ソフトウェア工学 129－16
（２０００．１１．１７）

XMLベースのエージェント動作記述言語CCPL

牛木一成，中村長寿，二宮智彦，金友大
ＮＴＴコムウェア
mailto: nari.nari@nttcom.co.jp

XMLを用いたエージェント動作記述言語であるCCPLを提案し、Ｓ式をXMLで表現させる試行をした。CCPLによりエージェントの内包ロジックの簡単化が図られ、短期間に設計～製造を行うことが出来た。またロジックを変更せずに仕様変更に対応することができるという効果もあった。

CCPL： XML based agent control language

Kazunari Ushiki, Nagatoshi Nakamura, Tomohiko Ninomiya, Dai Kanetomo
NTT Communicationware

In this report, we propose CCPL which is helpful to develop agent-based applications. CCPL is the expression of S formula with XML, and it simplify the internal logic of agents. As a result of our trial development we can develop our agent-based system in the short time. Moreover we can follow the change of specification easily without any modification about the logic for our trial system.

１．はじめに

近年エージェント技術を用いたシステム試作が盛んである。これは分散オブジェクトからの緩やかで自然な拡張と考えられる。他技術と比べたエージェント技術の特徴は、「適応性」「自律性」「協調性」である。

エージェント技術を用いたシステムの特徴としては、会話（メッセージの授受）をベースとした疎な結合、会話の意味付けと会話のやり取りの組合せによる協調活動、といったものがある。総じていえば「知的さ」である。

会話の意味付けにはFIPA[1] ACLの通信行為型やKQML[2]のperformative等の意味記述・交換用言語が開発され、標準化に向けた活動が行われている。会話内容の記述は、文字列の自由な処理が必要となるため、それが容易に記述できるLisp[5][6]言語のＳ式が広く使用されている。ただし昨今の情勢ではデータ形式としてXML[3]を用いることが盛んであり、提案も活発に行われている。

本研究報告では、従来のＳ式より簡単で読解性が高く、かつインターネットでの標準的情報交換形式と整合性の高いXMLベースの新言語CCPLを新たに提案する。以下の章で、その開発の背景、従来の問題点、解決策としてCCPLの提案、適用事例、を述べる。

２．開発の背景

我々は専用線ＳＯシステムを米stanford大で開発されたメッセージ配信基盤であるJATLite[4]を用いて試作してきた。

ここで、ＳＯとはサービスオーダの略称であり、窓口でお客様の要求に従って工事内容を受け付けて回線工事を発注する一連の流れを指し示す。

専用線ＳＯシステムでは、ＳＯの基本的なワークフローに加えて、お客様が要求した専用線の要求仕様とお客様別に蓄積された過去の工事情報を元に分岐・束ねなどの計算を行い、お客様に最適と思われる工事内容の提案を行うリコ

メンデーション機能を実装した。

JATLite に対しては、name service 機能と語彙変換機能を拡張することにより、求める機能を持つエージェントの名前の検索、と使用語彙の異なるエージェント間でのデータ授受を実現した。

エージェント間の会話においては、XML を用いたデータ交換を行った。

メソッドとデータの実装については、会話内容の簡素化とメソッドの複雑化を行った。これはエージェントについての理論的な理解が不足している開発者にも作成が可能なようにとの判断であった。複雑なデータ処理ロジックをエージェントの持つデータ処理メソッドに内包し、会話内容をデータのみと簡素化して、従来の分散オブジェクトと同様の考え方で実装できるようにした。

3．反省と課題の抽出

専用線ＳＯシステムでは設計製造者のエージェントについての理論的な理解不足を補うことを重要視した施策を行った。すなわち会話内容の簡素化とメソッドの複雑化である。これにより以下の問題が発生した。

（1）複雑なメソッドとなり肥大化した

専用線ＳＯシステムでは会話内容をデータのみと簡単化したことで、メソッドの構成は複雑化・多機能化して、エージェントは肥大化し、読解性が低下した。

（2）会話内容の知的さが不足した

メソッドの構成が複雑化･多機能化した分、会話内容はデータのみという単純さだった。これは分散オブジェクトのメソッド呼び出しとなんら代わらない。分散オブジェクトでのメソッド呼び出しと、エージェント間での会話による協調とでは知的さの観点で大きな差異がある。今回のシステムでは、会話内容に知的さが不足している。

（3）エージェント同士が密結合であった

メソッドが複雑化・多機能化したため呼び出し側が相手の持つメソッドの詳しいインタフェースを知らないと正常に呼び出せない。

4．解決策

前述の課題を以下の方針で解決する。

（1）エージェントをスリム化する

肥大化したエージェントをスリム化したい。

（2）知的な会話内容にする

単純なデータ授受であった会話内容を知的にしたい。

（3）XML を使用する

手馴れたデータ形式である XML は継続して使いたい。

上記方針に沿って解決策を策定する。

（1）エージェントをスリム化する

相手の持つメソッドを呼び出すにはインタフェースを知らなければならないことは避けられないが、複雑・多機能なメソッドではなく、単純なメソッド構成にし、エージェントのコード量を削減する。

（2）知的な会話内容にする

知的な会話内容とするために、会話内容には、データだけでなく相手エージェントへの指示を含める。指示記述にはＳ式を用いる。

（3）XML を使用する

指示を盛り込むために会話内容の記述にＳ式を導入するにしても手馴れた XML での表現の使用は継続する。

Ｓ式は Lisp を出自とする対のカッコでリストを表現するものであり、XML とは表現方法が異なる。これは XML で統合する。

上記について、XML をベースとした新規言語を開発し、これを実装することで具体化を図る。具体的には下記の項目を採用した言語とした。

（1）Lisp のエッセンス

Ｓ式を構成する要素として Lisp のリスト処理をエッセンスとして採用する。

（2）XML 表現

データ表現との親和性を保つために言語の全てを XML で表現する。

（3）メソッド呼び出し

エージェントの持つ情報にアクセスするためにエージェントの持つメソッドを呼び出す。

5．言語仕様

解決策検討に従い言語仕様を策定した。名称は Comware agent platform, Content Programming Language（以下 CCPL）とした。

・XML 採用：CCPL は XML をベースとする言語である
・S 式を表現：CCPL は S 式を XML で表現したものである
・移動 agent：データと処理命令を混在させた擬似移動エージェントでもある

文法構成要素は、3GL の基本構文（判定，繰り返し）と、リスト処理および演算である。特殊な用途用に XSLT [1] の呼び出しも可能とした。表 1 に言語仕様の概要を示す。

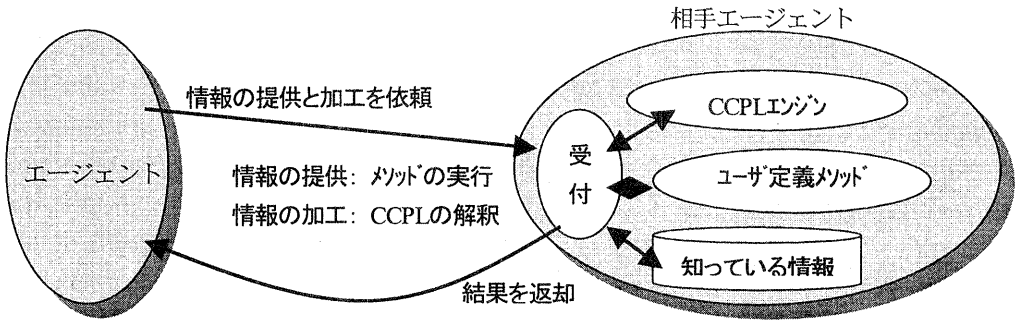
図 1 に CCPL の呼び出しイメージを示す。図 2 は使用例であり、データがどのように操作されていくかをこの図中で概説する。例は「東京都の 1999 年における最高気温を記録した月とその温度を知りたい」というものである。

JATLite は基本的にシングルスレッドの処理系であり、メッセージ処理中に他エージェントへメッセージを発出し結果を受け取ることはできない。しかし CCPL は処理中の他エージェントへのメッセージ発出を可能とする仕様を必要としている。CCPL の実装に伴い、CCPL 実行中の他エージェント呼び出しを可能とするため、JATLite への機能追加としてマルチスレッド化を行った。概要を図 3 に示す。

＜表 1　CCPL 言語仕様概要＞

種類	命令
判定	if, case 等
繰り返し	foreach, do while 等
リスト操作	car, cdr, ｻﾌﾞｾｯﾄ作成(sublist), ｱｲﾃﾑ追加(additem), 連結(connectlist) 等
演算	加減乗除(+, -, /, *, mod), 論理演算(and, or, xor, not) 等
代入	変数への代入(setq), 変数の参照(variable)

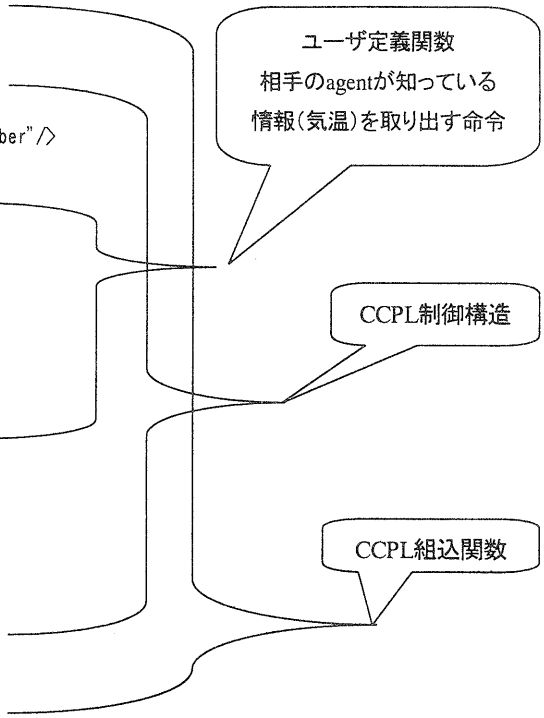
＜図 1　CCPL 呼び出しイメージ＞



＜図２　CCPL使用例＞

東京都の1999年における最高気温を記録した月とその温度を知りたい、という質問です

```
<ccpl:content>
 <ccpl:expression>
  <ccpl:command>CAR</ccpl:command>
  <ccpl:parameter>
    <ccpl:list>
     <ccpl:foreach>
        <ccpl:sort select="//気温" data-type="number"/>
        <ccpl:name>kion</ccpl:name>
        <ccpl:cond>
          <ccpl:expression>
            <ccpl:command>最高気温</ccpl:command>
            <ccpl:parameter>
              <最高気温の検索条件>
                <年>1999</年>
                <都市>東京</都市>
              </最高気温の検索条件>
            </ccpl:parameter>
          </ccpl:expression>
        </ccpl:cond>
        <ccpl:proc>
          <ccpl:item>
            <ccpl:variable>kion</ccpl:variable>
          </ccpl:item>
        </ccpl:proc>
      </ccpl:foreach>
    </ccpl:list>
  </ccpl:parameter>
 </ccpl:expression>
</ccpl:content>
```



まずもっとも内輪にあるユーザ定義命令を実行します

```
<ccpl:expression>
 <ccpl:command>最高気温</ccpl:command>
 <ccpl:parameter>
  <最高気温の検索条件>
   <年>1999</年>
   <都市>東京</都市>
  </最高気温の検索条件>
 </ccpl:parameter>
</ccpl:expression>
```

「最高気温」というユーザ定義命令によって得られる答えは以下で、リスト形式となっています

```
<ccpl:list>
 <ccpl:item><最高気温><月>1</月><気温>14</気温></最高気温></ccpl:item>
 <ccpl:item><最高気温><月>2</月><気温>6</気温></最高気温></ccpl:item>
 <ccpl:item><最高気温><月>3</月><気温>16</気温></最高気温></ccpl:item>
    :
 <ccpl:item><最高気温><月>12</月><気温>16</気温></最高気温></ccpl:item>
</ccpl:list>
```

↓続く

ユーザ定義命令によって得られるリストを並び替えします

```
<ccpl:foreach>
  <ccpl:sort select="//気温" data-type="number"/>      ←「気温」elementの内容でソートを指示
  <ccpl:name>kion</ccpl:name>          ←「kion」という変数でitemを受け渡すことを指示
  <ccpl:cond>
    <ccpl:list>
      <ccpl:item><最高気温><月>1</月><気温>14</気温></最高気温></ccpl:item>
      <ccpl:item><最高気温><月>2</月><気温>6</気温></最高気温></ccpl:item>
      <ccpl:item><最高気温><月>3</月><気温>16</気温></最高気温></ccpl:item>
          :
      <ccpl:item><最高気温><月>12</月><気温>16</気温></最高気温></ccpl:item>
    </ccpl:list>
  </ccpl:cond>
  <ccpl:proc>
    <ccpl:item>
      <ccpl:variable>kion</ccpl:variable>      ←ソートされたitemを順に展開
    </ccpl:item>                                 <ccpl:item>
  </ccpl:proc>                                     <最高気温><月>12</月><気温>16</気温></最高気温>
</ccpl:foreach>                                    </ccpl:item>
```

「最高気温」という命令によって得られる答え

並び替えした結果がCAR命令に渡されます
ここではもっとも高い気温を記録した月だけを取り出すという意味で得られたリストをCARします

```
<ccpl:expression>
 <ccpl:command>CAR</ccpl:command>
 <ccpl:parameter>
  <ccpl:list>
   <ccpl:item><最高気温><月>8</月><気温>35</気温></最高気温></ccpl:item>
   <ccpl:item><最高気温><月>9</月><気温>30</気温></最高気温></ccpl:item>
   <ccpl:item><最高気温><月>7</月><気温>28</気温></最高気温></ccpl:item>
     :
   <ccpl:item><最高気温><月>2</月><気温>6</気温></最高気温></ccpl:item>
  </ccpl:list>
 </ccpl:parameter>
</ccpl:expression>
```

並び替えした結果

リストが持つアイテムが展開されて実際にはこうなります

```
<ccpl:expression>
 <command>CAR</ccpl:command>
 <ccpl:parameter><最高気温><月>8</月><気温>35</気温></最高気温></ccpl:parameter>
 <ccpl:parameter><最高気温><月>9</月><気温>30</気温></最高気温></ccpl:parameter>
 <ccpl:parameter><最高気温><月>7</月><気温>28</気温></最高気温></ccpl:parameter>
    :
 <ccpl:parameter><最高気温><月>2</月><気温>6</気温></最高気温></ccpl:parameter>
</ccpl:expression>
```

展開した結果

「CAR」命令で得られる答えは <最高気温><月>8</月><気温>35</気温></最高気温> です
よって回答はこうなります

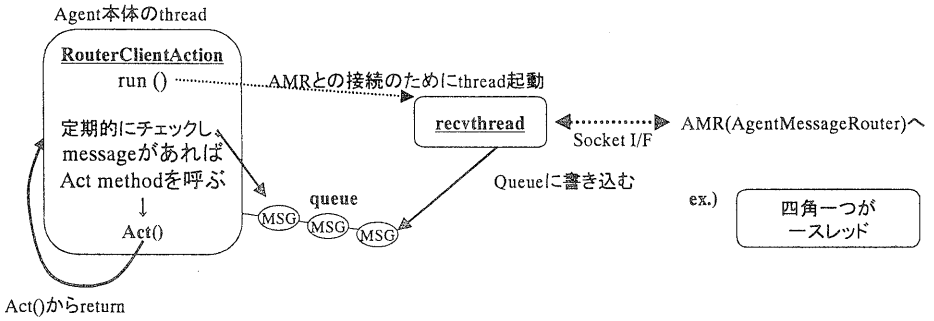
```
<ccpl:content>
 <最高気温><月>8</月><気温>35</気温></最高気温>
</ccpl:content>
```

<ccpl:content>は消去されて内部のデータだけが取り出されます

<最高気温><月>8</月><気温>35</気温></最高気温>

＜図３　JATLite の multi thread 化＞

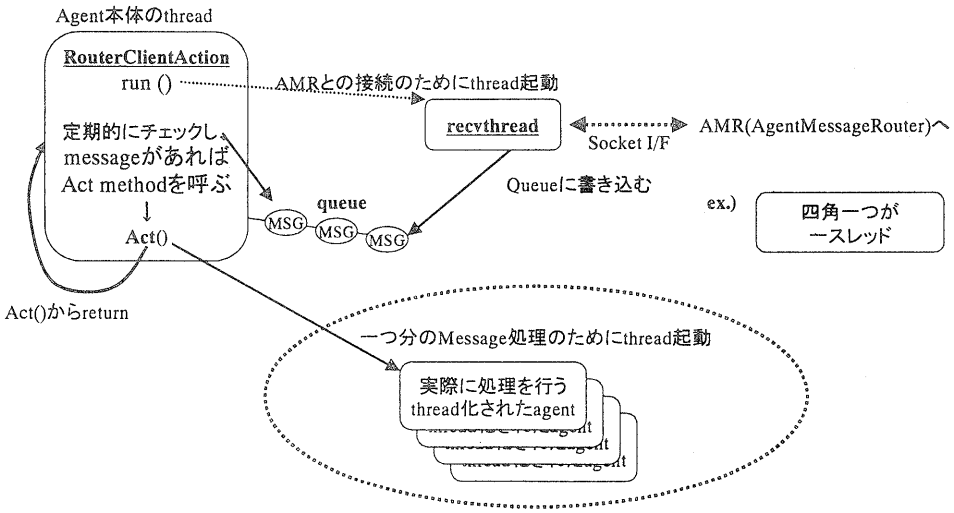
現状のJATLiteはシングルスレッド



- Agentの本体が RouterClientActionである
- RouterClientActionは、起動された後、AMRとのコネクションのためrecvthreadを生成する
- recvthreadはAMRとのコネクションを管理するthreadである
- recvthread はSocketを通じてmessageが送られてきたらそれを受け取って、queueに書き込む
- RouterClientActionはqueueを監視しており、queueにmessageがあれば取り出してAct methodを呼ぶ
- Act methodはuser programでoverrideされた実際の処理を行うmethodである

※ Agentに対して、同時に複数のmessageが送信された場合にも、AMR～Agent間の接続は一本しかないため、結局1 messageずつrecvthreadで受信され、queueに貯められる。RouterClientActionでの処理は直列（シーケンシャル）である。

CCPLの適用のためにJATLiteをマルチスレッド化

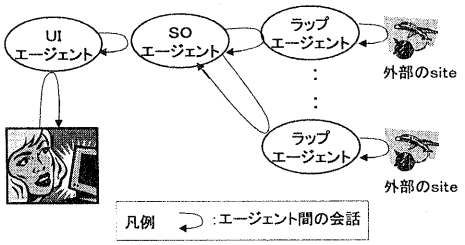


マルチスレッド化により、CCPL実行中の他エージェント呼び出しを可能にした

6．適用事例

我々は航空券オーダーのワンストップサービスシステムをCCPLを用いて試作した。図4にシステム構成を示す。

図4 航空券オーダーのワンストップサービスシステムの構成



航空券オーダーのワンストップサービスシステムはユーザが提示した条件（日時、行き先、価格等）を元に、ウェブページ（航空会社）にて検索を行い、検索結果（空席のある便）を精査して最適な案を提示する、というものである。

エージェントの構成は、航空会社をレガシーラッピングする「ラップエージェント」、複数のラップエージェントが返す結果を元に最適な案を検討する「ＳＯエージェント」、ユーザとのインタラクションとＳＯエージェントへの指示を行う「ＵＩエージェント」という簡潔なものである。

- ラップ：航空会社をレガシーラッピング
- ＳＯ：　最適な案を検討
- ＵＩ：　ユーザとのインタラクション，ＳＯエージェントへの指示

7．考察

課題の解決，定性的な評価について考察する。その後で他システムへの適用可能性と改善すべき点を述べる。

7.1 課題の解決

課題として挙げられた以下の点について適用の効果を考察する。

（1）複雑なメソッドを持ち肥大化した

CCPLの適用によりメソッドの単機能化が図られスリム化された。

処理の中心となるＳＯエージェントで比較すると、専用線ＳＯシステムでは約900行だったが、航空券オーダーシステムでは約300行と1/3に削減された。

（2）会話内容の知的さが不足した

CCPLの適用により、会話内容にデータだけではなく相手エージェントへの指示を含めることができた。

（3）エージェント同士が密結合だった

CCPLの適用により、メソッドの単機能化が図られ、呼び出しインタフェースが整理されたことにより、疎結合化が進んだ。

7.2 定性的な評価

エージェント部分の実装方式が事前に作成した専用線ＳＯシステムとは大きく異なっていたため、設計～実装方法（方式とクラス使用作法）の理解に時間を割き、開発担当者の立ち上がりに時間がかかった。しかしエージェントをひとつ作ったところ開発担当者は設計～実装方法の理解ができたようで、以降のエージェント作成は非常にスムースであり当初予定通りのスケジュールでシステム試作が終了した。

専用線ＳＯシステムとの比較では、エージェントが持つデータ操作用のコード量を多少なりとも減らすことができた。これはエージェントの行動を表現するのに適したＳ式を、使い慣れたXMLで扱えたため、作るのが楽だったからである。

また、試作途中でデータ操作の仕様が変更されたことがあったが、エージェント持つメソッドには手を加えずに、データ操作指示内容を書き換えることで済ますことができた。

今回は三種類のエージェント作成だったため目立つ効果が得られたとは言いがたいが、もっと多数のエージェントを作成する際には大きな差異が出てくることが期待できる。

7.3 他システムへの適用可能性

CCPL 自身は、JATLite 上でのみで実行可能な実装ではなく、独立している。いわば汎用スクリプト言語として利用可能である。適用例としては以下のようなものが考えられる。

（1）Web アプリケーション

・XML データを整形して HTML [1] 変換する

・XSLT では対応できない複雑なリスト処理は CCPL で行う

（2）一般アプリケーション

・業務での受け渡しデータはテーブルをイメージしたリスト構造であることが多いため、CCPL で複雑なリスト処理を行いロジックを簡単化する

7.4 改善すべき点

実行時の必要資源と処理速度の問題が大きい。また指示内容の記述量も問題である。

（1）資源を浪費

Java [TM] [7] 言語で DOM [1] による実装を行っているため実行に際して資源を浪費する傾向がある。

（2）処理時間がかかる

再帰処理を行っているためデータ量,ネスト階層が増えると処理時間がかかる傾向がある。

（3）指示内容の記述量が多い

XML 表現にしたために、総文字数が増加した。また attribute を積極的に用いず、element にて記述するようにしていたために行数が増えた。

これらの問題を避けるために現状では以下の回避策を施行する必要がある。

・非常に大量のトラフィックが予想される個所には使用は避けなければならない。

・高度なデータ処理が必要な個所に局所的に効果的に使用したい。

・前処理としてフィルタリングを行い、引き渡すデータを減らすことも時として必要となる。

・データ構造を単純化して負荷を減らすことも時として必要となる。

9．まとめ

XML を用いたエージェント動作記述言語である CCPL を提案し、S 式を XML で表現させる試行をした。CCPL によりエージェントの内包ロジックの簡単化が図られ、短期間に設計～製造を行うことが出来た。またロジックを変更せずに仕様変更に対応することができるという効果もあった。

参考文献：

[1] W3C http://www.w3.org/

[2] Labrou, Y., and Finin, T. A Proposal for a new KQML Specification, TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD 21250. 1997

[3] FIPA http://www.fipa.org/

[4] Jeon, H., Petrie, C., and Cutkosky, M. R. JATLite: A Java Agent Infrastructure with Message Routing.Internet Computing, March-April.2000.

[5] 湯浅太一，荻谷昌巳「Common Lisp ハンドブック」岩波書店 1987 ISBN4-00-007690-6

[6] 湯浅太一，荻谷昌巳「Common Lisp 入門」岩波書店 1986 ISBN4-00-007685-X

[7] SUN Microsystems http://java.sun.com/