ソフトウェア要求仕様書

「DiscussionVisualizerの改良」

東京工業大学工学部情報工学科　佐伯研究室

高橋　碧 (Takahashi Aoi)
梅田　侑 (Umeda Yuu)

2017年5月22日

目 次

第1章　仕様書の概要

1.1　目的

本要求仕様書は、Issue Tracking System (ITS) 上の issue の議論の流れを構造化して視覚的に表示するツールである「DiscussionVisualizer」を改良することによって、ITS 利用者が議論の流れをより把握しやすくするために、「DiscussionVisualizer」が満たすべき要求やその理由、制約などを明らかにすることを目的とする。本ツールの開発者及びステークホルダ、特に本ツールの使用者を対象読者とする。

1.2　範囲

本ツールは ITS 利用者の issue の内容把握を支援する。また、本ツールでの ITS は JIRA を対象とし、JIRA にある issue に関しての議論構造の生成や、編集を行う。本ツールを利用することで、issue 上の議論の流れを視覚的に捉えることができるという利点がある。本ツールに対して、ユーザビリティを向上させることでその利点を最大限生かせるようにする。さらに、協調作業支援機能 (CSCW) を付与することで、個人での issue の理解だけでなく、複数人での issue の理解を共有できるようにする。

これらの内容を満たすことで、「DiscussionVisualizer」の実用性の向上を図ることが目的である。

1.3 用語定義

表 1.1: 用語定義

用語	定義
DiscussionVisualizer	昨年度、佐伯研究室に所属していた大内さんが作成した issue 上の議論構造可視化と理解支援をするツールのこと
Issue Tracking System (ITS)	課題管理システムのこと
JIRA	ITS のツールの一つ
Ruby on Rails	Web アプリケーションフレームワークの一つ
Model View Controller (MVC)	アプリケーションソフトウェアを実装するためのデザインパターンの一つ Ruby on Rails ではこのデザインパターンが採用されている
Graphviz	グラフを描画するツール
プロジェクト	JIRA 上に設けられているプロジェクトのこと
issue	JIRA 上に設けられている issue のこと。プロジェクトごとに存在する
議論構造	issue 上の議論を Graphviz を用いて木構造として可視化したグラフのこと
Computer Supported Cooperative Work (CSCW)	協調作業支援機能。複数人でツールを利用し、作業を進めるための機能
コメント	JIRA の issue 上のコメントを抽出し、議論構造と同時に 「DiscussionVisualizer」に表示されているもの
タグ付け	コメントに対して、特定のタグを付けること
フィルタリング	コメントに付いているタグに従って、コメントの一部を表示する機能
議論参加者名	JIRA の表示名

1.4　概要

本要求仕様書では、第 2 章の全容で、要求分析の結果考えられる現状の問題点に関して、具体的に確認する。また本ツールの機能の概要を記述する。第 3 章の要求仕様では、本ツールの詳細な機能の仕様について記述する。

第2章　全容

2.1　製品の背景

現状のツールでは、個人での issue 上の議論構造の可視化を目的としている。また、元の「DiscussionVisualizer」の実装期間の関係で、機能はきちんと実装されているが、実際にツールとして使いやすいものとは言えない。そのため以下の問題点が存在する。

現状のツールに関する問題点は、大きく CSCW とユーザビリティである。

CSCW について、現状では個人での issue 管理を目的としてツールが作られているため、共同で issue 管理をすることができない。このことは、複数人で issue 管理をしたい場合には致命的な問題点である。

ユーザビリティについて、現状では議論構造を編集しづらく、タグ付けをしたコメントに対するフィルタリングの種類が少ないことによる、コメントの整理のしづらさがある。他には、issue が全て同じ階層で一覧になっているため、多くの issue を管理する際にとても見づらい。また、JIRA では issue ごとに issue 間の関係が定義されているが、その関係を「DiscussionVisualizer」では抽出できていないため、管理したい issue がどの issue と関係があるのか分からない。このような問題点がある。

これらの問題点を解決するために、いくつかの機能を実装する。

2.2　製品機能

ユーザビリティに関して、本ツール改良案では以下の 4 つの機能により、上記の問題点を解決する。

- 議論構造の編集・整理のユーザビリティ向上
  議論構造の編集のしやすさを向上させ、編集のコストを軽減させる。

- タグに対するフィルタリングの種類の追加
  フィルタリングの種類を増やすことで、タグ付けを最大限生かし、コメントを整理しやすくする。

- プロジェクトごとのグループ化
  issue を同じ階層で一覧にするのではなく、プロジェクトごとに issue を一覧にすることで、issue が多くなった場合にも見やすさを維持する。

- 複数 issue 間の関係の可視化
  JIRA に元々存在する関係を抽出し、表示することで、issue 間の関係を分かりやすくする。

また、自動でタグ付けをする機能を拡張し、ユーザが任意の議論参加者に対してタグ名を登録し、その議論参加者に自動でタグ付けをする機能も実装する。

CSCW に関して、本ツール改良案では以下の 4 つの機能により、上記の問題点を解決する。

- ユーザ名の追加
  アカウントに対して、ユーザ名を追加し共同作業の際に自身を特定する名前を設ける。

- プロジェクト/issue 一覧の共有
  プロジェクト/issue 一覧を共有することで、複数人で共有しながら、プロジェクト/issue を追加したり閲覧できたりする。

- 議論構造の共有
  議論構造を共有することで、複数人で共有しながら、議論構造を編集したり、整理できたりする。

- issue ごとに担当者・状態の管理
  issue ごとに担当者や状態を管理することで、複数人で共有しながら、issue を管理できる。

2.3　仮定および依存性

本ツールでは、ITS として JIRA を利用していること、また元の「DiscussionVisualizer」を利用することが可能である環境を仮定している。具体的には、Ruby on Rails 5.0.0、ruby 2.3.1、graphviz がインストールされている必要がある。

第3章　要求仕様

3.1　外部インターフェイス要求

3.1.1　ユーザインターフェイス要求

初回利用時には、新規アカウントを作成する。そうでない場合にはログインをして本ツールを利用する。新規アカウント作成の場合には、ユーザ名、メールアドレス、パスワードを入力する必要がある。ログインの場合には、メールアドレスとパスワードを入力する必要がある。ログイン後、新規プロジェクト追加のページから新規プロジェクトを追加し、Jira の issue の URL を新規 issue 作成のページから入力することで議論構造を自動生成し、その議論構造に対してユーザは閲覧・編集を行うことができる。また、issue 一覧ページから issue を閲覧できる。ログイン時にはどのページからもログアウトをすることができ、ログアウトをした場合には、新規アカウント作成もしくはログインをするページに戻ることができる。

3.2　機能要求

ここでは、具体的な機能について記述する。また、機能は実装の優先順位順に記述していく。1ヶ月の間で実装をするというスケジュールを考慮し、[最低限機能] と [発展機能] に機能を分類する。

3.2.1　[最低限機能] CSCW の向上

- ユーザ名を追加

  新規アカウント登録をする際に、ユーザ名を追加する欄を作り、アカウント情報にユーザ名を追加する。これにより、issue ごとに担当者を追加する際にも、

現在アカウント登録されているユーザ名の中から担当者を選択することが可能になる。他にも、issue 一覧ページにユーザ名を表示することで、自分のログインしたアカウントが本当に自分のアカウントであるのかというのも明確にわかるようになり、共同作業をする際には便利である。以下に、この機能を実装した場合の画面例を示す。

図 3.1: 新規アカウント登録ページ

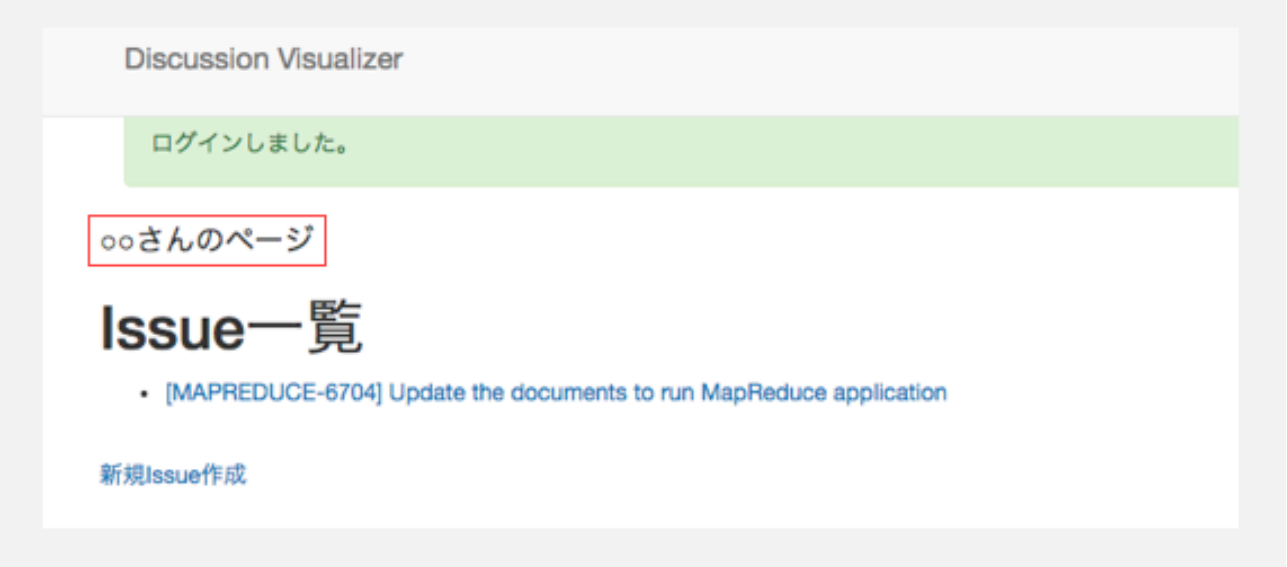

図 3.2: issue 一覧ページにユーザ名を表示

- プロジェクト/issue 一覧の共有

  複数人が issue を管理していても、その issue 一覧が共有されなければ共同で作業を進めることはできない。そのために、他の誰かが issue を追加した場合に、自分の issue 一覧にその issue が追加されていなければいけない。そのための機能である。すべての issue はプロジェクトごとに分類され、自分の issue 一覧のページに表示される。担当者機能を実装する場合、自分が担当している issue に絞って表示できる機能を実装する。以下に修正前と修正後の画面例を示す。

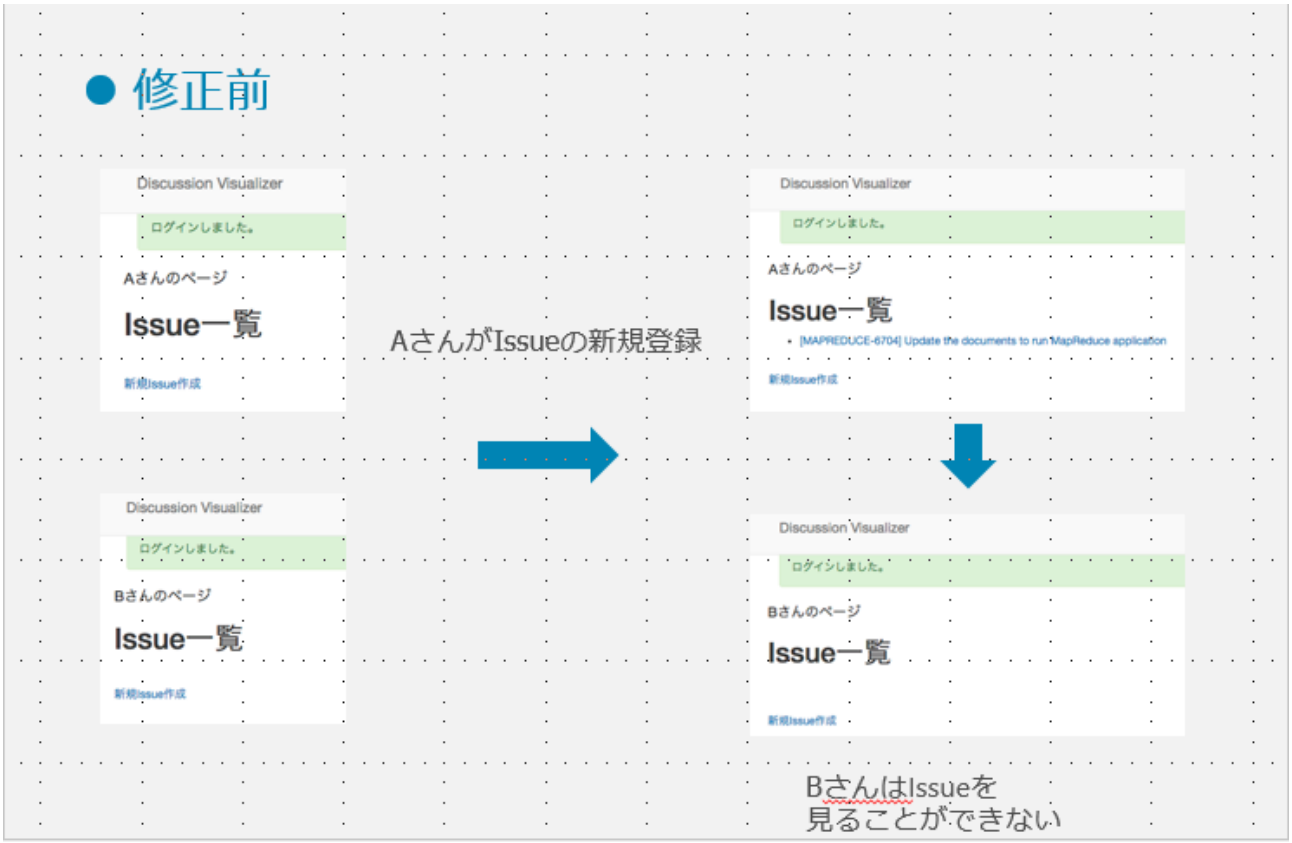

図 3.3: プロジェクト/issue 一覧の共有 (修正前)

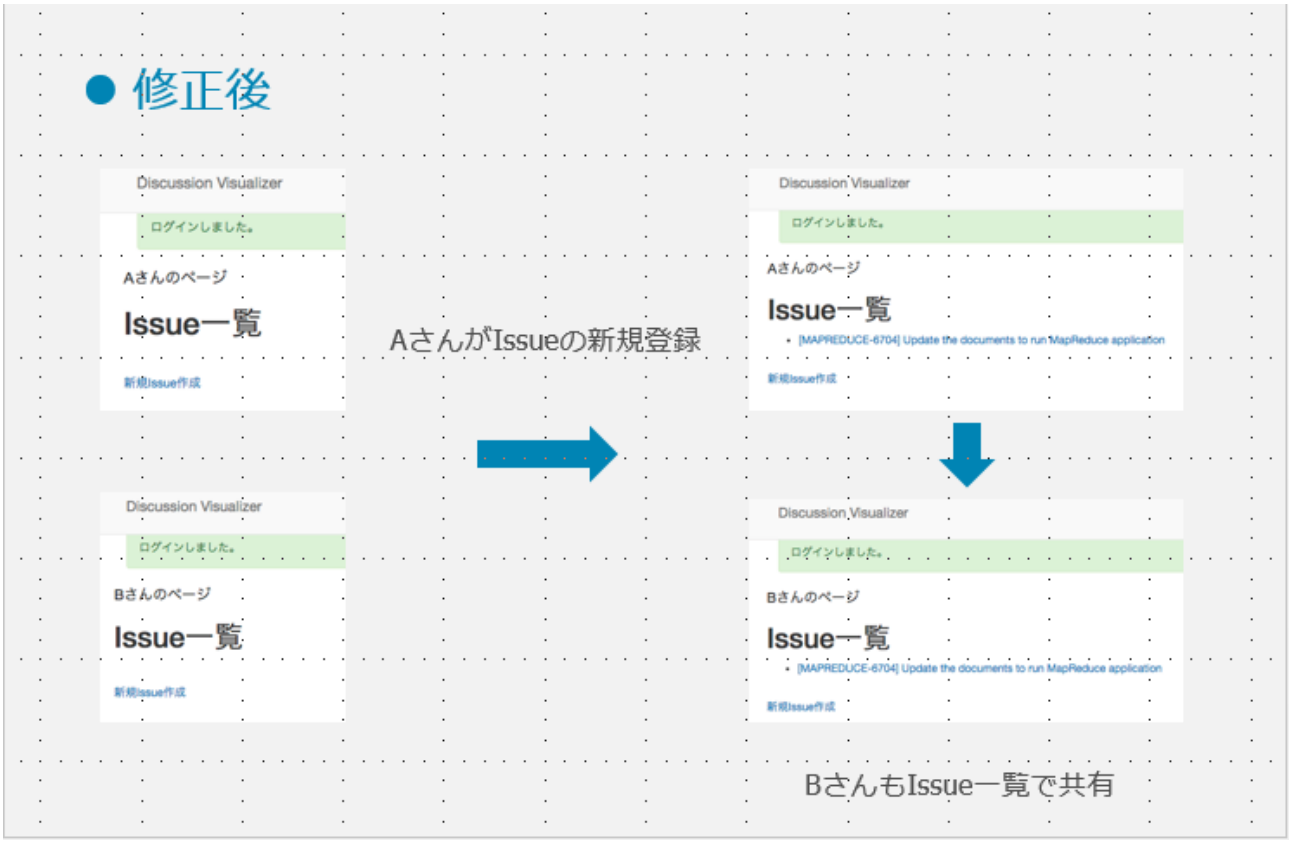

図 3.4: プロジェクト/issue 一覧の共有 (修正後)

- 議論構造の共有

  プロジェクト/issue 一覧の共有と同様に、議論構造に関しても、議論構造の変更が共有されていなければ、共同で作業を進めることはできない。そのための機能である。以下に、修正前と修正後の画面例を示す。

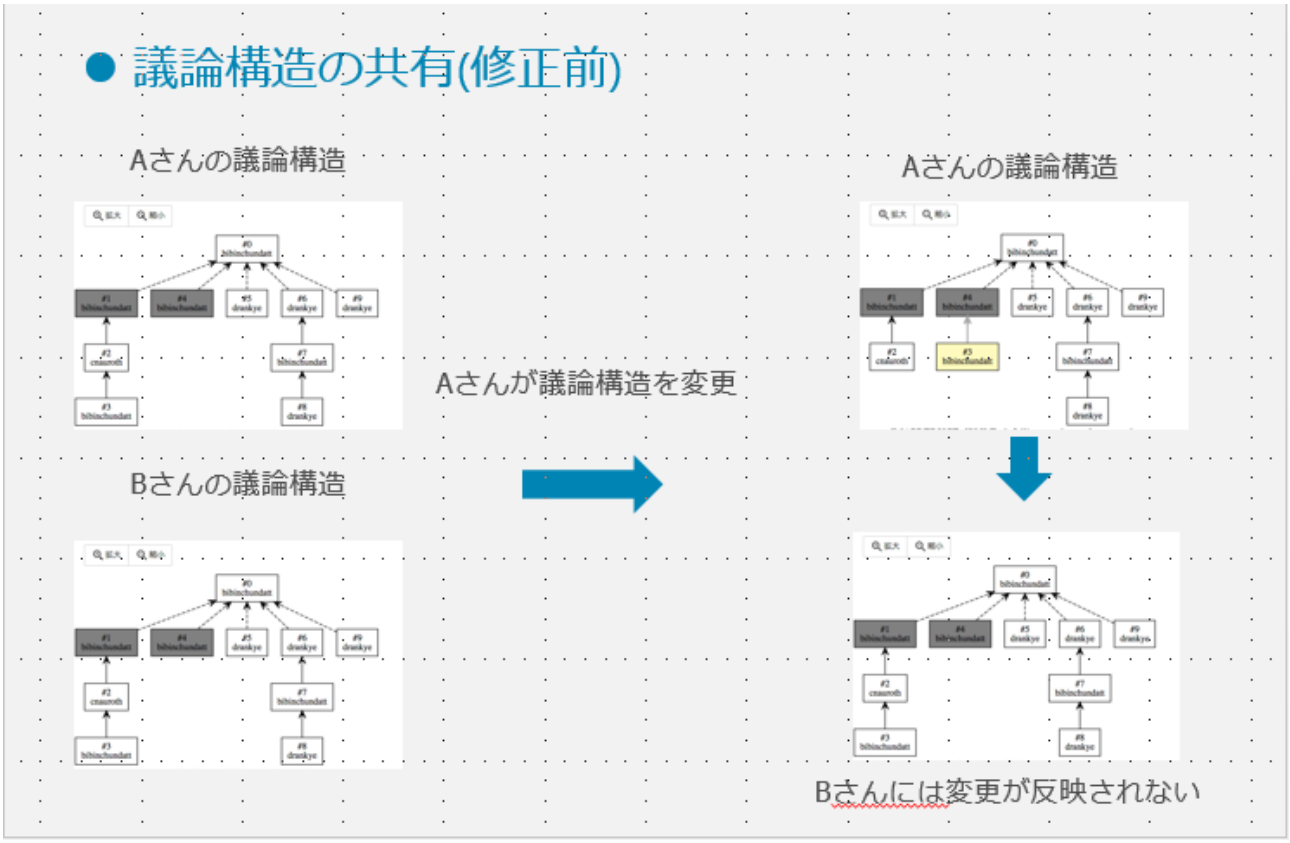

図 3.5: 議論構造の共有 (修正前)

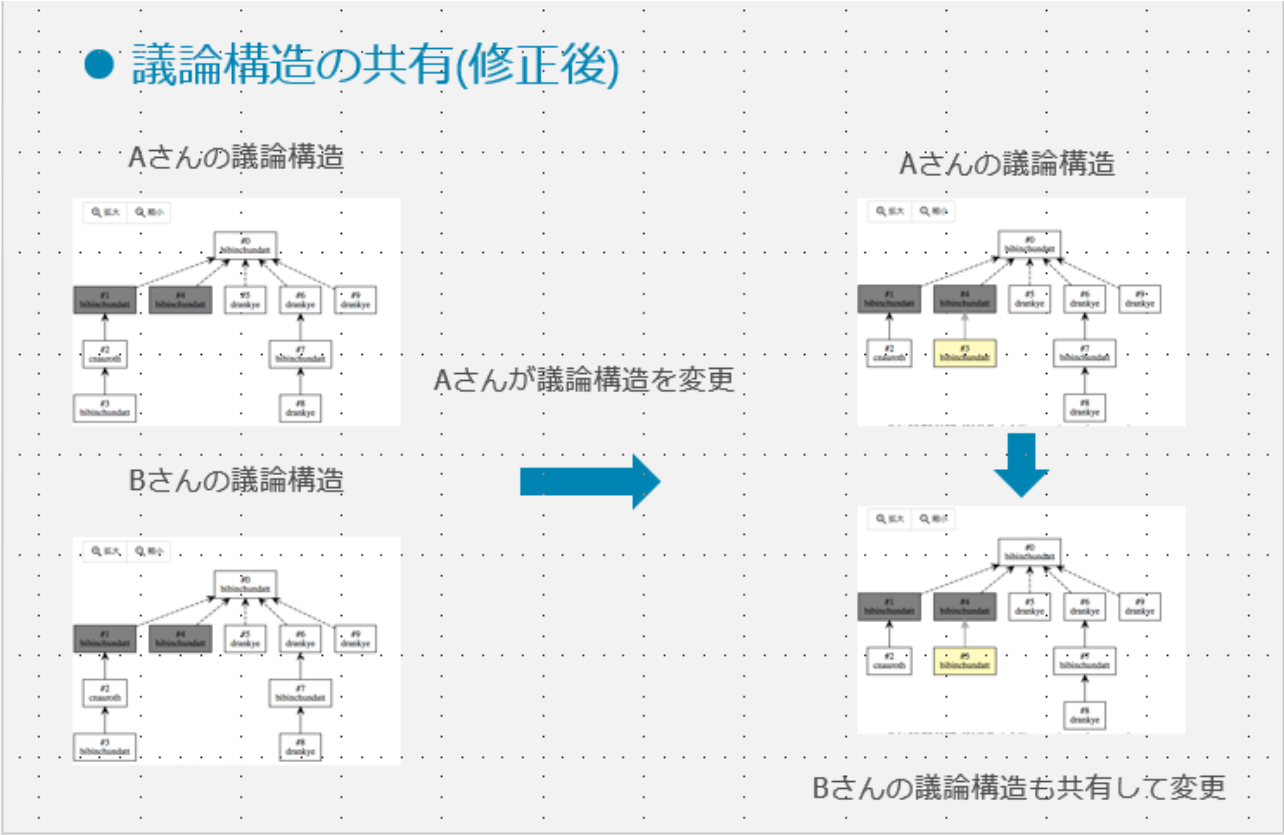

図 3.6: 議論構造の共有 (修正後)

共有に関して、現在の実装の説明では、ほぼ同時に複数ユーザが編集をした際に編集のコンフリクトが起こる可能性があるが特に対策は行わないため、注意して運用する。もし同時に二人のユーザーが議論構造を編集した場合、後の編集が反映される。

3.2.2 [最低限機能] プロジェクトごとのグループ化

多くの issue が追加された場合の見づらさを防止するために、issue をプロジェクトごとに管理する。具体的には、プロジェクトを手動で追加し、プロジェクトごとに issue を追加できるようにする。そうすることで、プロジェクトごとに issue をグループ化でき、見やすさが向上する。以下に、画面例を示す。

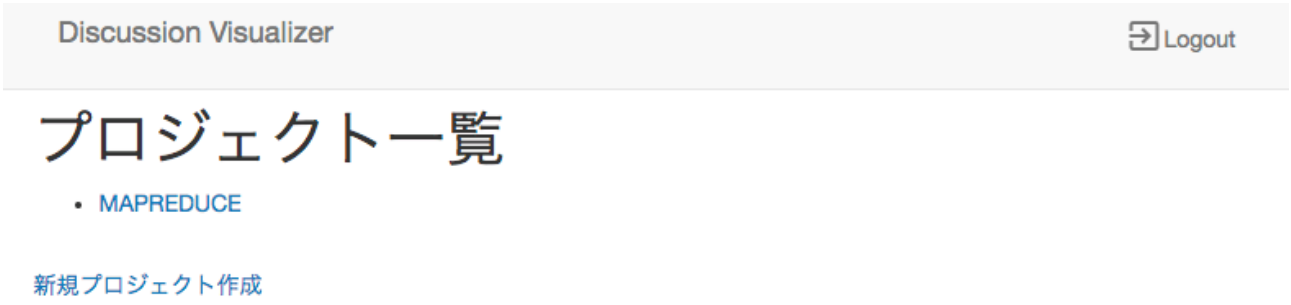

図 3.7: プロジェクト一覧ページ

まず、今までは issue 一覧であったページをプロジェクト一覧ページに変える。こうすることで、まず閲覧したいプロジェクトを選択することになる。

図 3.8: プロジェクト新規登録ページ

また、プロジェクト一覧ページでは新規にプロジェクトを追加することもできる。

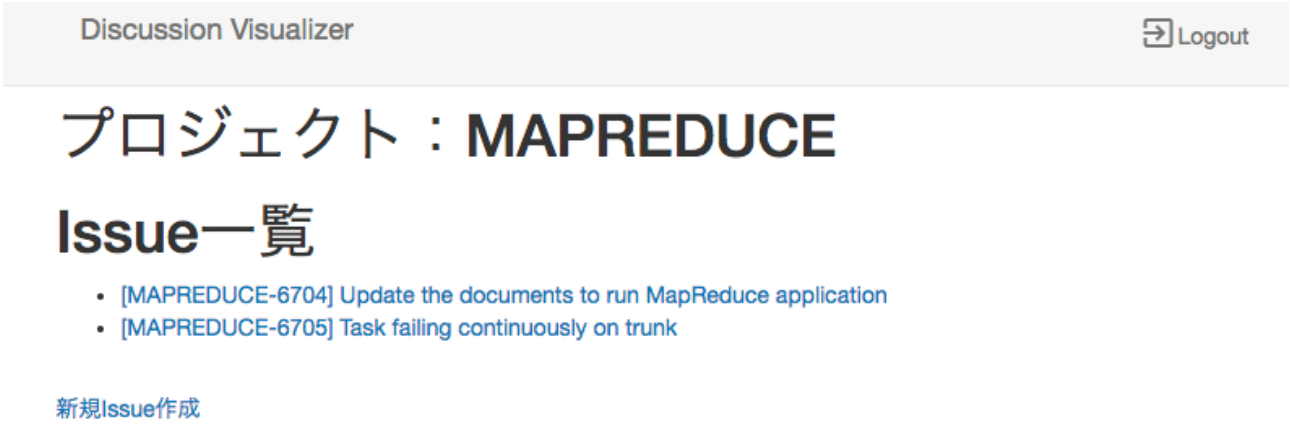

図 3.9: プロジェクトごとの issue 一覧

プロジェクト一覧ページのプロジェクトを選択すると、そのプロジェクト内の issue 一覧を閲覧することができる。また、issue を追加したい場合には、プロジェクト内の新規 issue 作成ボタンから、プロジェクトごとに issue を追加することができる。さらに、プロジェクト一覧ページにも新規 issue 作成ページを用意し、プロジェクト一覧ページの新規 issue 作成ページから新規 issue を作成した場合、issue 名などからプロジェクト名を自動で判定して適切なプロジェクトに割り当てる。

3.2.3 [最低限機能] プロジェクトごとに自動タグ付けを登録

現在では、issue を登録した際に、コメントに自動でタグ付けがされるようになっている。しかし、そのタグ付けはあらかじめ用意されているタグ付けのみである。そのため、上に述べたプロジェクトごとに、自動で、タグ付けをするものを登録できるようにする。登録ページでは、自動でタグ付けをしたい議論参加者名と、その議論参加者名 (JIRA の表示名) にどのようなタグ付けをしたいかを入力して登録する。登録すると、過去に登録したすべての issue について新しい自動タグ付け規則によってタグをつける。また、タグを削除した場合、過去に付けたタグも削除されるものとする。以下に、画面例を示す。

図 3.10: 自動タグ付け登録ページ

画面例として示しているが、実際にはもう少しユーザビリティを向上したいと考えている。この画面では一度に登録できるのが 1 組だけだが、そうではなく複数の登録をまとめてできるような入力フォームにすべきである。具体的には、カンマ区切りの入力、ボタンによる入力フォームの追加等が挙げられる。

3.2.4 [発展機能] タグに対するフィルタリングの種類の追加

現状では、フィルタリング機能はタグを一つ選択するとそのタグが付いているコメントがフィルタリングされる。しかし、そのようなフィルタリングだけでなく、選択したタグを持っているコメントだけを非表示にするフィルタリングや、複数タグを選択してフィルタリングをすることもできるようにする。

3.2.5 [発展機能] 複数 issue 間の関係の可視化

概要でも記述した通り、JIRA では issue 間に関係が存在する。その関係を現状では取り込むことができておらず、どの issue とどの issue が関係し合っているのかを確認できない。このことを改善するために、issue ごとに JIRA での issue 間の関係を表示するような機能をつける。以下に、画面例を示す。

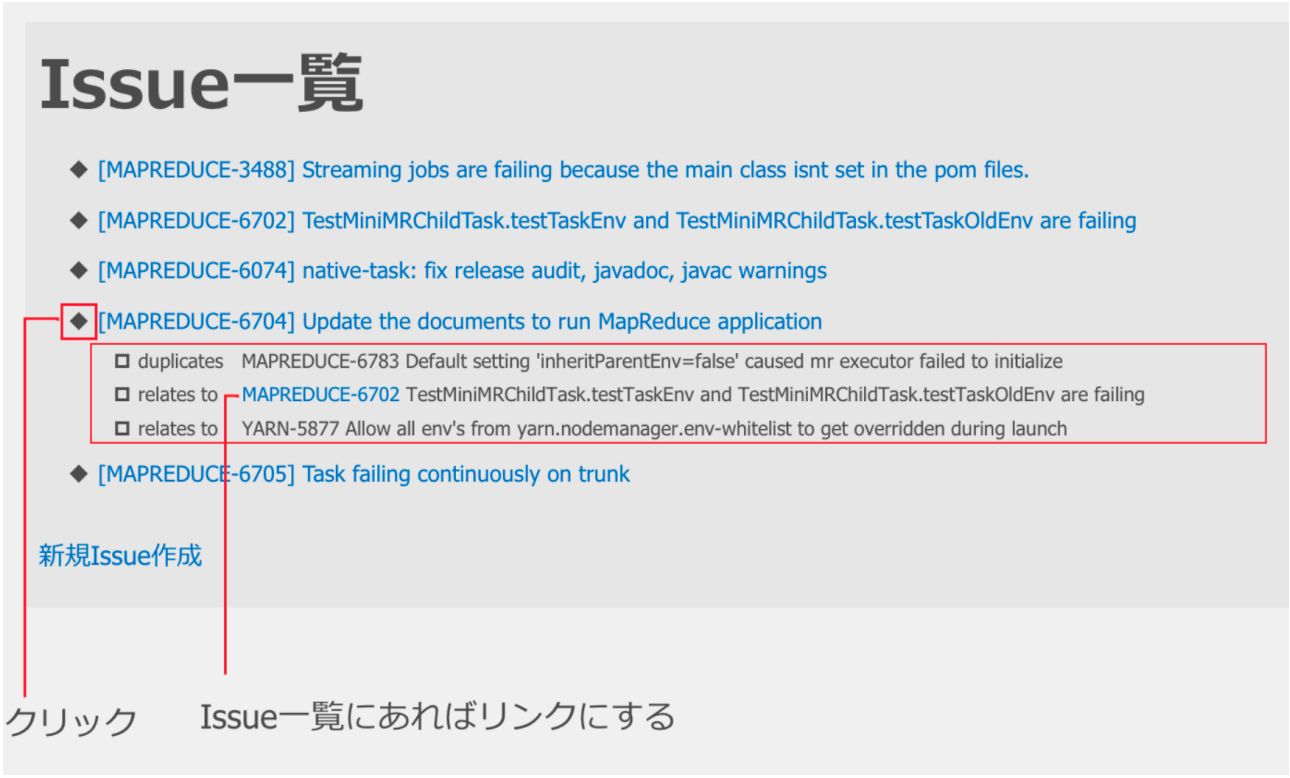

図 3.11: issue 間の関係の可視化

上に示す通り、issue 一覧でのそれぞれの issue について、ひし形のボタンをクリックすると、下にその issue の関係している issue を表示するようにしている。また、表示した issue がすでに issue 一覧に存在すれば、リンクを貼りその issue の議論構造に移動することができるようにする。この時、関係している issue が存在する場合は何らかの方法で関連 issue の存在を示すものとする。現状では菱形の横に [関連 issue 有り] と表示することで示すものとする。

3.2.6 [発展機能] issue ごとに担当者・状態の管理

ここでは、各 issue 画面において、issue 一つ一つに対して担当者・状態をつけることができるようにする。こうすることで、ユーザーは、その issue は誰が管理していて、どのような状態であるかを共有することができる。担当者は存在するアカウントのユーザ名の中から選択することができ、状態はこちらでいくつか用意した状態の中から選択する形にする。その情報は issue 一覧画面において issue の横に表示し、確認できるようにする。

3.2.7 [発展機能] 議論構造の編集・整理のユーザビリティ向上

全ての機能の実装が終わった場合に、実装する機能である。議論構造の編集は現在使いやすいものとは言えない。そのため、いくつかの機能を実装することで、議論構造の編集を簡単なものにする。現状では、議論構造の編集はコメントのノードをクリックした際に出るポップアップからしかできない。なので、議論構造の変更をドラッグなどの使いやすい実装に変えることを考えている。また、議論構造のリセットもできるようにすることで、議論構造を作り直したいときに便利である。他にも議論構造の編集・整理のユーザビリティが向上する実装を適用していくつもりである。

3.3 データベース要求

以下に、作成する機能に関わる、重要なテーブルとテーブルに設けるカラムを中心に内容を説明する。

3.3.1 users テーブル

このテーブルでは、アカウントごとにメールアドレス、ユーザ名、パスワードなどを持つ。

- id
- **email** (メールアドレス)
- **username** (ユーザ名)
- role
- **encrypted_passward** (パスワード)
- reset_password_token
- reset_password_sent_at
- remember_created_at

- sign_in_count
- current_sign_in_at
- last_sign_in_at
- current_sign_in_ip
- last_sign_in_ip
- created_at
- updated_at

3.3.2 projects テーブル

新しく作成したテーブルである。プロジェクトごとにプロジェクト名を持つ。

- id
- **name (プロジェクト名)**
- created_at
- updated_at

3.3.3 issues テーブル

このテーブルでは新しく、タイトルを含まない issue 自体の名前 (例、MAPREDUCE-8704) 、プロジェクト id、担当者のユーザ id、状態の種類を持つ。

- id
- url
- type
- title (タイトルを含めた issue の名前)
- created_at

- updated_at
- **name (issue 自体の名前)**
- **project_id (プロジェクト id)**
- **user_id (担当者のユーザ id)**
- **status_type (状態の種類)**

3.3.4 issue_relations テーブル

新しく作成したテーブルである。複数 issue 間の関係の可視化のために用意したもので、どの issue に関する関係なのか、どのような関係があるのかなどの情報を持つ。

- id
- **source_issue_id (どの issue に関する関係か)**
- **target_issue_title (関係先の issue の title)**
- **target_issue_ name (関係先の issue の name)**
- **type (関係の種類)**
- created_at
- updated_at

3.3.5 comments テーブル

このテーブルでは、所属している issue、コメントをした議論参加者名、コメントの内容などを持つ。

- id
- **issue_id (コメントの所属する issue)**
- **author (コメントをした議論参加者名)**

- **content (コメントの内容)**
- type
- JIRA_id
- internal_id
- created_at
- updated_at

3.3.6 edges テーブル

このテーブルでは、議論構造において、どのコメントとどのコメントをつないでいるのかなどの情報を持っている。

- id
- user_id
- **source_comment_id (どのコメントからの edge なのか)**
- **target_comment_id (どのコメントへの edge なのか)**
- type
- created_at
- updated_at
- reason

3.3.7 tags テーブル

このテーブルでは、タグがどのコメントについているのか、タグの内容は何かなどの情報を持っている。

- id

- user_id
- **comment_id** (どのコメントについているタグなのか)
- **content** (タグの内容)
- created_at
- updated_at

3.3.8 auto_tag_authors テーブル

このテーブルは元々、ツールがコメントをする場合の議論参加者名 (bot など) のみを対象としていたテーブルであったが、今回の機能の追加により自動タグ付けをする議論参加者名全体を対象とするテーブルに変更した。そのため、どのようなタグが議論参加者名に対して存在するのか、またプロジェクトごとに議論作者名を登録するため、どのプロジェクトに適用する自動タグ付けなのかなどを情報として持つようにした。

- id
- author_name
- created_at
- updated_at
- **tag_content** (自動タグ付けするタグの内容)
- **project_id** (どのプロジェクトに関するものなのか)

3.3.9 bookmarks テーブル

このテーブルでは、元から備わっているブックマーク機能のために、どのコメントに対するブックマークなのかなどを持っている。

- id

- user_id
- comment_id (どのコメントに対するブックマークか)
- created_at
- updated_at

3.3.10　その他のテーブル

そのほかにもいくつかテーブルがあるが、機能に関わるものでは無いのでテーブル名だけ列挙する。

- logs テーブル
- schema_migrations テーブル
- versions テーブル
- ar_internal_metadata テーブル

3.4　ソフトウェア属性

3.4.1　使用性

本ツール改良案では、ただ使いやすいものになっていたり多くの機能があることに比べて、複数の人が共同で作業できるツールにすることが求められている。そのため、ユーザビリティにも気を配るが、主として CSCW に着手しなければならない。そのことを意識した機能の実装順序にもなっている。また、ツール使用者の要求の優先順位も合わせて考慮し、より使いやすいツールを実現する。

3.4.2 拡張性

まず、本ツールは Ruby on Rails で MVC に基づいて設計がされている。このことは拡張性を高める。また CSCW の向上という機能の中で、共同作業支援に関する、別の機能を追加しようとした場合にも柔軟に対応できるようなデータベース設計をする。またもし今後、ITS として JIRA 以外に対してもこのツールを適用させようとした際に、プロジェクトごとに issue をグルーピングすることで、プロジェクトごとに ITS を対応づけることが可能である。

3.4.3 移植性

本ツールは、元々「DiscussionVisualizer」の実装にも使われていて、広く Web アプリケーション開発に使われている Ruby on Rails を使うことで移植性を高める。