Vol.2009-OS-112 No.10
2009/8/6

ネットワーク分断に対応した分散オブジェクトライブラリの提案

小長谷 秋雄[†1]　品 川 高 廣[†1]　加 藤 和 彦[†1]

近年，インターネットサービスの可用性に対する要求が高まっている．可用性を増加させるためには，サービスを分散して提供するという手法が有用だが，データの一貫性制御などの問題についてサービスの内部構造に応じた対応を取る必要があるために，開発コストが増加してしまうという問題がある．本論文では，分散型のインターネットサービスを構築する際に有用な分散オブジェクトライブラリを提案する．本ライブラリでは，汎用的に利用可能なデータ構造と一貫性制御に関する処理を提供しており，サービス開発者は提供したいサービスに応じて適切なデータ構造を利用することで，分散型のサービスを容易に構築することができる．本稿では，提供すべきデータ構造と各データ構造に必要となる一貫性制御について分析を行い，設計を行った．

Suggestions of Distributed Object Library for Network-partition

Akio Konagaya ,[†1] Takahiro Shinagawa [†1]
and Kazuhiko Kato[†1]

Recently, the demand for the availability of internet services is increasing. In order to improve the availability, the technique for distributing service is useful. However, there is a problem that the cost of development may increase due to that it is necessary to concern the consistency of data individually. In this paper, we propose a distributed object library to support constructions of distributed internet services. This library presents an integrity control of each data structure so that the developer can construct distributed internet services easily by using an appropriate data structure. We have designed the data structure and its needed integrity control of each data structure.

1. はじめに

近年，様々なインターネットサービスが提供されており，個々のサービスに深く依存している場合も多いことから，サービスの可用性に対する要求が増大しつつある．しかし，インターネットの構造上，専用線を用いた場合に比べて比較的信頼性の低い通信路が用いられることが多いために，ネットワークの分断という障害が起こる可能性がある[8]．この場合，特にサービスを単一のホストあるいはそれに類する構成で提供している場合，継続してサービスを提供することができなくなり，可用性の低下につながる．

サービスの可用性を向上させるための手法の一つに，複数のホストを地理的に離れた場所に配置し，各ホストの内部状態を同期させることで，ホスト全体で一貫したサービスを提供するという手法がある[2),3),7)]．この手法の場合，どのような一貫性制御手法を用いてホスト間の内部状態を同期させるかを決定する必要がある．しかし，ホスト間が地理的に距離が大きく離れているような広域ネットワークを想定すると，従来の分散ロックなどを用いた一貫性制御は，通信の遅延による著しい情報伝達の遅れやネットワーク分断によって情報伝達自体が不可能になった場合，ホスト間の同期をとることは難しくなる．これは，Brewer ら[1)]によって CAP 定理という形で提案されている問題の一つで，一貫性（Consistency），可用性（Availability）及びネットワーク分断への対応（network Partition）のうち，同時に満たすことができる性質は二つまでであるとしている，したがって，分散ロックなどの強い一貫性制御を適用する場合，ネットワーク分断と可用性の要求を同時に達成することは困難であるということは CAP 定理からも知られている事実である．

従来の強い一貫性制御を用いる場合と比較して，通信の遅延やネットワーク分断といった問題を克服する為に，一貫性に対する制約を緩めた同期手法が提案されている[2),3),5),7)]．この手法は，Eventually Consistency と呼ばれる概念に基づいており，ホスト間が常に一貫性のとれた情報を保持することはできないが，十分な時間が経てば最終的にすべての情報が同期されるため，可用性を高く維持したまま分散型のシステムを構築することができる．ただし，この同期手法はサービスの内部構造に深く依存している場合が多く，個々のサービスに対して個別に一貫性制御を設定しなければならないことから，サービス開発の高速化が求まれている現在では，開発コストの増加などの理由から採用することが困難な状況にある．

†1 筑波大学大学院 システム情報工学研究科
Graduate School of Systems and Infomation Engineering, University of Tsukuba

本研究では，分散型のサービス開発に有用な分散オブジェクトライブラリを提案する．このライブラリでは，分散型システムの構築支援とネットワーク分断時の可用性維持を目的として設計されている．まず，分散型システムの構築を支援する為に，様々なサービスにおいて共通に用いることができる汎用的なデータ構造をライブラリにて提供する．このデータ構造には個別に必要となる一貫性制御が定義されており，サービス開発者はサービスの構造に適合したデータ構造を選択し利用するだけで，独自に適切な一貫性制御を実装することなく分散型のシステムを構築することができる．また，ネットワーク分断時にも可用性を維持する為に，データ構造に定義している同期手法には緩い一貫性制御を採用している．これにより，ネットワーク分断によって情報伝達が一時的に不可能になった場合にもいずれすべての情報の一貫性が保たれることが可能となり，かつ可用性の低下を最小限に抑えることができる．

本稿では，まず第 2 章にて既存研究と本研究との違いについて述べる．続いて，第 3 章にて本研究で提案する分散オブジェクトライブラリの設計について説明する．その後，第 4 章で実際のインターネットサービスに対して本研究で提案するライブラリを具体的にどのように用いればよいのかについて考察する．最後に，本研究全体のまとめを第 5 章に示す．

2. 関連研究

本研究では，インターネットサービスの可用性を維持する為に分散型のシステムにすることを提案しているが，この手法は他の研究でも現在までに広く提案されている．しかし，広域ネットワーク上でインターネットサービスを提供するという状況の基で，可用性の維持とサービス開発者側の負担軽減を目的とするという点において他の研究とは異なる．

Petersen らは Bayou[5] というシステムにおいて，緩い一貫性制御を組み込んだ分散型のシステムを提案している．Bayou では，一貫性制御を SQL に近い文法で記述することでき非常に汎用性の高い開発が可能となるが，個々のシステムに対して適切な一貫性制御を逐次組み込まなければならない為サービス開発者側の負担が大きい．

Gribble ら[4] は，ハッシュ構造を用いた汎用的かつスケーラブルな分散オブジェクトを設計，実装している．しかし，このシステムではクラスタ環境での運用を想定しており，クラスタ内部のネットワークは信頼性が高いことを仮定しているため，各ホスト間のネットワーク分断などの障害には対応できない．

Gao らの研究[2],[3] では，インターネットサービスの可用性を高めるために緩い一貫性制御を組み込んだ分散オブジェクトを提案している．この研究では，広域ネットワーク環境を想定しているためネットワーク分断などの障害にも対応可能だが，書籍販売サイトに特化した設計になっている為，汎用的なサービスへの適用は難しい．

Steen らは Globe[7] というシステムにおいて，広域のネットワーク上でインターネットサービスを提供するために有用な一貫性制御などの機能を組み込んだ汎用的な分散オブジェクトを提案しており，本研究で提案するライブラリを用いたシステムの構想にかなり近い．ただし，目的とするシステムに対してどのオブジェクトが適切であるかは，サービス開発者側で選択しなければならない．本研究ではサービス開発者の選択を支援するために，インターネットサービスに用いられるデータ構造やデータ構造に対する操作をその性質によっていくつかの種類にあらかじめ分類することで，サービス開発者はどのデータ構造を使用するべきなのかをその性質に沿って適切かつ容易に判別できる．

3. 設　　計

3.1 概　　要

本システムで提案するライブラリを用いることで構築可能な分散型システムの全体像を**図 1** に示す． 本システムでは，クライアントからの操作を受け付けるサービス提供レイヤー，分散オブジェクトライブラリ，ストレージによって構成されている．これらはホスト内に格納されており，地理的に離れた場所に複数台分散させて設置する．

サービス提供レイヤーは，Web サービスであれば HTTP サーバとアプリケーションサーバなどを用いて構成することで，クライアントからのリクエストを処理する機能を担っている．サービス提供レイヤーでは，リクエストを処理する為に分散オブジェクトに格納された情報を用いて必要となる情報をクライアントに提供する．分散オブジェクト自体は，サービス提供レイヤー側からは Map などの一般的なコレクション，あるいは Integer などの単一の数値として扱うことができるため，分散オブジェクト内部でどのような処理を行っているかを知る必要はない．

分散オブジェクトライブラリは，分散オブジェクトと分散オブジェクトを管理する為の管理マネージャによって構成されている．分散オブジェクトは，サービス提供レイヤー及び管理マネージャから非同期にアクセスされ，その内部状態を変更させる．また，管理マネージャは，ストレージへの永続化，他ホストとの更新情報伝達機能を備えており，これらは各オブジェクトのセマンティクスによって設定されたポリシーに沿って実行される．この分散オブジェクトは，サービス開発者が提供したいシステムに適合したオブジェクトを選択できるように，オブジェクトの表すデータ構造の種類に応じて分類されている．また，データ構

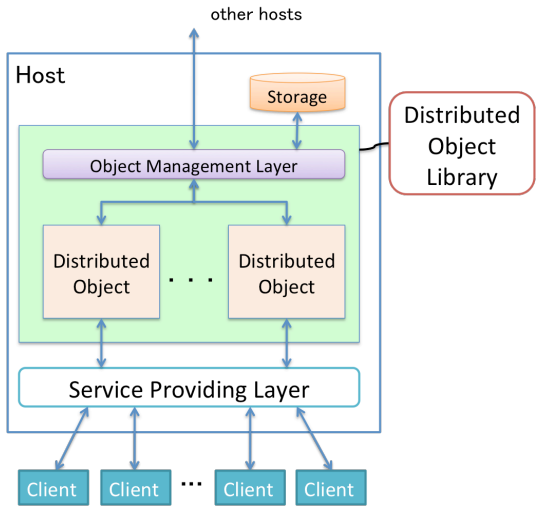

図 1 システム構成
Fig. 1 SystemArchitecture

造に適した一貫性制御をサービス開発者側で実装するコストを削減するため，あらかじめオブジェクト内部に一貫性制御の為の処理とそれを司るポリシーが定義されており，サービス開発者は一貫性に関するポリシーを設定するだけでホスト間のオブジェクトの同期処理が可能となる．この分散オブジェクトの詳細な構造については，第 3.2 章にて説明する．

3.1.1 ネットワーク分断

本研究におけるネットワーク分断とは，サービスを提供するホスト間のネットワークが，何らかの通信経路上のトラブルによって著しく遅延が増大する，あるいは通信ができなくなる状態とする．クライアントとホスト間のネットワーク分断については対象としていないが，これは，本研究で対象としている問題の一つとして，ネットワーク分断時のホスト間の一貫性制御を挙げているためである．したがって，クライアントとホスト間のネットワーク分断に関しては，本研究とは別のアプローチが必要になると考えられる．また，ストレージ障害などのホスト障害についても，障害に移行する前にシステムのメモリ上のデータがすべて永続化されていれば，バックアップなどを用いて再度起動することで対応することは可能だが，基本的にはホストに障害が起こった場合については本研究において対象としていない．

3.2 分散オブジェクトライブラリ

本ライブラリは，大きく分けてオブジェクト自体のデータ構造部分と，他ホストとの更新情報伝達やストレージへの永続化などの作業を担うオブジェクト管理部分によって構成されている．本章では，本研究で提案する分散オブジェクトライブラリの設計について示すために，提案するライブラリの設計コンセプトを示した上で，実際にコンセプトに沿ってデータ構造部分とオブジェクト管理部分について説明を行う．

3.2.1 設計のコンセプト

分散型のシステムにおいて，可用性を保ちつつ強い一貫性を維持することが困難であることは第 1 章で述べた．本研究では可用性の維持が目的であるから，Eventually Consistency に基づいた一貫性を緩める手法を用いる．この Eventually Consistency に基づいた一貫性制御手法に関しては，第 2 章で紹介したように既存研究にていくつか提案されており，これらの研究で提案されている手法は更新の衝突を解消する為にシーケンサやタイムスタンプなどを用いて更新情報の順序付けを行い，最も優先度の高い更新を有効としている．しかしこの手法では，ユーザの操作のタイミングによっては行った操作が一瞬有効になり，しばらくの後に無効となるという事態が起こりえる．

本研究では，先のような事態を避ける，すなわち一貫性をある程度満たしながら可用性を最大限上げるために，データ構造のセマンティクスに注目した一貫性制御を用いる．データ構造のセマンティクスに注目することで，データ構造に対して満たすべき状態や想定される操作を規定することができるため，このような状態や操作のレベルを変えることで一貫性と可用性をサービスが満たすべき状態まで高めることができる．具体的には，データ構造の種類によって可用性への影響が小さい範囲で操作あるいは状態に制限を設けることで，更新の衝突が起こらない状況を作り出すことが可能となり，一貫性をある程度まで満たせるだけでなく，タイムスタンプだけでないデータ構造固有の順序付けを定義することもでき，より柔軟な一貫性制御が行える．さらに，データ構造が満たすべき状態や想定される操作はデータ構造固有のものであるので，一貫性制御をあらかじめ定義した状態で提供することができ，開発者側の負担を減らすことができる．

以上のコンセプトに基づいて，具体的な設計について次章より説明していく．

3.2.2 データ構造部分

本研究で提案するライブラリでは，抽象的なデータ構造に対して満たすべき状態や想定される操作を定義できる抽象部分と，このようなデータ構造を基にして実際にサービスを開発する際に有用に使用可能な具象部分に分かれている．

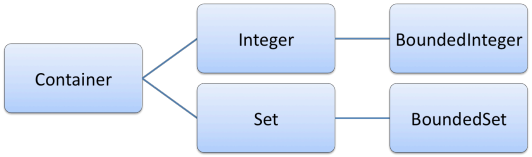

図 2 抽象クラス
Fig. 2 Abstract Classes

抽 象 部 分

本ライブラリにおける抽象部分のクラス構造を**図 2** に示す．抽象部分のクラスについては，直接開発者側が使用することはできず，あくまでデータ構造を分類する為の指標として定義している．したがって，実際にサービスに適用する際には，後で説明するこれら抽象的なクラスを実装した具象クラスを，サービスにうまく適合するものを選び出して使用することになる．

Container クラス — Container クラスは，すべてのクラスの基底となるクラスであり，すべての下位クラスで必要となる機能が備わっている．主な機能としては，すべてのホストの情報が同期されている最終の更新時刻を保持する機能がある．この機能によって，完全に同期が取れている情報と同期が確認できていない情報とを区別することができ，用途に応じて使い分けることができる．

Integer クラス — Integer クラスは，単一の数値を扱う為のクラスである．このクラスには，以下の 3 つのメソッドが使用可能である．

- add
- sub
- set

ここで，add は値のインクリメント，sub は値のデクリメント，set は指定した数値をセットすることができるメソッドである．Integer クラスでは，上限や下限が設けられていないため，満たすべき状態や操作に対して制限はかけられていない．分断中に行われた操作が衝突した場合の解消手段としては，以下の操作が定義できる．

- sum 操作
- max（min）操作
- タイムスタンプなどによる順序付け

add，sub メソッドのみ使用可能であれば，すべてのホストで行われた値の加減を足し合わせる sum 操作を使用することができる．set が使用可能であればすべてのホストでの現在値のうち最大（もしくは最小）のものを有効にする max（min）操作，および操作が実行されたタイムスタンプなどあらかじめ定義された順序付けによって最も優先度の高いものを有効とする操作が定義できる．これらの解消手段は，サービス内部での使用目的によって選択する必要がある．

BoundedInteger クラス — BoundedInteger クラスは，Integer クラスと同じく単一の数値を扱う為のクラスであるが，上限や下限が設定できるようになっている．そのため，add，sub，set の 3 つのメソッドの実行に制限がかかる．例えば上限が設けられている場合，Integer クラスのように無制限に add 操作を許可すると，ネットワーク分断中に上限を超えてしまう場合がある為である．

上限や下限などの状態の制限と add や set などの操作の制限との関係について，さらに詳しく述べる．まず，設定された状態の制限と逆方向の操作に関しては制限する必要はない．例えば下限が設定されている状態では add 操作は制限しない．しかし設定された状態の制限方向への操作および set 操作を行いたい場合，操作について何らかの制限を設ける必要がある．add，sub 操作について制限を設ける場合，その変化量について制限を設けることで，システム全体で設定された状態の制限を超えないようにすることができ，かつ行われた操作が無効になることはない．set 操作について制限を設ける場合は，add，sub 操作と違いある値が set されるか否かという二つの状態しかないために，max（min）操作やタイムスタンプによる順序付けを行うことでしか衝突を解消できず，行われた操作が無効になる可能性がある．

Set クラス — Set クラスは，複数のデータが集まったデータ集合を扱うためのクラスである．このクラスでは，以下の 3 つのメソッドが使用可能である．

- create
- delete
- modify

ここで，create は集合へ新たなデータの追加，delete は集合内の任意の要素の削除，modify は集合内のデータの変更を行うことができるメソッドである．Set クラスではいずれの制限も設けられていない為，自由にデータの追加，変更，削除を行える．ここで，本ライブラリの Set は通常の集合と違い，データの重複も許可されている特殊な集合としているため，論理的に等価なデータであっても追加することができる．分断中に行われた操作が衝突した場合の解消手段としては，以下のものがある．

- sum 操作
- タイムスタンプなどによる順序付け

これら解消手段は，想定される操作が create だけであれば，sum 操作によってすべてのホストで追加されたデータを足し合わせて一つの集合とすればよいが，delete，modify 操作が想定される場合は，タイムスタンプなどあらかじめ定義された順序付けによって最も優先度の高いものを有効にするという手段が考えられる．

BoundedSet クラス — BoundedSet クラスは，Set クラスと同じくデータの集合を扱う為のクラスであるが，データの重複がないという制限を設けた Set クラスになる．そのため，create 操作や modify 操作が制限される．

重複の禁止などの状態の制限と create や modify などの操作の制限との関係について，さらに詳しく述べる．重複が禁止された Set では，create や modify 操作によって等価なデータが集合内に複数存在しないようにしなければならない．Set クラスにおいては，タイムスタンプなどの順序付けに従って優先度の高いものを有効にするという方式しか定義できなかったが，BoundedSet クラスにおいては操作範囲を制限するという手法によって操作の衝突を回避することができる．例えば，ホストによって集合内の操作できるデータの範囲を制限することで，更新の衝突自体を回避することが可能となる．このように，操作範囲を制限することで更新の衝突を防ぐことができ，かつ行われた操作が後に無効になるという事態は起こりえない．

具象部分

本ライブラリにおける具象部分のクラス構造を**図 3** に示す．具象部分では，抽象部分のクラスを基にして，インターネットサービスにおいて有用であると考えられるデータ構造に特化してクラスを実装している．サービス開発者は，基本的にサービスに適合するように具象クラスを選択し使用するだけで分散型のシステムを構築することができる．具体的には，抽象部分において示された各クラスの構造に基づき，使用したい構造がどのクラスに属しているかを見極め，そのクラスを実装している具象クラスのうち適合しているクラスを用いればよい．

Counter クラス — Counter クラスは，BoundedInteger クラスを実装したクラスである．Counter クラスには，下限として 0 が，操作として add のみが定義されている．したがって，衝突の解消手法としては各ホストで行われた add 操作の値を合計するという操作が定義されている．実際に使用する際は，Web ページのカウンタなど，単調増加する数値が必要な場合に用いることができる．

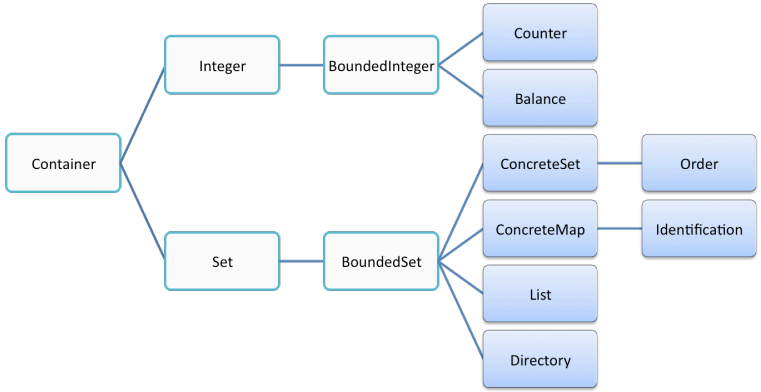

図 3 具象クラス（色付き部分）
Fig. 3 Concrete Classes (colored)

Balance クラス — Balance クラスは，BoundedInteger クラスを実装したクラスである．Balance クラスには，下限を任意に定義することができるため，sub 操作に制約が生じることになる．衝突の解消手法としては，各ホストで行われた sub 操作の値を合計して元の値から減じるという操作が定義されている．実際に使用する際は，商品の在庫数など値が増減するが 0 以下にはならないような場合に用いることができる．

ConcreteSet クラス — ConcreteSet クラスは，BoundedSet クラスを実装したクラスである．ConcreteSet クラスは，一般的な Set 構造と同じく，重複の許可されていない集合を表している．したがって，create および modify 操作に制限が生じる．実際に使用する際は，ConcreteSet クラスそのものを使用するというよりも，ConcreteSet クラスから派生したクラスの中で適合するものを選択した方が制限などの定義の手間を低減できる．

ConcreteMap クラス — ConcreteMap クラスは，BoundedSet クラスを実装したクラスである．ConcreteMap クラスは，一般的な Map 構造と同じく，キーと値がペアとなったデータが格納されている集合であり，キーの重複は許可されていないため，create 操作に制限が生じる．実際に使用する際は，ConcreteMap クラスそのものを使用するというよりも，ConcreteMap クラスから派生したクラスの中で適合するものを選択した方が制限などの定義の手間を低減できる．

Order クラス — Order クラスは，ConcreteSet クラスから派生したクラスである．Order

クラスでは，create 操作のみが定義されており，create 操作については重複の生じないデータの追加が可能になっている．ただし，データの追加には上限が設けられており，あらかじめ定義された順序付けに従って最も優先度の高いものを無効にする操作が定義されている．実際に使用する際は，商品の注文リストなど基本的にデータの追加のみで構成されるような場合に用いることができる．

Identification クラス — Identification クラスは，ConcreteMap クラスから派生したクラスである．Indentification クラスでは，create，delete，modify 操作が定義されており，create 操作に関しては設定されたキーが重複しないように制限がされている．さらに，ホスト毎に操作できるキーの範囲を制限することもでき，各操作が衝突しないような状況を作り出すこともできる．実際に使用する際は，ユーザ情報など重複しない何らかのキーがあるようなデータ構造に対して用いることができる．

List クラス — List クラスは，BoundedSet クラスを実装したクラスであり，データ間の関連性が強い集合を扱う場合に用いることができる．List クラスでは，create，delete，modify 操作の他に，insert 操作も定義されており，指定したデータとデータの間へ挿入することができる．そのため，操作が衝突する可能性があるが，操作できる List 中の範囲を制限することで衝突を回避することができる．

3.2.3 オブジェクト管理部分

オブジェクト管理部分では，主に「他ホストとの更新情報伝達」と「ストレージへの永続化」の二つの機能を備えている．これらの機能をいつ使用するかについては，オブジェクト内のポリシーによって決められており，定義されたポリシーに従って情報の送受信や永続化が行われる．

他ホストへ更新情報を送る為に，本システムではデータ構造に対して行われた操作をログとして保存している．このログに登録された各情報には，システム全体でユニークとなる ID と登録日時が定義されており，衝突回避操作を行う際に他の情報と区別することができる．このログを他ホストへ送信することによって，システム全体の同期を行っている．この更新情報を送信するタイミングは，分散オブジェクトに定義されたポリシーによって決められており，サービス開発者はこのポリシーを設定することで同期の頻度を調整することができる．

ストレージへの永続化は，通常データ構造へ何らかの操作が行われたときと，他ホストへ更新情報を送信するときに行われる．具体的な永続化手法としては，本研究ではオブジェクトのシリアライズを用いてファイルあるいはデータベースへ保存する手法を用いている．

内部情報	説明
ユーザ情報	ユーザの名前，ユーザ ID，パスワード，居住地域，出品者としての評価などの情報
商品情報	商品名，出品個数，現在価格，即落札価格，出品者，落札日時などの情報
入札情報	入札者，入札商品，入札価格，入札個数，入札日時などの情報
即落札情報	即落札者，即落札商品，即落札個数，即落札日時などの情報
コメント情報	出品者に対するコメントと出品者への評価に関する情報
カテゴリー情報	出品商品のジャンル情報
地域情報	ユーザの居住地域情報

表 1 RUBiS で取り扱う情報
Table 1 Informations used by RUBiS

4. サービス適用例

本章では，我々が提案するシステムを用いて実際にネットワーク分断にも対応した分散型のインターネットサービスが構築できるかどうかを検証する為に，実際のインターネットサービスへ本ライブラリを適用させることを考える．今回は，オープンソースのインターネットオークションである RUBiS というアプリケーションに対して，具体的に本ライブラリをどのように適用すればよいかについて考察する．

4.1 RUBiS[Rice University Bidding System]

RUBiS[6] とは，eBay というオークションサイトをより簡略化したものを基にして設計されてた，オークションサービスの性能評価を目的としたアプリケーションである．前述のように，簡略化されたシステム構成になっているが，一般的なオークションサービスで用いられている機能はほぼ搭載されており，オープンソースで公開されているために内部構造の変更が容易である，作成したシステムの性能評価を行いやすいなどの観点から，分析対象として採用した．

4.2 RUBiS への適用

RUBiS では，内部で扱う情報を**表 1** で示す 7 つに分類している． 今回は，表 1 の 7 つの情報をそれぞれオブジェクトとするために，今回ライブラリにて提供しているクラス構造のどの部分に属しているかを考察し，適用した．以下に，この 7 つの情報が今回ライブラリで提案するデータ構造のどのクラスに属しているか，情報の特徴を踏まえた考察を詳細に示す．

ユーザ情報 — ユーザ情報では，ユーザ ID を主キーとして区別しており，一度作成された情報をユーザ自身が変更することはできない．ただし，ユーザの出品者としての評価

が数値で記録されており，落札者からの評価によって出品者評価が変化する．

以上の特徴から，ConcreteMap を実装している Identification クラスを用いることにする．このクラスに許可する操作としては create 操作のみを定義し，変更される可能性のある出品者評価の数値は上限下限を設けない Balance クラスを用いることで対応する．同じユーザ ID で同時に登録しようとした場合の衝突回避手段としては，タイムスタンプによって登録された日時の早いものを有効にするか，あるいは登録できるユーザ ID の範囲をホスト毎に定義してしまうという制限が考えられる．

商品情報 — 商品情報では出品されている商品の情報が格納されており，一度登録された情報が出品者によって変更されることはない．ただし，入札や即落札によって現在価格や出品個数が変化する．

以上の特徴から，BoundedSet を実装している Order クラスを用いることにする．このクラスに許可する操作としては，create 操作のみを定義し，変更される可能性のある現在価格は Counter クラスを，出品個数については下限として 0 を定義した Balance クラスを用いることで対応する．現在価格で用いる Counter クラスでは set 操作のみを定義し，操作が衝突した場合には値の大きい方，すなわち金額の大きい方を有効とすることで衝突を解消する．出品個数で用いる Balance クラスでは sub 操作のみを定義し，0 より小さくなるような操作が行われた場合はタイムスタンプが遅い方の操作を無効とすることで衝突を回避する．Order クラスの場合，上限を設ける必要があるが，ある程度十分な量を定義しておき，上限を超えた場合は既に出品期限が過ぎている商品の中から古い情報を削除するようにすれば，削除操作の頻度を低下させることができる．

入札情報 — 入札情報では，出品されている商品に対して行われた入札の情報が格納されており，一度登録された情報は出品者や入札者が変更/削除を行うことはできない．また，入札金額については必ず以前の金額よりも大きくなければならないという制限がある．

以上の特徴から，BoundedSet を実装した ConcreteMap を用いることにする．このクラスに許可する操作としては，create 操作のみを定義し，重複のない追加を可能とするためシステム内部で定義されている固有の ID をキーとして用い．同時に入札された金額を商品情報の現在価格にセットする．このとき，以前の入札金額よりも大きくなければならないという制限を満たすため，入札金額によって順序付けを行い，入札日時が最近のものであっても入札金額が以前の金額よりも低いデータを無効とする処理を衝突回避処理として実装する．

即落札情報 — 即落札情報では，出品されている商品に対して行われた即落札操作の情報が格納されており，一度登録された情報は出品者や即落札者によって変更/削除を行うことができない．また即落札の個数は，出品されている個数を超えてはならないという制限がある．

以上の特徴から，BoundedSet を実装した ConcreteMap を用いることにする．このクラスに許可する操作としては，create 操作のみを定義し，重複のない追加を可能とするためシステム内部で定義されている固有の ID をキーとして用い，同時に即落札された個数の分だけ商品情報の出品個数から減ずる．このとき，減じた結果が 0 以下にならないという制限を満たすため，即落札処理が行われた時間によって順序付けを行い，先に行われた操作を有効とする処理を衝突回避処理として実装する．

コメント情報 — コメント情報では，落札者の出品者に対する取引の対応などに関するコメントとその評価が格納されており，一度登録された情報は落札者によって変更/削除を行うことができない．

以上の特徴から，BoundedSet を実装した ConcreteSet を用いることにする．このクラスに許可する操作としては，create 操作のみを定義し，同時に評価値をユーザ情報の評価値へ加える．この評価の加算操作は，ユーザ情報の評価に上限下限が設けられていないため，自由に行うことができる．

カテゴリー情報 — カテゴリー情報では，書籍関係や服飾関係など出品された商品のジャンルに関する情報が格納されており，出品者は出品する際にこのカテゴリーを選んで出品を行うことになる．ただしこのカテゴリー情報は，システム運用中に変更されることがないという特徴を持っている．

以上の特徴から，BoundedSet を実装した ConcreteMap を用いることにする，このデータ構造は他の構造と違い，データ構造に対する操作が想定されず，操作が衝突することはないため，このデータ構造に対する一切の操作を許可せず，衝突回避処理は実装しない．

地域情報 — 地域情報では，ユーザの居住地域に関する一覧が格納されており，ユーザ登録の際にユーザが住んでいる地域を選んで登録することになる．ただし地域情報は，システム運用中に変更されることがないという特徴を持っている．

以上の特徴から，BoundedSet を実装した ConcreteMap を用いることにする．このデータ構造はカテゴリー情報と同じく，データ構造に対する操作が想定されず，操作が衝突することはないため，このデータ構造に対する一切の操作を許可せず，衝突回避処

理は実装しない．

5. まとめと今後の予定

本論文では，ネットワーク分断にも対応する可用性の高い分散型のシステムを容易に構築することのできるライブラリを提案した．通常，可用性を高めるために緩い一貫性を採用する場合，サービス毎に内部構造に適した一貫性を逐次実装する必要があるため，開発者の負担が増加する．本論文では，データ構造のセマンティクスに注目して，それぞれのデータ構造に対して必要だと考えられる一貫性制御をあらかじめ組み込むことで，サービス開発者が提供したいサービス内部の構造に応じて適切なデータ構造を用いるだけで分散型のシステムを構築することが可能となり，一貫性制御を逐次実装するコストを軽減させることができる．

今後の予定としては，まず本論文で提案した設計を基に実装を行い，実際に RUBiS に適用させることでその評価を行う．その他，本論文で提案したデータ構造ではまだサービス開発者に対して一貫性制御を個別に実装してもらう必要がある部分があったため，本論文で提案したデータ構造の分類をさらに進め，汎用的に用いることのできるデータ構造と一貫性制御を洗い出し実装を進める．また，本論文では RUBiS というオークションサービスに対して提案したライブラリを適用したが，オークションのようなインターネットショッピングサービスだけでなく，その他の種類のサービスにも同様に適用できるかを実際に本ライブラリを組み込んで実験を行う必要がある．

参 考 文 献

1) Brewer, E.A.: Lessons from Giant-Scale Services, *IEEE Internet Computing*, Vol.5, pp.46–55 (2001).
2) Gao, L., Dahlin, M., Nayate, A., Zheng, J. and Iyengar, A.: Improving Availability and Performance with Application-Specific Data Replication, *IEEE Trans. Knowl. Data Eng.*, Vol.17, No.1, pp.106–120 (2005).
3) Gao, L., Dahlin, M., Nayate, A., Zheng, J. and Iyengar, A.: Application specific data replication for edge services, *WWW '03: Proceedings of the 12th international conference on World Wide Web*, New York, NY, USA, ACM, pp.449–460 (2003).
4) Gribble, S.D., Brewer, E.A., Hellerstein, J.M. and Culler, D.: Scalable, Distributed Data Structures for Internet Service Construction, *Proceedings of OSDI 2000.*, pp. 319–332 (2000).
5) Petersen, K., Spreitzer, M., Terry, D. and Theimer, M.: Bayou: replicated database services for world-wide applications, *EW 7: Proceedings of the 7th workshop on ACM SIGOPS European workshop*, New York, NY, USA, ACM, pp.275–280 (1996).
6) Team, R.: RUBiS, http://rubis.ow2.org/index.html.
7) van Steen, M., Homburg, P. and Tanenbaum, A. S.: Globe: A Wide-Area Distributed System., Technical report, Technical report (1999).
8) Zhang, Y., Paxson, V. and Shenker, S.: The Stationarity of Internet Path Properties: Routing, Loss, and Throughput, Technical report, In ACIRI Technical Report (2000).