に評価基準の例を示す． 表3.2中の足跡は，自分のページを訪れた人が誰であるかを示すもので， 訪れた際に残る記録を意味する．また，表3.3に本論文で扱う評価基準のEALP表現とその規則の読みを示す．SNSへの応用では，ユーザの評価基準において「とても」,「やや」,「あまり」などのあいまいな表現を扱うので，用いる真理値の完備束は0から1までの実数値区間 $\Gamma = \Re[0,1]$ とする．この真理値の完備束を図3.1に示す．

ユーザが真理値を決定する際の判断基準として，信念の度合いが強い場合は1.0，やや強い場合は0.7の様に信念の度合いを段階的に分けた枠組みをユーザに提供し，判断してもらうことを考える．mixiではプロフィールとして現住所，性別，誕生日，生まれた年，血液型，出身地，趣味，職業をリストから選択可能である．趣味は24個のアイテムから，職業は20個のアイテムからの選択式でユーザ同士の比較が容易である．表3.3中の「趣味があうこと」を意味する述語 $corrrespond\_hobby(W)$ はエージェントがユーザと登録依頼者のプロフィールを比較し合致した度合いによって注釈を自動的に与え生成する．ここでは，ユーザは特定の趣味が一致した場合に高い度合いの注釈を与えるか，趣味がどれだけ一致しているかの割合で注釈を与えるかのいずれかを選択可能である．また職業が同じことを示す述語 $same\_job(W)$ と出生地が同じであることを示す述語 $same\_birthplace(W)$ は両者の職業，出生地が同じであった場合に注釈として1.0を付加し生成する．また日記の更新頻度を示す述語 $update\_interval(W)$ は，エージェントが登録依頼者の日記の日付から更新頻度を計算し，その割合から注釈を計算し自動生成する．例えば「一ヶ月に一回日記の更新があれば更新頻度が高い」と設定した場合，実際一ヶ月に一回更新があれば $update\_interval(W)$:1.0，二ヶ月に一回更新があれば $update\_interval(W)$:0.5となる．

3.1　Mymixiへの追加の審査

エージェントの議論によるMymixi承認審査フレームワークを図3.2をもとに説明する．エージェントはあらかじめ議論エンジン (LMA) を保持している．図中の矢印は動作主のアクセスを示す．

STEP1 ユーザから評価基準を与えられるとエージェントはこれをEALPへ変換する．

STEP2 ユーザの元へMymixi登録依頼メッセージが届くと，登録依頼者のプロフィールページへアクセスする．登録依頼者のプロフィールページから必要な情報を抽出しEALPへ変換する．

表 3.1: プロフィール

名前	Kenji	Yui	ユーザ
年齢	38	20	23
性別	男性	女性	男性
出身地	東京	沖縄	新潟
職業	会社員	アーティスト	学生
趣味	音楽	音楽	音楽
更新頻度	高	高	高
Mymixi	35	45	22
コミュニティ数	27	15	76

STEP3 EALP へ変換された登録依頼者のプロフィール情報とユーザの評価基準を合わせた KB を作成する．KB が作成されると，登録依頼者を Mymixi に登録するか否かについて議論を展開する．

STEP4 議論結果に基づき審査結果をユーザに示す．W というユーザから登録依頼メッセージが来たとすると，$register(W):\mu$ を結論にもつ論証が正当化された場合，W を Mymixi へ登録するように薦める． それ以外は，登録を薦めない．

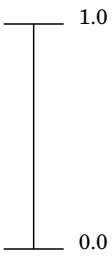


図 3.1: 真理値の完備束 Γ

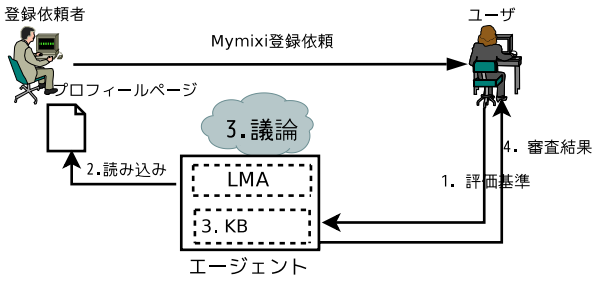


図 3.2: 議論による Mymixi 承認審査フレームワーク

例 Kenji と Yui から Mymixi 登録依頼メッセージが届いたと仮定し，表 3.3 に示した評価基準の EALP 表現を使用して議論により審査する．Kenji と Yui とユーザ， それぞれのプロフィールを表 3.1 に示す．Kenji についての議論の結果の木表現を図 3.3 に示す．対話木の枝の終端がすべて提案者なので $register(kenji):0.8$ を結論に持つ論証が正当化され，エージェントはユーザに対し Kenji の Mymixi への登録を薦める．対話木において， フレームは論証，太枠のフレームは正当化された論証， 矢印は攻撃関係（論破）を示す．同様に Yui についての議論の結果の木表現を図 3.4 に示す．対話木の枝の終端がひとつ提案者ではないので $register(Yui):0.8$ を結論に持つ論証が正当化されず，エージェントは Yui の Mymixi への登録は薦めない．

表 3.2: 評価基準の例

趣味がある程度同じ方が良い.
Mymixi が多くなければコメントしてくれる.
出生地が同じ方が良い.
職業が同じ方が良い.
mixi へのログイン率が高い方が良い.
自分に興味を持っているなら登録する.
足跡数が多ければ， 自分に興味を持っている.
mixi に消極的ならばコメントしてくれない.
日記の更新頻度が低い人は mixi に消極的

表 3.3: 評価基準の EALP 表現

$register(W):0.8 \leftarrow correspond\_hobby(W):0.8 \,\&\, comment(W):0.7.$
(W と趣味がある程度合っていて，W が日記にコメントを書いてくれるならば Mymixi に登録する．)
$correspond\_hobby(W):0.8 \leftarrow hobby(W, music):1.0.$
(W の趣味が音楽ならば， ある程度趣味が合う.)
$comment(W):0.7 \leftarrow my\_mixi(W, X):1.0 \,\&\, (X < 70):1.0.$
(W の Mymixi の数が 70 より少なければ，W はコメントを書いてくれる．)
$register(W):0.7 \leftarrow same\_job(W):1.0.$
(W と職業が同じならば Mymixi に登録する．)
$register(W):0.6 \leftarrow same\_birthplace(W):1.0.$
(W と出生地が同じならば Mymixi に登録する．)
$\sim comment(W):0.5 \leftarrow \textbf{not}\ positive(W):0.7.$
(W が mixi に積極的であるということがわからなければ，W はコメントを書いてくれない．)
$positive(W):0.7 \leftarrow update\_interval(W):1.0.$
(W の日記の更新頻度が高ければ，mixi に積極的である．)
$\sim positive(W):0.7 \leftarrow community\_num(W, X):1.0 \,\&\, (X < 20):1.0.$
(W のコミュニティの数が 20 より少なければ，mixi に積極的でない．)

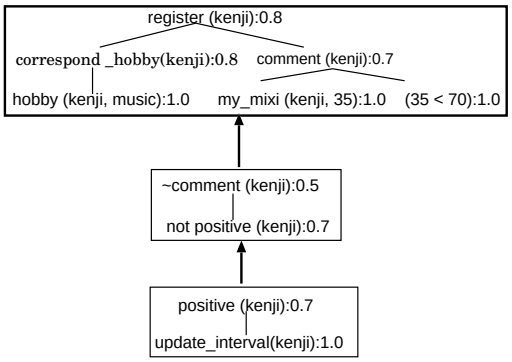


図 3.3: Kenji についての議論の対話木

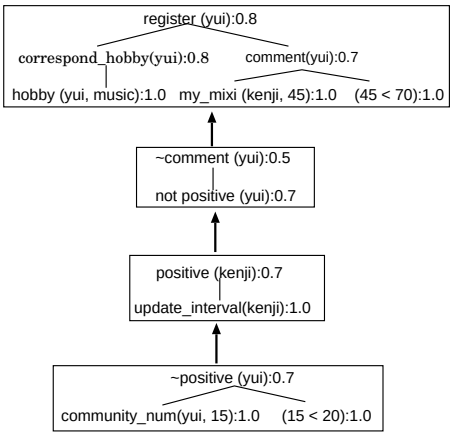


図 3.4: Yui についての議論の対話木

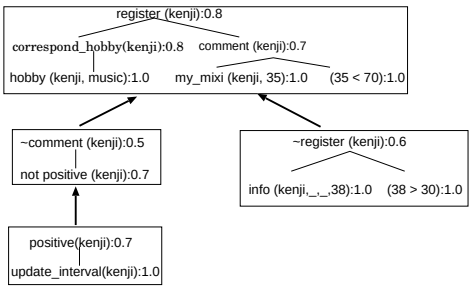


図 3.5: Kenji に対する再審査の議論の対話木

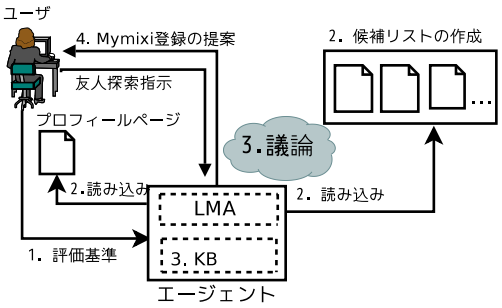


図 3.6: Mymixi 追加提案のフレームワーク

3.2 Mymixi からの削除の審査

ユーザの趣味の変化などにともなってユーザの評価基準が変化することがある．ユーザの評価基準が変化するとエージェントは， 変更された評価基準をもとに新たに KB を作成し Mymixi の審査を行う．エージェントは 3.1 節と同様の手順で KB を作成し，議論を展開する．そして $register(W):\mu$ を結論にもつ論証が正当化されなかった場合，エージェントは W を Mymixi から削除することを薦める．削除の審査の例を以下に示す．

例 ユーザが表 3.2 の評価基準に対して，新たに

$\sim register(W):0.6 \leftarrow info(W,\_,\_,Age):1.0\,\&\;\;(Age > 30):1.0.$

（W の年齢が 30 歳より高ければ，Mymixi に登録することは認めない.）という評価基準を追加したとする．このとき Kenji を再審査すると，議論の結果の木表現は図 3.5 のようになる．対話木の枝の終端がひとつ提案者ではないので $register(Kenji):0.8$ を結論に持つ論証が正当化されず，エージェントは Kenji の Mymixi からの削除を薦める．

3.3 Mymixi への追加の提案

3.1 節と 3.2 節では，他のユーザから登録依頼メッセージが来た時にこれを議論によって審査する方法とユーザが評価基準を変更した場合の再審査の方法を示した．本節では Mymixi を辿ることで，ユーザが入力した評価基準を満たす人を自動的に探す Mymixi 追加提案のフレームワークを説明する．図 3.6 に追加提案のフレームワークを示す．図中の矢印は動作主のアクセスを示す．ユーザから友人を探す指示がでると，エージェントは Mymixi を辿り Mymixi 追加候補リストを作成する．そしてリストの人物全てに対して 3.1 節で示した議論による審査を行う．審査の結果 $register(W) : \mu$ を結論に持つ論証が正当化された W の集合に対して Mymixi への追加を薦める．

第4章　Wikipediaへの応用

本論文では分散した複数の EALP を個々のエージェントと捉え，EALP の集合をマルチエージェントシステムとして扱う．EALP の集合を $MAS = \{KB_1, \cdots, KB_n\}$ のように表現する．LMA と削除依頼の議論の記録から抽出した情報をもとに，エージェントは議論を展開し，削除依頼されたページ・画像を削除するか存続するか審査する．

4.1　削除の審査

エージェントの議論による削除審査フレームワークを図 4.2 をもとに説明する．図 4.2 の矢印は動作主のアクセスを示す．各エージェントはあらかじめ議論エンジン (LMA) を保持している．

STEP1 ユーザが削除依頼の議論の記録から必要な情報を抽出し，EALP に変換する．これをエージェントの KB とする．

STEP2 STEP1 で作成した KB をエージェントに与える．

STEP4 KB をもとに議論を展開.

STEP5 議論結果に基づき審査結果をユーザに示す．P というページ・画像を審査したとすると， $delete(P)$: **t** を結論に持つ論証が正当化された場合，P を「削除する」ことを薦める．それ以外の場合は,「存続する」ことを薦める．

図 4.1: 真理値の完備束 $\mathcal{TWO}$

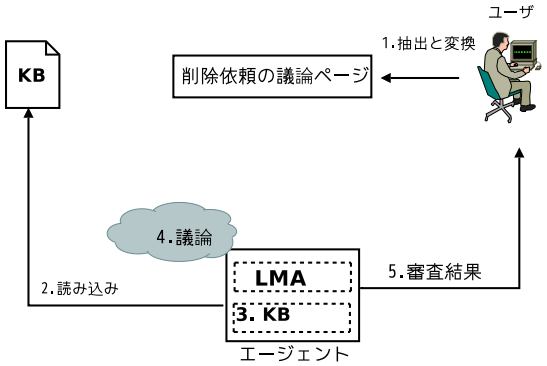


図 4.2: 議論による削除審査フレームワーク

例　過去に削除依頼され「存続する」という結論が下されたページ「ロボット工学三原則 [17]」を審査する．$MAS = \{KB_1, \cdots, KB_4\}$ である．ページ「ロボット工学三原則」の削除依頼の議論から， 削除審査に必要な情報を抽出する．抽出した情報を表 4.1 に示す．抽出した情報は時系列順であり,（1）から（4）は異なる発言者による発言である．発言（1）が議題である．表 4.1 中の（削除）,（存続）は以下の意味を持つ.

表 4.1: 抽出した情報

(1)(削除) 引用の要件不足.
(2)(存続) 引用の要件を満たしている.
(3)(削除) 具体的にどの作品からの引用であるか書かれていない.
(4)(存続) 削除依頼の濫用.

表 4.2: エージェントの KB

$(1) KB_1 = \{delete(page\_r) : \mathbf{t} \leftarrow quotation\_requirement(page\_r) : \mathbf{f}.\}$
$(2) KB_2 = \{\sim delete(page\_r) : \mathbf{t} \leftarrow quotation\_requirement(page\_r) : \mathbf{t}.\}$
$(3) KB_3 = \{delete(page\_r) : \mathbf{t} \leftarrow quotation\_requirement(page\_r) : \mathbf{f}.\}$
$(4) KB_4 = \{\sim delete(page\_r) : \mathbf{t} \leftarrow abuse\_of\_delete\_request(page\_r) : \mathbf{t}.\}$

- （削除）: 任意のユーザによる削除依頼されたページ・画像を削除する理由を述べた文章
- （存続）: 任意のユーザによる存続する理由を述べた文章

上記の他に以下のものが削除依頼の議論に存在するが, (削除) , (存続) 以外の文章からは情報は抽出しない.

- （コメント）: 任意のユーザによる（削除）, (存続）に対するコメントを述べた文章
- （保留）: 任意のユーザによる削除依頼を保留した方が良いという意見を述べた文章

表 4.1 の EALP 表現を表 4.2 に示す. 表 4.2 中の (1) から (4) は，各エージェントの KB である.

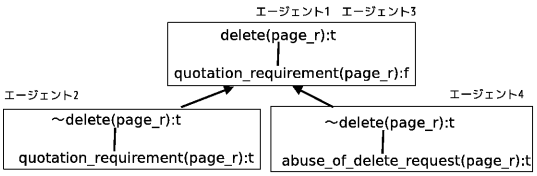


図 4.3: ページ「ロボット工学三原則」に対する議論の対話木

図 4.3 に，ページ「ロボット工学三原則」を審査したときの対話木を示す. 議論の結果, 対話木の枝の終端すべてが提案者ではないので， $delete(page\_r) : \mathbf{t}$ を結論に持つ論証が正当化されず，LMA のもとページ「ロボット工学三原則」を「存続する」ことをエージェントは薦める. 対話木において，フレームは論証，太枠のフレームは正当化された論証, 矢印は攻撃関係（論破）を示す

実際の議論の流れは以下である.

1. 引用の要件不足で削除するという意見が議題として提出される．
2. 引用の要件を満たしているので存続するという意見が提出される．
3. 具体的にどの作品からの引用であるか書かれていないので削除するという意見が提出される．
4. 削除依頼者のコメントから，削除依頼を濫用している意図が感じられるので，存続するという意見が提出される．

以下に長期にわたり対処が決定していない議論を審査する例を挙げる．

例　長期にわたり対処が決定していないページ「一撃必殺 [17]」を審査する．$MAS = \{KB_1, \cdots, KB_3\}$ である．ページ「一撃必殺」に対する削除依頼の議論から，削除審査に必要な情報を抽出する．抽出した情報を表 4.3 に示す．表 4.4 にエージェントの KB を示す．

図 4.4 に，ページ「一撃必殺」を削除するか存続するか議論したときの対話木を示す．議論の結果，対話木の枝の終端すべてが提案者でないので，LMA のもと $delete(page\_h):\mathbf{t}$ を結論に持つ論証が正当化されず，ページ「一撃必殺」を「存続する」ことをエージェントは薦める．

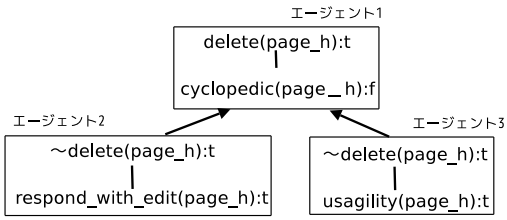


図 4.4: ページ「一撃必殺」に対する議論の対話木

表 4.3: 抽出した情報

(1)(削除) 百科事典的でない記事.
(2)(存続) 編集で対処すべき案件.
(3)(存続)Wikipedia 内に残しておいた方が有用.

表 4.4: エージェントの KB

$(1) KB_1 = \{delete(page\_h):\mathbf{t} \leftarrow cyclopedic(page\_h):\mathbf{f}.\}$
$(2) KB_2 = \{\sim delete(page\_h):\mathbf{t} \leftarrow respond\_with\_edit(page\_h):\mathbf{t}.\}$
$(3) KB_3 = \{\sim delete(page\_h):\mathbf{t} \leftarrow usability(page\_h):\mathbf{t}.\}$

実際の議論の流れは以下である．

1. 百科事典的でない記事であるので，削除するという意見が議題として提出される．

2. 空手道、剣術において使用される術語であり，編集で対処すべき案件であるので，存続するという意見が提出される．

3. 削除するよりも Wikipedia に残しておく方が有用性があるので，存続するという意見が提出される．

第5章　SNSへの応用の実装

本章では、第3章の「SNSへの応用」の中でも3.3節のエージェントの機能「Mymixiへの追加の提案」の実装方法を提案する．実装のための主なツールとしてRuby on Rails[11]を使用する．これが持つ特徴を活用することにより，簡単かつ迅速にウェブアプリケーションを作成できることを以下に述べる．

5.1　Ruby on Railsについて

Ruby on Railsは，スクリプト言語のRuby[13]により構築されたWebアプリケーション開発のためのフレームワークで，実アプリケーションの開発を他のフレームワークより少ないコードで簡単に開発できるよう考慮し設計されている．「RoR」または単に「Rails」と呼ばれることもある．以下，Ruby on RailsをRailsと呼ぶ．

Railsは，MVC（Model-View-Controller）アーキテクチャをサポートしており，データベースに関する処理を担うモデル，データの表示を行うビュー，それらを制御するコントローラの雛形を自動生成する機能を持っている．データの作成・読み込み・更新・削除(CRUD)のみを行うような単純なWebアプリケーションであれば，データテーブルの作成と雛形の自動生成を行うのみでほとんどの部分が開発できる．Webアプリケーションの多くはCRUD処理を中核としているので，Railsを使うことによって多くのコードの作成を自動化することができ，開発期間を短縮できる．他にもWebアプリケーションの動作テストを行うためのWebサーバや，テストのためのコードを自動生成する機能，Webアプリケーションの配置を自動化するツールなどのツールキットも同時に提供している．

5.1.1　Ruby on Railsの使用方法

Railsの使用方法を「Ruby on Rails入門[21]」をもとに説明する．

プロジェクトの作成と実行手順

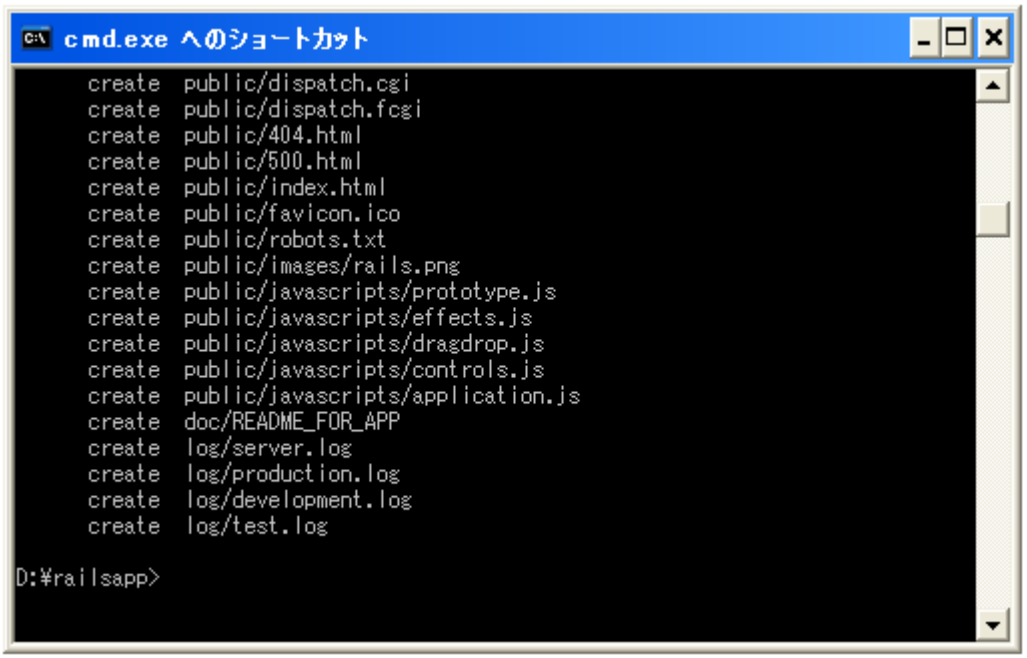
まずRailsでアプリケーションを作成するためにプロジェクトを作成する．新しいプロジェクトの作成は以下のように行う．「rails」の後にスペースを1つ空けて作成したいプロジェクト名を入力する．

```
rails プロジェクト名
```

今回はプロジェクト名として「hello」と言うプロジェクトを作成する．そして，コマンドプロンプトを立ち上げて作成したディレクトリに移動する．そして次のコマンドを実行する．

```
rails hello
```

実行するとコマンドプロンプトに以下のようなメッセージが表示される．プロジェクトを作成すると自動的に必要となるディレクトリやファイルが作成される．



ディレクトリ構成

以下に，プロジェクトを作成した際に作成された各ディレクトリについての簡単な説明を記述した表を載せる．

ディレクトリ名	目的
app	プログラムを配置します
components	コンポーネントを配置
config	設定ファイルが格納
db	データベース関係のファイルを配置
doc	ドキュメントを配置
lib	ライブラリを配置
log	ログファイルを格納
public	Web サーバに対するドキュメントルート
script	スクリプトファイルを格納
test	テスト用のファイルを配置
tmp	テンポラリファイルを格納
vendor	プラグインファイルを配置

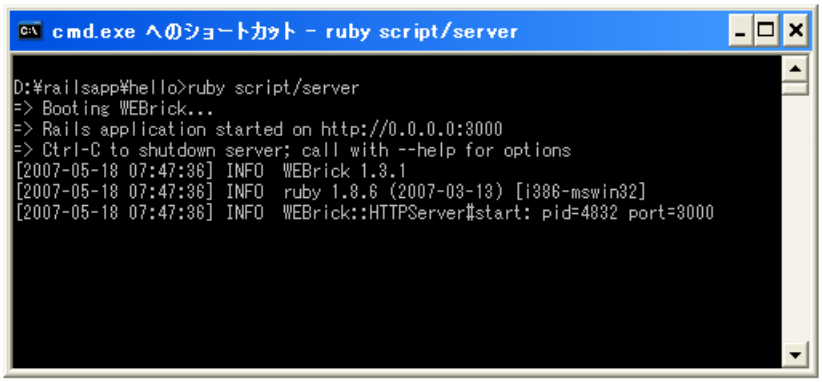
WEBrick ライブラリを使った Web サーバの起動

Rails を使って Web アプリケーションを作成するためにはクライアントからの要求を受け付ける Web サーバと連携させる必要がある．Ruby には標準で Web サーバの機能を実現する WEBrick ライブラリがインストールされており，Rails プロジェクトを作成すると自動的に WEBrick ライブラリを使って Web サーバを起動するスクリプトも用意されている．実際には下記の場所にスクリプトが用意されている．

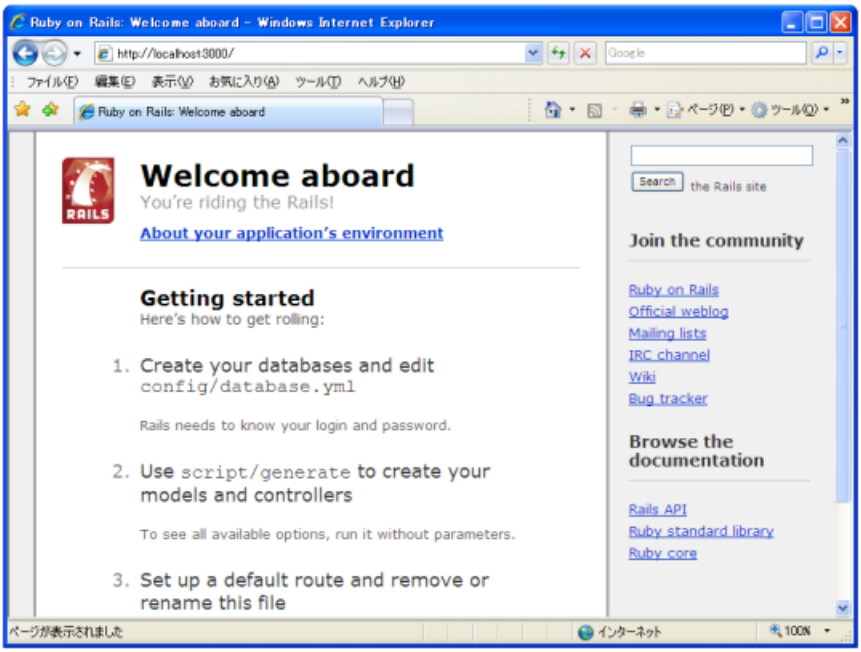
```
(プロジェクトディレクトリ)\ script \ server
```

コマンドプロンプトを立ち上げてプロジェクトのルートディレクトリに移動した後で次のコマンドを実行.
これにより Web サーバが起動する（デフォルトのポート番号は 3000 番).

```
ruby script/server
```



ブラウザを開いて「http://localhost:3000/」と入力すると以下のような画面が表示される．以下のように表示されれば Web サーバの起動に成功している．現在表示されているページはプロジェクトを作成した時にデフォルトで作成された画面である．



5.1.2 アプリケーションの作成

Rails は MVC フレームワークを用いているが，ここではコントローラーを重点に説明する．コントローラーはブラウザなどのクライアントから要求の処理を担当する．要求に対してモデルからデータを取り出したり，データを元にビューに対して画面表示を指示したりする部分を担当するのがコントローラーとなる．

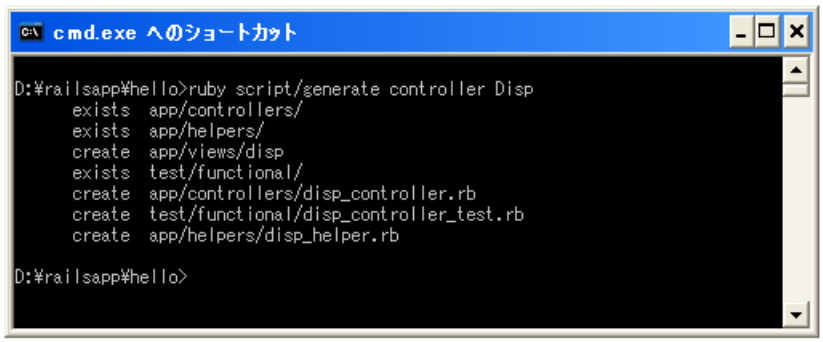
コントローラーの中にはさらに具体的な処理を記述したアクションと呼ばれるものを記述する．ユーザーはコントローラーを指定してさらにコントローラーの中のアクションを指定することでどのような処理をしたいのかを指示する．アクションはコントローラーの中に複数記述が行える．コントローラーはアクションをある程度まとめたものであると考えられる．

コントローラーの作成

「Disp」と言うコントローラーを作成する例を挙げる．プロジェクトを作成した時と同様にコントローラーを作成する際にもスクリプトが用意されている．

プロジェクトを作成した際に自動で作成された「script」ディレクトリの中に「generate」と Ruby プログラムが用意されている．コントローラーを作成するには次のように実行する．

```
ruby script/generate controller コントローラ名（今回の例では Disp を入力）
```



「generate」を実行すると上記のようにディレクトリやファイルが作成される．プロジェクト内の「app \ controllers」ディレクトリの中に，新しく「disp_controller.rb」と言うプログラムファイルが作成される．このファイルが「Disp」コントローラーとなる．

アクションの作成

次にアクションの作成を説明する．アクションを作成するには使用するコントローラークラスの中に public なメソッドを定義することで作成する．この時、アクション名がメソッド名となる．

今回はコントローラーとして「Disp」コントローラーを使うので「D:\ railsapp\ hello\ app\ controllers\」ディレクトリにある「disp_controller.rb」内に記述する．例として「helloRails」アクションを作成する．

disp_controller.rb

```
class DispController < ApplicationController
  def helloRails
  end
end
```

アクション名をメソッド名として使うので今回は「helloRails」がメソッド名となる．クライアントからアクションが呼ばれた場合、このメソッド内に定義されたプログラムが実行された後に対応するビューが呼び出される．このビューにクライアントに表示する内容を記述する．ビューではメソッド内で設定されたインスタンス変数などを参照することや，ビュー内で Ruby のプログラムを記述することも可能である．

ビューの作成

アクションを指定してクライアントから要求があった場合、まずコントローラー内で定義されたアクションに対応するメソッドが実行される．その後，アクションに対応するビューが呼び出される．

ビューはMVCフレームワークの中で画面表示を担当する部分である．結果としてクライアントに表示するHTML文書（テンプレートと呼ぶ）をビューとして用意しておく．

単なるHTML文書だけでは毎回同じ表示しか出来ないので、何らからの処理を行った結果をビューに反映させられるようにHTML文書内にRubyのプログラムを記述することが可能である．ビュー内でプログラムを記述できるが，アクションメソッド内でプログラム部分は処理しておいて，その結果をテンプレートから参照することも可能である．

ビューのファイル名は以下のようになる．

アクション名.rhtml

拡張子は「rhtml」となる．ファイル名はビューが呼び出される元になるアクション名となる．例えば「helloRails」アクションに対応するビューファイル名は「helloRails.rhtml」である．

設置する場所はコントローラーを作成すると「app\views\コントローラー名」と言うディレクトリが自動的に作成されるのでその中に配置する．今回の例であればコントローラー名が「Disp」なので「app\views\disp」というディレクトリが作成されている．

それではアクション「helloRails」に対応したビューである「helloRails.rhtml」を作成する．

helloRails.rhtml

```
<html>
<head>
<title>Hello Rails</title>
</head>
<body>

<p>Hello Rails</p>

</body>
</html>
```

以上でコントローラー、アクション、そしてビューの準備が完了した．

クライアントからの呼び出し

コントローラー、アクション、そしてビューの準備が出来たのでクライアントから要求を行ってみる．今回は「Disp」コントローラーに含まれている「helloRails」アクションを呼び出す．

http://url/ctr/action

「http://url/」の部分はWebサーバのURLとなる．今回の例では「http://localhost:3000/」が該当する．

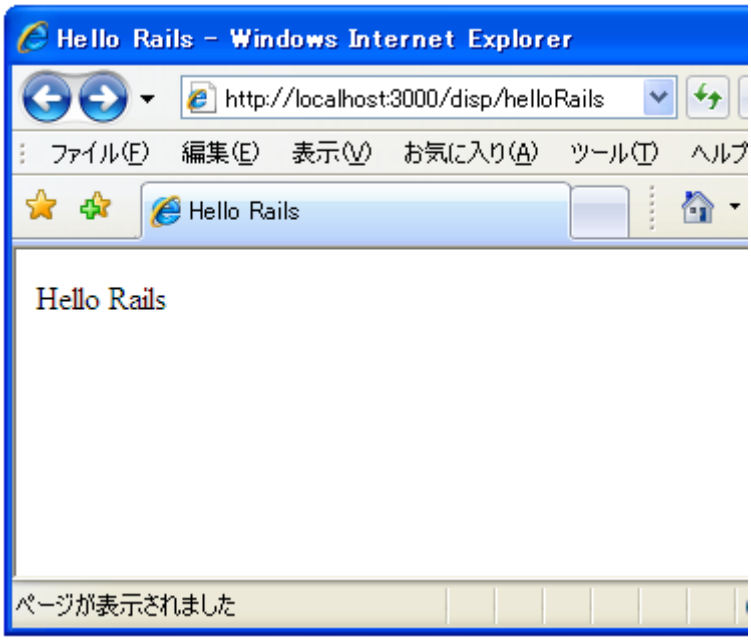
次に「ctr」の部分は使用するコントローラー名を記述する（今回は「disp」を記述）．最後に「action」の部分は実行するアクション名を記述する（今回は「helloRails」を記述）．

よってクライアントから「helloRails」アクションを呼び出す時の URL は以下のようになる．

http://localhost:3000/disp/helloRails

URL だけを見るとサブディレクトリ内のファイル名を指定しているように見えるが，Rails を使用している場合はディレクトリという概念は無く呼び出すべきコントローラーとアクションをディレクトリ階層のように指定しているだけである．

実際に試してみる．まずプロジェクト内で Web サーバを立ち上げる．そしてクライアントから「http://localhost:3000/disp/helloRails」を呼び出す．アクションが指定されて呼び出されると結果として対応するビューがクライアント側に以下のように表示される．



5.2 Ruby on Rails による実装

Rails にはあらかじめ Rails で構築したアプリケーションに独自機能を追加できる Ruby on Rails プラグインというものが存在する．ここでは，数あるプラグインの中でも通常のセッションを用いた認証，ベーシック認証，Cookie による認証が利用できる認証プラグイン restful_authentication プラグインを利用する．そして，それに独自機能を追加し実装を行う．

本節では，restful_authentication プラグインを利用してどのように実装を行うかを図 5.1 の本研究で実装した Rails アプリケーションのモデル，ビュー，コントローラの各役割と関係をもとに述べる．

上述したように，Rails は MVC に基づく Web アプリケーションである．クライアントから HTTP サーバを通じて送られてきたリクエストは，ルータと呼ばれるプログラムによって解析され，コントローラのメソッド（アクション）が呼び出される．コントローラはモデルを通じてデータベースの操作を行い，モデルのデータを反映したビューを生成してクライアントに返す．以下に，実装の流れの概要を示す．

1. アプリケーションを作成するためのプロジェクトを作成
2. 認証プラグイン restful_authentication プラグインをインストール
3. 認証用のコンポーネント一式を生成．（プロジェクトを作成した際に自動で作成された「script」ディレクトリの中に「generate」と Ruby プログラムが用意されているので，それらを用いてデータベースに関する処理を担うモデルとセッション管理用のコントローラを作成．この段階で図 5.1 中のモデル，ビュー，コントローラの黒字部分の機能が作成される．）
4. 3. によってユーザ情報テーブルの migration ファイルが作られているので，rake により migration タスクを実行しユーザ情報テーブルを作成．（この段階で図 5.1 中のデータベースが作成される．）
5. 図 5.1 中のコントローラとビューの赤字部分を独自機能として作成

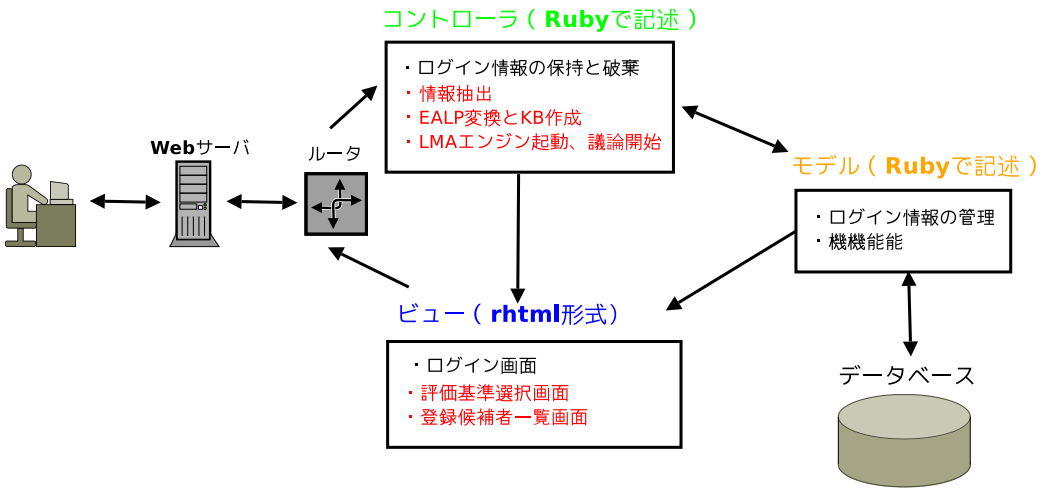


図 5.1: モデル，ビュー，コントローラの各役割と関係

restful_authentication プラグインに独自機能を追加した Rails アプリケーションの動作フレームワークを図 5.2 に示す．図中の矢印の上に記述された文章はユーザの動作，矢印の下に緑色で記述された文章は Rails の動作である．このように，Rails はモデル，ビュー，コントローラの雛形を自動生成する機能やプラグインを提供しているので，それを利用し独自機能を追加することでユーザが求める機能を短期間で開発できる．

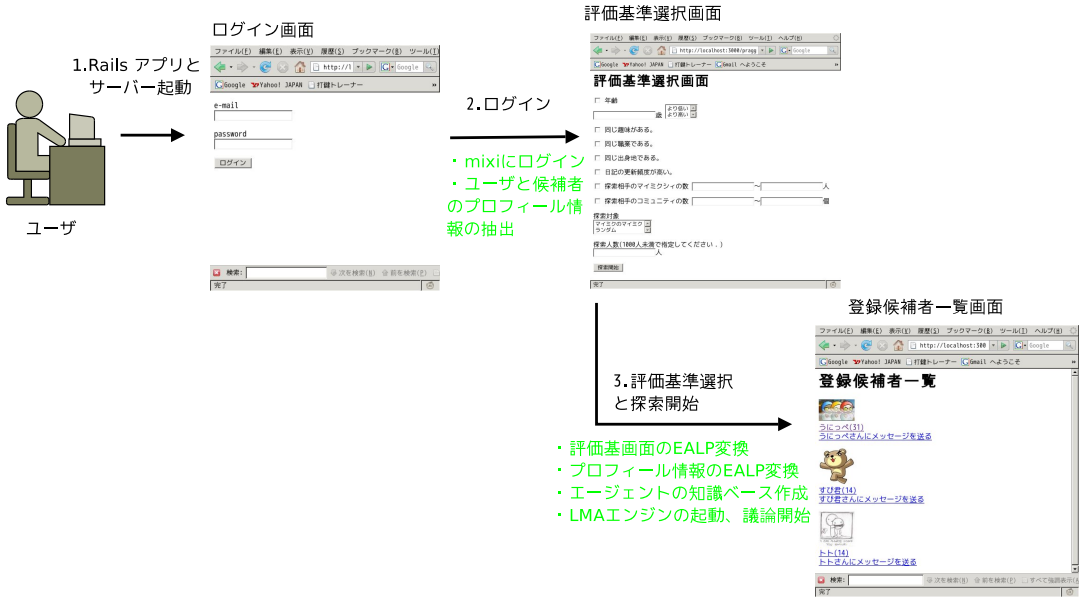


図 5.2: アプリケーションの動作フレームワーク

第6章　まとめと今後の課題

本研究では，数ある SNS の中の mixi と Wikipedia というソーシャルウェブアプリケーションに注目し，それらに我々が提案する拡張注釈付き論理プログラム EALP と多値議論の論理 LMA を適用した．真理値として SNS の mixi の問題では実数値を，Wikipedia の問題では 2 値を用いた．このように EALP は，問題に応じた真理値を選択できるという多様性を持つ．

mixi では各ユーザのプロフィールページから情報を抽出し，マイミクシィの管理をサポートするエージェントを提案した．エージェントはマイミクシィへの追加の審査，マイミクシィからの削除の審査，マイミクシィへの追加の提案の 3 つの機能を持ち，ユーザの意思決定を支援すると共に，プロフィールの合いそうなユーザを提案することでコミュニケーションの輪を広げる効果を期待できる．また例を通し議論がマイミクシィの管理に効果的であることを示した．

本研究で提案したエージェントはユーザのプロフィール情報は常に真であると考えるが，プロフィールを偽装するユーザもいるので状況に応じて判断できる様々な視点を持ったエージェントを今後考える必要がある．現在，プロフィールの情報だけでなく，紹介文の一覧や参加しているコミュニティ，さらにはそのコミュニティ内での評判などをもとに多角的に評価するために研究中である．また，本研究では潜在的有効性についてしか言及していないので，作成したアプリケーションに対してアンケート評価などを行い実際的有効性について言及することも今後の課題である．

一方 Wikipedia では記事の削除についての議論の記録を用いて議論を行い，議論の分析と意思決定を支援するエージェントを提案した．自然言語の議論を論理プログラムへ変換することで議論の論理的構造がわかりやすくなり，さらに議論結果を木構造で表現することによって直観的な議論分析が可能になる．本研究では長期にわたり対処が決定していない議論に対して実際に適応し，削除するか存続するかという意思決定を支援できることを示した．ここでは，一般的な削除の方針のみを扱ったが，Wikipedia ではその他にも「即時削除の方針」,「リダイレクト削除の方針」などが存在するので，今後はそれらについても考慮しなければならない．Wikipedia への応用における大きなボトルネックは，議論ページ中の自然言語を人間が EALP に変換しなければならないことである．しかし，これを克服する二つの試みがある．一つ目は，Araucaria という議論マッピングシステム上で行われた自然言語による議論と LMA における形式的な議論との相互変換システム [16] の研究である．この論文では触れられてないが，現在はある程度の制約のもとで自然言語を EALP に変換することも可能になった．二つ目は，関係データベースを利用して議論をマイニングする研究 [1] である．これにより，Wikipedia の議論ページで行われている議論と同様の議論をマイニングすることが可能であると考える．

SNS，Wikipedia ともに EALP で記述された知識ベースの存在を仮定しているが，現在

オントロジーや，セマンティック Web が注目を集め盛んに研究され [12][4]，我々も Web オントロジーを知識ベースに用いたエージェントシステム [14] や知識ベースのデザインをサポートするシステム [7] を提案した．そのためこれは非現実的な仮定ではないだろう．

現在，企業が社内で SNS や Wiki をグループウェアとして利用したり，マーケティングなどに利用したりするケースが目立ってきている．また熊本県八代市の地域 SNS「ごろっとやっちろ [20]」のように自治体も住民との情報や意見の交換の場として SNS を注目しているため，今後も SNS，Wiki ともに社会のインフラストラクチャーとして発展していくだろう，今後，ソーシャルウェブが社会基盤となった時に情報格差を生まないためにも，本研究のようにソーシャルウェブにおける生活をサポートするエージェントはますます必要になっていくと考える．

関連図書

[1] Safia Abbas and Hajime Sawamura. Towards argument mining from relational argument database. In *Proc. of the 2nd Int. Workshop on Juris-Informatics (JURISIN2008) Workshop*, pages 22–31, 2008.

[2] Amazon!Japan. http://www.amazon.co.jp/.

[3] C. I. Chesñevar, G. Maguitman, and R. P. Loui. Logical models of argument. *ACM Computing Surveys*, 32:337–383, 2000.

[4] C. I. Chesñevar, Jarred McGinnis, Sanjay Modgil, Iyad Rahwan, Chris Reed, Guillermo Simari, Matthew South, Gerard Vreeswijk, and Steven Willmott. Towards an argument interchange format. *The Knowledge Engineering Review*, 21:293–316, 2006.

[5] Flickr. http://www.flickr.com/.

[6] Google. http://www.google.co.jp/.

[7] Takashi Isogai, Taro Fukumoto, and H. Sawamura. An integrated argumentation environment for arguing agents. In *2006 IEEE / WIC / ACM International Conference on Web Intelligence (WI 2006)*, pages 1077–1078. IEEE Computer Society, 2006.

[8] Mixi. http://mixi.jp/.

[9] MSN!Japan. http://jp.msn.com/.

[10] MySpace!Japan. http://jp.myspace.com/.

[11] Ruby on Rails. http://rubyonrails.org/.

[12] C. Reed and T. J. Norman, editors. *Argumentation Machines*. Kluwer Academinc Publishers, 2004.

[13] Ruby. http://www.ruby-lang.org/ja/.

[14] H. Sawamura, T. Wakaki, and K. Nitta. The logic of multiple-valued argumentation and its applications to web technology. In *P. E. Dunne and T. J. M. Bench-Capon, editors, Computational Models of Argument*, pages 291–296. IOS Press, 2006.

[15] T. Takahashi and H. Sawamura. A logic of multiple-valued argumentation. In *Proceedings of the third international joint conference on Autonomous Agents and Multi Agent Systems (AAMAS'2004)*, pages 800–807. ACM, 2004.

[16] Y. Takahashi, H. Sawamura, and J. Zhang. Transforming natural arguments in araucaria to formal arguments in lma. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 668–678. IEEE Computer Society, 2006.

[17] Wikipedia. http://ja.wikipedia.org/wiki/.

[18] Yahoo!JAPAN. http://www.yahoo.co.jp/.

[19] Youtube. http://www.youtube.com/.

[20] ごろっとやっちろ. http://www.gorotto.com/.

[21] Ruby On Rails 入門. http://www.rubylife.jp/rails/index.html/.