Vol.2013-SE-179 No.23
2013/3/12

Linked Data を用いた サービス連携アーキテクチャの提案と評価

小島弘誉[†1]　中道上[†1]　青山幹雄[†1]

Linked Data を用いてサービス間のスケーラブルなデータ連携を実現するサービス連携アーキテクチャを提案する．近年，クラウドなどのサービスを連携して利用する方法が注目されつつある．その連携には大量のデータを扱えるスケーラビリティが求められる．しかし，ESB(Enterprise Service Bus)などの連携ではブローカに負荷が集中するため，スケーラブルな連携が困難となっている．本研究では，Linked Data を用いたサービス連携アーキテクチャを提案する．アーキテクチャを制御と処理の二層に分け，Linked Data で振舞いを制御することにより，スケーラブルなデータ連携を実現する．さらに，提案アーキテクチャのプロトタイプを開発し，連携のスケーラビリティを評価することで提案アーキテクチャの有効性を示す．

A Service Integration Architecture with Linked Data

HIROTAKA KOJIMA[†1]　NOBORU NAKAMICHI[†1]　MIKIO AOYAMA[†1]

We propose a service integration architecture with Lined Data. The architecture enables scalable data integration. Currently, the service integration method is in the spotlight because cloud computing is widely used on the business. The integration method needs scalability to integrate large amounts of data. However, the methods based on ESB(Enterprise Service Bus) make broker burdened with concentration payload. It is difficult to architect scalable integration method. In this thesis, we propose a service integration architecture with Linked Data. We separate the architecture into control layer and transaction layer. Control layer employs Linked Data. Therefore, the service integration architecture is scalable. We developed a prototype of the architecture. We evaluated scalability of the architecture with respect to data integration and evaluated effectiveness of the proposal architecture.

1. はじめに

クラウドコンピューティングが普及し，クラウドコンピューティングが提供するサービスの連携に注目が集まっている．従来から，ESB(Enterprise Service Bus)や EAI(Enterprise Application Integration)など，異なるアプリケーションや Web サービスを連携する研究が行われてきた．しかし，クラウドコンピューティングではプロバイダが多数存在し，各プロバイダが様々なサービスを提供しているため，既存の技術だけでは実現できない要求がある [5]．その中でも，各サービスが作成する多様なデータを多量に交換することは利用者にとって重要な要求である．そのため，スケーラビリティが必要とされている．

本研究は Linked Data を用いた連携アーキテクチャを提案する．サービス連携を制御と処理の二層に分け，Linked Data で振舞いを制御することにより，連携の負荷が非集中な連携を可能とする．また，処理層のデータ交換はデータ受信者が配置されているリソースリポジトリから直接データを取得するため，データサイズに対してスケーラブルな連携を実現する．

2. 研究課題

2.1 スケーラブルなデータ連携が困難

不特定多数のサービス連携では，連携に参加しているサービスの負荷がブローカに集中する．これにより，スケーラブルなデータ連携が困難になっている．

2.2 クラウドサービスの疎結合な連携が困難

不特定多数のサービス連携では，連携参加者の時間，位置，状態によらずデータを送受信する必要がある．

3. 関連研究

(1)　サービスブローカ

ESB[6]など，SOA(Service-Oriented Architecture)に基づきエンタープライズ全体のサービス統合技術が利用されている [13]．このため，プロトコルの整合，メッセージルーティングなどのサービス連携技術が提案されている．しかし，スケーラビリティを保証する具体的なアーキテクチャは提案されていない．

(2)　Linked Data を用いたデータ連携

Linked Data[4]を用いて多様なデータを連携する技術が提案されている [10]．この連携方法では Linked Data のシステムアーキテクチャに基づき，様々なドメインのデータを Linked Data で管理する．これにより，ドメインの知識領域外の関連する知識のデータも取得可能にする．しかし，この研究はコンセプトにとどまっており，莫大なデータの検索を統一的なインタフェースで行うことから負荷がブローカに集中することがわかる．しかし，スケーラビリティに関しては議論されていない．

(3)　WS-Brokered Notification

†1 南山大学
Nanzan University.

ⓒ 2013 Information Processing Society of Japan

Publish/Subscribe Architecture[8]を実現する仕様である[12]．メッセージ配信者とメッセージ受信者の間にブローカを配置することで，配信者と受信者はメッセージの送受信を分離できる．そのメッセージは非同期で交換されるため，ブローカ，メッセージ受信者の状態によらず連携が可能である．また，ブローカで配信時間，受信者の位置を管理することで，配信者と受信者は時間と空間から分離される．このアーキテクチャではブローカに負荷が集中し，スケーラブルな連携は困難である．

4. 連携アーキテクチャのアプローチ

動的に連携の振舞いを変更させるため，アーキテクチャをメタレベルとベースレベルの二層に分けるアプローチをとる[7](図 1)．メタレベルは連携の制御，ベースレベルは連携の処理を担う．このアプローチをとり，メタレベルで Linked Data[4]を用いて連携の振舞いを制御することにより，ブローカへの負荷の集中を各クラウドサービスに分散させることが可能となり，連携の負荷が非集中のアーキテクチャとなる．これを二層 LDA(Linked Data Architecture)と呼ぶ．

制御層は Publish/Subscribe Architecture に基づき連携参加者の時間分離，空間分離を可能とする．処理層はリソース指向[9]に基づきデータ交換することで連携参加者の状態分離を可能とし，疎結合な連携を実現する[14]．

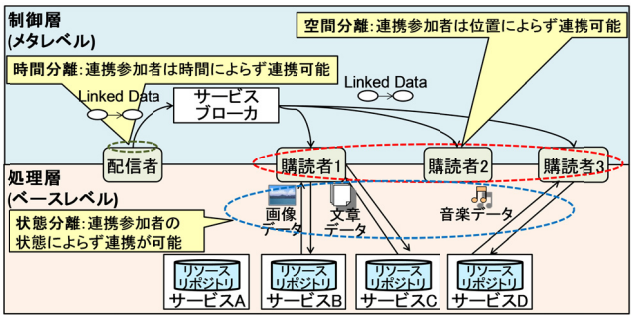

図 1　提案のアプローチ

Figure 1　Approach of the proposal

5. 二層 LDA の提案

5.1 二層 LDA の構造

二層 LDA の構造を図 2 に示す．メタレベルで連携の制御，ベースレベルで連携の処理を行う．各レベルの機能を説明する．

5.1.1 メタレベル: サービスブローカ

(1)　購読管理サービス

購読管理サービスの役割は二つある．

(a)　購読情報の登録

購読者から購読登録を受ける．購読者は少なくとも購読者の取得したいデータ，購読者のエンドポイント，購読者の振舞いを登録しなければならない．取得するデータの取捨選択はメタ購読情報に定義された選択肢に従って定義する．

(b)　購読情報の参照

フィルタリング時にはフィルタリングサービスから購読管理サービスに配信者から取得した Linked Data に関連する購読情報があるか問い合わせがある．問い合わせがあると，購読情報データベースを参照し，関連する購読情報がないか検索し，ある場合は購読情報をフィルタリングサービスに返す．

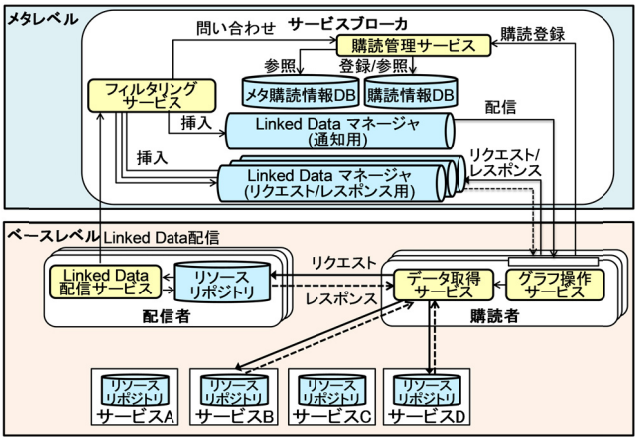

図 2　二層 LDA の構造

Figure 2　Structure of the proposal

(2)　メタ購読情報 DB

取得するデータの取捨選択を可能とするプロパティを定義しておくリポジトリである．メタ購読情報 DB に登録しておくことで Linked Data を詳細的にフィルタリングすることが可能になる．

(3)　購読情報 DB

購読情報を保存しておくリポジトリである．連携に参加する全ての購読者の情報がこのリポジトリに保存される．

(4)　フィルタリングサービス

フィルタリングサービスの役割は購読情報に合わせて Linked Data をフィルタリングし，Linked Data マネージャに登録する．Publish/Subscribe Architecture に定義されているフィルタリング方法を三つ示す．

(a) トピックベースフィルタリング

トピックベースフィルタリングは Liked Data の種類を分類してフィルタリングする方法である．トピックは抽象度が高いまとまりを表現可能なため，多数の Linked Data をやりとりする場合に向いている．

(b) コンテンツベースフィルタリング

コンテンツベースフィルタリングはプロパティを用いて Linked Data が持つリンクやリテラルの値でフィルタリングする方法である．プロパティの粒度でフィルタリングするため，具体的な条件を記述可能だが，

多数の Linked Data をやりとりする場合には購読情報が複雑化する．

(c) タイプベースフィルタリング

タイプベースフィルタリングはトピックベースフィルタリングとコンテンツベースフィルタリングの二つの方法を併せ持つ方法である．Linked Data の種類が事前に定義された型と一致することを利用している．これにより，Linked Data を型で分類し，さらにその型のプロパティで分類することができる．

Linked Data マネージャに Linked Data を登録する際には購読情報に従って通知用とリクエスト/レスポンス用に分類して行う．

(5) Linked Data マネージャ

購読情報に登録されたタイミングでメッセージを配信するサービスである．

二層 LDA では異なる振舞いに対応するためにキューを二つに分類して管理する．

(a) 定期的なタイミング

購読者の振舞いが通知型の場合，ブローカは購読登録されたタイミングに合わせて Linked Data を配信する．そのため，異なる購読者の Linked Data を一つのキューで管理することが可能である．

(b) ランダムなタイミング

購読者の振舞いがリクエスト/レスポンスの場合，購読者が任意のタイミングで Linked Data を要求する．ブローカは購読者がリクエストしたタイミングで Linked Data を配信する必要があることから，購読者毎にキューを作成し，Linked Data を管理する．

5.1.2 ベースレベル: 配信者

(1) Linked Data 配信サービス

Linked Data の作成と配信を行うサービスである．作成する Linked Data は次の二種類ある．

(a) 配信者へのリンク

配信者が保持しているデータへのリンクである．この Linked Data を作成する場合は配信者がリソースリポジトリを外部に公開する必要がある．

(b) 他のサービスへのリンク

外部のサービスが保持しているデータへのリンクである．

本研究ではこれらの Linked Data を組み合わせて利用することを想定している．

(2) リソースリポジトリ

配信者が保持しているリソースを保管する領域である．二層 LDA の連携を実現するためには HTTP で外部のネットワークからアクセスを可能にする必要があるため，リソースリポジトリに保管されているデータを外部に公開する必要がある．

5.1.3 ベースレベル: 購読者

(1) グラフ操作サービス

配信された Linked Data から取得したいデータの URI を抽出する．Linked Data では RDB ライクにクエリを用いて問い合わせることでデータを抽出することが一般的である．そのため，グラフ操作サービスでも同様の方法でデータを抽出する．

(2) データ取得サービス

グラフ操作サービスで取得した URI を用いてリソースリポジトリにアクセスし，データを取得するサービスである．

5.2 二層 LDA の振舞い

複数の購読者と複数の配信者のデータ交換を例に二層 LDA の振舞いを図 3 に示す．配信者(サービス X)，購読者(サービス A，サービス B)を記述する．購読者は異なる振舞いを持つサービスである．サービス A はリクエスト/レスポンス型，サービス B は通知型のサービスである．また，サービス Y はサービス X が配信した Linked Data が指すリソースリポジトリを保持しているとする．

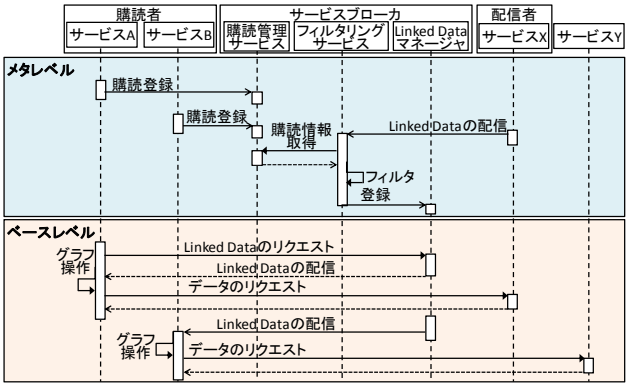

図 3 二層 LDA の振舞い

Figure 3 Behavior of the proposal

5.2.1 全体の振舞い

連携全体の振舞いを説明する．

(1) 購読登録

購読者のサービス A とサービス B がサービスブローカに購読登録する．

(2) 配信者の Linked Data 配信

配信者のサービス X がサービスブローカに Linked Data を配信する．

(3) Liked Data マネージャへの登録

サービスブローカは購読管理サービスから購読情報を要求し，取得する．購読情報が記述されている情報に従って Linked Data をフィルタリングし，Linked Data に購読者の振舞い，配信時間，購読者の宛先などを指定して，購読管理サービスに登録する．

(4) サービスブローカの Linked Data 配信

サービス A のようなリクエスト/レスポンス型のサービ

スの場合は購読者から Linked Data を要求し，Linked Data を取得する．そして，グラフ操作サービスを用いて Linked Data から要求する URI を取得する．その URI を用いてサービス X のリソースリポジトリにアクセスし，データを取得する．サービス B のような通知型のサービスの場合はサービスブローカから購読情報に記載されたトリガに従って Linked Data を配信する．

(5) データの取得

グラフ操作サービスを用いて，Linked Data から要求する URI を取得する．取得した URI を用いて，サービス Y からデータを取得する．サービス Y は Linked Data を配信していないが，サービス X が配信した Linked Data がサービス Y のリソースリポジトリを指しているために，サービス Y のリソースリポジトリにアクセスし，データを取得する．

5.2.2 二層分割による効果

連携アーキテクチャを二層に分けることによる効果をメタレベル，ベースレベルに分けて示す．

(1) メタレベル

メタレベルでは連携参加者が疎結合な連携を可能とするために振舞いの制御を行う．疎結合な連携を可能とする二つの性質を示す．

(a) 空間分離

連携参加者はお互いの位置を知らなくても連携することを可能とする．ブローカが連携参加者のエンドポイントを管理し，登録された購読要求に従って，Linked Data を配信するため，連携途中で配信者や購読者のエンドポイントが変更されても，連携が可能である．

(b) 時間分離

連携参加者は Linked Data の配信，取得を任意のタイミングで行うことを可能とする．ブローカは配信された Linked Data をメッセージキューで保持するため，Linked Data を任意のタイミングで連携することが可能となる．

(2) ベースレベル

ベースレベルでは異なるリソースリポジトリに配置されたデータをリンクさせた Linked Data を用いてデータを取得する．それにより，データ交換の分散とデータ保持の分散を可能とし，負荷が非集中の連携を実現する．また，データ交換では状態分離を可能とする．

(a) データ交換の分散

購読者が Linked Data の指すリソースリポジトリから直接データ交換する．そのため，不特定多数の連携でも，ブローカに負荷は集中せず，購読者とリソースリポジトリを保持するサービスに負荷分散する．

(b) データ保持の分散

二層 LDA ではブローカがデータを保持しない．データは配信者，または，リソースリポジトリを保持しているサービスに配置されるため，ブローカに集中していたデータ保持の負荷を配信者と他のリソースリポジトリに分散させることが可能となる．

(c) 状態分離

連携参加者はお互いの状態によらず連携が可能である．購読者は REST でデータを取得するため，リソースリポジトリを保持する配信者やサービスの状態によらず，連携が可能である．

6. プロトタイプによる実証

6.1 プロトタイプ開発の目的

(1) 実現可能性の確認

プロトタイプを開発することで，二層 LDA の実現が可能であることを確認し，連携アーキテクチャとして妥当であることを示す．

(2) データサイズに対するスケーラビリティの評価

プロトタイプを用いてデータサイズ毎の連携時間を計測することで，データサイズに対するスケーラビリティを評価する．また，関連研究と比較することで二層 LDA の有効性を示す．

6.2 プロトタイプの構成と実現

実装したプロトタイプの構成を図 4 に示す．

連携時間を正確に計測するため，配信者と購読者は同じ PC 上に配置した．また，ネットワーク遅延などで連携時間の計測結果に影響を与えないため，全てのアクタを同一の Web 上に開発した．プロトタイプの実行環境を表 1 に，開発したサービスの LOC を表 2 に示す．

(1) Web サーバ

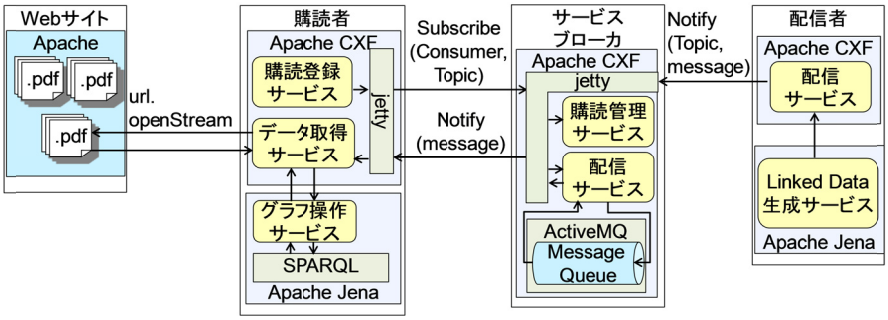

図 4 二層 LDA プロトタイプの構成

Figure 4 Prototype structure of the proposal architecture

配信者，購読者，サービスブローカには Web サーバに Jetty を用い，Web サイトには Apache を用いた．近年，Web サービスは Apache Maven などのプロジェクト管理ツールを用いて開発される．その際には Web サーバに Jetty を用

表 1　プロトタイプの実行環境

Table 1　Environment of the prototype

Linked Data 処理機能	Apache Jena (SPARQL)
メッセージキュー	Apache ActiveMQ
Web サービスフレームワーク	Apache CXF
Web サーバ	Jetty，Apache
OS	Windows

表 2　各サービスの LOC

Table 2　Environment of the prototype

サービス名	LOC
配信サービス	102
Linked Data 生成サービス	28
購読管理サービス	30
配信サービス	23
購読登録サービス	21
データ取得サービス	87
グラフ操作サービス	44
合計	335

いることが一般的である．また，Web サイトでは Web サーバに Apache を用いることが一般的であるため，Apache を用いた．

(2)　Web サービスフレームワーク

Web サービスは Apache CXF[3)]を用いて開発した．Apache CXF には Publish/Subscribe Architecture を実現するライブラリが用意されているため，Apache CXF を用いた．

(3)　Linked Data 処理機能

Linked Data 処理機能は Apache Jena[2)]を用いて開発した．プロトタイプでは Linked Data を作成する機能と Linked Data から要求する URI を抽出する機能を開発する必要があったため，モデル作成のライブラリや SPARQL のライブラリが用意されている Apache Jena を用いた．

7. 例題への適用による評価

共著の文書や画像アルバムなど複数のデータを交換する例題に適用して，提案した二層 LDA を評価する．次の二つの方法で評価する．

(1)　アーキテクチャの振舞いの比較：二層 LDA と Publish/Subscribe Architecture の比較

(2)　プロトタイプによるスケーラビリティの評価

7.1 アーキテクチャの比較による評価

二層 LDA と Publish/Subscribe Architecture を比較することでスケーラビリティを評価する(図 5，表 3).

(1)　振舞いの比較

Publish/Subscribe Architecture ではブローカを介してデータ交換を行う．そのため，データ交換がブローカに集中する．しかし，本研究が提案する二層 LDA ではブローカを介さずデータ保持者と直接データ交換する．そのため，データ交換が各サービスに分散する．

(2)　データ配置の差異

Publish/Subscribe Architecture では連携する全てのデータを一度ブローカに保持する必要がある．そのため，データの配置がブローカに集中する．しかし，二層 LDA ではブローカがデータを保持する必要はない．また，Linked Data を配信するサービスは必ずしもデータを保持する必要はなく，配置の異なるソースリポジトリ指す Linked Data を配信することが可能である．そのため，二層 LDA はデータの配置を分散させることが可能である．

以上から，二層 LDA はデータ交換の分散とデータ配置の分散を可能にし，全体として負荷が非集中の連携を実現する．よって，二層 LDA は他のアーキテクチャよりスケーラビリティが高いと評価できる．

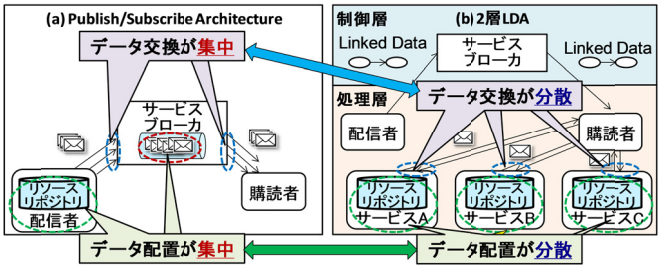

図 5　各アーキテクチャの振舞いの比較

Figure 5　Comparison of the behavior

表 3　各アーキテクチャにおける負荷の比較

Table 3　Comparison of the architecture

アーキテクチャ	振舞い	データ配置
(a)　Publish/Subscribe	ブローカに集中	ブローカに集中
(b)　二層 LDA	サービスに分散	サービスに分散

(3)　連携参加者の結合度の評価

不特定多数の連携アーキテクチャを実現するためには連携参加者の状態によらず連携が可能である必要がある．連携参加者がエンドポイントを変更しても連携を可能としなければならない．二層 LDA は制御層を Publish/Subscribe Architecture に基づくことで，連携参加者の時間分離，空間分離を実現した．処理層はリソース指向に基づいてデータを取得するため，状態分離を可能とする．二層 LDA は時間分離，空間分離，状態分離を可能とし，疎結合な連携を実現した．そのため，結合度は低い．

7.2 プロトタイプによるスケーラビリティの評価

プロトタイプにユースケースを適用し，提案した二層 LDA のスケーラビリティを評価する．

7.2.1 適用ユースケース

(1)　ユースケース

二層 LDA を定期購読ユースケース(図 6)に適用する．アクタは執筆者，編集者，e コマース，購読者の四つである．編集者は Linked Data を用いて三人の執筆者の情報や原稿のデータを管理しているとする．執筆者は画像や PDF ファ

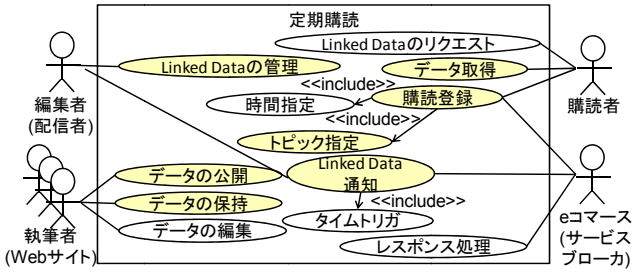

図 6　定期購読ユースケース

Figure 6　Subscription Use Case

イルなどサイズの大きいデータを保持し，それらを Web サイトで公開しているとする．ユースケースの中で実装した機能は Linked Data の管理，データ取得，購読登録，トピック指定, Linked Data 通知, データ公開, データ保持である．

今回はデータサイズに対するスケーラビリティの評価で用いる機能を優先的に開発した．

(2)　ユースケースシナリオ

編集者は e コマースに異なるエンドポイントへのリンクを複数持つ Linked Data をトピック指定で配信する. そして, e コマースは配信された Linked Data のトピックに興味がある購読者に Linked Data を配信する．そして，購読者は Linked Data の中で必要なデータを取得する．このユースケースシナリオを表 4 に示す．

表 4　定期購読のユースケースシナリオ

Table 4　Use Case Scenario of the subscription

ユースケース名: 定期購読ユースケース
アクタ: 執筆者，編集者，e コマース，購読者
事前条件: 編集者は Linked Data で執筆者のデータを管理 共著の電子書籍が完成
事後条件: 購読者がデータを保存
基本フロー 1.　購読者が好きなトピックを購読登録 2.　編集者は完成した書籍の Linked Data を e コマースに通知 3.　e コマースは Linked Data のトピックを購読登録している購読者に Linked Data を通知 4.　購読者は Linked Data から取得したデータの URI を抜き出す 5.　購読者は執筆者からデータを取得 6.　3 から 6 を繰り返す

7.2.2 例題の Linked Data の構造

例題の Linked Data の構造を図 7 に示す. ユースケースに基づき，三人の執筆者が共著の本を執筆することを例に Linked Data を定義した [1]．構造は Linked Data Platform[11]に基づいている．この Linked Data は http://JavaBook がコンテナであり，その中に section1.pdf，section2.pdf，section3.pdf が含まれていることを表現している．ユースケースでは各 pdf ファイルが異なるサーバに配置されていることになっているが，今回は連携時間を正確に計測するため，全てのファイルを同一のサーバ上に配置した．

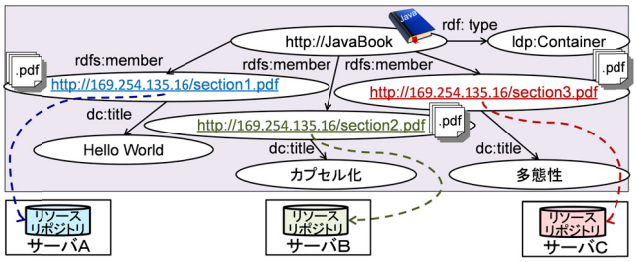

図 7　例題の Linked Data の構造

Figure 7　Structure of the Linked Data

7.2.3 プロトタイプの振舞い

プロトタイプの振舞いを図 8 に示す．

編集者は e コマースが公開するメソッドを用いて，Topic と編集者のエンドポイントを Subscription 型に格納し，購読登録サービスに送信する．その後，Apache Jena のライブラリを用いて図 7 の Linked Data を作成する．Linked Data は Apache Jena が提供するモデル型として作成される．それを String に変換し，e コマースに配信する．

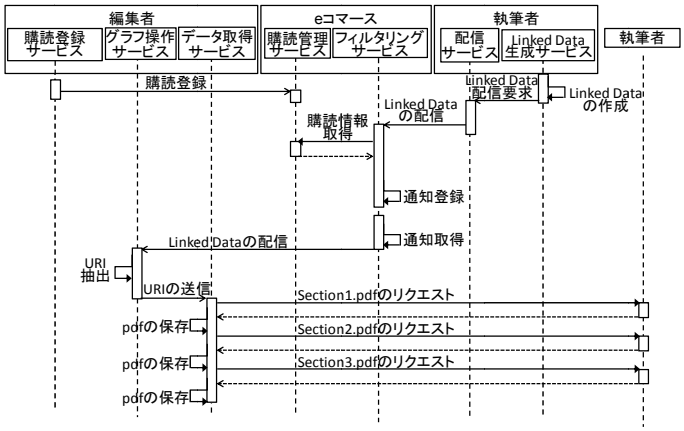

図 8　二層 LDA プロトタイプの振舞い

Figure 8　Behavior of the prototype

表 5　二層 LDA プロトタイプの連携時間

Table 5　Integration time of the prototype

データサイズ(KB)	1	25	50	100	200	300	500
連携時間(ms)	74.61	112.68	183.6	385.88	424.0	576.75	1094.09
標準偏差	8.93	11.26	8.74	19.77	86.21	76.23	140.52

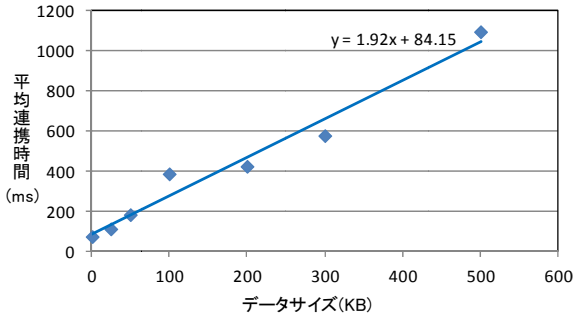

図 9　二層 LDA プロトタイプの連携時間の推移

Figure 9　Integration time of the proposal

編集者は Apache Jena が提供する SPARQL のライブラリを用いて，URI を抽出する．そして，その URI から pdf ファイルをダウンロードする．定義した Liked Data では三つのファイルへのリンクが含まれているため，ファイルのダウンロードは三回実行される．

7.2.4 連携時間の計測

データサイズを 1KB，25KB，50KB，100KB，200KB，300KB，500KB と変更し，連携時間を計測した．Linked Data を 3,000ms 間隔で 100 回送信して全体の連携時間を計測した．計測結果を表 5 と図 9 に示す．全体の連携時間とは配信者が Linked Data を配信してから，購読者が全てのデータを保存するまでの時間である．

7.2.5 Publish/Subscribe Architecture プロトタイプの評価

二層 LDA のスケーラビリティの有効性を示すために Publish/Subscribe Architecture の連携時間を計測する．Publish/Subscribe Architecture は Apache CXF で公開されているサンプルを利用する．サンプルでは String で文字列を交換するプログラムとなっているが，ユースケースに基づき三つのファイルデータを送信できるように拡張した．Publish/Subscribe Architecture 全体の振舞いを図 10 に示す．

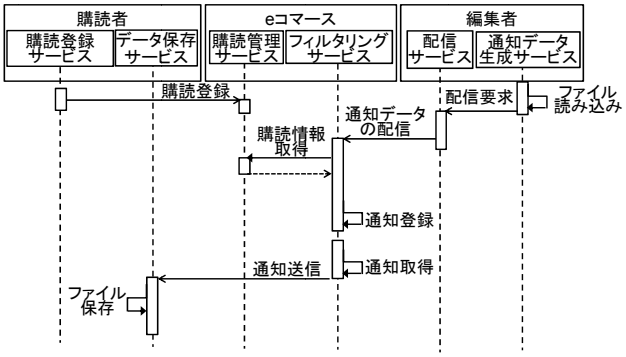

図 10　Publish/Subscribe Architecture プロトタイプの振舞い
Figure 10　Behavior of Publish/Subscribe Architecture

表 6　Publish/Subscribe Architecture の連携時間の推移
Table 6　Integration time of Publish/Subscribe Architecture

データサイズ(KB)	1	25	50	100	200	300	500
連携時間(ms)	55.96	361.46	834.14	1699.8	3908.54	4630.39	8698.52
標準偏差	10.99	73.61	95.89	158.95	434.31	287.56	856.66

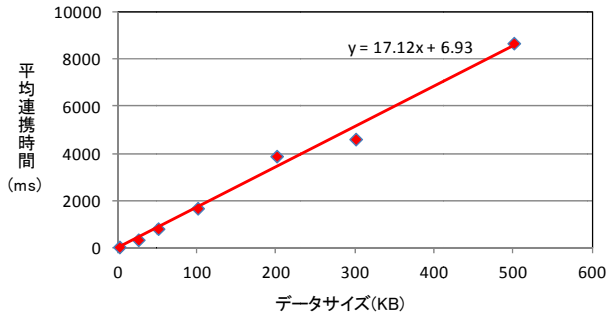

図 11　Publish/Subscribe Architecture の連携時間の推移
Figure 11　Integration time of Publish/Subscribe Architecture

配信者が三つのファイルをバイトコードとして読み込み，それを Base64 にエンコードし，String 型で送信する．購読者は受け取ったデータをバイトコードにデコードし，ファイルを保存する．

データサイズを 1KB，25KB，50KB，100KB，200KB，300KB，500KB と変更し，Publish/Subscribe Architecture の連携時間を計測した．三つのファイルを一つの String 型として 1KB, 20KB, 100KB, 200KB のときは 3,000ms 間隔で，300KB と 500KB のときは 8,000ms 間隔で 100 回送信し，全体の連携時間を計測した．計測結果を表 6 と図 11 に示す．全体の連携時間とは配信者がデータを配信してから，購読者が全てのデータを保存するまでの時間である．

7.3　二層 LDA と Publish/Subscribe Architecture の比較

二層 LDA と Publish/Subscribe Architecture の連携時間を比較することでスケーラビリティを評価する．

データサイズに対するスケーラビリティをアーキテクチャの連携時間の静的比較，連携時間の動的比較に分けて評価する．

(a)　連携時間の静的比較

二層 LDA の Linked Data 配信時間 T1 (ms)，データ取得時間 T0+T2 (ms)，Publish/Subscribe Architecture のデータ配信時間 T2 (ms)と定義する(図 12)．二層 LDA の全体の連携時間は T1*2+T0+T2 となる．Publish/Subscribe Architecture の全体の連携時間は T2*2 となる．これらから T2*2>T1*2+T2+T0 より，T2/2>T1 が求まる．そのため，データサイズが大きい連携では二層 LDA の連携時間の方が小さい．

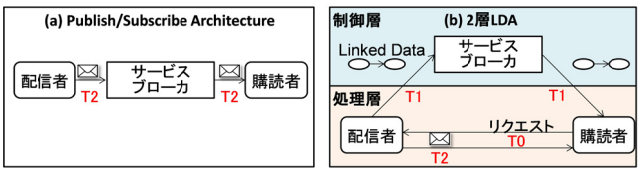

図 12　連携時間の比較
Figure 12　Comparison of the integration time

(b)　連携時間の動的比較

二層 LDA のプロトタイプと Publish/Subscribe Architecture プロトタイプの計測結果を用いて，静的な評価が正しいことを確認し，二層 LDA の有効性を示す．データサイズを 1KB，25KB，50KB，100KB，200KB，300KB，500KB と変更し，連携時間を計測する．Linked Data を 3,000ms 間隔で 100 回送信して平均の連携時間とその近似式を求めた(図 13)．

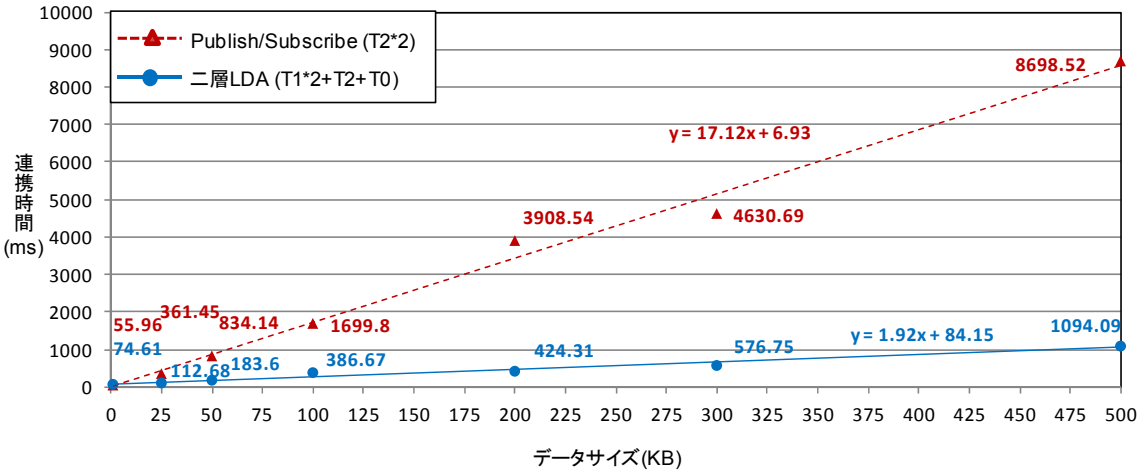

図 13　二層 LDA と Publish/Subscribe Architecture のスケーラビリティ比較
Figure 13　Comparison of scalability of prototypes

近似式の交点が(x, y) = (5.08, 93.90)であるため，データサイズが 5.08KB 未満では二層 LDA の方が Publish/Subscribe Architecture より連携時間が大きいが，5.08KB 以上では二層 LDA の方が Publish/Subscribe Architecture より連携時間が小さい．二層 LDA と Publish/Subscribe Architecture の傾きは 9.0 倍異なるため, サイズが 1 00KB のデータでは二層 LDA の連携時間が Publish/Subscribe Architecture の約 1/4 倍となる．

以上の結果より，提案した二層 LDA は Publish/Subscribe Architecture に比べ，スケーラビリティが高いといえる．

8. 今後の課題

8.1 複数の連携参加者用いた負荷の計測

多数の実機を用いて，サービス，ブローカにかかる負荷を計測することが必要である．

8.2 Linked Data の信頼性管理

Linked Data が指すデータがリポジトリに存在していない場合や Linked Data の指し先が異なっていることが考えられるため，Linked Data の信頼性を管理する必要がある．

9. まとめ

本研究は連携を動的に変更させるため，連携をメタレベルとベースレベルの二層に分け，Linked Data を用いたサービス連携アーキテクチャを提案した．

メタレベルでは Publish/Subscribe Architecture に基づくことで，連携参加者の時間分離と空間分離を実現し，ベースレベルでは Linked Data を用いて振舞いを制御することで異なるリソースリポジトリからデータを取得できる．それにより，連携の負荷が非集中なアーキテクチャとなり，スケーラブルなデータ連携を実現した．また，そのデータ取得はリソース指向に基づいているため，連携相手の状態によらず連携が可能であり，全体として結合度の低い連携を実現した．

提案したアーキテクチャのプロトタイプを開発し，ユースケースに適用することで連携のスケーラビリティを評価し，アーキテクチャの有効性を示した．

参考文献

1) D. Allemang and J. Hendler, Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL, Morgan Kaufmann, 2008 [荻野 達也(監訳), 実践セマンティック Web, ジャストシステム, 2010].
2) Apache, A SPARQL Processor for Jena, http://jena.apache.org/documentation/query/.
3) Apache, CXF User's Guide, http://cxf.apache.org/docs/index.html.
4) C. Bizer, T. Heath and T. Berners-Lee, Linked Data - The Story So Far, Int'l J. on Semantic Web & Information Systems, Vol. 5, No. 3, Jul. 2009, pp.1-22.
5) R. Buyya, J. Broberg and A. M. Goscinski, Cloud Computing: Principles and Paradigms, Wiley, 2009.
6) D. A. Chappell, Enterprise Service Bus, O'Reilly, 2004.
7) F J. Dowling and V. Cahill, The K-Component Architecture Meta-Model for Self-Adaptive Software, Proc. Reflection 2001 LNCS Vol. 2192, Springer, Sep. 2001, pp. 81-88.
8) P. T. Eugster, et al., The Many Faces of Publish/Subscribe, ACM Computing Survey, Jun. 2003, pp. 114-131.
9) R. T. Fielding and R. N. Taylor, Principled Design of the Modern Web Architecture, ACM Trans. on Internet Technology, Vol. 2, No. 2, May 2002, pp. 115–150.
10) M. Graube, et al., Linked Data as Integrating Technology for Industrial Data, Proc. NiBS 2011, Sep. 2011, pp. 162-167.
11) Linked Data Platform Working Group, http://www.w3.org/2012/ldp/wiki/Main_Page.
12) OASIS, WS-Brokered Notification, Ver. 1.3, Oct. 2006, http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf.
13) M. P. Papazoglou, Web Services: Principles and Technology, Prentice Hall, 2007.
14) L. Richardson, and S. Ruby, RESTful Web Services, O'Reilly, 2007.

ⓒ 2013 Information Processing Society of Japan