マルチエージェント系協同開発環境 GuardNet

鷹合 基行・武田 英明・西田 豊明
(奈良先端科学技術大学院大学 情報科学研究科)

GuardNet : A Cooperative Development Environment for Multiple Agent Systems

Motoyuki TAKAAI, Hideaki TAKEDA, Toyoaki NISHIDA
Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-01 Japan

Abstract: In this paper, we propose a new environment that supports cooperative development of multiple agent systems (MAS). Since it is natural that MAS is developed cooperatively by multiple programmers, it is important to provide environments that support their cooperative work. For this purpose, we suggest a system called *GuardNet*. In *GuardNet*,which is also realized as MAS, there exists an agent called *Guardant* for every agent in target MAS. *Guardants* can store requirement specifications of their for target and relevant agents, and exchange them with each other on behalf of agent programmers. Another feature of *GuardNet* is *agent-imitating* facilities for debugging and testing. In *agent-imitating*, a *Guardant* behaves as a proxy either for the target agent or relevant agents. The proxy simulates the agent based on the requirement a specifications determined by *Guardants'* communication. Using *agent-imitating* facilities, we can test the agent system which includes incomplete agents.

keyword: Multiple Agent System, Agent Specification, Cooperative Development, Agent-imitationg

e-mail: motoyu-t@is.aist-nara.ac.jp

1 はじめに

近年マルチエージェント系による問題解決や、知識情報処理についての研究が盛んに行なわれている。マルチエージェント系とは、いくつかのエージェントと呼ばれる情報処理システムを連結しそれらの相互作用によってより大きな問題を解決していくものである。我々は知識コミュニティと呼ぶ、マルチエージェント系による大規模知識ベースの実現にむけての知識の共有と再利用の枠組を提唱し、システムの開発を行ってきた [4]。

マルチエージェント系は複数の開発者によって開発されることが多いが、開発者同士の協調が十分なされていないとエージェント同士の相互作用が開発できない。また、エージェントは他のエージェントが存在して初めて機能するので、エージェント単体での動作チェックが困難である。

そこで、本研究ではエージェント開発者同士の協同開発を支援する環境をマルチエージェント系で構成した GuardNet と呼ぶシステムを提案する。

GuardNet ではエージェントが開発者に代わって開発中のエージェントの仕様を交換することにより、仕様の決定を支援する。こうして決定された仕様をもとに、エージェントのテンプレートを作成することによりコーディングを支援する。また、 GuardNet は、そこでエージェントの仕様をもとに、エージェント系の動作のシミュレーションを実現することにより、エージェントのデバッグを支援する。

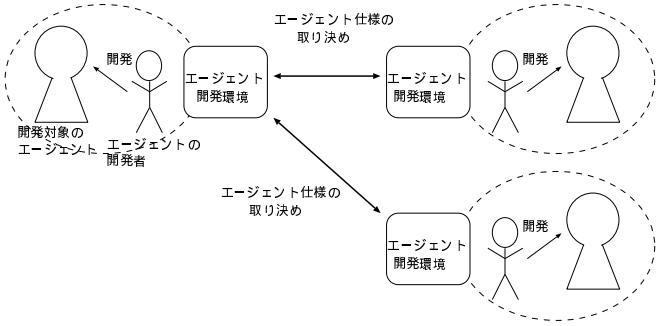

Fig 1: GuardNet による開発者同士の結び付け

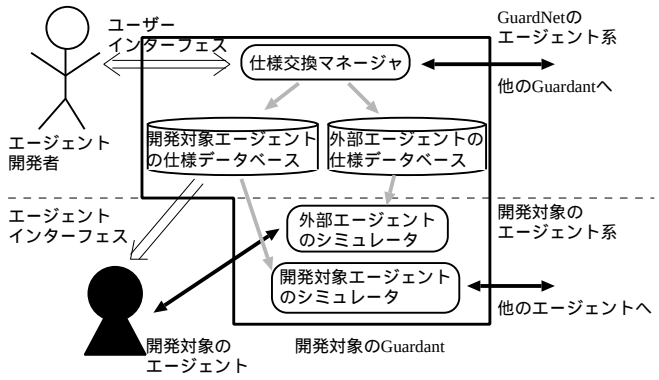

Fig 2: Guardant

2 GuardNet による開発者同士の協同開発の支援

GuardNet は Fig 1のようにエージェント開発者ごとに開発環境エージェント (Guardant と呼ぶ) を配置し、それらのメッセージ交換によってエージェントの協同開発を支援する。

マルチエージェント系の開発環境には柔軟な環境が必要である。よって、 GuardNet では開発者同士の協同開発を支援する環境をマルチエージェント系によって実現した。その結果、エージェントの開発者は、エージェントの仕様の決定とその仕様にもとづく実行シミュレーションとを同一の開発環境で行えるようになった。

GuardNet のエージェント系は、開発対象のエージェント系とは別に構成されている。その理由は、開発におけるメッセージの混入による開発対象のエージェント系の混乱を避けるためである。

Guardant は開発対象のエージェントごとに配置され、一つの開発対象エージェントにつき (開発者, 開発エージェント,Guardant) の組が開発の単位となる。

Guardant の機能を以下に挙げる。

- 他の Guardant との通信によるエージェントの仕様の決定
- エージェントの仕様にもとづくエージェントプログラミングの支援
- 虚像化と呼ぶエージェントの動作シミュレーション

Fig 2に Guardant の構成を示す。 Guardant は GuardNet を構成するエージェントであると同時に、開発対象のエージェント系のエージェントでもある。

GuardNet ではエージェントは何らかの知識をメッセージ交換を通して提供するものであるので、エージェントの仕様を交換可能なメッセージ (メッセージ仕様) の集合と同一視する。

仕様交換マネージャは GuardNet のエージェント系を用いてエージェント開発者同士の協調によるメッセージ仕様の決定を支援し、メッセージ仕様を仕様データベースに登録する。外部および開発対象のエージェントのシミュレータは、それぞれ外部および開発対象の仕様データベースをもとに開発対象のエージェント系に対し虚像化を実現する。

3 Guardant によるメッセージ仕様の決定

一般にエージェントは複数のエージェントとさまざまなメッセージを交換する。よってエージェントの開発においてどのエージェントとどのようなメッセージを交換するのか (メッセージ仕様) を捉えることは必須である。あるエージェントのメッセージ仕様の決定において他エージェントの開発者との議論が必要な場合、メッセージの仕様ごとにそれぞれの議論を捉える必要がある。 GuardNet はエージェントの仕様とその決定の議論の過程を管理することによって、エージェント開発者のメッセージ仕様の決定を支援する。

3.1 メッセージ仕様

GuardNet におけるメッセージ仕様は以下の二つの要素からなる。

メッセージクラス: 受信可能なメッセージの定数を抽象化したもの

メッセージインスタンス: メッセージクラスの具体化例

メッセージの通信プロトコルとして KQML[1]、知識表現言語として KIF[2] を用いた場合のメッセージ仕様の例を Fig 3に挙げる。

この仕様を要求されたエージェントはメッセージクラスの $name に寺の名前が具体的に代入されたメッセージの内容を、処理することができなければならない。また、このメッセージ仕様は質問メッセージ (ask-if) なので、メッセージインスタンスにもとづいて具体化して返答しなければならない。つまり、例えば Fig 4の上の質問に対して下の返答をしなければならない。また、 $name に todaiji や saidaiji 以外の寺の名前であっても適切な返答ができることが期待される。

3.2 メッセージ仕様決定の過程

メッセージ仕様のメッセージクラスを決定するために以下の過程を用いる。

開発者 U がエージェント A の Guardant G を用いて他のエージェント開発者 U' (エージェント:A',Guardant:G') にメッセージの仕様を要求している場面を考える。そのときの GuardNet における仕様決定の流れは以下のようになる。

```
仕様 1(寺の電話番号)
 コメント:
   電話番号は市内局番から
 メッセージクラス:
   (ask-if :content  (and (temple ?t $name)
                          (tel-number ?t ?tel))
           :language kif)
 メッセージインスタンス:
   ($name = todaiji)  => (?t = t1, ?tel = "12-3456")
   ($name = saidaiji) => (?t = t2, ?tel = "23-4567")

仕様 2(寺の拝観時間の情報)
 コメント:
   $name は寺の名前
 メッセージクラス:
   (ask-if :content  (and (temple ?t $name)
                          (open-time ?t ?o-time))
           :language kif)
 メッセージインスタンス:
   ($name = todaiji)  => (?t = t1, ?o-time = "AM:9:00")
   ($name = saidaiji) => (?t = t2, ?o-time = "AM:9:30")
```

Fig 3: メッセージ仕様の例

```
質問メッセージ:
(ask-if :content (and (temple ?t todaiji)
                      (tel-number ?t ?tel))
        :language kif)

返答メッセージ:
(reply  :content (and (temple t1 todaiji)
                      (tel-numver t1 ``12-3456''))
        :language kif)
```

Fig 4: 質問・返答の例

1. U が要求するメッセージ仕様の概要をコメントとして G に登録
2. メッセージクラスの取り決め
   (a) U から G へ要求するメッセージクラスを入力
   (b) G から G' へ要求したメッセージクラスの伝達
   (c) G' から U' へ要求したメッセージクラスの提示
   (d) U' から G' へ要求したメッセージクラスに対する返答 (受諾 or 反論) を入力
   (e) G' から G へ返答の伝達
   (f) G から U へ返答の提示
3. G, G' の仕様データベースに要求中または決定済の仕様を記録

ここで、(a) から (f) の過程は会話の状態遷移モデル [6] に基づいて Guardant 内の仕様交換マネージャを用いて実行され、双方が受諾するまで繰り返される。またこの過程中, 要求中あるいは決定済の仕様は G,G' の仕様データベースに随時記録される。状態遷移は登録されたメッセージ仕様ごとに用意されるので、ユーザーは複数のメッセージクラス決定の過程を並行して行うことができる。

外部エージェントに要求したメッセージクラスは外部エージェントの仕様データベースに蓄えられる。逆に外部から仕様要求された場合、その要求されたメッセージクラスは開発対象エージェントのデータベースに置かれる。

```
(defagentkc foo
  :mbus-port 2392
  :message-handler 'foo-handler)
(setq foo-handler
  '((((:performative . (?or ask-if))
      (:kif-content . (and (temple ?t $name)
                           (tel-number ?t ?tel)))
      (:language . kif)
     ) . do-foo-telnumber)
    (((:performative . (?or ask-if))
      (:kif-content . (and (temple ?t $name)
                           (open-time ?t ?o-time)))
      (:language . kif)
     ) . do-foo-open-time)))
(defun do-foo-telnumber (env)
 "Title: 寺の電話番号
  Comment: 電話番号は市内局番から "
  (let ((binding *kif-binding*))
  ))
(defun do-foo-open-time (env)
 "Title: 寺の拝観時間の情報
  Comment: $name は寺の名前 "
  (let ((binding *kif-binding*))
  ))
```

Fig 5: Guardant によって生成されたメッセージ受信部分

3.3 決定されたメッセージ仕様の使用

上記の過程でメッセージクラスが決定されたメッセージ仕様は他の全てのエージェントが使用することができる。

あるメッセージ仕様を使用すると宣言することは、そのメッセージ仕様のメッセージクラスにしたがってその仕様が決定されたエージェントにメッセージを送信する可能性があることを意味する。

メッセージ仕様を使用すると宣言したエージェントはそのメッセージ仕様にメッセージインスタンスを追加することが許される。

4 Guardant によるエージェントコーディングの支援

GuardNet はメッセージ仕様を用いてエージェントのコーディングを支援する。我々は GuardNet によって支援されるエージェントプログラミング言語として、 Common Lisp をベースにした簡易言語 KC-Basic を開発した。

KC-Basic はエージェントのメッセージ受信部分を以下のように記述する。

1. エージェント登録部
2. メッセージクラスと対応する関数の登録部
3. 受け取ったメッセージに応じて実行される関数本体の記述

このようにして作られたエージェントは 1. で指定される名前を持つ。このエージェントに外部からやってきたメッセージの内容が 2. に書かれたメッセージクラスをみたすなら対応する 3. に書かれた関数本体が呼び出される。

Guardant は保持している開発対象エージェントのメッセージ仕様データベースから KC-Basic のエージェントプログラムのメッセージ受信部分を自動生成する事ができる。 Fig 3のメッセージ仕様から Guardant が作ったエージェントプログラムのメッセージ受信部分を Fig 5に挙げる。エージェント

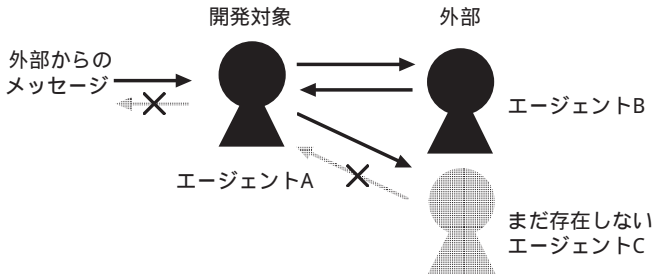

Fig 6: エージェントの依存関係

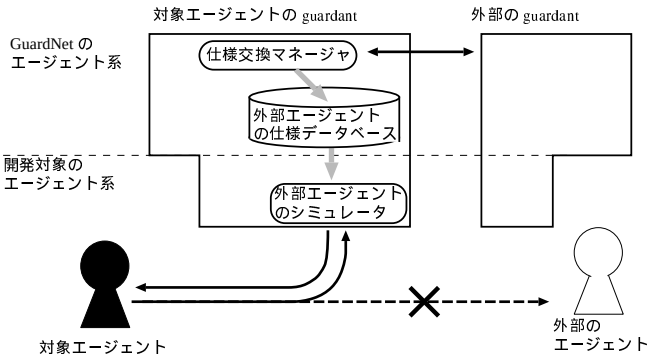

Fig 7: Guardant による外部エージェントの虚像化

開発者は関数 do-foo-telnumber や do-foo-open-time の本体にそれぞれ対応する処理部分 (例えば返事を返す) を記述する事によってエージェントを完成させることができる。

5 エージェントの虚像化

エージェントの虚像化とは、存在しないあるいは不完全なエージェントを他のエージェントがあたかもそのエージェントであるかのように振舞わせることである。この方法を用いることで存在しないあるいは不完全なエージェントを含むエージェント系の実行シミュレーションが可能になる。ここでは 3 節で述べたメッセージ交換で得られる仕様をもとに虚像化を実現する。

我々は次の二つのタイプの虚像化を行う。

外部エージェントの虚像化 開発されたエージェントを単独に動作チェックする目的で、そのエージェントが依存する外部エージェントを虚像化する。

開発対象エージェントの虚像化 トラブルによるエージェントの欠損に対する応急処置やエージェント系全体の動作チェックを行う目的で、動作していない開発対象のエージェントを虚像化する。

5.1 外部エージェントの虚像化

開発対象のエージェントが他のエージェントの知識に依存している場合、 Fig 6のように開発の済んでいないエージェントがあると開発対象のエージェントの動作チェックができない。依存関係のある開発の済んでいないエージェントを虚像化により実行シミュレーションすることによって開発対象のエージェントの動作チェックを可能にする。

対象エージェントからみた外部のエージェントのメッセージ行為は次の四つに分類される。

質問 外部へ質問メッセージを送り返答が返ってくるのを期待する。

返答 外部からの質問メッセージに対し返答メッセージを送り返す。

情報発信 メッセージ送信によって外部に対象エージェントの情報を伝える。

情報受信 外部からのメッセージによって情報を受け取る。

エージェントの実行シミュレータはこれら四つのメッセージ行為を処理しなければならない。特に、返答は他の三つと違い、質問に対応する返答メッセージを送らねばならず、実行シミュレーションも複雑である。以下では、返答のシミュレーションについて述べる。

エージェント開発者自身とメッセージ仕様を使用する他のエージェントとから、メッセージインスタンスを収集して、メッセージ仕様データベースに追加する。このようにして得られたメッセージインスタンスをもとに、エージェントの実行シミュレーションを行う。

返答の場合、メッセージインスタンスから質問に対する応答を検索してシミュレーションを行う (Fig 7参照)。

メッセージフローの制御

虚像化を行うためにはエージェントの実行シミュレータとメッセージが交換ができるようにメッセージフローを制御する必要がある。

外部エージェントが虚像化されている場合、開発対象エージェントからその外部エージェントへ送られるメッセージの全ては対応する Guardant の外部エージェントシミュレータによって受信され、実際の外部エージェントが受け取ることはない。外部エージェントシミュレータからの返答には、差出人 (KQML での sender) を虚像化されている外部エージェントとしたメッセージを用いることで、実際の外部から出されたメッセージを装うことができる。

連邦アーキテクチャ [5] を採用しているエージェント系では、全てのメッセージは一旦 facilitator と呼ばれる特殊なエージェントに送られる [3]。そこで、このようなメッセージフローの制御は facilitator を用いて実現される。

5.2 開発対象エージェントの虚像化

開発対象のエージェントが欠けているとエージェント系全体の実行が不可能となる。開発の完了していない開発対象エージェントを虚像化により実行シミュレーションすることによってエージェント系全体の実行を可能にする。

外部エージェントの虚像化と同様に実行シミュレーションしなければならないメッセージ行為は、質問・返答・情報発信・情報受信の四つが挙げらる。シミュレーションの方法も同様に、 Guardant の持つメッセージ仕様データベースの情報をもとに行われる。

Table 1: 実装された Guardant

	コミュニケーション相手	機能
Guardant	GuardNet のエージェント系, 開発対象のエージェント系	エージェント仕様の管理・虚像化機能。
GuardantUI	GuardNet のエージェント系, ユーザー (人間)	Guardant のユーザーインタフェス

Fig 8: 旅行者エージェントの開発者が Guardant を用いて仕様交換している様子

メッセージフローの制御

開発対象エージェントが虚像化されている場合、開発対象エージェントへ送られる全てのメッセージは、その Guardant が持つ開発対象エージェントシミュレータによって受信される。もし開発対象エージェントが存在していてもそれが受け取ることはない。開発対象エージェントシミュレータからのメッセージは、外部エージェントの虚像化の時と同様に差出人の書き換えによって開発対象エージェントからのメッセージであるかのように装ってメッセージが送られる。

6 知識コミュニティにおける実装

我々は GuardNet を知識コミュニティにおいて実装した。

Guardant は実際には 2 つのエージェントから構成される (Table 1)。

Guardant は GNU Common Lisp を用いて実現されている。 Guardant はユーザーが開発環境を使用していない時でも動き続け、他の Guardant からのメッセージなどに

Fig 9: 東大寺エージェントの開発者が Guardant を用いて仕様交換している様子

備える。メッセージ仕様の状態やデータなどはすべてこの Guardant が保持し、 GuardantUI が持つことはない。

GuardantUI は Emacs Lisp (ver 19) によって実現されている。ユーザーは好きな時にいつでも GuardantUI を立ち上げ・終了が可能である。

システムの実行例

Fig 8は、旅行者エージェントの開発者が Guardant を用いて東大寺エージェントの開発者に拝観時間についての仕様を要求している様子である。上のウィンドウには現在登録されているメッセージ仕様のリストが表示されている。その中の寺の拝観時間の情報に関する仕様が選択 (ハイライトの部分) されており、そのメッセージ仕様についての仕様交換の内容が、下のウィンドウに表示されている。ここでは、旅行者エージェントの開発者が東大寺エージェントの開発者に、 Fig 3のメッセージ仕様を要求 (request) している。

要求した内容は東大寺エージェントの Guardant のほうに送られる。

Fig 9は、東大寺のエージェント開発者の Guardant がその要求を受け取り、それに対して逆提案 (counter) している様子である。上のウィンドウには現在登録されているメッセージ仕様のリストが表示されている。中央のウィンドウに受け取った要求の内容が表示されている。東大寺エージェントの開発者が入力した逆提案の内容が表示されている。

Fig 10は、旅行者エージェントの開発者が外部のエージェ

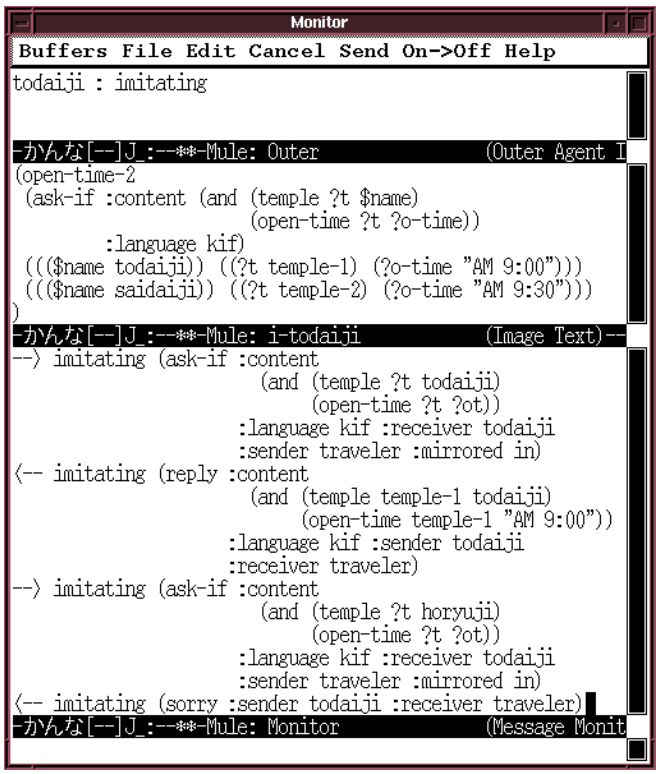

Fig 10: 旅行者エージェントの開発者が Guardant を用いて外部エージェントの虚像化をしている様子

ント (ここでは東大寺エージェントのみ) を虚像化して、開発された旅行者エージェントとのメッセージ交換を行っているところである。上のウィンドウは東大寺エージェントが現在虚像化中であることを示している。中央のウィンドウには、東大寺エージェントの Fig 3のメッセージ仕様のデータが表示されている。メッセージインスタンスとして東大寺と西大寺の拝観時間が与えられている。下のウィンドウは虚像化による東大寺エージェントの実行シミュレーションの様子である。旅行者エージェントから東大寺エージェントへ出されたメッセージを、東大寺エージェントに代わって旅行者エージェントの Guardant が受け取り、返答を行っている。

7 結論と展望

本研究ではエージェントの協同開発を支援するマルチエージェント系 GuardNet について述べた。本システムを用いるとエージェント開発者同士が離れていてもエージェント開発が可能になり、エージェントのコーディング・デバッグも効率良く行える。

参考文献

[1] T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, J. McGuire, P. Pelavin, S. Shapiro, and C. Beck. Specification of the KQML agent-communication language. Technical Report EIT TR 92-04, Enterprise Integration Technologies, 1992. (Updated July 1993).

[2] Michael R. Genesereth and Richard Fikes. Knowledge Interchange Format version 3.0 reference manual. Technical Report Logic-90-4, Computer Science Department, Stanford University, 1990.

[3] 飯野健二, 武田英明, 西田豊明. 知識コミュニティにおけるメッセージ仲介機構. 人工知能学会全国大会 (第 8 回) 論文集, pp. 287–290, 1994.

[4] 西田豊明. 知識コミュニティ. 人工知能学会全国大会 (第 8 回) 論文集, pp. 77–80, 1994.

[5] R. S. Patil, R. E. Fikes, P. F. Patel-Schneider, D. McKay, T. Finin, T. R. Gruber, and R. Neches. The DARPA knowledge sharing effort: Progress report. In Charles Rich, Bernhard Nebel, and William Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*. Morgan Kaufmann, 1992.

[6] Terry Winograd. A language / action perspective on the design of cooperative work. In Irene Greif, editor, *Computer-Supported Cooperative Work: A Books of Reedings*. Morgan Kaufmann, 1988.

著者略歴

鷹合 基行
1993 年 3 月 大阪府立大学工学部数理工学科卒業。
1995 年 3 月 奈良先端科学技術大学院大学情報科学研究科博士前期課程修了。
現在 同博士後期課程在学中

武田 英明
1986 年 3 月 東京大学 工学部卒業。
1988 年 3 月 東京大学大学院工学系研究科修士課程修了。
1991 年 3 月 東京大学大学院工学系研究科博士課程修了 (吉川・冨山研究室)。
1991 年 4 月 財団法人日本システム開発研究所嘱託研究員。
1992 年 2 月 ノルウエー工科大学 (Norwegian Institute of Technologoy, Norges Tekniske Hoegskole, NTH) postdoctoral fellow。
1992 年 7 月 ノルウエー王立科学技術研究会議 (Royal Norwegian Council for Scientific and Industrial Research, Norges Teknisk-Naturvitenskapelige Forskningsraad, NTNF) research fellow (至 1993 年 9 月)。
1993 年 4 月 奈良先端科学技術大学院大学 情報科学研究科助手。
1995 年 4 月 同助教授 (知能情報処理学講座)。

西田 豊明
1977 年 京都大学工学部情報工学科卒業。
1979 年 同大学院修士課程修了。
1980 年 同大学博士課程退学、同年より京都大学工学部助手。
1988 年 助教授。
1993 年 奈良先端科学技術大学院大学情報科学研究科教授、現在に至る。
1984 年 1 年間 Yale 大学客員研究員。
1995 年 科学技術庁金属材料研究所技術研究所客員研究官。
京都大学工学博士。知識の共有と再利用、知識メディア、定性推論の研究に従事。
著書：自然言語処理入門 (オーム社)、定性推論の諸相 (朝倉書店) 等。