XML - KQML ビジネスコミュニケーション

松村　英孝, 小西　修
高知大学理学部数理情報科学科
780-8520 高知県高知市曙町 2-5-1
{98ss077, konishi}@is.kochi-u.ac.jp

概要

ワークフローや電子商取引など実世界でのビジネスコミュニケーションをモデルとして, 複数のエージェントがコミュニ ケーションを行うことにより問題を処理する過程を表現する. 人間同士が言葉で会話をするように, エージェントも他のエージェントに「何をしたい」, 「何をしてほしい」のかが明確に分かるように, 通信手段として KQML というコミュニケーション言語を用いる. 拡張可能で, 厳密な構造を持つ XML により, ビジネスコミュニケーションで使用するさまざまな要素を表現する. それを KQML に組み込むことで, 再帰的表現といったより複雑な表現が可能になり, より表現豊かなコミュニケーションを行うことができる. KQML と XML を併せて使うことによって, 現実の EDI で使われるようなエージェントコミュニケーションに近づける.

キーワード：KQML，XML，ビジネスコミュニケーション，RELAX, 再帰的表現

XML - KQML Based Business Communications

Hidetaka MATSUMURA, Osamu KONISHI
Dept.of Information Science,Faculty of Science,Kochi University
2-5-1 Akebono-cho Kochi 780-8520 JAPAN
{98ss077, konishi}@is.kochi-u.ac.jp

Abstract

In this paper, we describe a cooperative agent model of business communications based on XML - KQML. We expressed required events by XML for the business communications. Furthermore, it has more precise structure by using RELAX for the schema declaration. The focus here is to enables a complex expression like recursive expression. We use XML in the content of KQML. The message can have recursive or iterated content is one in which a message can have content composed of another message. The message is sent through the facilitator for facilitating the coordination and interaction among the agents. The communications among the business is done with the facilitator. The benefit of this communication system is that it makes the communication infrastructure more flexible, easier to modify, easier to expand and more capable.

Key words：KQML, XML, Business Communications, RELAX, Recursive Expression

1 はじめに

従来商品を注文するといったやりとりは電話やFAXを用いて行われていたが, コンピュータの普及に伴い, そのようなやりとりをネットワークを介し, 電子データを用いて行うようになってきた. このようなネットワーク上で行う商取引をEC(Electronic Commerce：電子商取引) という. 特に図 1 のように, 企業間の EC を B2B(Business to Business), 企業対消費者の EC のことを B2C(Business to Consumer) と呼んでいる.

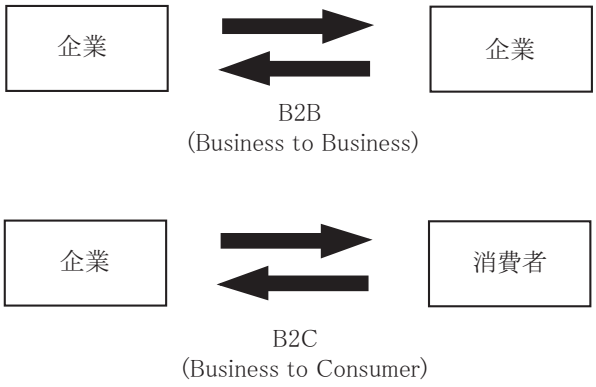

図 1: EC の形態

B2C はオンラインショッピングなどで日常生活でも身近なものになりつつあるが, B2B はそれまでオペレータによって行っていた処理を自動化することによりデータ精度の向上や費用削減が望める等の利点が明確であったため, 早いうちから行われていた. B2B の中で取引にまつわる規約を定め, それらに則って EC を行うことを EDI(Electronic Data Interchange：電子データ交換) と呼んでいる.

このような, 商取引において企業間など取引の当事者間で交換される情報は製品名やその仕様, 納期, 見積価格など多岐に渡る. 商取引を行うためには, これらの情報は当事者双方で相違のないよう厳密に定義され, ネットワークを通じて正しく交換されなければならない. しかし, それぞれの企業が独自のデータの形式を使用した場合には, 取引先毎に専用の端末を設置しなければならなかったり, 取引先毎に異なる形式のデータを自社システム用のデータ形式に変換しなければならない等の弊害が生じてしまう. これらの問題を解決するために広く合意され, 標準に基づいたデータ交換を行い, 正確に自分の意図を伝えなければならない.

関連研究として, Speech Act Theory を基盤としたコミュニケーション言語の FLBC[7] や, インターネット上でショッピングや銀行口座取引を行うための取引手順や手続きを標準化するための通信プロトコルである IOTP(Internet Open Trading Protocol)[10] が挙げられる. エージェントのコミュニケーションにおいても, このようなプロトコルに従ったデータ交換が求められる.

このような, B2B, B2C のコミュニケーションで起こりうる購入, 発注, 支払い, といったビジネスコミュニケーションを対象として, エージェントによる通信を行って表現した. その中で次の点に注目して行った.

より正確に相手へ自分の意図を伝えなればならないビジネスコミュニケーションで扱うため, 再帰的表現などのより表現豊かなメッセージ交換を行う.

本研究では, エージェントのコミュニケーション言語に人間の視点から見ても分かりやすく, デファクトスタンダードとなっている KQML(Knowledge Query and Manipulation Langage)[5] をベースとして使用した.

エージェントのコミュニケーションで扱う言葉も交換するデータと考え, 拡張可能で, 厳密な構造を持つ XML によってコミュニケーションの要素を作成した. そして, KQML のコンテンツの中で, ビジネスコミュニケーションの要素を記述した XML を使用することで表現豊かなメッセージ交換を行った. 本稿の流れとして, 以下のようにこの KQML - XML 表現の作成に沿って述べる.

1. 対象となる世界を表現する XML を考える.
2. その XML に必要な要素を DTD で記述.
3. データの型, 定義域を決め RELAX に変換.
4. KQML に記述できる情報を分離.
5. KQML で表現できない情報をコンテンツとして表す.

2 ビジネスモデル

一般にビジネスモデルとは, 情報技術を駆使した新しいビジネスの方法やそのシステムとされ, インターネットを利用したオンライン・ショッピングやオークション方式などが続々と登場し注目を集めている.

ビジネスコミュニケーションのデータ交換に XML を適用する企業が増えてきた. XML が使われる点として, 以下のような特徴が挙げられる.

1. タグ名を自由に設定できる
   データの意味がタグになっているので, 人間の目で見て内容が理解できるため, メンテナンスが容易になる.

2. 構造が厳密である
   構造が厳密であるため処理を行う際, プログラムの中で扱い易い.

3. テキストデータである
   テキストファイルであるため, ネットワークによる転送が容易となる.

4. 拡張可能である
   タグセット等を簡単に追加することで, システム変更などが非常に柔軟に出来る.

5. プログラムからの構文解析が容易である
   DOM や SAX を使うことにより, さまざまなアプリケーションから XML 文書の内容を操作できる.

このような特徴を持つため, ビジネスコミュニケーションにおいて XML が普及してきた. 本研究では図 2 のようなビジネスコミュニケーションをモデルとし, 主に使用されるようなパフォーマティブなどの要素を DTD で作成した. その DTD を以下に示す.

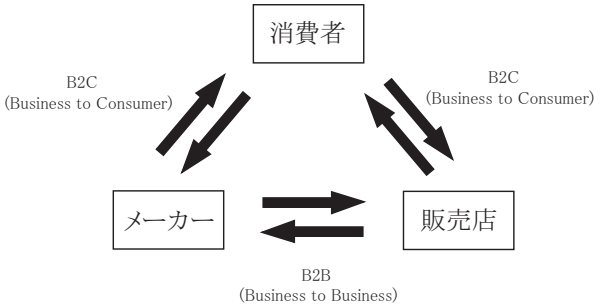

図 2: モデル

```
<!ELEMENT Msg (Agent, context?) >
<!ATTLIST Msg
  msgID ID  #REQUIRED >

<!ENTITY % Logical.Forms " andMsg | orMsg | isNotMsg | iffMsg
                         | ifThenMsg | ifThenElseMsg " >
<!ENTITY % Simple.Content " document | %Logical.Forms; ">
<!ENTITY % Content "Msg | Agent | %Simple.Content; ">

<!ELEMENT Agent (performative+) >
<!ATTLIST Agent
  sender   NMTOKEN #REQUIRED
  receiver    NMTOKEN #REQUIRED >
<!ELEMENT performative ( %Content; )>
<!ATTLIST performative
    act ( advise | assent | assert | buy | charge | confirm | deliver
          | deny | dissent | offer | order | pay | permit | predict
          | promise | question | refuse | request | report| return
          | sell) #REQUIRED>

<!ELEMENT document (#PCDATA)>
<!ATTLIST document
  language  ( prolog | lisp | kif | xml ) "xml"
  ontology CDATA #IMPLIED>
<!ELEMENT andMsg (%Content;)+>
<!ELEMENT isNotMsg (%Simple.Content;)>
<!ELEMENT iffMsg ((%Simple.Content;), (%Simple.Content;))>
<!ELEMENT ifThenMsg ((%Simple.Content;), (%Content;)) >
<!ELEMENT ifThenElseMsg ((%Simple.Content;),
                         (%Content;), (%Content;))>
```

```
<!ELEMENT context (resources?, reply-with?, in-reply-to?, (timeSent
                   | alsoSentTo | alsoAddressedTo | forwardedBy )*)>
<!ELEMENT resources (actors?) >
<!ATTLIST resources
  id    ID  #IMPLIED >
<!ELEMENT actors EMPTY >
<!ATTLIST actors
  id    ID  #IMPLIED
  name  CDATA #IMPLIED >
<!ELEMENT reply-with EMPTY >
<!ATTLIST reply-with
  msgID   NMTOKEN #REQUIRED >
<!ELEMENT in-reply-to EMPTY >
<!ATTLIST in-reply-to
  msgID   NMTOKEN #REQUIRED >
<!ELEMENT timeSent (time) >
<!ELEMENT alsoSentTo (person+)>
<!ELEMENT alsoAddressedTo (person+) >
<!ELEMENT forwardedBy (person+) >
<!ELEMENT time EMPTY>
<!ATTLIST time
  year  NMTOKEN #REQUIRED
  month NMTOKEN #REQUIRED
  dayOfMonth    NMTOKEN #REQUIRED
  hour  NMTOKEN #REQUIRED
  min   NMTOKEN #REQUIRED
  sec   NMTOKEN #REQUIRED >
<!ELEMENT person EMPTY>
<!ATTLIST person
  name  CDATA   #IMPLIED
  URL CDATA #IMPLIED >
```

3 RELAX での表現

現在一般に XML のスキーマ言語として用いられている DTD には, 以下のような問題点が挙げられる.[3]

1. XML 文書ではない
   DTD を操作するためのツールは少なく, 品質も満足できるものではない. その原因は, DTD が独自の構文 (<!ELEMENT ....>や<!ATTLIST ....>など) を持つことにある.

2. データ型がない
   XML で数字を使用する際, DTD では NMTOKEN という型を用いるが, 数字のみのデータでも 1 文字の文字列として扱われるため, 厳密さに欠ける.

3. 名前空間が扱えない
   DTD では, 名前空間の「接頭辞は文書中で任意に変えられる」といった本来の趣旨に従った形での運用は不可能であるため, 整合性に欠ける.

現在, DTD に代わるスキーマ言語として XML Schema, RELAX, TREX, RELAX と TREX を統一した RELAX NG などがある. 本研究ではこの中で理解しやすく, DTD よりも詳細にスキーマを定義するため, RELAX を使用した. 上記の DTD で表現した要素を RELAX によりデータ型, 定義域を追加した表現を以下に示す.

```
<module
        moduleVersion="1.0"
        relaxCoreVersion="1.0"
        xmlns="http://www.xml.gr.jp/xmlns/relaxCore">
```

```
<interface>
  <export label="Msg"/>
</interface>

<tag name="Msg">
  <attribute name="msgID" required="true" type="ID"/>
</tag>
<elementRule role="Msg">
  <sequence>
    <ref label="Agent"/>
    <ref label="context" occurs="?"/>
  </sequence>
</elementRule>

<tag name="Agent">
  <attribute name="sender" required="true" type="string"/>
  <attribute name="receiver" required="true" type="string"/>
</tag>
<elementRule role="Agent">
  <ref label="performative" occurs="+"/>
</elementRule>
```

Agent 要素ではメッセージを送る人の名前, メッセージを受け取る人の名前を表す. KQML の sender, receiver を示している.

```
<tag name="performative">
  <attribute name="act" required="true" type="NMTOKEN">
    <enumeration value="advise"/>
    <enumeration value="assent"/>
    <enumeration value="assert"/>
    <enumeration value="buy"/>
    <enumeration value="charge"/>
    <enumeration value="confirm"/>
    <enumeration value="deliver"/>
    <enumeration value="deny"/>
    <enumeration value="dissent"/>
    <enumeration value="offer"/>
    <enumeration value="order"/>
    <enumeration value="pay"/>
    <enumeration value="permit"/>
    <enumeration value="predict"/>
    <enumeration value="promise"/>
    <enumeration value="question"/>
    <enumeration value="refuse"/>
    <enumeration value="request"/>
    <enumeration value="report"/>
    <enumeration value="return"/>
    <enumeration value="sell"/>
  </attribute>
</tag>
```

---

perfomative 要素に, ビジネスコミュニケーションで使用される行為をいくつか表現した.

```
<elementRule role="performative">
  <choice>
    <ref label="Msg"/>
    <ref label="Agent"/>
    <ref label="document"/>
    <ref label="andMsg"/>
    <ref label="orMsg"/>
    <ref label="isNotMsg"/>
    <ref label="iffMsg"/>
    <ref label="ifThenMsg"/>
    <ref label="ifThenElseMsg"/>
  </choice>
</elementRule>

<tag name="document">
  <attribute name="language" type="string">
    <enumeration value="prolog"/>
    <enumeration value="lisp"/>
    <enumeration value="kif"/>
    <enumeration value="xml"/>
  </attribute>
  <attribute name="ontology" type="string"/>
</tag>
<elementRule role="document" type="string"/>
```

---

document 要素は, 子ノードにエージェント間で交換される知識の表現形式 KIF や, 取引で使用される XML 文書などのテキストが来る. そのテキストの内容で使用されている言語, オントロジーを表現している. これは KQML のコンテンツの内容で使用されている言語, オントロジーを示している.

```
          (中略)

<tag name="context"/>
<elementRule role="context">
  <sequence>
    <ref label="resources" occurs="?"/>
    <ref label="reply-with" occurs="?"/>
    <ref label="in-reply-to" occurs="?"/>
    <choice occurs="*">
      <ref label="timeSent"/>
      <ref label="alsoSentTo"/>
      <ref label="alsoAddressedTo"/>
      <ref label="forwardedBy"/>
    </choice>
  </sequence>
</elementRule>

<tag name="resources">
  <attribute name="id" type="ID"/>
</tag>
<elementRule role="resources">
  <ref label="actors" occurs="?"/>
</elementRule>

<tag name="actors">
  <attribute name="id" type="ID"/>
  <attribute name="name" type="string"/>
</tag>
<elementRule role="actors">
  <empty/>
</elementRule>

<tag name="reply-with">
  <attribute name="msgID" required="true" type="string"/>
</tag>
<elementRule role="reply-with">
  <empty/>
</elementRule>

<tag name="in-reply-to">
  <attribute name="msgID" required="true" type="string"/>
</tag>
<elementRule role="in-reply-to">
  <empty/>
</elementRule>
```

KQML で用いられている reply-with, in-reply-with は主にメッセージの前後関係などを表す context 要素の中で表現する.

```
<tag name="timeSent"/>
<elementRule role="timeSent">
  <ref label="time"/>
</elementRule>

<tag name="alsoSentTo"/>
<elementRule role="alsoSentTo">
  <ref label="person" occurs="+"/>
</elementRule>

<tag name="alsoAddressedTo"/>
<elementRule role="alsoAddressedTo">
  <ref label="person" occurs="+"/>
</elementRule>

<tag name="forwardedBy"/>
<elementRule role="forwardedBy">
  <ref label="person" occurs="+"/>
</elementRule>

<tag name="time">
  <attribute name="year" required="true" type="year"/>
  <attribute name="month" required="true" type="month"/>
  <attribute name="dayOfMonth" required="true" type="date"/>
```

```
    <attribute name="hour" required="true" type="int">
      <minInclusive value="0"/ >
      <maxInclusive value="23"/ >
    </attribute>
    <attribute name="min" required="true" type="int"/>
      <minInclusive value="0"/ >
      <maxInclusive value="59"/ >
    </attribute>
    <attribute name="sec" required="true" type="int"/>
      <minInclusive value="0"/ >
      <maxInclusive value="59"/ >
    </attribute>
  </tag>
  <elementRule role="time">
    <empty/>
  </elementRule>
```

DTD では使用できなかったいくつかのデータの型を RELAX では使用できる． time 要素の year, month, dayOfMonth などはそれぞれ year, month, date といった型を使用できる．専用の型がない hour, min, sec などは int 型で表現し最大値, 最小値といった定義域を設定した．

```
  <tag name="person">
    <attribute name="name" type="string"/>
    <attribute name="URL" type="string"/>
  </tag>
  <elementRule role="person">
    <empty/>
  </elementRule>

</module>
```

4 パフォーマティブ

ここで, 今回作成したビジネスコミュニケーションで考えられる全ての行為を以下に示す．

- advise :sender は receiver に＜内容＞を勧める．
- assent :sender は receiver の＜内容＞について同意する．
- assert :sender は receiver に＜内容＞を主張する．
- buy :sender は receiver の＜内容＞を購入する．
- charge :sender は receiver に＜内容＞を請求する．
- confirm :sender は receiver に＜内容＞について確認する．
- deliver :sender は receiver に＜内容＞を配達する．
- deny :sender は receiver に＜内容＞を否定する．
- dissent :sender は receiver に＜内容＞について異論を唱える．
- offer :sender は receiver に＜内容＞を提供する．
- order :sender は receiver に＜内容＞を依頼する．
- permit :sender は receiver に＜内容＞を許可する．
- predict :sender は receiver に＜内容＞を期待する．
- promise :sender は receiver と＜内容＞を契約する．
- pay :sender は receiver に＜内容＞を支払う．
- question :sender は receiver に＜内容＞を質問する．
- refuse :sender は receiver に＜内容＞を断ることを伝える．
- report :sender は receiver に＜内容＞を知らせる．
- request :sender は receiver に＜内容＞を要求する．
- return :sender は receiver に＜内容＞を返却する．
- sell :sender は receiver に＜内容＞を販売する．

5 KQML - XML

```
=========================================
(performative
      :sender A
      :receiver B
      :in-reply-to P)
      :reply-with W
      :language L
      :ontology N
      :content X
=========================================
```

図 3: KQML メッセージの形式

メッセージのベースとなる KQML の形式を図 3 に示す．このメッセージは何を行動するかをパフォーマティブ performative で表し, メッセージの送信者が A, 受信者が B である．そして, コンテンツ X は言語 L によって書かれ, オントロジー N により制限されている．ID が P のメッセージに対する回答を送るときは :in-reply-with を使用し, メッセージの ID を W として現在のメッセージに対する回答がほしいときは :reply-with を使用する．

ビジネスコミュニケーションの要素を KQML -

XML の形式で表す際, 上記のような KQML で表すことができる情報とそうでないものに分割する. KQML で表現できない情報はコンテンツの中に記述する. この KQML - XML 作成の流れは図 4 のようになる.

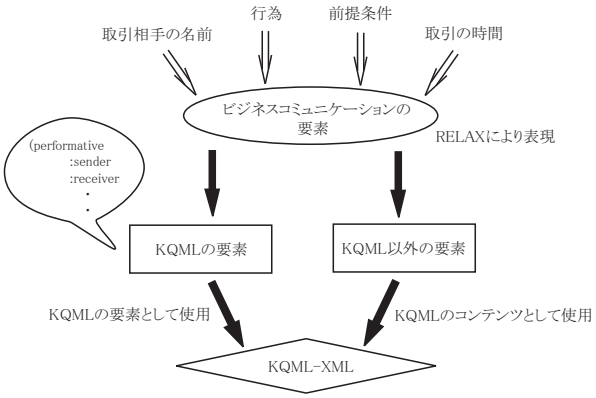

図 4: KQML - XML 作成の流れ

実際に作成した KQML - XML のメッセージを図 5 に示す.

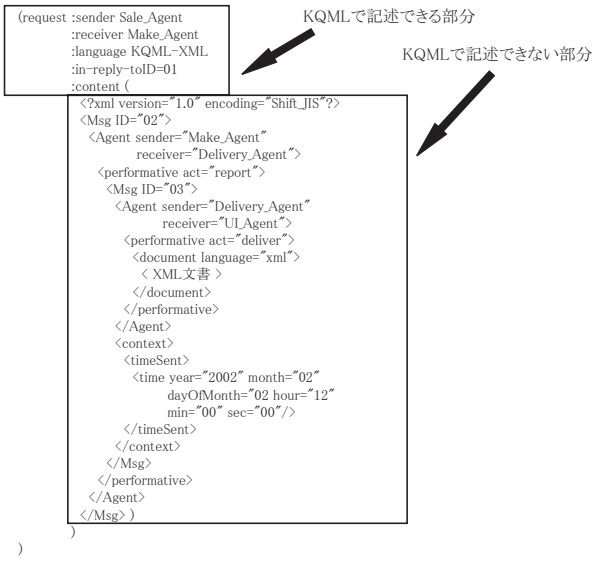

図 5: KQML - XML の記述例

この KQML - XML 表記を行うことにより, 次のような特徴が生まれる.

1. 一つのメッセージの情報量の増加.
2. 再帰的などの複雑な表現を行える.
3. 厳密な構造定義のため, システムを柔軟に変更できる.
4. データ型などのスキーマを明確にすることで, エージェントによる正確な処理を行える.

6 再帰的表現

ここで例として, KQML - XML 記述の最大の特徴となる再帰的なメッセージの例をいくつか挙げる.

最初に, 「「「DeliverAgent が UIAgent に商品の配達をする」ということを MakeAgent が知らせる」ということを SaleAgent が要求する」という再帰的表現を以下に示す.

「「「DeliverAgent が UIAgent に商品の配達をする」ということを MakeAgent が知らせる」ということを SaleAgent が要求する」

```
(request :sender Sale_Agent
         :receiver Make_Agent
         :language KQML-XML
         :reply-with ID=01
         :content (
           <?xml version="1.0" encoding="Shift_JIS"?>
           <Msg ID="02">
             <Agent sender="Make_Agent"
                    receiver="Delivery_Agent">
               <performative act="report">
                 <Msg ID="03">
                   <Agent sender="Delivery_Agent"
                          receiver="UI_Agent">
                     <performative act="deliver">
                       <document language="xml">
                         < 内容 >
                       </document>
                     </performative>
                   </Agent>
                   <context>
                     <timeSent>
                       <time year="2002" month="03"
                             dayOfMonth="04 hour="12"
                             min="30" sec="00"/>
                     </timeSent>
                   </context>
                 </Msg>
               </performative>
             </Agent>
           </Msg> )
         )
)
```

図 6 の様に, ビジネスコミュニケーションにおいて, エージェントは相手企業の全てのエージェントと直接コミュニケーションを行うことができない場合がある. そこで, 他企業との窓口となるエージェントを介してメッセージを送らなければならない. このような状況において, より正確に自分の要求を相手に伝えるためにこの再帰的表現を用いたメッセージが使用される.

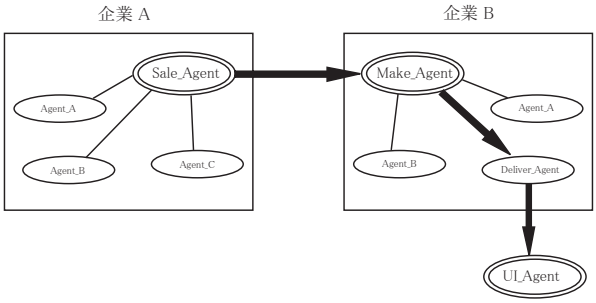

図 6: エージェントの取引形態

次に,「「「SaleAgent が UIAgent に＜内容＞を販売すること」に対して MakeAgent は同意している」ということを知らせる」といった表現を以下に示す.

「「「SaleAgent が UIAgent に＜内容＞を販売すること」に対して MakeAgent は同意している」ということを知らせる」

```
(report :sender Make_Agent
        :receiver Sale_Agent
        :language KQML-XML
        :in-reply-to ID=01
        :content (
          <?xml version="1.0" encoding="Shift_JIS"?>
          <Msg ID="02">
            <Agent sender="Make_Agent"
                   receiver="Sale_Agent">
              <performative act="assent">
                <Msg ID="03">
                  <Agent sender="Sale_Agent"
                         receiver="UI_Agent">
                    <performative act="sell">
                      <document language="xml">
                        < 内容 >
                      </document>
                    </performative>
                  </Agent>
                </Msg>
              </performative>
            </Agent>
          </Msg> )
        )
)
```

上記の表現は, ある二者の間で行われる行為に対して, 第三者の意見を伝える際に, この再帰的表現を使用した. 最後に, 契約を行う際のメッセージのやり取りの例を以下に示す.

「「「MakeAgent が SaleAgent と＜内容＞を契約する」ということを要求する」

```
(request :sender Make_Agent
         :receiver Sale_Agent
         :language KQML-XML
         :reply-with ID=01
         :content (
           <?xml version="1.0" encoding="Shift_JIS"?>
           <Msg ID="02">
             <Agent sender="Sale_Agent"
                    receiver="Make_Agent">
               <performative act="promise">
                 <document language="xml">
                   < 内容 >
                 </document>
               </performative>
             </Agent>
           </Msg> )
         )
)
```

「「「SaleAgent は MakeAgent に＜内容＞について異論を唱える」ために契約する」ということを断る」

```
(refuse :sender Sale_Agent
        :receiver Make_Agent
        :language KQML-XML
        :in-reply-to ID=01
        :content (
          <?xml version="1.0" encoding="Shift_JIS"?>
          <Msg ID="02">
            <Agent sender="Sale_Agent"
                   receiver="Make_Agent">
              <performative act="promise">
                <Msg ID="03">
                  <Agent sender="Sale_Agent"
                         receiver="Make_Agent">
                    <performative act="dissent">
                      <document language="xml">
                        < 内容 >
                      </document>
                    </performative>
                  </Agent>
                </Msg>
              </performative>
            </Agent>
          </Msg> )
        )
)
```

この表現は, ただ契約することを断るのではなく, なぜ断るかという理由を伴って相手に伝えることができる. 今までのコミュニケーションでは, ただ端的に回答するだけだった. 何がどうなのかという理由を明確に伝えることで, 相手はその理由に対して改善し次のコミュニケーションへ繋げることができる.

7 実装

米国スタンフォード大学で開発された JATLite は Java で記述されたクラスライブラリパッケージで, インターネット上で相互に通信ができるエージェントを作成することができる. 本研究ではテンプレートとして, この JATLite を使いエージェントの作成を行った.

全てのメッセージは AMR(Agent Message Router) と呼ばれるエージェントの登録や, メッセージキューを行うメッセージルータでメッセージの送付先を判定してエージェントへ転送している. RELAX によ

りスキーマ定義された KQML - XML メッセージは, この AMR によって メッセージの構文解析が行われる. 従来の AMR は図 7 のように 1 つのホスト上で使用し, それぞれのホストにあるエージェントは AMR があるホストへ接続し登録を行っていた.

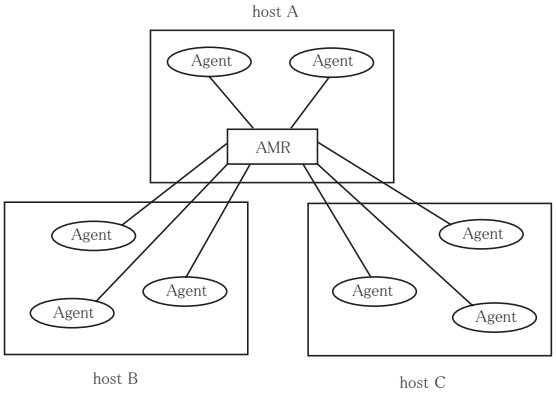

図 7: 従来の AMR

本研究では, より現実のビジネスコミュニケーションで使用することを意識し, 各ホストにそれぞれ AMR を配置した. 他のホストのエージェントへメッセージを送る際には, この AMR で判断して相手ホストの AMR へ送るように改良した. システム構成を図 8 に示す.

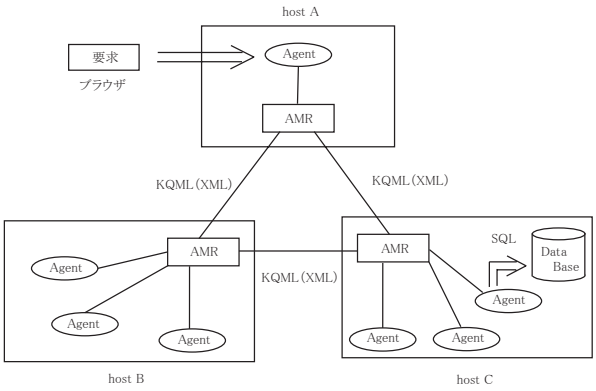

図 8: システム構成

8 おわりに

エージェントによるビジネスコミュニケーションを表現する中で, XML と KQML を併せて使用することで, データフォーマットの標準化, メッセージ表現の拡張を行うことができた. KQML の良さとして, パフォーマティブを拡張することでコミュニケーションの幅を広げることが可能である. XML の拡張可能な特徴や, RELAX を使ったスキーマ定義を行うことによって, システムの変更をより柔軟に行うことができるようになった. そして, 再帰的表現といった複雑な表現も簡単に扱うことができるようになった.

実応用として, RosettaNet や ebXML などで取引のプロトコルを規定することが考えられる. 今後の課題としては, このようなプロトコルを想定した実験を重ね, 標準的なフォーマットに近づけることである.

参考文献

[1] 長尾確: エージェントテクノロジー最前線, 共立出版 (2000).

[2] 岡部 惠造: XML がビジネスを変える!, 翔泳社 (2000).

[3] 村田 真: XMLSchema と RELAX, 電子情報通信学会誌, Vol. 84, No. 12, pp. 890–894 (2001).

[4] 浦本 直彦: XML とデータベース, 電子情報通信学会誌, Vol. 85, No. 1, pp. 45–53 (2002).

[5] Yannis Labrou, Tim Finin: A Proposal for a new KQML Specification, Computer Science and Electrical Engineering Department (CSEE) University of Maryland Baltimore County (UMBC) (1997)

[6] Scott A. Moore: A foundation for flexible automated electronic communication, University of Michigan Bussiness School (2000).

[7] Scott A. Moore:KQML & FLBC: Contrasting agent communication languages, University of Michigan Bussiness School (2000).

[8] 稲森 真二郎: 協調情報エージェントと通信に関する研究, 高知大学大学院理学研究科情報科学専攻 1997 年度修士論文 (1997).

[9] Asuman Docac, Ilker Durusoy, Sena Arpinar, Nesime Tatbul, Pinar Koksal, Ibrahim Cingil, Nazife Dimiler: A Workflow-based Electronic Marketplace on the Web.

[10] D. Burdett: Internet Open Trading Protocol - IOTP Version 1.0 (2000).

[11] Stanford University, JATLite: (http://java.stanford.edu/) (1996)

[12] RELAX 公式サイト: (http://www.xml.gr.jp/relax/) (2001)