B-009

異なるチャットシステム間でメッセージ送信可能なシステムの構築
A system for bridging the different chat systems to send the messages

宮崎 光 †
Ko Miyazaki

玉田 春昭 ‡
Haruaki Tamada

1. はじめに

今日，社内の連絡にビジネスチャットを使う事例が数多く報告されている．例えば，JR 西日本は，社員間，駅間の連絡に direct[1]を導入すると報告されている[2]．
このように，ビジネスチャットが広く使われるようになっている．

チャットシステムの導入により，日常的な連絡がカジュアルかつ迅速に行えるようになり，一般的にメールに比べ，コミュニケーションオーバーヘッドが軽減できる．しかし，ビジネスチャットでの社内連絡が当たり前になると，２つの会社が合同でプロジェクトを実施する時に問題がある．各会社それぞれのチャットシステムへの相互参加が難しいことである．一方のチャットシステムにユーザ登録することは，会社のセキュリティの問題上，許されないことも考えられる．また，別のチャットシステムを使う場合であっても，アプリケーションや画面の切り替えなど，メッセージの確認／送信に多少なりともオーバーヘッドが生じる．さらに，従来のようにメールでコミュニケーションを行うと，さらにコミュニケーションオーバーヘッドが大きくなる．普段利用しているチャットシステムに，ユーザ登録の必要なく，会社間（チーム間）を安全に結ぶ必要がある．

この問題を解決するために，チャットシステムのブリッジサービス，sameroom.io [3]が存在する．sameroom.io を利用することで，異なるチャットシステムでもメッセージのやり取りが行えるようになる．そのため，上記問題を解決できる．しかし，サービスであることから，オンプレミスに未対応である点，そして，リアクションに未対応である点に改善の余地がある．リアクションとは，チャット上のメッセージに各ユーザが付けられる絵文字によるフィードバックである．このリアクションは，いいね（👍）に代表されるように，メッセージ入力を考える手間を軽減させることにより，コミュニケーションオーバーヘッドを下げられる．

そこで，本稿では，オンプレミスも可能なように，ブリッジシステムを再構築し，絵文字によるリアクション機能に対応する方法について議論する．つまり，図 1 に示すように，ボットが中継し，チャットシステムを超えてメッセージの送受信を行えるようにする．図 1 では，チャットシステム A と B を繋げている．まずユーザ A がチャットシステム A にメッセージを投稿する（1-1）．チャットボットは，ユーザ A の投稿をチャットシステム B に代理で投稿する（1-2）．同様に，チャットシステム B に投稿されたユーザ B のメッセージもチャットシステム A にチャットボットが代理で投稿する（2-1, 2-2）．

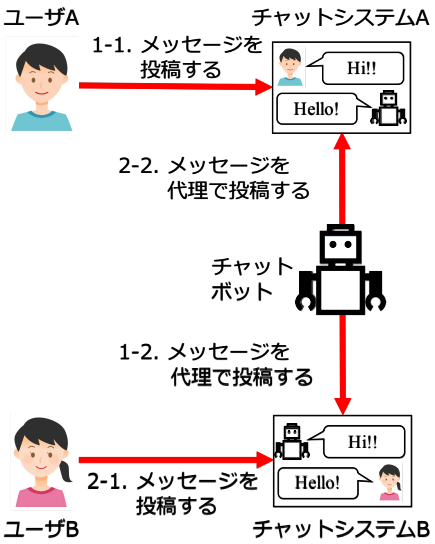

図1. 提案システムの概要

2. 提案手法

2.1 チャットシステム

本稿では，チャットシステムを次のように扱う．チャットシステムは，複数のチームを持つ．チーム間はつながりを持たず，メッセージの送受信が行えない．このチームは，部署単位や開発チーム単位，また企業など，特定の団体と紐付くことが多い．

また，１つのチームには，複数のルームが存在する．このルームは特定の話題に紐付くことが多い．

そして，チーム内には，複数のユーザが登録されており，その中の有志がルームに参加する．一般的にルームへの参加，不参加はユーザが任意に行える．また，一人のユーザが複数のチームに所属する場合もある．

そして，各ユーザは自身が参加するルームに対してメッセージを発信できる．メッセージは発信時間を表すタイムスタンプ，発信者などの情報を含む．発信されたメッセージは，そのルームに参加しているユーザ全員に公開される．

また，リアクションはメッセージに付けられる．いいね（👍）や OK（🆗）など，１ユーザが複数のリアクションを付けられる．また，別のユーザが同じ種類のリアクションを同じメッセージに付けられる．そして，メッセージを閲覧可能であれば，誰がそのリアクションを行ったのか判別可能とする．

† 京都産業大学大学院，Graduate School of Kyoto Sangyo University.
‡ 京都産業大学，Kyoto Sangyo University.

[1] https://direct4b.com/ja/
[2] https://japan.cnet.com/article/35100987/
[3] https://sameroom.io/

本稿では，異なるチームのルーム同士を結びつけることを考える．ここで，異なるチームは同じチャットシステム上に構築されているとは限らないことを前提とする．

なお，チャットシステムによっては，チームに相当する概念が存在しないものもある（Skype など）．そのような場合，ルームがチームの役割も持ち，チームとルームが一対一対応しているものと扱うことで，提案手法は適用可能である．

2.2 利用シナリオ

ここで，提案手法の利用シナリオを考える．チーム A とチーム B が持つルーム r_a, r_b をブリッジする．ここで，チーム A に所属するユーザ a が r_a にメッセージを投稿する．この時，投稿に反応して，ブリッジシステムが起動する．ブリッジシステムは投稿されたメッセージをブリッジ先である r_b にユーザ a に代わり投稿する．チーム B にユーザ a が所属していないためである．なお，本来の投稿ユーザである a の情報を失うと，コミュニケーションの大きな問題に繋がることから，投稿ユーザ a の情報をメッセージに含める．

次に，ユーザ a により，r_a に投稿されたメッセージ m_a に対してリアクション r が行われたことを考える．この時もメッセージの投稿と同様にリアクションに反応してリアクションをブリッジする．r_b に m_a に対応するメッセージ m_b が存在しているはずである．そのため，ブリッジシステムが m_b に対して，リアクション r を行う．なお，ブリッジシステムが代理でリアクションを行うため，元のユーザ a の情報が失われてしまう．この点は本システムの限界であり，今後，何らかの別の対応が必要となる．

もちろん，リアクションシステムに対応していないチャットシステムとブリッジする場合も同様に対応を考える必要がある．

2.3 提案システム

前節で述べた利用シナリオを実現するため，提案システムは，３つのコンポーネントを用意する．２つのチャットボットとブリッジシステムである．２つのチャットボットはチーム A, B のそれぞれが利用するチャットシステムにインストールする．このチャットボットはメッセージの通達と代理投稿の２つの役割を実施する．

メッセージの通達では，ボットは，メッセージの投稿や，ユーザのリアクションに反応して起動する．そして，投稿内容やリアクションの情報をブリッジスステムに必要な情報を含めて通知する．

一方の代理投稿では，ブリッジシステムからのメッセージを受け取れることにより起動する．そして，受け取った情報から，メッセージを必要な情報を含めたものに再構成し，代理でルームに投稿することである．

そして，ブリッジシステムは，多数のチャットボットの対応を管理する．そして，あるチャットボットから情報が送られてくると，対応するチャットボットに対して情報を伝達する．このように，ブリッジシステムを構築する．

3. 実装

3.1 概要

提案手法を実現するために，2 つの Slack のチームを結ぶことを考える．Slack は，Hubot[1]など，ボットの作成

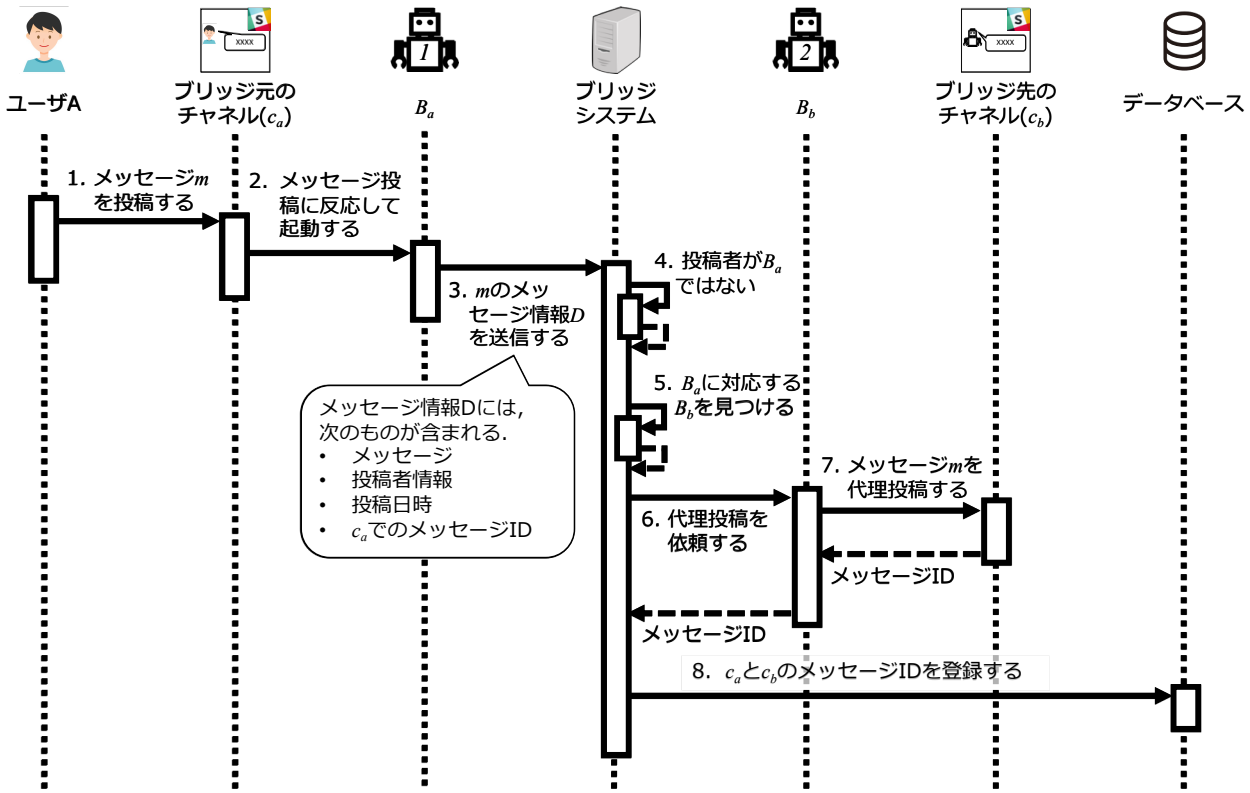

図2. メッセージ投稿時の提案システムの振る舞い

[1] https://hubot.github.com

が容易であることから選択した．なお，今回は，チャットシステムを Slack に限定しているが，異なるチャットシステムであっても，チャットボットの実装をそのチャットシステムに合わせる必要がある程度で，実装の内容自体は同様である．

Slack には，第 2.1 節で述べたチームがあり，そして，チームにはルームに相当するチャネルがある．そこで，異なるチームのルーム同士（それぞれ c_a, c_b とする）を結びつける．

3.2 メッセージのブリッジ

両チームに，今回作成したチャットボット B_a, B_b をインストールする．チャットボットの実装の詳細は後述する．ブリッジシステムには，B_a と B_b がインストールされたチームのルーム c_a, c_b を結びつける設定が行われる．これにより，両ルームがブリッジシステムにより繋げられた．

次に，チャットボットの実装の詳細を述べる．チャットボットの役割は大きく投稿通知と代理投稿の 2 つがある．投稿通知では，ルーム内でのメッセージの投稿に反応して起動する．そして，ブリッジシステムに投稿されたメッセージ，投稿ユーザ，投稿日時，メッセージ ID をデータ D として，ブリッジシステムに送信する．

一方の代理投稿では，ブリッジシステムからの通知に反応して起動する．投稿通知で B_a から送られたデータ D が B_b に送信される．その時の情報を元に，c_b に代理で投稿する．

そして，ブリッジシステムは，B_a から D が送られてくると，対応する B_b を探す．そして，B_b が見つかれば，D を送信する．さらに，D から c_a におけるメッセージ ID（$id_a(m)$）と c_b におけるメッセージ ID（$id_b(m)$）を取得する．そして，両メッセージ ID をデータベースなどに保存しておく．リアクションはメッセージ ID に対して付与するものであるが，c_a と c_b 間でブリッジしたメッセージ，すなわち，同じメッセージであってもメッセージ ID が異なる．リアクションをブリッジするためには，メッセージ ID の対応付けが必要であるためである．なお，代理投稿の場合は，投稿通知の処理が実行されないようにする必要がある．そのため，投稿者が B_a の場合は，何もせずに処理を終える．

これら一連のメッセージをブリッジする処理を図 2 にシーケンス図として示す．上記の通り，ユーザの投稿によりボット B_a が起動し，ブリッジシステムにメッセージ情報 D を渡す．ブリッジシステムは D が送られてくるとまず代理投稿の可否を判定する．代理投稿を行う場合は，B_a に対応する B_b を探し，B_b に対して，代理投稿のリクエストを投げる．リクエストの結果として，c_a と c_b における同一メッセージのメッセージ ID を取得し，データベースに保存する．この一連の処理により，メッセージのブリッジが完了する．

3.3 リアクションのブリッジ

リアクションについては，ブリッジシステムは，メッセージの送受信とほぼ同様に処理できる．B_a, B_b が行う処理が多少異なる．投稿通知においては，どのメッセージにどのようなリアクションが行われたのかをメッセージの代わりにブリッジシステムに通知する．そして，B_b では，D を受け取ったあと，どのメッセージに付けられたリアクションであるのかを判別し，代理でリアクションを行う点が異なる．

図 3 にリアクション投稿時の提案システムの振る舞いをシーケンス図として示す．図 3 から分かるように，図 2 と同じように処理を進める．ただし，ブリッジするリア

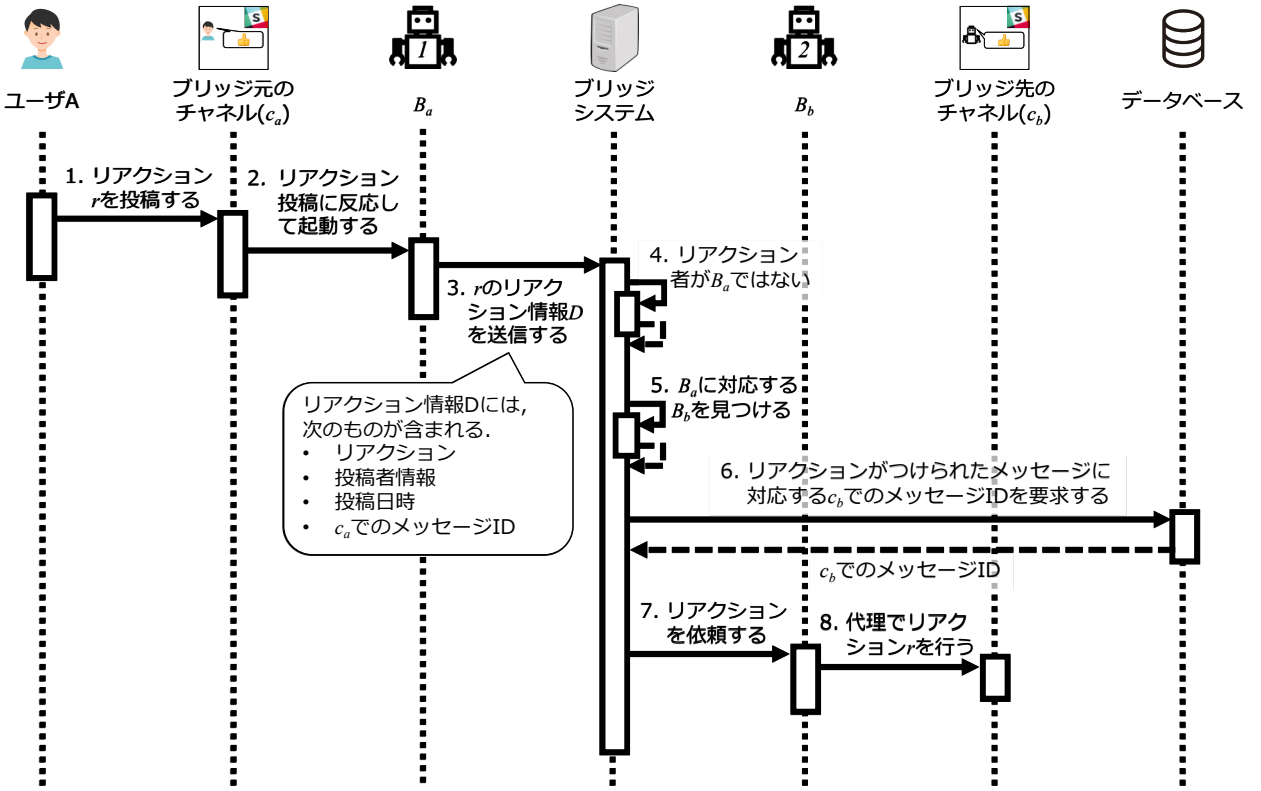

図3. リアクション投稿時の提案システムの振る舞い

クションがどのメッセージに付けられるのかをデータベースに問い合わせ，得られたメッセージ ID を元にリアクションを実行する点が異なる．この場合も，B_b による代理リアクションにより，投稿通知の処理が実行されないようにする必要がある．図 3 上の 4 がその判定処理に相当し，リアクション者がインストールしたチャットボットでないことで判定している．

3.4 CiBridge

上記の内容を元に，CiBridge を作成した．チャットボットは，Slack API を利用しアプリケーションを作成した．また，ブリッジシステムは，Google App Script として作成した．

なお，ブリッジシステムは REST サービス[1]として構築しており，B_a からのデータは単純な HTTP POST リクエストとしてブリッジシステムに送られる．加えて，代理投稿のための B_b でのデータ受信についてもブリッジシステムと同様に，B_b 内の HTTP サーバ上に REST サービスとして構築される．そして，HTTP GET として受け取る．もちろん，実際の運用においては，不正なアクセス防止のためアクセストークンでの認証や https での通信を行う．

4. 議論

提案手法では，c_a で行われたアクション（メッセージの送信，リアクションの実行）をチャットボット B_b が c_b で実行するものである．そのため，ブリッジしたメッセージ，リアクションには，次に述べる制限がある．

1. メッセージの送信者をアイコンや名前で識別できない．
2. リアクションを 1 つのチャットボットが行うことにより，投票や既読確認が行えない．

まず，1 のアイコンや名前でメッセージ送信者を識別できないという問題について述べる．この問題は，c_a で投稿されたメッセージを c_b にチャットボット B_b が投稿することに起因する．c_b に c_a のユーザは登録されていないためでもある．ただし，B_b が投稿するメッセージは編集できるため，投稿者情報を含めたメッセージに変更することで対応できる．

次に，2 の投票や既読確認が行えない問題について述べる．リアクションを使って，投票や既読確認を行う場合がある．しかし，リアクションをブリッジした場合，c_a でのリアクションを c_b で行うのは，B_b である．ca で複数人がいいね（👍）のリアクションを行なったとことを考える．この場合，c_a では，誰が👍を行なったのかを確認できる．しかし，c_b では B_b がリアクションを行うことから，誰が行なったか，何人が行なったかを確認できない．この問題は，リアクションのみでは解決できない．c_a のユーザは c_b に存在しないかつ，登録しない前提であるため，B_b が代表してリアクションを行うことになるためである．そこで，一般的にメッセージが書き換え可能であることに着目し，リアクションの詳細を示すメッセージ m_r をつけることが考えられる．リアクションをブリッジする度に，m_r の内容を書き換えて詳細を示すようにすることで対応できる．この方法は，リアクションに未対応のチャットシステムに対しても応用可能である．

また，提案手法では，チャットシステムごとに対応したチャットボットを作成する必要がある．あらゆるチャットシステムへの対応を維持するのは非常に困難である．そこで，Hubot などボットライブラリを利用することで複数のチャットシステムへの対応コストを下げられる．

5. 関連研究

Slack には，共有チャネルがある．共有チャネルは，Slack の異なるチーム同士が共有して使えるチャネルである．しかし，Slack 以外のチャットシステムとの連携は不可能である．そのため，Slack 以外のチャットシステムともメッセージをやりとりする場合は，提案手法や sameroom.io のようなブリッジシステムを利用する必要がある．しかし，sameroom.io では，前述のようにリアクションやオンプレミスには未対応なため，それらへの対応が必要であれば，提案手法である CiBridge を利用する必要がある．

6. まとめ

本稿では，チャットシステムを跨いでメッセージを送受信するブリッジシステムに着目した．そして，オンプレミスも可能なように，ブリッジシステムを再構築し，絵文字によるリアクション機能にも対応する方法を提案した．そのために，投稿通達と代理投稿を担うチャットボットと多数のチャットボットの対応を管理するブリッジシステムを用意した．そして，提案手法を元に，まずは Slack に向けて，CiBridge を作成した．CiBridge を動作させることで，実際にメッセージ並びにリアクションのブリッジが可能であることを確認した．最後に，現状の問題点とその解決方法について示した．

今後の課題として，CiBridge の客観的な評価や，実践的な環境での利用を踏まえた考察を行う．

謝辞

本研究の一部は JSPS 科研費 17K00196，17K00500，17H00731 の助成を受けたものです．

参考文献

[1] Roy Fielding, “Architectural Styles and the Design of Networ-based Software Architectures,” Doctoral dissertation, University of California, 2000.