ソフトウェアアーキテクチャ記述言語変換系の設計と実装

3K−5

金丸恵祐* 原田賢一†
慶應義塾大学大学院 理工学研究科‡
email: kana@hara.cs.keio.ac.jp

1 はじめに

ソフトウェアシステムの大規模化と複雑化に伴い、システム設計者にとっては、計算アルゴリズムやデータ構造の選択よりもむしろ、システムの全体構造の設計がより重要な問題となる。これらはソフトウェアアーキテクチャレベルの設計と呼ばれている [1]。

こうした流れを受けて、ソフトウェアアーキテクチャを形式的に表現する必要性が高まり、様々なソフトウェアアーキテクチャ記述言語 (SADL) が開発されてきた。

2 研究の目的と意義

Wright[2] は、CSP に基づいた SADL であり、高い汎用性とコネクタ表現力をもつ。そのため、ソフトウェアアーキテクチャを記述するのに特に有効である。

しかしながら、CSP 単位の静的な分析は可能であるが、シミュレーションや視覚化などといった点では弱い。また、抽象度の高い言語でもあるので、実装への結びつけが困難である。

そこで、他の SADL へと変換し、それらの点を補うことを考える。変換対象としては、振舞い記述の類似性、ツールセットの充実などの点から Rapide[3] を選択した。

Wright-Rapide 変換系の実装によって、Wright を用いて信頼性のあるシステムを設計し、Rapide によってシミュレーションや視覚化、そして実行コードの生成を行うといったシステム構築が可能となる。

3 Wright および Rapide の概要

3.1 Wright

Wright は、CMU の David Garlan らによって 1994 年から研究開発が行われている SADL である。

システム構造 Wright では、システムはコンポーネント＋ポート、コネクタ＋ロールの組によって記述される。ポートはコンポーネントの、ロールはコネクタのインタフェースをそれぞれ表現する。

CSP による振舞いの記述 コンポーネントおよびコネクタの振舞いは、すべて CSP(Communicating Sequential Processes) のプロセスモデルによって記述される。次に WRIGHT コネクタの記述例を示す。

$$\begin{aligned}\text{Glue} = \text{Client.request?x} &\rightarrow \overline{\text{Server.invoke!x}} \\ &\rightarrow \text{Server.return?y} \rightarrow \overline{\text{Client.result!y}} \rightarrow \text{Glue} \\ &\quad \Box\surd\end{aligned}$$

Wright では、CSP と異なり、発生イベント (Initial Event) と受理イベント (Observe Event) の区別を明確にしている。

3.2 Rapide

Rapide はスタンフォード大学の David Lackhum らによって 1994 年から研究開発が行われている SADL である。

システム構造 Rapide はシステムをインタフェース、コネクション、リアクションルールによって記述する。

インタフェースには、イベントインタフェースとして、外部から受理するイベントおよび外部へ向けて発生させるイベントを記述する。それらを結びつけるのがリアクションルールである。

リアクションルールによる振舞いの記述 リアクションルールは、受理したイベントに対してどのようなアクションを行うかを記述したパターンの集合である。それぞれのパターンは並列に動作し、個々の受理イベントは複数のパターンに参加することができる。

受理イベントおよび発生イベントは、Rapide の提供する Pattern Language によって記述される。

4 変換系の設計

4.1 変換の基礎的概念

CSP は基本的にシーケンシャルなイベントの処理を記述する。一方、リアクションルールは単純なイベント受理、発生の処理を行うルールが並列に動作しているので、一般に処理の順序を制御することはできない。

そこで、本研究ではリアクションルールを制御するための内部変数を導入し、これをプロセスポインタと呼ぶことにする。

*Keisuke Kanamaru
†Kenichi Harada
‡Faculty of Science and Technology, Keio University

4.2 基本パターンへの分解

CSP 記述は、リアクションルールに直接変換できるように、単純な基本パターンに分解される。主な基本パターンには、次のようなものがある。

受理イベント、発生イベントパターン リアクションルールへ直接変換できるパターンであり、最終的にすべてのパターンはどちらかの形に変換される。それぞれ、次のような変換を行うことができる。

- 発生イベントパターン
  $P = \underline{a} \rightarrow \underline{b} \cdots \triangleright \quad (pp = P) \Rightarrow a \rightarrow b \cdots$
- 受理イベントパターン
  $P = a \rightarrow \underline{b} \cdots \triangleright \quad a \text{ where } (pp = P) \Rightarrow b \cdots$

CSP 演算子 決定的選択子、非決定的選択子、シーケンス演算子、並列演算子などが含まれる。

多重ラベルつきプロセス演算子 $L_x : P$ のような多重ラベル付きプロセスに関する演算子であり、CSP 演算子と同様なものが含まれる。

次節では、これらパターンの中で、やや特殊な処理が必要となる並列オペレータについて述べる。

4.3 並列演算子の処理

並列演算子は、複数の並列に動作するプロセスを記述する時に用いられる。

プロセスポインタの多重化 本来リアクションルールは並列動作するものであり、プロセスポインタによって、疑似的にシーケンシャルな動作を表現している。

したがって、並列プロセス (マルチスレッド) を実現するためには、プロセスポインタを増やしてやればよいことになる。

イベントの同期処理 並列動作するプロセスそれぞれに同じイベントが含まれる場合は、それらイベント間で同期をとる必要がある。

イベントの同期処理は受理イベント、発生イベントに分けて行うが、ここでは受理イベントの処理についての説明をする。

次のようなプロセス P,Q を並列プロセスとする。
$P = a \rightarrow \underline{b} \cdots$, $Q = a \rightarrow \underline{c} \cdots$
それぞれ次のように変換される。
a where ($pp_1 = P$) $\Rightarrow$ b ;;
a where ($pp_2 = Q$) $\Rightarrow$ c ;;
イベントの同期をとるには、これらルールのガードを合成する。
a where ($pp_1 = P$ and $pp_2 = Q$) $\Rightarrow$ b ; c ;;

5 実装と変換試験

変換処理系は、lex と yacc を用いて実装した。この変換系の評価をとるために、いくつかのサンプルについての変換試験を行った。以下にその一部を示す。

並列プロセスを持つサーバモデル 並列演算子処理の評価をとるために、内部でプロセスが並列動作するサーバ記述を変換を行った。本処理系により変換された Rapide アーキテクチャの実行によって、並列依存性を示す POSET が生成できる。

AEGIS システム AEGIS は、艦船制御のための大規模なソフトウェアシステムである。本処理系によって、AEGIS の記述を Rapide に変換することができる。

6 関連研究

SADL 間の変換を扱った研究には、CMU の Robert Monroe らによる ACME[4] がある。現在、ACME を経由した変換系の一つとして、Wright → Rapide 変換のプロトタイプシステムがあるが、入力の言語仕様に制約が多く、現在のところ実用的ではない。

7 まとめ

本研究では、SADL である WRIGHT と Rapide 間の変換系の設計し、それに基づいて処理系を実装した。また、実装した処理系を用いて変換試験を行い、性能評価を行った。

参考文献

[1] D. Garlan and M. Shaw: "An Introduction to software architecture," Advances in Software Engineering and Knowledge Engineering, World Scientific, Vol. 1, 1993, pp.1-39.

[2] R. Allen and D. Garlan: "A Formal Basis for Architectural Connection," ACM Trans. on Software Eng. and Methodology, July 1997.

[3] D. C. Luckham, et al. : "Specification and Analysis of System Architecture Using Rapide," IEEE Trans. Software Eng., Vol. 21, No. 4, April 1995, pp.336-355.

[4] R. Monroe, et al. : "ACME: An Architecture Description Interchange Language," *Proceedings of CASCON 97*, Nov. 1997.