自律 Agent-Flow 型処理を実現する開発環境構築への提案

水谷晃三† 小林俊裕† 山口大輔† 永井正武‡

†帝京大学大学院理工学研究科 ‡帝京大学理工学部

1. はじめに

ソフトウェア品質と生産性向上のため，筆者らは視覚的開発手法と環境構築方法を研究している。当初のオブジェクトフロー型では，並列処理プログラムを GUI によって生成する開発手法[1]と，数式から並列処理型オブジェクトフローを直接自動生成する手法[2]を提案している。オブジェクトフロー型につづいて，Agent 技術を適用した Agent-Flow 型[3]を提案し，Agent の自律性によって自動的にプログラムを生成する研究を行っている。

本稿では Agent-Flow 型処理を実現する開発システムを提案する。入出力対応情報を利用する自律的再利用手法を提案し，プログラム自動生成方法としての利点について評価し考察する。

2. 基礎理論

2.1 Agent-Flow 型処理と Agent-Factory[3]構造

本提案は，固有の基本演算処理を持つ Agent を複数種類定義し，Worker と名づける。各 Worker には次に示す基本動作を定義する。

①外部からの入力メッセージを受け Worker 自身の固有処理を行う。

②あらかじめ指定された別の Worker に向け処理結果を出力する。

入出力メッセージは ACL(Agent Communication Language)によって行われる。Worker は互いに独立し自身の処理も独立して行われるため，並列処理が行われる。このような複雑な処理を行う Worker の連携を Agent-Flow 型処理と呼ぶ。Agent-Flow 型処理では，ソースコードやネットリストを同時に出力することが可能である。

Agent-Flow の管理 Agent として，Manager を定義する。Manager は配下の Worker を管理するだけでなく，他の Agent-Flow において Worker として機能する。これを Agent-Flow の再利用という。複数の Manager を管理する BaseManager と外部とのインタフェースを担当する Main を図 1 のように定義し，このように体系化された MAS(Multi-Agent System)型を Agent-Factory として形成する。

2.2 Agent-Factory 制御手法

Agent-Factory を制御するために，XML ベースの EIF(Execute Information File)を定義する。Agent-Factory に対する依頼，問合せなどの専用タグを定義し，Main を通して Agent-Flow の定義や自律的実行を行う。

2.3 入出力対応情報による自律的再利用方法

Agent-Factory に存在する複数の Agent-Flow から，要求仕様だけを満たすものを自律的に再利用する手法として，入出力対応情報を用いる方法を提案する。本手法では，

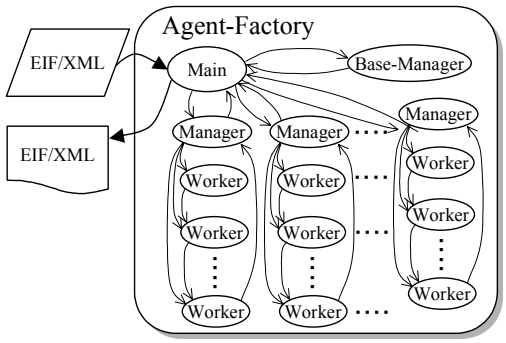

図1 Agent-Factory概念図

契約ネットプロトコル[4]を利用することによって，ユーザ要求仕様と合致する Agent-Flow を自律的に選択する手法である。プログラムの単体テスト仕様書の利用を視野に入れ，近い将来にはさらに自然言語が含まれる仕様書を，入出力対応情報として使用可能とする。

本提案手法は以下の手順で行われる。

①テストケースとして入出力対応情報を含む EIF/XML を，Agent-Factory 外部より Main が受信。

②受信データを BaseManager に転送し，入出力対応情報をマルチキャスト通信方式で Manager に送信。

③Manager は受信した入出力対応情報を自身の Agent-Flow に代入し，出力結果の適用度を算出。

④算出結果を BaseManager に送信し，最も適用度の高い Agent-Flow を BaseManager が自律的に選択。

手順③における適用度の算出は，入出力対応表に対する合致項目数で判断する。全項目が合致する場合は 1 を BaseManager に返す。適合する Agent-Flow が複数見つかった場合は，ユーザにダイアログを表示して選択を依頼する。入出力対応情報は新規 Agent-Flow の定義時に埋め込むことができるため，Agent-Flow の自律的な再利用が可能となる。

3. Agent-Flow 型開発システム

3.1 開発システム概要

EIF/XML を用いることにより，Agent-Factory を用いた Agent-Flow の構築と実行が可能となる。しかし，Agent-Flow 型開発の利点を活かすためには，GUI による視覚的な開発システムが必須である。そこで，2.3 で述べた自律的再利用に対応した Agent-Flow 型開発システムを構築する。

Agent-Flow 型開発システムは図 2 に示すように，基本的には①Flow 検索，②Flow ツリー，③Flow 編集の 3 つのペインからなる JAVA アプリケーションである。ソフトウェアなどの開発者は，Flow 編集ペイン上に端子と Worker を配置し，コネクタにより接続する。操作はマウスだけで行うことが可能であり，感覚的に Agent-Flow を定義することができる。

編集エリアに配置する Worker の検索は，検索ペイン内のテキストボックスにキーワードを入力して行う。この

Proposal for the Development System of Autonomous Agent-Flow Processing

†Science and Engineering, Graduate School of TEIKYO University

‡School of Science and Engineering, TEIKYO University

場合，各 Agent-Flow の付加情報を検索する。詳細検索機能では，端子の付加情報などを組み合わせて検索することができる。

3.2 動的処理 Worker と自律的再利用機構

自律的再利用への対応として，図 2 の Flow ツリーペインに動的処理 Worker の選択肢を用意する。

動的環境選択 Worker は，従来どおり Worker 名を直接指定し，Agent-Flow 実行時に最適な動作環境を自律選択する。数式入力選択 Worker は，オブジェクトフロー型処理順序発見法[2]を適用して既存のフローを検索し，存在しない場合は自律的に生成する。

入出力対応情報による自律的再利用は，テストケース入力選択 Woker を使用する。他の Worker と同じように編集ペインに配置し，入出力対応情報を CSV 形式で読込み割当てる。新規フロー定義時に，割当てられた情報に基づいて自律的再利用が行われる。

4. 実験・評価

入出力対応情報による自律的再利用の具体例として，組合せ論理演算の Agent-Flow 生成を行う。定義済み Worker としてあらかじめ NAND と NOR の 2 項演算処理を用意し，これらを組み合せた等価論理式処理フローを表 1 に示すように定義する。次に，テストケース入力選択 Worker を含む Agent-Flow を定義し入出力対応情報をロードする。

図3 の例は，全加算回路論理式を実現するフロー定義において，OR に関する Worker が自律的に再利用される Agent-Flow 実験の一例である。TCase_Wk_00 に入出力対応情報としてリスト 1 を割当てる。Agent-Factory へ入力すると，複数 Agent-Flow の適合を知らせる専用ダイアログが表示される（図 4）。これは，NAND と NOR による OR の等価論理式フローが存在し競合するため，ユーザに選択を依頼するものである。適合 Agent-Flow が 1 つしかない場合は，自律的再利用されるため，ユーザの選択は必要ない。

5. 考察

開発現場における単体テストやテスト駆動型開発[5]では，テスト対象のコードレビューと単体テスト結果により品質を向上させる試みが行われている。本提案手法では，単体テスト仕様書に基づいた再利用対象の選択により開発工程の一部削減が可能になる。特に Agent の自律性と移動性により，ネットワーク上に分散している Agent-Flow を対象とすることができ，開発プロジェクト間でリソース共有の可能性が見込まれる。4.で実験に用いた組合せ論理演算でも，予想どおりの結果を得ることができた。しかし，小数演算や文字列など，演算対象となるデータ系列が複雑化した場合の自律的再利用精度については，なお評価を行う必要がある。

自律的再利用時には，契約ネットプロトコルの性質上複数 Agent が同時に活動状態になる。Agent-Factory を構成する各 Agent は理論上独立しているため，入出力対応情報に対する適用度の算出時間は Agent-Flow の数に関係なく一定である。しかし，Agent-Factory が実装された JAVA VM 上の現時点では，ハードウェアリソースが各 Agent に分割されるため，Agent-Flow 数の増加に対して性能の低下が見込まれる。今後の課題として，検証を行う必要がある。

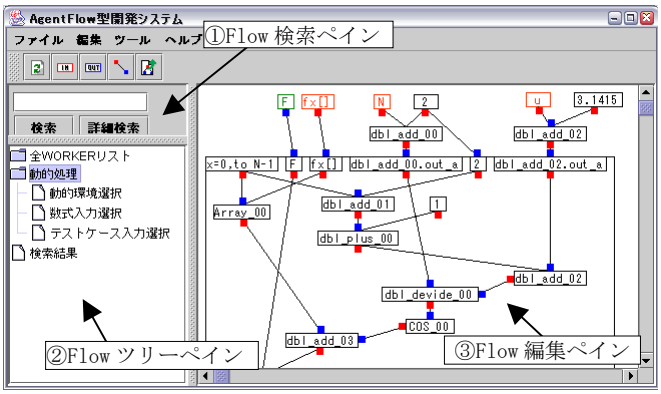

図2 Agent-Flow型開発システム

表1 等価論理式問題

構成演算子	等価演算子	式	Agent-Flow名
NAND	NOT	$F=\overline{A\cdot A}$	NAND_NOT
	OR	$F=\overline{\overline{A}\cdot\overline{B}}$	NAND_OR
	AND	$F=\overline{A\cdot B}$	NAND_AND
	XOR	$F=(A+B)\overline{A\cdot B}$	NAND_XOR
NOR	NOT	$F=\overline{A+A}$	NOR_NOT
	OR	$F=\overline{\overline{A+B}}$	NOR_OR
	AND	$F=\overline{\overline{A}+\overline{B}}$	NOR_AND
	XOR	$F=\overline{A+B}+\overline{A+B}$	NOR_XOR

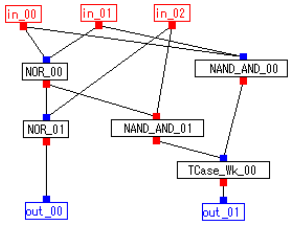

図3 テストケース入力選択を含むAgent-Flow例

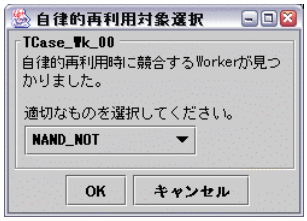

図4 再利用対象の再選択

```
<worker type="dynamic" name="TCase_Wk_00">
  <testcase>
    <terminal type="in" name="*0", value="0" />
    <terminal type="in" name="*1", value="0" />
    <terminal type="out" name="*", value="0" />
  </testcase>
  <testcase>
    <terminal type="in" name="*0", value="1" />
    <terminal type="in" name="*1", value="0" />
    <terminal type="out" name="*", value="1" />
  </testcase>
  <testcase>
    <terminal type="in" name="*0", value="0" />
    <terminal type="in" name="*1", value="1" />
    <terminal type="out" name="*", value="1" />
  </testcase>
  <testcase>
    <terminal type="in" name="*0", value="1" />
    <terminal type="in" name="*1", value="1" />
    <terminal type="out" name="*", value="1" />
  </testcase>
</worker>
```

リスト1 入出力対応情報のEIF/XML例

6. おわりに

本論文では，GUI 導入による Agent-Flow の開発システムと自律的な再利用手法を巧みに組み込み，入出力対応行列を用いる手法を提案した。今後は，継続して実用性を向上させる研究を行い，ソフトウェア開発の V モデル（図 5）において実装工程を省略できる手法として有効性を高めていきたい。

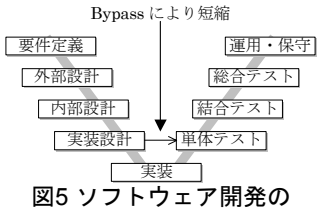

図5 ソフトウェア開発のVモデル

参考文献

[1] 平嶋史武, 池本悟, 山口大輔, 小田智大, 島田馨, 永井正武:"オブジェクトフロープログラム自動作成方法への一提案", FIT2002, B-20, 2002.

[2] 山口大輔, 池本悟, 小林俊裕, 平嶋史武, 島田馨, 永井正武:"オブジェクトフロー型処理順序発見方法への一提案", 電子情報通信学会総合大会, D-8-3, 2003

[3] 水谷晃三, 山口大輔, 小林俊裕, 伊藤貴雄, 永井正武:" オブジェクトフロー型処理順序発見方法へのマルチエージェントの適用", FIT2003, B-040, 2003

[4] Reid G. Smith:"Contract Net Protocol : High-Level Communication and Control in a Distributed Problem Solver", IEEE Transactions on Computers, Vol.C-29, No.12, pp.1104-1113, 1980.

[5] Kent Beck:" Test Driven Development: By Example ", Addison-Wesley Pub Co, 2002.