統合学習による機能モデルの自動構成に関する一考察

4E−6

小橋哲郎　山田宏之　相原恒博

愛媛大学 工学部 情報工学科

1　はじめに

我々は，オブジェクト間の機能的関係(変更に伴う相互作用の発生に本質的な関係)を表現するモデル(機能モデル)に基づき，変更時の相互作用を指摘することにより，ユーザの変更作業を支援するための研究を行っている[1]．その中で，従来ソフトウェア開発者自身が行っていた機能モデルの構成を，機械学習パラダイムを用いることにより，自動化するための方法論の確立を目指している[2]．そこで本稿では，専門家の行う変更過程の正当性の説明に基づく学習と，観察からの学習とを統合した学習により，オブジェクト間の機能的関係を獲得し，機能モデルを自動構成する手法について報告する．

2　機能モデルの自動構成法

2.1　機能モデル

機能モデルは，ノードとリンクとからなるネットワーク形式で表現される．ノードは対象ソフトウェアのクラスオブジェクトに対応し，リンクはノードに対応するオブジェクト間の機能的関係を表現する．但し，より詳細な変更支援を実現するために，複数のノードを1つのクラスに対応させることが許されている．ノードには，基本ノードと応用ノードとがあり，それぞれ対象領域で共有されるクラス・個々のソフトウェアでユーザが定義するクラスに対応する．ノード内には，変更時に有効になるリンクを検出するための情報(相互作用検出情報)と，リンクが有効になり，たどられてきたときに，相互作用に対処するための情報(相互作用対処情報)とが格納されている．なお，機能モデルに基づく変更支援の詳細は，文献[1]を参照されたい．

2.2　機能モデル構成法

本構成法では，以下の手順で機能モデルを構成する．なお，現段階では，変更内容を応用ノードに対応するクラスの追加・修正・削除に限定している．

(1) ノードの作成: 対象ソフトウェアを構成する各クラスに対応するノードを作成する．

(2) 訓練例の獲得: 専門家の行う変更過程を観察し，操作可能な述語で表現する．

(3) 説明の構成: 観察した変更過程の正当性を，OOP及び対象領域に関する知識を用いて説明する．なお，本構成法では，文献[3]で示されているアルゴリズムを用いて説明を構成する．操作性基準は，獲得された概念が，既存クラスに関する事実を示す述語または訓練例を示す述語で表現されるものとしている．

(4) 概念記述の獲得: (3)で得られた説明を，文献[3]のアルゴリズムを用いて一般化することにより，機能的関係を表現する操作可能な概念を獲得する．

(5) ノード表現への変換: (4)で得られた概念から，相互作用検出情報・相互作用対処情報を生成し，ノード表現を形成する．ノード表現は，出現回数を記録するためのカウンタを持つ．

(6) モデル構造の洗練化: (5)で形成されたノード表現が，既にノードに格納されているかどうか調べる．格納されていない場合は，新たに格納し，カウンタを1に設定する．格納されている場合は，カウンタを1増やし，この表現と同じ相互作用検出情報を持つが，同時には有効にならないノード表現を検索する．ここで，検索されたノード表現及び(5)で形成されたノード表現のカウンタが定められた値を越えていれば，このノードに必要なだけ下位ノードを作成し，それらのノード表現を格納する．

(7) 操作性基準の変更: (6)で新たなノード表現がモデルに格納された場合，ノード表現に関連する述語を，操作可能な述語の集合に加える．

(8) (2)へ戻る．

3　評価実験

本研究では，本構成法の有効性を調べるため，Sparc Station 1上のSmalltalk\Release 4を用いて実験システムを構築している．ここでは，待ち行列シミュレーションソフトウェアに対する機能モデルの構成例を示す．

3.1　実験1：ノード記述の獲得

2.2 (2)において，実験システムが以下のような専門家の変更過程を観察したとする：”インスタンスメソッドtasksを持つクラスFerryを，クラスEventMonitorの下位クラスとして定義し，次に，クラスFerrySimulationのメソッドdefineArrivalScheduleを修正する”．

```
defClsAsSubCls('Truck' 'EventMonitor').
deflMtd('Truck' 'tasks').
addMsg('schedule:at:' 'defineArrivalSchedule'
  'FerrySimulation').
include('schedule:at:' 'Truck').
```

この変更過程の正当性を，次に示す領域知識を用いて説明する(3)．

A Study on Automatic Construction of a Functional Model by Integrated Learning.

Tetsuo KOBASHI, Hiroyuki YAMADA, Tsunehiro AIBARA

Faculty of Engineering, Ehime University

```
functionalRelation(x y) ←
    hasIMtd(x 'tasks') & appCls(x) &
    subCls(x 'SimulationObject') &
    subCls(y 'Simulation') & appCls(y) &
    hasIMtd(y 'defineArrivalSchedule') &
    sendMsg('new' y x).
subCls('FerrySimulation' 'Simulation').
subCls('EventMonitor' 'SimulationObject').
appCls('FerrySimulation').
hasIMtd('FerrySimulation' 'defineArrivalSchedule').
hasIMtd(a a1) ← deflMtd(a a1).
subCls(c t) ← defClsAsSubCls(c t).
subCls(b b2) ← subCls(b b1) & subCls(b1 b2).
sendMsg('new' d e) ← aKindOf(f 'scheduling') &
    include(f e) & addMsg(f 'defineArrivalSchedule' d).
aKindOf('schedule:at:' 'scheduling').
...
```

次に，構成された説明を一般化し，次のような記述を得る (4)．

```
deflMtd(x 'tasks') & defClsAsSubCls(x b1) &
subCls(b1 'SimulationObject') &
subCls(y 'Simulation') & appCls(y) &
hasIMtd(y 'defineArrivalSchedule') &
aKindOf(f 'scheduling') & include(f x) &
addMsg(f 'defineArrivalSchedule' y).
```

さらに，この記述からノード表現を形成する (5)．

```
NodeName: SimulationObject
Ir: (and (defClsAsSubCls) (deflMtd 'tasks'))
...
NodeName: Simulation
Ir: (addMsg('scheduling' 'defineArrivalSchedule')
...) ...
```

これは，インスタンスメソッド tasks を持つクラスを，クラス SimulationObject の下位クラスとして定義すれば，クラス Simulation の下位クラスのインスタンスメソッド defineArrivalSchedule に，定義するクラスをスケジューリングするメソッドを追加しなければならないことを示している．

3.2 実験 2：ノード階層の洗練化

2.2 (6) では，ノード表現形式で格納された知識の構造を洗練する．本作成例の場合，初期設定では，機能モデルのノード階層のうち，リソースの下位クラスは非同期型リソース及び同期型リソースである (図 1.(a))．ここで，学習が繰り返されると，非同期型リソースを示すノード内に，リソース生成・獲得に関する表現と，リソース生成・獲得・解放に関する表現の 2 種類のノード表現が格納される．これらのノード表現は，同じ相互作用検出情報を持つが，同時には有効にならない．そこで，これらの表現がそれぞれ 3 つ以上格納された段階で，このノードの下位ノードとして，2 つのノード (消費型リソース・非消費型リソース) が作成され，これらの表現が格納される (図 1.(b))．この時，新しいノード名はユーザにより入力される．

4 検討・課題

本構成法では，領域知識を使用することにより，1 つの正事例から，操作可能な機能的関係を表現する知識を獲得することができる．さらに，獲得された知識をネットワーク形式で格納することにより，学習の有効性問題の発生を避けることが可能であると思われる．

また，ノード階層をクラス階層 (文法的構成を表現) に一致するものから，ソフトウェアの意味的構成を表現するものへと，動的に洗練することにより，より適切な助言をユーザに与えることができる．

今後の課題として，まず，多くの例を用いた領域知識の妥当性の評価が挙げられる．

現段階では，OOP 共通の知識と，対象領域である待ち行列シミュレーションの知識とが混在している．そこで，これらの知識を分離し，OOP 共通の知識を抽出することにより，本手法の適用範囲を広げる必要がある．

本構成法では，領域知識は陽に外部から与えられるものとしているため，機能モデルの構成を完全に自動化するには至っていない．そこで現在，オブジェクト間のメッセージ通信列の解析結果を用いた類似に基づく学習により，必要な知識を獲得する手法を考察中である[2]．

5 おわりに

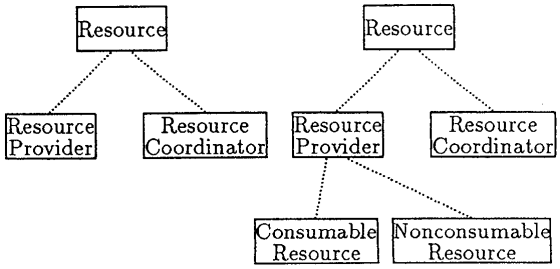
本稿では，学習パラダイムを用いて機能モデルを自動構成する手法について報告した．今後は，4 で列挙した課題を克服していく予定である．

参考文献

[1] 小橋，山田，相原: "機能モデルに基づくソフトウェアの変更支援", 情報処理学会オブジェクト指向ソフトウェア技術シンポジウム論文集, pp.13-22, 1991.

[2] T. Kobashi, H. Yamada and T. Aibara: "A Constructing Method of a Functional Model by Learning from Examples", JCSE'92, pp.55-62, 1992.

[3] DeJong, G. and Mooney, R.: "Explanation-Based Learning: An Alternative View", *Machine Learning*, Vol.1, No.2, pp.145-176, 1986.



(a) 学習前　　(b) 学習後

図 1: ノード階層の洗練化