広域および小規模分散コンポーネント間通信機構の評価

岩井 将行[1] 中澤 仁[1] 徳田 英幸[1 2]
[1] 慶應義塾大学大学院 政策・メディア研究科 [2] 慶應義塾大学 環境情報学部

近年の多くの分散システム開発の現場では，独立するソフトウエアコンポーネントを複数組み合わせて一つのシステムが構築されている．独立コンポーネントを統合させるには，コンポーネント間を接続する通信機構を的確に認知し，設置しなければならない．本稿は，CORBA, Jini, Mobile Agent, SOAP, JXTA, JECHO, SIENA などの分散協調ミドルウェアを『分散コンポーネント間イベント通信』という切り口から複数の項目に渡り分類・比較を行う．これにより，各ミドルウェアに最適なアプリケーションドメインを明らかにする．その上で近年注目を集める情報家電間を相互に協調動作させる場面に適した通信ミドルウェアを提案し，本稿で設計と実装を述べる．

Evaluation of Distributed Components Communication Framework in Wide Area and Local Area

Masayuki Iwai[1] Jin Nakazawa[1] Hideyuki Tokuda[1 2]
[1]Graduate School of Media and Governance, Keio University
[2]Faculty of Environmental Information, Keio University

Nowadays, many distributed systems are developed by joining independent software components distributed throughout the network. To integrated these components, it is necessary to recognize and establish an inter-component communication framework. In this research, we classify and compare major distributed middleware such as Jini, JXTA, CORBA, SOAP, Mobile Agent, SIENA, by communication methods, and topologies of the connection between components. This will help clarify the appropriate application domain of each middleware. Finally, we present the design and implementation of a new middleware suitable for collaboration between networked appliances.

1 はじめに

近年の多くの分散システム開発の現場では，複数の独立するソフトウエアコンポーネントを組み合わせて一つのシステムが構築されている．例えば，商品データベース，商品発注，決算処理，社内経理などそれぞれ単独でも動作をしているが，それらの部分部分を統合した新しいWEBサービスを導入させたい場合に，既存のソフトウエアコンポーネントをそのまま利用することでで低開発コストのインテグレーションを実現できるのである．こういった独立するソフトウエアコンポーネントを再利用した統合システム開発においては，コンポーネント間の通信機構が重要な役割を持つ．コンポーネント間の複雑なデータの交換と協調処理を行い，同時に信頼性と互換性を保持する必要があるからである．

本稿では，それらのコンポーネント間通信の主要な要素を抽出し，分散ネットワーキング技術あるいは分散協調支援技術と呼ばれる既存のミドルウェアの特徴を検討する．代表的なものとして、CORBA[9], Jini[12], JMS[11], ECJ[15], MobileAgent, SOAP[8], JXTA[13], JECHO[17], SIENA[1], MQ[3]をあげる．いずれも分散されるコンポーネントを有機的に結び付け意味ある協調動作システムを支援を目的とした，多くの分散システムで用いられている基盤ソフトウェアである．本稿では，上記の基盤ソフトウェアを，コンポーネント間イベント通信の観点でいくつかの分類を行い，これらミドルウェアに不足する技術やミドルウェアを利用するプログラマが留意しなければならない点を明らかにし，適切で信頼性のある分散システムを構築を支援する．

さらに各分散ミドルウェアのターゲットとするアプリケーションドメインを明らかにする．近年注目を集めている家庭内情報家電ネットワークというアプリケーションドメインに最適なミドルウェアが備えるべき機能を検証し，情報家電ネットワークに最適なミドルウェアを設計，実装する．

以下に本稿の構成を示す．第2章では，コンポーネント間通信の基本要素を明らかにする．第3章では，コンポーネント間通信機構で用いられるトポロジを検証する．第4章では，各ミドルウェアのアプリケーションドメインを明らかにし，第5章で，情報家電ネットワークに適したイベント配送モデルの設計を述べる第6章では，本稿のまとめを行う．

2 コンポーネント通信に関する基本要素

2.1 前提条件

一つのコンポーネントは，独立したアドレス（IPとポート番号および名前の組み合わせ）を持ち単体で動作することが可能であるソフトウェアあるいはソフトウェアとハードウェアが一体化されたも

のとして位置付ける．それらのコンポーネント間で通信を行う場合，何からのメッセージ，遠隔制御命令，オブジェクトのやり取りが行われる．本稿ではそれらを**イベント** (Event) という言葉で概念化し議論を進める．つまり，メッセージ通信により何らかをデータがやり取りされることや，遠隔メソッドが起動されることを含めて広義にイベント通知と定義する．イベント通知は，ほとんどの場合アプリケーション層における通信を指しているが，アプリケーション層以下の層においても同様の言葉で扱う．その分散コンポーネントは大まかに**イベント生産者** (Supplier, Producer, Publisher, Generator, Sender)，**イベント消費者** (Consumer, Sink, Interest Party, Subscriber, Listener，Receiver) に分けることが出きる．生産者とは，イベントを他者に送ることが可能な分散コンポーネントである．消費者とは，生産者からのイベントを受け取り，身らの内部でそのイベントに基づいた処理を行うコンポーネントである．この両方の性質を持つコンポーネントも多く存在する．この特別なコンポーネントは詳しく3章で後述する．

2.2 コミュニケーションの同期性とパターン

本節では，コンポーネント間通信における同期性と分散デザインパターンについて議論する．さて今二つのコンポーネントが通信を行う場合，大まかにpull型とpush型に分類できる．

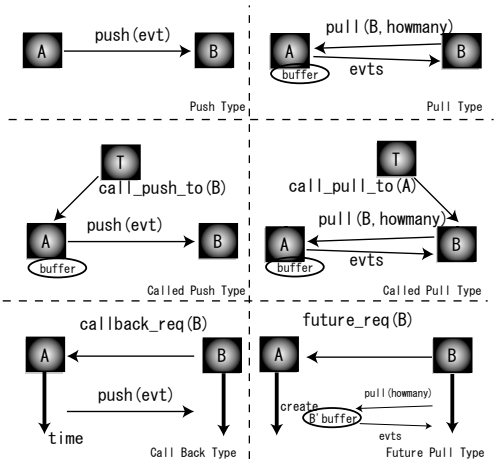

図 1: 6種類の同期性

図1の上に示すように，push型は非同期に (asynchronous) イベントの受けてにイベントを送りつけるものである．push型においてBは，マルティスレッディングをサポートしていなければ円滑に非同期通信を受け取ることができない．非同期イベントとスレッドとの関係は，[10]で詳しい考察が行われている．push型には，完全に相手がイベントの通知を受けたかを確認しないパターンおよび，確認を即座に返してもらう実装パターンがある．後者は，厳密には非同期とは呼べないが，マルティスレッドを利用した返り値なしの一部のRMIなどは，本稿では，意味的な非同期としてpush型に含める．

pull型は，情報を要求する者が，先にイベント提供者に，生産者内のメモリやバッファに存在するイベントの送付を同期的に (synchronous) 要求するものである．通常メソッドの返り値として，イベントが送られ，返答が終了するまでBは待機しているので，時間的な遅延は少ないほうが良い．通常の返り値があるRMI，HTTPのGET呼び出しなどがある．

さらにpush型とpull型を拡張させ，その発生条件 (trigger) を第3者 (third party) に委託する場合がある．例えば図1の左下はAがBに対してpushを行うタイミングをTが制御している．また右下は，BがAに対してイベントの要求を行うことタイミングをTが制御強いるパターンである．いずれもTとBが同一分散コンポーネントであれば，通常のpush型やpull型になる．中左の図はcalled push型，中右の図はcalled pull型と呼ぶ．called pull型は，第3者:TがAにBへの参照を渡す．Aはその参照を元にBへのpush型を通知する．called pull型は，TがBにAへの参照を渡す．Bはその参照を元にAに対してpullを実行する．これらの第3者が介するパターンは，Peer to Peerのコミュニケーションや，サービスディスカバリの用途に頻繁に用いられる．

時間軸の視点を考慮すると，将来的にpullメソッドを呼び出す，またpushされることを予約（promise）する方式がある．左下図は，はじめのイニシエーションの時点で，タイムアウトを通知し，その間だけ特別にpush型のイベントを配送の要求できる．これは，call backパターン呼ばれる．一度Bがpushの要請を行い，Aから連続して返答がある場合を特に，BからAへの購読要求（Subscribe）とAからBへの発行（Publish）と呼ぶ．Java AWT等で用いられるListenerパターンもcall back方式にあたる．call back方式でも，送信先の不在を検知して一時的に送信を見合わせるパターンは，Time Independent Invocationと呼ばれる拡張がある．下右図は，future pull型と呼ぶパターンで，まずBからAにB参照の登録を行う．そののちA側でB用のイベントバッファを用意し，Bからの将来的に呼び出されるpull型通信を待つ．こもパターンの利点は，Aが通知相手ごとにpullされる内容を用意できる点にある．さらに，Bの外部にポーリングを行う役割をもつポーラを実装しておき，Bからは外部ポーラに問い合わせることで，分散デッドロックの起こりにくいパターンにすることが可能である．しかし，この場合イベントの伝播速度としては平均的に遅くなってしまう．またメモリ空間が有限であることに注意して，実装しなければならない．

2.3 1対多，多対1イベント送受信方式

図1で示したすべての同期パターンでは，それぞれのイベントの生産者とイベントの消費者が各一つづつのコンポーネントであった．しかし，実際の分散システムの構築の際は，送信先あるいは送信元が

複数ある場合を想定しなければならない．例えば，push型では，Bが複数存在し，一度にすべてのBに対してイベントの送信を行う必要がある．これらの1対多，多対1イベント送受信方式は，図2に示す8種類が主にあげられる．

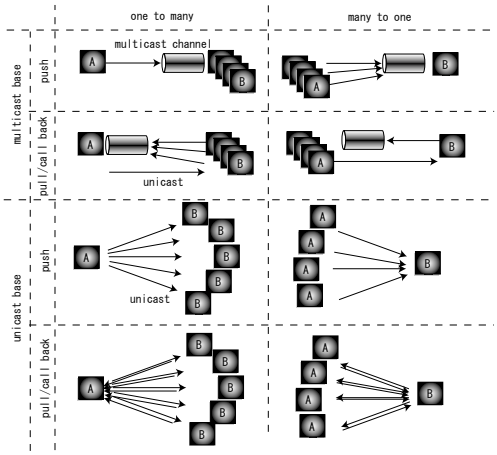

図 2: 1対多，多対1のイベント送受信方式

マルティキャストを用いた通知は，一度に複数の消費者へのイベント通知を可能にする．しかし，マルティキャストの届かない範囲にあるコンポーネントとの通信は行うことができない．またコンポーネントとチャネルの関係を決定する枠組みが必要になる．一方ユニキャストによるイベント通知は，ファイヤウォールやマルティキャストの有効範囲を意識しなくてよい反面，すべてのイベントを配送し終える間でに比較的時間がかかる．そのためスケーラビリティや配送順序を考慮しなければならない．

2.4 コミュニケーションポート

各コンポーネントには，イベントの送受信を行うモジュールがある．本稿では、それらのモジュールをポートと抽象化して呼ぶ．ポートは，文字通り，Socketのportをさしている場合もあれば，notify(Object obj)などのメソッドインタフェースとしての記述をさしている場合もある．図3にあるように入り口ポートが複数あるモデルと一つしかないモデルがある．入り口ポートが複数あるモデルでは，同じコンポーネント間を複数のポートでつなぎあわせるができる．しかしポートと内部的に呼び出すメソッドの関係を定義しておかなければならない．一方入り口ポートが一つしかないモデルでは，内部に持つフィルタによって，呼び出すメソッドを決定しければならない．フィルタは，ポートとメソッドのマッピングのほかに，セキュリティのチェックやビジネスロジックチェックを行うミドルウェアもある．出口ポートも同様に複数あるモデルと一つしかないモデルがあり，複数あるモデルでは，各出口ポートを通知先のコンポーネントの関係を定義しなければならない．一方で口ポートが一つしかないモデルでは，必然的にすべての登録されているコンポーネントに通知を行うことになる．

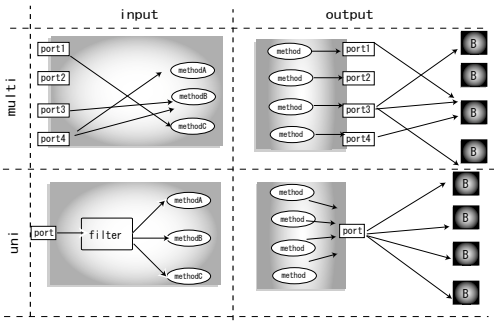

図 3: マルチポートとユニポート

3 トポロジ

分散コンポーネントシステムでのイベント通信の多くは，ワークフロウに基づいているため，複数のコンポーネントにわたってイベントが伝播する可能性もある．こういった分散イベントドリブンプログラミングをサポートする機構を整理するため，分散システムのトポロジを比較検討する．トポロジを考察した先行研究として，モビリティを中心に分散イベントモデルを分類検討したものがある[6]．本章では分散トポロジを仲介者有無で分類した方法をとる．

3.1 仲介者なしトポロジ

仲介者をおかず生産者と消費者間の通信しか行わないモデルである．Jiniでは通常，サービスとクライアント間はこのトポロジにより，イベントの通信を行う．それぞれのイベント生産者コンポーネントが誰に何のイベントを配送するのかを認識している必要がある．また，別の視点で見ると一つの生産者が複数の消費者を支配（dominate）しているモデルとも考えられるが，ともに仲介者のないトポロジである．仲介者がいない場合，[22]が示すようにいくつかのアダプタモデルを適応し，相手のコンポーネントを紹介あるいは推薦することができる．図4に紹介モデルと推薦モデルを示した．

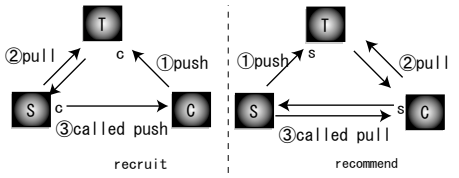

図 4: 二つのアダプタパターン

3.2 仲介者のあるトポロジ

図5の上に示すトポロジは，生産者と消費者間に仲介者を一つおきイベントの配送を制御させるモデルである．CORBA Event Service は，このモデルであり，push型だけでなくpull型を備えている．さらにCORBAのEventService間をIIOPを用いて通信を行わせることにより，図5の下の示す二つの仲介者を用いたトポロジを設定できる．仲介者はそれ

ぞれ役割が異なっている．二つの仲介者が存在するモデルでは，メッセージが一方通行に流れるものと双方向に流うことが可能なものがある．

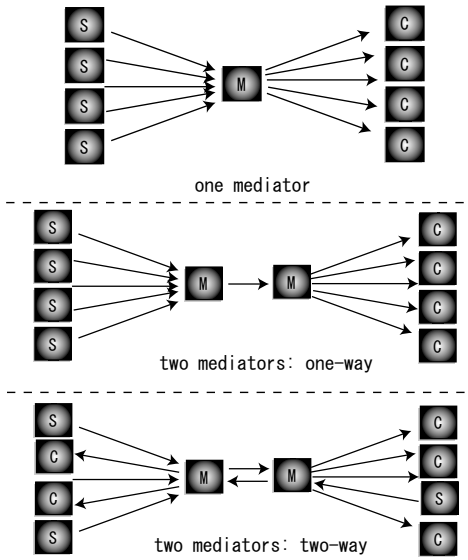

図 5: 単一及び二つの仲介者を持つトポロジ

3.3 複数の仲介者が存在するトポロジ

JXTA，SIENA などのいくつかのミドルウェアは，複数の分散する仲介者が存在するトポロジを構成させる．これは仲介者の役割を分割しシステムの構成に柔軟性を持たせている．この種類の研究をアプリケーションレベルルーティングと呼ぶ場合もある．

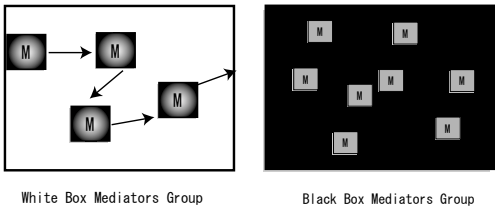

図 6: 仲介者が複数存在するトポロジ

3.3.1 明示的な仲介者間コミュニケーション

仲介者間の組み合わせをプログラマに明示的な（explicit）提示し，設定を外部委託させるトポロジである．経路設定者が設定を行い，このトポロジが生成されるまでは，イベントの伝達を行うことはできない．また経路設定者の経路決定を促すために，現在分散仲介者がどのように存在するのかを複数の設定者コンポーネントに提示する必要がある．提示の方法は，マルティキャストの 1 チャネルを共有する方法，一つのプロキシソフトウエアを共有する方法，すべての設定者コンポーネントがそれぞれ自分自身以外の仲介者コンポーネントの存在を認識している方法などがある．

3.3.2 暗黙的な仲介者間コミュニケーション

暗黙的な（implicit）なトポロジでは，明示的なトポロジよりもインテリジェンスをもち，入ってくるイベントの内容やネットワークの状況によって動的にトポロジ構成を変えることができることを実現しているモデルである．自分の近隣の，あるいはすべての仲介者を認識し，次に伝えるべき仲介者をその中から取捨選択しなければならない．このため独自の経路決定アルゴリズムを持つ．構造的なメッセージの内容，メタ情報，宛先アドレス，フラッディングに基づいた経路決定を行っている．イベントの到着以前にルーティングが確定しているモデルと到着時にルーティングが確定するモデルがある．

4 アプリケーションド メインの比較

各ミドルウェアのコンポーネント間通信機構の比較を図1に示した．ECJ は，小さい粒度にデータを非同期に配送するミドルウェアである．非常に高速な，メッセージ通信を可能にしており，ユニキャストによる 1 対多の push 通信が可能になっている．大規模システムでの中で，単純かつ大量な発生する一方向メッセジ伝達手段として有効である．RMI は，返り値を void にし，何らかの情報を通知先に非同期にコピーすることができる．また返り値として処理結果を待つ同期的な通信も行える．しかし，非常に低速な Unicast による 1 対多，多対 1 のユニキャスト通信をしなければならない．Jini は，RMI を基盤としたミドルウェアで，Lease による分散ガベージコレクション，クラスの型による Lookup などの機構を備えているミドルウェアである．Jini で採用されている，Java Distributed Event Model では，Lookup Service を中心とした recommend の call back パターンのイベント通知が可能である．SOAP は，UPnP[16] の基盤プロトコルとして採用された XML と HTTP を用いた通信機構である．SOAP は，HTTP の POST を用いてリクエスト行い，結果を受け取る．異なるシステム間や，企業間の大規模なシステムの通信には適しているが，複雑な参照関係のある分散コンポーネント間通信や，ワークフローに基づいたシステムへの応用は課題も多い．CORBA は，様々な通信技術が用意されている．CORBA Messaging, CORBA Event Service, CORBA Notification Service, また IIOP で ORB 間の通信やファイヤウオールも考慮されている．そのため {non,1,2}-mediators のトポロジを構築でき多彩な同期性をサポートしている．そのため様々なアプリケーションド メインで使用できる一方，非常にミドルウェアとして複雑なものになった．JMS，MQ などのトランスポートミドルウェアの比較は [14] で行われている．これらのミドルウェアは，トランザクション処理，ビジネスロジックの管理を考慮しいる．1 対多，多対 1 の通信機構で PriorityBase の順序配送を実装している．JECho は，マルティキャストのチャネルマネジメンを行い．White Box 型のトポロジで動作する．このため 1 対

多，多対1の通信機構が非常に効率的である．同一ネットワークドメインで多くのコンポーネント間で通信しなければならない場面で有効である．SIENAは，Black Boxのトポロジを組みpublish/subscribe型のメッセージ配送，ルーティング機構である．構造化されたメッセージをフラッディングすることなく適切に配送できる．インターネットスケールでの数値情報を扱うシステムなどに適している．Mobile Agentは，Agent自身をメッセージと考えると，メッセージ自身のメタ情報に，どこを循環する宛先がすでに記入されているモデルである．移送に時間がかかるのであまりpull型の通信には向かない．継続処理を伴うシステムで，ワークフローに基づいたシステムなどに有効である．JXTAは，Peer 2 Peer ネットワークとXMLを基盤とした分散ミドルウェアである．JXTA-pipeの機構を用いて複数の暗黙的に仲介者を経由して処理を行うモデルを適応できる．大規模で移動が頻繁に発生する不安定な環境化のシステム構築にも適している．

5 情報家電間協調分散ミドルウェア

近年情報家電機器が，ネットワーク通信インタフェースを持ち，遠隔制御可能になってきている．さらに，こういった機器にJaba VMが組み込まれていることが可能となった．またセンサも小型化し，低価格化しているため家庭内に偏在しておかれる可能性は高い．このような情報家電機器とセンサを協調動作させる目的とした情報家電間協調分散ミドルウェアを提案する．

5.1 Dragon

情報家電機器やセンサは，豊富なコンピューティング能力は，期待できない．我々は，その様な環境下で，有効なミドルウェアを開発している．**D***ynamic* **r***e-configura***a***ble event delivering framework with smart* **g***enerati***on** (**Dragon**)[20, 18, 4, 5]は，Jiniあるいは VNA[21, 19, 7]を基盤としイベント配送を実現するミドルウェアである．Dragonは，なるべく情報家電やセンサ側の処理を簡略化するため，複数の仲介者間ネットワークを持ちUnicast call backによる１対多の通信行う．Dragonは，図7に示すように3種類のコンポーネントからなる．Event Supplierでは，Sensor Moduleがイベント発生させ，ユニポート型のEvent Output Moduleから配信する．Event Mediatorは，Filter Moduleを持ちイベントを通過させるべきかどうかを決定し，イベントに対して加工を行う．Event Consumerは，Event Input Moduleからイベントを受け取った後，機器を動作させ実世界に対してAction Moduleから何らかの動作を行う．

仲介者ネットワークは小規模のネットワークを想定しているため，White Box型で自らルーティングや接続切り替え等は行わない．しかし，家庭内で利用するシステム構成は，頻繁に変化させることをユーザが望むことを考慮して，図8のような応用

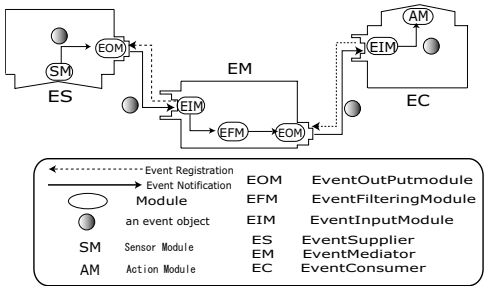

図 7: Dragonコンポーネント間の構成

ツールを用意している．このツールは，carp@[2]などのJiniビジュアルプログラミングツールとは，異なり常に複数の設定者からの変更命令を想定して実装されている．また，複数の仲介者に渡りイベントの伝播を設定し，そのイベント伝播にかかる実時間を表示できるため，より信頼性のあるシステムが構築できる．またDragonは，ユニキャストによる通知のため時間が通信に比較的長くかかる．複数の1対Nのイベント配送を行う場合には，優先度をつけている．図9では，EC_2 には最優先でイベントを配送し，他の消費者を後回しにできる．

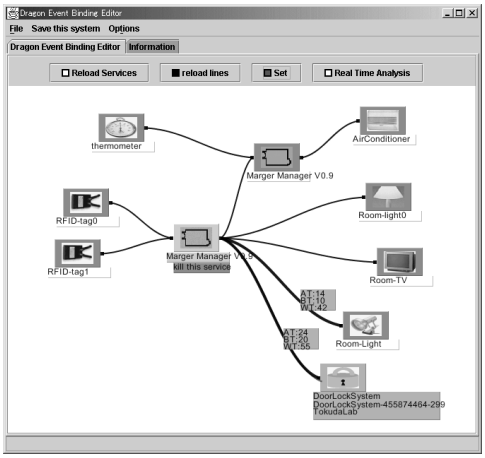

図 8: 明示的に仲介者間を結び合わせている画面

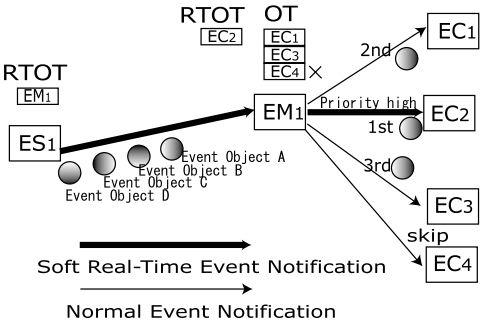

図 9: 時間的な制約を持たせたイベント配送

middleware	Topology	push	pull	called push	called pull	call back	future pull
ECJ	non-mediator	√					
RMI	non-mediator	√	√				
Jini	non-mediator	√	√			√recommend	
Dragon	N-mediators WhiteBox	√		√		√recruit	
SOAP	non-mediators		√				√
CORBA	{non,1,2}-mediator(s)	√	√			√	
MQ,JMS	N-mediators WhiteBox	√	√			√	
JECho	N-mediators WhiteBox	√	√				
JXTA	N-mediators BlackBox	√	×	√	×	√	×
SIENA	N-mediators BlackBox	√	×	√	×	√	×
Mobile Agent	N-mediators Gray Box	√	×	√	×	√	×

√：サポートされている, 空欄：実現可能, ×：ふさわしくない

表 1: 同期性サポート比較表

6 今後の課題とまとめ

本稿では，コンポーネント間を接続する通信機構を同期性，1対多通信，トポロジなどの側面で比較し，既存のミドルウェアについて比較した議論した．また，近年注目を浴びている情報家電ネットワークにおけるコンポーネント間イベント通信を行うミドルウェア：Dragonを提案した．Dragonを基盤としたアプリケーション例であるツールを用いることにより，ユーザが思い描く新しい役割を情報家電機器間に動的に割り振る事も可能になる．

今後は，本稿で議論した同期性，1対多通信，マルティポート，イベントコンテンツの構造化，Black-Box型の仲介者間通信などをサポートして行き，より多くのアプリケーションドメインでDragonが使用されることを目指す．また他のミドルウェアと連携も考慮して行く．

参考文献

[1] A. Carzaniga, D.S.Rosenblum, and A.L.Wolf. Achieving expressiveness and scalability in an internet-scale event notification service. In *Nineteenth ACM Symposium on Principles of Distributed Computing (PODC2000)*, 7 2000.

[2] Michael Fahrmair, Chris Salzmann, and Maurice Schoenmakers. Carp@ - Managing Dynamic Jini Systems. In *Middleware 2000*, April 2000.

[3] IBM. MQ Series Family. http://www-4.ibm.com/software/ts/mqseries/.

[4] Masayuki Iwai, Jin Nakazawa, and Hideyuki Tokuda. Dragon: Soft Real-Time Event Delivering Architecture for Networked Sensors and Appliances. In *The 7th International Conference on Real-Time Computing System and Applications(RTCSA2000)*, pp. 425–432, December 2000.

[5] Masayuki Iwai, Jin Nakazawa, and Hideyuki Tokuda. Flexible Distributed Event-Driven Programming Framework for Networked Appliances and Sensors. In *3rd International Symposium on Distributed Objects and Applications (DOA'01) Short Papers Proceedings*, pp. 61–68, September 2001.

[6] R. Meier. State of the art review of distributed event models. In *Technical report TCD-CS-00-16, Dept. of Computer Science, Trinity College Dublin, Ireland*, 2000.

[7] Jin Nakazawa, Tadashi Okoshi, Masahiro Mochizuki, Yoshito Tobe, and Hide Tokuda. Vna: An object model for virtual network appliances. In *IEEE International Conference on Consumer Electronics (ICCE2000)*, June 2000.

[8] W3C Note. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/SOAP/.

[9] Object Management Group. Common object request broker architecture and specification, 1996.

[10] R. Guerraoui P. Th. Eugster and J. Sventek. Distributed asynchronous collections: Abstractions for publish/subscribe interaction. In *14th European Conference on Object Oriented Programming (ECOOP 2000)*, June 2000.

[11] Sun Microsystems, Inc. Java Message Service Documentation Version 1.0.2b.

[12] Sun Microsystems Inc. *Jini Architecture Specification*. http://www.sun.com/jini/specs/.

[13] Sun Microsystems, Inc. JXTA v1.0 Protocols Specification. http://www.jxta.org/project/www/docs/ProtocolSpec.pdf.

[14] Stefan Tai and Isabelle Rouvellou. Strategies for integrating messaging and distributed object transactions. In *Middleware 2000 Proceedings*, 4 2000.

[15] 宮澤隆幸, 海邊裕. Javaによる非同期メッセージ配送フレームワーク ECJ. 情報処理学会 システムソフトウエアオペレーティングシステム. 情報処理学会, May 1999.

[16] Universal Plug and Play Forum. Universal Plug and Play (UPnP), 1999. http://www.upnp.org.

[17] Dong Zhou, Karsten Schwan, Greg Eisenhauer, and Yuan Chen. *JECho - Supporting Distributed Collaborative Applications with Java Event Channels*. College of Computing Georgia Institute of Technology, 2000.

[18] 岩井将行, 楠本晶彦, 中澤仁, 徳田英幸. 情報家電機器間の動的なイベントバインディング機構の構築. 情報処理学会 システムソフトウェアとオペレーティングシステム研究会, May 2000.

[19] 大越匡, 中澤仁, 田村陽介, 望月祐洋, 戸辺義人, 西尾信彦, 徳田英幸. VNA：仮想情報家電の実現へ向けて. 第59回情報処理学会全国大会, September 1999.

[20] 岩井将行, 大越匡, 中澤仁, 徳田英幸. 家電ネットワークにおけるサービスの動的提供機構. 全国大会. 情報処理学会, September 1999.

[21] 中澤仁, 大越匡, 望月祐洋, 徳田英幸. VNA構築用ライブラリの設計と実装. 全国大会. 情報処理学会, September 1999.

[22] 飯島正. 分散オブジェクトコンピューティング:第2章モデルとアーキテクチャpp31-pp81 ISBN4-320-02775-2. 共立出版, 東京, 1999.