人工知能学会研究会資料
SIG-SW&ONT-A301-04

DAML-S に基づくサービス連携エージェントの実現について

福田 直樹[1]　和泉 憲明[2]　山口 高平[1]

[1]静岡大学情報学部
〒432-8011 静岡県浜松市城北 3-5-1
{fukuta, yamaguti}@cs.inf.shizuoka.ac.jp

[2]産業技術総合研究所サイバーアシスト研究センター
〒135-0064 東京都江東区青海 2-41-6
niz@ni.aist.go.jp

あらまし: DAML-S におけるプロセス記述を利用して Web サービスを連携させるソフトウェアを実装する場合，DAML-S の記述内容を解釈し，実際に Web サービスを呼び出すためのコーディングが必要であり，開発者にとって負担となっていた．本発表では，Web サービス連携の記述言語として DAML-S に着目し，DAML-S で定義されたサービス連携を実装したエージェントのコードを自動生成するための枠組みを提案する

１．はじめに

Web サービス(Web Services)は，アプリケーション統合の強力な手段の1つとして注目されている． Web サービスでは，アプリケーション間の疎な結合を前提としているため，アプリケーション構築基盤を自由に選択できる利点があり，連携のためのプロトコル標準化が進んでいるため，企業を中心に導入が進んでいる．今後，企業内アプリケーション統合から企業間アプリケーション統合(EAI, Enterprise Application Integration から B2BI, Business to Business Integration)へ発展させる際に重要となるのが，Web サービス連携である．

複数のサービスを知的に連携させることをめざして，セマンティック Web コミュニティを中心に セマンティック Web サービス[McIlraith 2001]が提唱されている．セマンティック Web サービスでは，Web サービスの意味記述を用意することで，複数の Web サービスを知的に連携させることを目指している．Web サービスの意味記述を行うためのサービス記述言語 DAML-S[DAML-S Coalition 2002]が開発されている．DAML-S では，1つのサービスを複数のサービスの組み合わせによって実現するための，サービスプロセス記述を用意している．

サービスのプロセスとは，そのサービスを利用するために経るべき過程である．例えば「6月19日に名古屋発，6月19日ロサンゼルス着の航空便を予約する」というサービスを行う場合に，「6月19日に名古屋発で6月19日にロサンゼルス着の航空便の便名と空席状況を調べる」，「その便に予約を入れるかどうかを決定する」，「その便に予約を入れる手続きをする」という3つのプロセスからなる．ここで注意すべき点は，プロセスの内部に判断および決定を行うプロセスが含まれる点である．非常に単純なプロセスを除き，サービスのプロセスを考慮してサービスを利用する場合， その過程における判断および決定はサービスの利用者側で行われるべきものである．サービスプロセスにおける判断および決定はサービス利用者が行うものであるため，サービス提供者か

```
 1  <daml:Class rdf:ID="BravoAir_Process">
 2    <daml:subClassOf rdf:resource="&process;#CompositeProcess"/>
 3    <daml:subClassOf>
 4      <daml:Restriction>
 5        <daml:onProperty rdf:resource="&process;#composedOf"/>
 6        <daml:toClass>
 7          <daml:Class>
 8           <daml:intersectionOf rdf:parseType="daml:collection">
 9             <daml:Class rdf:about="&process;#Sequence"/>
10             <daml:Restriction>
11              <daml:onProperty rdf:resource="&process;#components"/>
12              <daml:toClass>
13                <daml:Class>
14                 <process:listOfInstancesOf
15                    rdf:parseType="daml:collection">
16                   <daml:Class rdf:about="#GetDesiredFlightDetails"/>
17                   <daml:Class rdf:about="#SelectAvailableFlight"/>
18                   <daml:Class rdf:about="#BookFlight"/>
19                 </process:listOfInstancesOf>
20               </daml:Class>
21              </daml:toClass>
22            </daml:Restriction>
23           </daml:intersectionOf>
24         </daml:Class>
25       </daml:toClass>
26     </daml:Restriction>
27   </daml:subClassOf>
28  </daml:Class>
```

図 1：DAML-S によるサービスプロセス記述例(部分)

ら提供されるものではない．つまり，サービスを利用する場合，単にサービス利用プロセスの構成要素を並べてその入出力をつなぎ合わせるだけでは不十分である．

サービスをソフトウェアが自動的に利用するためには，なんらかの基準に基づいて判断・決定するための機構が必要である．その機構を持つソフトウェアは，エージェントと呼ぶにふさわしいと考える．我々は，サービス連携ソフトウェアはエージェントによって構成されるべきであると考える．サービス連携エージェントには，企業内・企業間でのサービス連携のみでなく，ユーザ・企業間でのサービス連携への発展が考えられる．企業内でのサービス連携では，サービスの連携プロセスはそれほど動的に大きく変化するものではない．この場合，WSDL による記述に基づいて Web サービス間の連携を作り込むことは，十分に現実的な方法である．企業間でのサービス連携を考えた場合，企業内での場合と比較して，サービスの連携プロセスは複雑になり，その変化も大きくなると予想される．ユーザのニーズを駆動源として動的にサービスを連携させる場合には，サービスの連携プロセスは動的に構成され，その連携先も状況に応じて変化すると考えられる．駆動源となるユーザのニーズを集団から個人のレベルに詳細化していくと，もはや固定的なプロセスでサービスの連携を記述していくことは困難となり，エージェントによる自律的なサービス連携の実現が必須となる．サービスの利用対象を企業からユーザへと変えていくことでサービスの連携が複雑化するというのはあくまでも仮説であるが，近年の商品・サービスの個人化の傾向から，我々はこの仮説が十分な妥当性を持つものであると考えている．さらに，サービスの連携をエージェントが行うことにより，状況に応じた代替サービスの検索と利用，サービスの利用条件（利用料金など）の動的な変更やそのための交渉などが実現できる可能性がある．我々がサービス連携エージェントの実現を目指すのは，これらの可能性を追求することを目的としているからである．

サービス連携エージェントを実装する場合，サービスの利用プロセス中に含まれる判断・決定プロセスの部分は自動的に構成

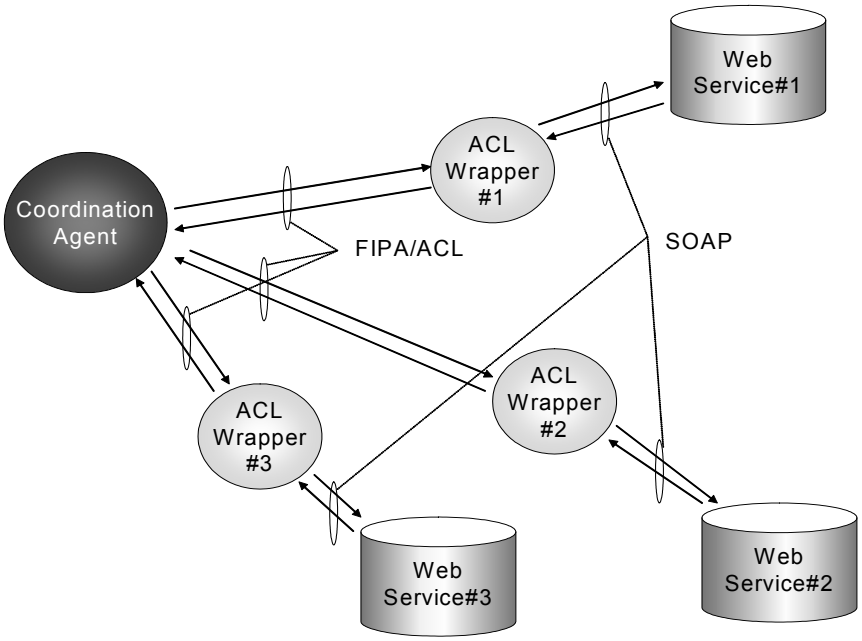

図 2: JADE エージェントによる Web サービス連携

することが容易ではないが，それ以外の固定的なプロセスについてはその実装を全体あるいは部分的に自動化することができると考えられる．例えば，DAML-S におけるプロセス記述を利用して Web サービスを連携させるエージェントを実装する場合，エージェント内部に，DAML-S の記述内容を解釈し，実際に Web サービスを呼び出すためのコーディングが必要であり，この部分を自動化できればエージェント開発者の負担を軽減できると考えられる．本論文では，Web サービス連携の記述言語として DAML-S に着目し，DAML-S で定義されたサービス連携を実装したエージェントのコードを自動生成するための枠組みを提案する．

２．Web サービス連携コードの生成

２．１　DAML-S とプロセス

DAML-S では，サービスを3つの側面(プロファイル，プロセス，接地)から記述する．プロファイルでは，そのサービスが持つ静的な性質（例えば，サービスのカテゴリ，入出力およびサービス提供者に関する情報など）を記述する．プロセスでは，そのサービスの実現方法を，プロセスの構成（Composition）として記述する．接地（Grounding）では，サービスを構成する個々のプロセスの Web サービスとの対応づけについて，WSDL(Web Service Description Language)を補助的に用いながら記述する．

図 1 に，DAML-S による航空券販売サービスのプロセス記述例[1]の一部を示す．図 1 では，航空券販売プロセスが，GetDesiredFlightDetails, SelectAvailableFlight, および BookFlight の3つのプロセスから構成されることを示している．図1の1行目は，このプロセスが BravoAir Process という識別子で定義されることを示している．このプロセスが複数のプロセスによって構成される（複合プロセスである）ことを2行目で示している．3行目から26行目までが，このプロセスの構造に関する制約を示している．この制約では，このプロセスの構成に対する制約として（5行目），その構成がプロセスの逐次処理(sequence)であり（9行目），その構成要素が3つのプロセス GetDesiredFlightDetails, SelectAvailableFlight, および BookFlight からなる（12～21行目）ことを示している．このように，DAML-S におけるサービスのプロセスは，いくつかの子プロセスに分解されて

[1] http://www.daml.org/services/daml-s/0.9/BravoAirProcess.daml より一部を引用

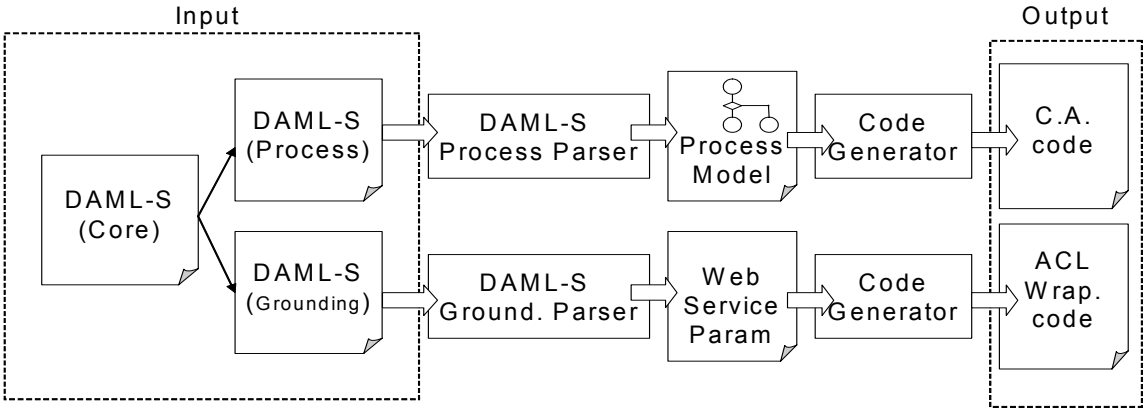

図 3: 試作システムの構成

いき，最終的に Web サービスと同等な粗さのプロセスの組み合わせに展開される．図1の記述では省略したが，各プロセスに対する入力，出力，前提条件，作用については，別途記述される．

サービスをプロセスにしたがって実際に実行するには，各プロセスの実行以外に，プロセスと次のプロセスの実行の間に推論やユーザとの対話を行う場合がある．例えば，航空券に関する希望の詳細を伝え予約可能な便の一覧を得るプロセスから，予約すべき便名を選択してその航空便の予約プロセスに進むためには，予約可能な便の一覧から予約すべき便を選び出すプロセスが必要となる．DAML-S では Web サービスとの対応づけからプロセスが記述されるため，この判断・決定プロセスは明示的にモデル化されない．本研究では，明示化されない判断・決定プロセスを除いた，サービスの実行プロセス全体のひな形と，その構成要素となる Web サービスの呼び出し部分について，コードを自動生成する機構を用意することにより，開発者の負担を軽減することを目指す．開発者は，自動生成されたコードに推論やユーザとの対話のためのコードを追加することにより，目的のサービスを実行するソフトウェア（エージェント）の実装が可能となる．

２．２ JADE に基づくエージェントの生成

JADE [Bellifemine 2001] は FIPA 仕様に基づくエージェントを JAVA 言語で実装するためのフレームワークであり，エージェント開発コンペティション Agentcities 等で主要なエージェント開発フレームワークの1つとして用いられている．図 2 に，本論文で扱う Web サービス連携コードの構成（すなわち，本システムで生成したいコードのシステム構成）を示す．本コードでは，複数のエージェントを用いて Web サービス連携を実現する．具体的には，1つの Web サービスに対して1つの JADE ラッパーエージェント (ACL Wrapper)を割り当て，Web サービス連携プロセスを管理するエージェント (Coordination Agent)として別途 JADE エージェントを1つ割り当てる．Web サービス連携管理エージェントは，JADE に内蔵された FIPA-ACL 互換のエージェント間通信機構を用いてラッパーエージェントと通信し，間接的に Web サービスにアクセスすることで，複数の Web サービスを連携させたサービスプロセスを遂行する．この構成をとることにより，Web サービスアクセスのためのコードを Web サービス連携のコードから効果的に分離することができ，生成したコードの加工およびメンテナンスに役立つ物と考えられる．

DAML-S で記述されたサービスプロセスは，各プロセス構成要素に対応する状態を用意することで，1つの状態遷移図として記述できる．JADE では，エージェントの振る舞い(Behavior)が一連の複雑なプロセスとなる場合には，複数の振る舞いに分割し，状態を示す変数によってそれらの振る舞いを切り替えるようにコーディングを行う必要

がある. 本システムでは, DAML-S による記述からサービスプロセスを表現した状態遷移図を構築し, 各状態における処理を振る舞いに対応させる.

図 3 に, 本システムの構成を示す. 本システムの試作には, JAVA 言語および JAVA 上で動作する論理型言語 MiLog[Fukuta 2001]を用いている. 本システムのユーザインタフェース, および JADE エージェントのサンプルコードを, 付録1および付録2に示す.

３. 関連研究

DAML-S におけるプロセス記述ツールとして, ペトリネットに基づくプロセス記述・視覚化ツール KarmaSIM [Narayanan 2002] が提案されている. KarmaSIM では, ペトリネット理論に基づいたグラフ記述から DAML-S プロセスの生成, および生成したプロセスのシミュレーションを行うことが可能である. 本研究では, DAML-S におけるプロセスの記述そのものを支援するのではなく, 記述されたプロセスを実際に実行するためのコード生成を支援する点で異なっている.

JADE に基づく Web サービスエージェント構築支援ツールとして WSDL Tool [SZTAKI 2003] が提案されている. WSDL Tool では, WSDL 定義から Web サービスにアクセスするための JADE コードを生成することが可能となっている. 本研究では, Web サービス自身に対するアクセスよりもむしろ, 複数の Web サービスを連携させるためのプロセス記述のサポートに比重を置いており, この点で本研究は WSDL Tool と異なる.

４. まとめ

本論文では, JADE エージェントによる Web サービス連携のためのコード自動生成機構の設計およびその試作について述べた. 本機構では, サービス連携に関するプロセス記述を参照し, 適切なサービス連携コードを生成可能である. 本機構では, サービス連携のためのひな形コードを生成可能であるが, あらかじめサービス連携プロセスが与えられている必要がある. また, サービス連携プロセスを含んだ完全な形のアプリケーションを自動生成できるわけではなく, 開発者によるコーディングが別途必要となっている. 動的に必要なサービスを見つけ, それらのサービス連携プロセスを自動構築すること, およびサービス連携プロセス自身の記述を支援することは, 本研究の扱う範囲を超える内容であるが, 重要な課題である. これらの課題と本提案手法との連携については今後の課題としたい.

参考文献

[DAML-S Coalition 2002] DAML-S Coalition: “DAML-S: Web Service Description for the Semantic Web”, In Proc of 1st International Conference on Semantic Web (ISWC2002), pp.348—363. (2002)

[McIlraith 2001] S. A. McIlraith, T. C. Son, and H. Zeng: “Semantic Web Services”, IEEE Intelligent Systems, pp.46—53, March/April, 2001. (2001)

[Bellifemine 2001] F. Bellifemine, A. Poggi, and G. Rimassa: “JADE: a FIPA2000 compolant agent development environment”, In Proc. of International Conference on Autonomous Agents 2001 (Agents2001), pp.216—217. (2001)

[SZTAKI 2003] System Development Department, Computer and Automation Research Institute (SZTAKI): “Tool for Deploying Web Service Wrapper Agents”, In Proc. of 2003 Agentcities Aent Technology Exhibition, pp.44—46. (2003)

[Narayanan 2002] S. Narayanan, and S. A. McIlraith : “Simulation, Verification and Automated Composition of Web Services”, In Proc. of 11th International World Wide Web Conference (WWW2002), pp.77—88. (2002)

[Fukuta 2001] N. Fukuta, T. Ito, and T. Shintani: “An Approach to Building Mobile Intelligent Agents Based on Anytime Migration”, In R. Kowalczyk et, al. (Eds.) LNAI, Vol2112, pp.219—228. (2001)

付録１：試作した JADE コード生成システムのユーザインタフェース

ブラウザより，コード生成対象とする DAML-S ファイルへの URL を指定する．

指定した DAML-S ファイル内部で定義されているサービス一覧から，コードを生成したいサービスの名前を指定する．

コードがシステムによって自動生成され，生成されたコードを JAR 形式でまとめたファイルをダウンロードすることができる．

付録２：DAML-S プロセス実行用 JADE エージェントのサンプルコード（BravoAirProcess プロセス実行用）

```
package jadesample1;

import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.DFService;
import jade.domain.FIPAException;


public class BravoAirProcessAgent extends Agent {

   public static void main(String args[]) {
      jade.Boot.main(args);
   }
   public static AID toAID = null;
   class BravoAirProcessAgentBehaviour extends jade.core.behaviours.CyclicBehaviour {
      public BravoAirProcessAgentBehaviour(Agent a) { super(a); }
      static final int Idle = 0;
      static final int GetDesiredFlightDetails = 1;
      static final int SelectAvailableFlight = 2;
      static final int BookFlight = 3;
      static final int End = -1;
      int state = 0;

      public void action() {
         ACLMessage msg = null;
         switch(state) {
         case Idle:
            // wait for requests...
            msg = blockingReceive();
            // switch to next
            if( msg != null ) {
               //TODO: write program here
            }
            state = GetDesiredFlightDetails;
            break;
         case GetDesiredFlightDetails:
            // send message to process GetDesiredFlightDetails
            sendMessageToGetDesiredFlightDetailsProcess();
            // receive reply
            receiveMessageFromGetDesiredFlightDetailsProcess();
            // switch to next
            state = SelectAvailableFlight;
            break;
         case SelectAvailableFlight:
            // send message to process SelectAvailableFlight
            sendMessageToSelectAvailableFlightProcess();
            // receive reply
            receiveMessageFromSelectAvailableFlightProcess();
            // switch to next
            state = BookFlight;
            break;
         case BookFlight:
            // send message to process BookFlight
            sendMessageToBookFlightProcess();
            // receive reply
            receiveMessageFromBookFlightProcess();
            // switch to next
            state = End;
         case End:
            if( msg != null ) {
               ACLMessage reply = msg.createReply();
               reply.setContent(generateReplyMessage());
               send(reply);
            }
            msg = null;
            state = Idle;
         default:
            break;
         }
      }
      String generateReplyMessage() {
         // TODO: write program here
         return("hello");
      }
      void sendMessageToGetDesiredFlightDetailsProcess() {
         ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
         msg.setSender(getAID());
         msg.addReceiver(new AID("GetDesiredFlightDetailsProcess",AID.ISLOCALNAME));
         String cont = getMessageBodyForGetDesiredFlightDetailsProcess();
         msg.setContent(cont);
         send(msg);
      }
      void receiveMessageFromGetDesiredFlightDetailsProcess() {
         ACLMessage msg = blockingReceive();
         if( msg != null ) {
            if( msg.getPerformative() == ACLMessage.INFORM    ) {
               storeReplyMessageFromGetDesiredFlightDetailsProcess(msg.getContent());
            } else {
```

```
            // failed
            state = End;
          }
        }
      }
      void sendMessageToSelectAvailableFlightProcess() {
        ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
        msg.setSender(getAID());
        msg.addReceiver(new AID("SelectAvailableFlightProcess",AID.ISLOCALNAME));
        String cont = getMessageBodyForSelectAvailableFlightProcess();
        msg.setContent(cont);
        send(msg);
      }
      void receiveMessageFromSelectAvailableFlightProcess() {
        ACLMessage msg = blockingReceive();
        if( msg != null ) {
          if( msg.getPerformative() == ACLMessage.INFORM    ) {
            storeReplyMessageFromSelectAvailableFlightProcess(msg.getContent());
          } else {
            // failed
            state = End;
          }
        }
      }
      void sendMessageToBookFlightProcess() {
        ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
        msg.setSender(getAID());
        msg.addReceiver(new AID("BookFlightProcess",AID.ISLOCALNAME));
        String cont = getMessageBodyForBookFlightProcess();
        msg.setContent(cont);
        send(msg);
      }
      void receiveMessageFromBookFlightProcess () {
        ACLMessage msg = blockingReceive();
        if( msg != null ) {
          if( msg.getPerformative() == ACLMessage.INFORM    ) {
            storeReplyMessageFromBookFlightProcess(msg.getContent());
          } else {
            // failed
            state = End;
          }
        }
      }
      String getMessageBodyForGetDesiredFlightDetailsProcess() {
        // TODO: write program here
        return("hello");
      }
      void storeReplyMessageFromGetDesiredFlightDetailsProcess(String cont) {
        // TODO: write program here
      }
      String getMessageBodyForSelectAvailableFlightProcess() {
        // TODO: write program here
        return("hello");
      }
      void storeReplyMessageFromSelectAvailableFlightProcess(String cont) {
        // TODO: write program here
      }
      String getMessageBodyForBookFlightProcess() {
        // TODO: write program here
        return("hello");
      }
      void storeReplyMessageFromBookFlightProcess(String cont) {
        // TODO: write program here
      }
    }
    protected void setup() {
      DFAgentDescription dfd = new DFAgentDescription();
                ServiceDescription sd = new ServiceDescription();
                sd.setType("BravoAirProcess");
                sd.setName(getName());
                sd.setOwnership("Owner");
                //sd.addOntologies("AnOntology");
                dfd.setName(getAID());
                dfd.addServices(sd);
                try {
                  DFService.register(this,dfd);
                } catch (FIPAException e) {
                  System.err.println(getLocalName()+
          " registration with DF unsucceeded. Reason: "+e.getMessage());
                  doDelete();
                }
              BravoAirProcessAgentBehaviour behaviour= new    BravoAirProcessAgentBehaviour(this);
                addBehaviour(behaviour);
    }
}
```