形式的マイクロプロトコル組立てフレームワーク

東原大記 and Xavier Défago
School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
Email: {d-higa, defago}@jaist.ac.jp

一般的なソフトウェアでは，単独のプロセスやノード上に配置されたソフトウェアコンポーネントが相互作用することで，ひとつのソフトウェアとしての振る舞いをする．分散システムでは，複数のノードやプロセス上に配置されたソフトウェアコンポーネントどうしが相互作用することにより，ひとつのソフトウェアとして振る舞うため，副作用を考慮する事が重要となる．　以上の理由により，分散システムの開発やデバッグ，テストは複雑性が高い．

マイクロプロトコルμP [2], [4] は，分散システムを基本的な機能に分割し，独立したソフトウェアコンポーネントとして開発し，開発する分散システムの仕様に適した　μP を組立てることで分散システムを開発する．　μP は，低粒度のビルディングブロックとして扱うことができるように，インタフェースと各プロトコルの仕様を定義する事で，μP の再利用性と柔軟な開発及び組立てを保証することができる．　再利用性と柔軟性を考慮されたμP では，μP の開発者とμP を用いて分散システムを構築する構築者の役割を分離することができる．　μP の開発の役割を分離するためには，各μP の独立性を考慮することも重要よなり，各μP は自己完結する事が望ましい．　以上のようなμP においては，μP の開発者は，開発するμP がどのようなコンテキスト，タイミングで利用されるかを意識することなく，μP 外部である環境をブラックボックス化する事ができる．　また，分散システムを開発するμP 構築者は，各μP の振る舞いに関する仕様とインタフェースが解って居る状態で，全てのμP の実装を意識することなく内部をブラックボックス化する事ができる．

各分散アルゴリズムの抽象化レベルは分散アルゴリズムの開発者毎に異なり，実際に利用するためにμP として開発する際には，アルゴリズムの記述方法と実装の間には大きな差が存在する場合がある．　よって，開発時にアルゴリズムの正しさを証明された分散アルゴリズムを実際のμP として開発する際には，μP のプログラムコードの正しさをモデルチェックなどの何らかの方法を用いて検証する必要がある．　また，既存のμP フレームワークでは，何らかの方法を用いて書くμP の正しさを保証できるが，複雑な分散システムを構築した際の，分散システム全体の正しさを保証することは困難である．　単一のμP に副作用が存在しないことが保証された場合でも，複数のμP を結合させた場合には，副作用が存在するかどうかは別に検証する事が重要となる．

Cactus[2]やAppia[4]は，μP　　を組立てることにより分散システムを構築するためのフレームワークである．Cactus は，柔軟なμP 開発をサポートするためのフレームワークである．μP の実装を，μP の組立てと分離することにより，μP の柔軟な開発及び組立を保証する．　一方で，複雑な分散システムを開発する場合には，副作用が原因となり，分散システムの構築者の予期しない振る舞いをする場合がある．

Appia　は，耐故障性を考慮した分散システムをグループコミュニケーションを基本として開発するツールキットである．　Appia は，μP の組立てを簡易化し，副作用が発生しないようにするために，システム内に単独のスレッドの実行のみを許可する．　Appia　は，安全な分散システムを開発することができるが，柔軟性な分散システムの開発には注目していない．

信頼性の高いμP の組立てを行うためには，各μP を形式的に記述し正確性を保証し，形式的に記述されたμP を組み立てることで，アルゴリズムの正確性を検証できる事が重要となる．　分散システムを形式的に記述する方法に (Input / Output Automata) I/O Automata [5] がある．

I. I/O Automata

I/O Automata は，分散アルゴリズムの正確性と平等性を保証するために，非同期の分散アルゴリズムを形式的に記述するモデリングフレームワークである．　我々は[3]において，I/O Automata 及び組立に関して述べた．

I/O Automata は，リアクティブなμP の入出力アクションと状態，状態遷移を形式的に定義することで，分散アルゴリズムの仕様を形式的に記述し，入力を与えることで振る舞いを確認することができる．　I/O Automata は，分散アルゴリズムの記述にイベントモデルを利用ているが，分散アルゴリズムの中にはプロセスモデルを用いて記述されている場合もある．　既存の　μP　フレームワークは，アルゴリズムの記述自体には注目していないため，プロトコルの開発者がモデルを変更し，変更した分散アルゴリズムの検証をした上でμP　として開発し，開発したμP　の正しさを検証する必要があった．　μP　を　I/O Automata　に適応させた場合には，開発者がプロセスモデルのアルゴリズムをイベントモデルとして変換・記述した場合に，開発したμP 自体がI/O Automata として記述されているため正確性を検証する事ができる．　また，I/O Automata を適応したμP　は，μP　自体が形式的に記述されているため，モデルチェックなどを用いるために容易に変換する事ができる．

I/O Automata は，2つ以上の I/O Automata を組合わせることにより新たな I/O Automata を構築[1]する事ができ，組合わせたI/O Automata の正確性と公平性を形式的に示すことができる．　組立てる全ての　I/O Automata の持つ入出力イベントを適切に確認する事で新

たなI/O Automata として表現可能である．しかし，I/O Automata の組立て全体の複雑な構造に対して形式的に検証を行う手法は無いため，安全な組立てや並行処理などを考慮した検証が重要となる．

II. μP 組立て

複雑な分散システムを I/O Automata を用いて組立てる場合には複雑性があがるため，I/O Automata の結合全体をμP のグラフとして扱う．

μP の組立ての構造のグラフは，μP をノード，ノード間 の結合をエッジとしたμP の組立てのラベル付きの多重辺有向グラフとする．μP の組立てグラフを用いることにより，組立自体が安全であるかどうかを確認する事ができる．例えば，どのμP とも繋がりのない出力がある組立グラフの場合，組立グラフを用いてシステムを実行した場合には構築者の意図しない振る舞いをする場合がある．

組立てグラフにおいて，ソースノードからシンクノードへのイベントの流れをevent waveとする．ある分散システムを構成する全てのμP が，各々1つだけの出力を保つ場合には，event wave は単一の経路となる．しかし，複数の出力をもつμP が存在する場合には，event wave の範囲は複雑な経路となる．ソースノードが複数存在し，複数の出力を持つμP が存在する場合には，並行処理(concurrency)の存在する範囲は，2つのevent wave が重なり合う部分となる．

μP の組立グラフを用いる事により，μP の組立てに対する動的な側面と静的な側面から組立ての安全性を確認することができる．動的な側面とは，前述の複数のevent wave が存在し並行処理が存在する場所では，副作用が起きる場合がある，また，組立てグラフの構造によっては，deadlockが起きる場合もある．一方で，静的な側面とは，既存のμP の組立てでは，インタフェースを基にして複数のμP の結合を行う．しかし，柔軟性のためには，高い抽象化レベルでインタフェースを定義する事が重要となり，分散システムの構築者の意図しない組み合わせで組立てることもできる．

a) Concurrency: 複数のsourceノードが組立てグラフ内に存在する場合には，組立てられたグラフを用いて実行したシステム内に並行処理が存在する．μP は，入力に対してリアクティブに反応し，入力されたイベントを1つずつ処理する事ができる．

Deadlock: 組立てグラフ内にサイクルが存在し，プロトコル内の入力イベントバッファーが有限で，入力されるイベントにプライオリティが存在しない場合には，deadlockが発生する場合がある．サイクル内の全てのプロトコルにおいて，入力イベントバッファーが飽和状態であるが，一つ前のプロトコルからの入力を必要としている場合，全てのプロトコルが入力の待ち状態になり，deadlockが起きる．Deadlockへの対応は，組立てグラフ内におけるサイクルを禁止する方法や，データベースのトランザクションのようにしてイベントウェーブを制御する方法が考えられる．

Isolation: μP の組立ては，組立てられたプロトコルの正しさは形式的に確認する事が可能であるが，実行時の副作用に関しては確認する事ができない．複数のソースノードが存在し，2 つ以上のevent wave が共通のμP を保つ場合には副作用が起きる場合がある．例えば，logical clock は送受信するメッセージの順番にタイムスタンプをつける分散アルゴリズムである．複数のevent wave の共通点にlogical clock の機能を提供するμP が存在する場合，送信したメッセージや受信したメッセージに対してタイムスタンプをつける．しかし，タイムスタンプを利用するμP が，タイムスタンプを参照する時点で他のevent waveからのメッセージによりlogical clock内の状態が既に遷移している場合があり，タイムスタンプが受信時のタイムスタンプと同じであるとは限らない．上記のような場合にも，event waveをデータベースのトランザクションのように扱い，各event waveのisolationを保つことが重要となる．

b) Compatibility: 組立てグラフの構造が，動的要素の観点から安全であるとわかった場合でも，既存のフレームワークでは，μP 間の意味的なつながりに関しては注目していない．

Type safety: Broadcastのインタフェースを持つアルゴリズムに，atomic broadcastとreliable broadcastがある．2つのアルゴリズムの実装であるμP は同一のbroadcastインタフェースを持ち，インタフェースを用いる組立てではどちらのμP も結合可能である．しかし，atomic broadcastの保証が必要な場所にreliable broadcastのμP を結合した場合には，保証が充分でないため正しい振る舞いを行わない場合がある．type safetyのためには，インタフェースの定義のみではなく，μP が提供もしくは要求する保証の定義を行うことが重要となる．

III. Conclusion

分散システムを構築する手法として，ビルディングブロックとして取り扱うことができる分散アルゴリズムを実装したプロトコルを組立てる手法がある．プロトコルを組立てて信頼性の高い分散システムを構築するためには，インタフェースの組み合わせのみでは不十分である．プロトコルを組立てる場合には，concurrencyとcompatibilityをの問題を考慮することが重要になる．

我々は，上記の2点に対して考えられる問題点をあげ，考えられる解決手法の概要に関して述べた．また，組立ての問題点の解決方法が適応可能であると考える既存のソフトウェアなどに関して述べた，たま，信頼性の高い分散システムの構築のためには，耐故障性を考慮したアルゴリズムを用いたプロトコルを用意することが重要となる．

Acknowledgment

本研究はJSPS科研費 23500060の助成を受けたものです．

References

[1] E. Amoroso and M. Merritt. Composing system integrity using I/O automata. In Proc. of ACSAC, 1994.

[2] N.T. Bhatti and R.D. Schlichting. A system for constructing configurable high-level protocols. SIGCOMM Comput. Commun. Rev., pages 138--150, 1995.

[3] D.Higashihara and X.Défago. Formal framework for micro-protocols composition. In PRDC, 2012. to appear.

[4] A. Pinto H. Miranda and L. Rodrigues. Appia: A flexible protocol kernel supporting multiple coordinated channels. In ICDCS '01, pages 707--710, 2001.

[5] Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In Proc. of the sixth annual ACM Symposium on PODC, pages 137--151, 1987.