

Enterprise JavaBeansTM サーバ仕様の形式化と検証

中島 震

NEC 情報通信メディア研究本部

玉井 哲雄

東京大学大学院 情報学環

あらまし 分散コンポーネント基盤フレームワークでは構成コンポーネント間でのイベント系列(振舞い仕様)が重要な側面であり、厳密に規定されている必要がある。本稿では実用的な分散コンポーネント基盤である Enterprise JavaBeansTM (EJB) サーバ仕様を形式化し、モデル検査ツール SPIN を用いて解析を行なうことで、EJB 仕様書の正しさを振舞い仕様の観点から確認する。SPIN による形式化と検証によって、仕様書が明記していない暗黙の状態管理が必要であること、等が判明した。また、本具体例を通して、SPIN が分散アーキテクチャ検証のためのデザイン言語として良い特徴を持つこと、ならびに試行錯誤的な検証過程を容易にする「軽い (Lightweight)」デザインカリキュレータとして有用であること、等がわかった。

キーワード コンポーネント、形式手法、振舞い仕様、モデルチェック

Formalization and Verification of the Enterprise JavaBeansTM Server Specification

Shin NAKAJIMA

C&C Media Research Laboratories
NEC Corporation

Tetsuo TAMAI

Interfaculty Initiative in Information Studies
Graduate School of The University of Tokyo

Abstract Rigorous description of protocols (a sequence of events) between components is mandatory for specifications of distributed component server frameworks. This paper reports an experience in formalizing and verifying behavioural aspects of the Enterprise JavaBeansTM server specification with the SPIN model-checker. The formal model helps uncover server states necessary for a proper management of components. The present case study shows that SPIN has useful features for analyzing behaviour of distributed software architecture, and that SPIN can be used as a lightweight design calculator in the verification process.

Keywords Component, Formal Methods, Behavioural Specifications, Model-Checking

1 はじめに

E-Commerce等の分散アプリケーションシステム構築技術としてコンポーネント技術が注目されている。コンポーネントはシステムの構成要素であって、外部とのやりとりを行なうための明確なインタフェースを持つ再利用単位である[14]。コンポーネントを作動させるためには特定の情報交換プロトコルやポリシーを持つコンポーネント基盤が必要である。代表的なコンポーネント基盤として、COM、Enterprise JavaBeans (EJB)、OMG CORBAのコンポーネントモデル、等がある。共通の仕様を満たすコンポーネントが多くベンダから提供されると、コンポーネントを組み合わせるだけで所望のシステムを構築できる。そのためには、コンポーネントやコンポーネント基盤のモデルを厳密に曖昧さなく規定する必要がある[7]。

COMやEJBの仕様書は、従来技法で書かれたインフォーマルなものであり厳密性に欠ける。実際、COMコンポーネントの組み合わせモデルを厳密に形式化することで、同モデルに曖昧さがあることを指摘した研究がある[13]。また、コンポーネント基盤は基盤内部の処理単位や基盤上で作動するコンポーネントを構成要素とする分散システムと見做すことができる。構成要素間の情報交換プロトコル(イベント系列)が整合性を持ち、そのプロトコルが厳密に曖昧さなく規定されていなければならない。すなわち、イベント系列に着目した振舞い検証が重要となる。有限状態遷移モデルに対するモデルチェッキングに基づく検証手法が有効であることがわかっていて[2][8][12]。

本稿では、実用化されている分散コンポーネント基盤であるEnterprise JavaBeansサーバ仕様[1]を形式化し、モデル検査ツールSPIN[4]を用いて振舞い検証することで、EJBサーバ仕様の正しさを確認する研究について報告する。特に、SPINの従来からの適用対象である通信プロトコル[4]と異なり、分散アーキテクチャを対象とする場合のモデル化の工夫ならびに検証技法の役割を考察する。

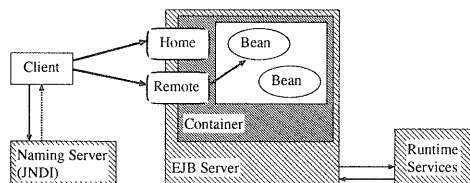


図 1: EJB サーバ

2 EJB サーバ

2.1 EJB の概要

EJBはコンポーネントベースの分散システムのアーキテクチャであり、トランザクション処理を含むエンタープライズ・アプリケーションを想定する。コンポーネントに相当するBeansは比較的大きな粒度の実体である。図1に示すように、BeanはEJBサーバの管理下で実行され、特に、Containerが実行管理を司る。Clientは、2つのインタフェース(HomeとRemote)を通してBeanにアクセスする。

EJB1.1仕様[1]では、Entity BeansとSession Beansと呼ぶ異なる性格を持つ2種類のBeansを規定している。Entity Beansはビジネスデータを表現するコンポーネントであって、実体をデータベースに格納する等の永続性を持つ。一方、Session BeansはClientの代理となってEJBサーバ内部で作動する実体であり、複数のEntity Beansにアクセスする一連のビジネスロジックを担う。以下、本稿では、Entity Beansを対象として具体的に議論を進める。

2.2 振舞い仕様

EJBサーバの振舞い仕様を仕様書[1]の記述内容を引用して説明する。「振舞い仕様」とは、メソッド起動をイベントとするイベント系列として表現した外部仕様のことである。

図2にEntity Beansのライフサイクル(p.102, Figure 23)を示した。概念的には3つの状態からなることを示し、状態遷移を起こすメソッドを明示する。たとえば、Clientが起動するビジネスメソッド(一般のメソッド)はready状態の

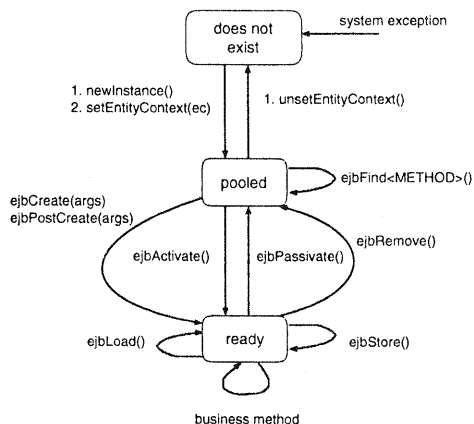


図 2: Entity Beans のライフサイクル

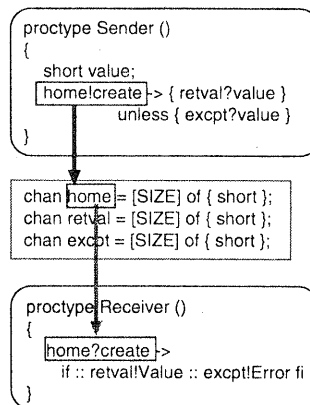


図 4: チャネル通信プロセス

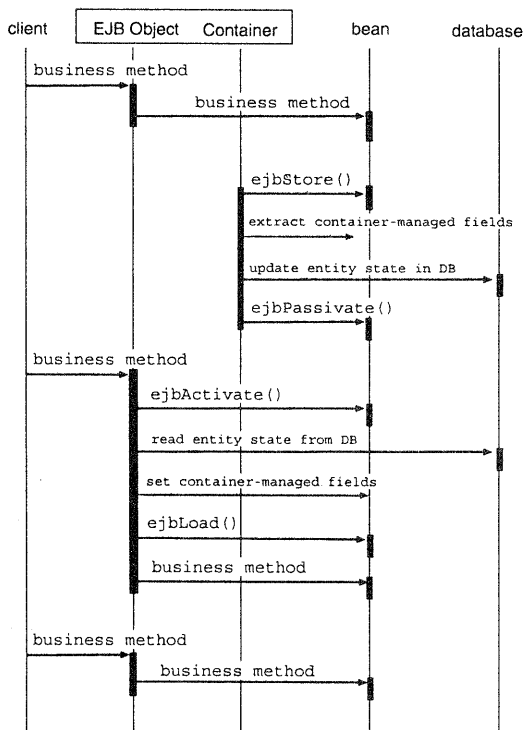


図 3: イベントトレースの例

時にのみ実行することができる。

また、仕様書では、典型的な振舞いをイベントトレースで説明する。図 3に Entity Beans のオブジェクト相互作用ダイアグラム例 (p.139, Figure 29 の一部を省略) を示した。ここで、EJB Object は図 1の Remote インタフェースに相当する仮想的な実体、Container は対応する実体である。Client が BusinessMethod を起動する場合、状況に応じて種々のイベントが発生することを示す。Container の自律的な動作によって Beans に ejbPassivate 等の処理を行なうと、Beans はready からpooled の状態に遷移する (図 2参照)。この後に、Client がビジネスメソッドを起動すると、Container はビジネスメソッドを委譲する前に、ejbActivate を発生して Beans の状態をready に復帰させ、さらに ejbLoad を起動する。

本稿の目的は、EJB サーバの構成要素 (図 1) 間で交換するイベント系列 (メソッドの起動系列) を明示・形式化し、モデル検査手法を用いて検証することで、振舞い仕様の観点から仕様書記載内容の正しさを確認することである。

3 SPIN を用いた形式検証

3.1 SPIN の概要

SPIN はチャネル通信プロセスでシステムを

モデル化するための仕様記述言語 Promela を提供する [4]。図 4 は送信プロセス (Sender) と受信プロセス (Receiver) が有限バッファ付き通信チャンネル home を通してイベントを通知する例を示す。特に、この例ではイベントを定数 create によって表した。また、Receiver はメッセージ受け付け後の動作として if ... fi による条件分岐を持つ。条件部はチャンネル送信であり常に「真」であるので、2つの分岐を非決定的に選択することを表す。正常終了する場合と例外が発生する場合が非決定的に起こることを表現する。また、Sender はイベント送信後に retval チャンネルからの返却値と except チャンネルからの例外を待つ。unless 構文を用いて簡便に記述することができる。

Promela で記述されたプロセスはラベル付き遷移システム (LTS) に変換される。SPIN の安全性 (safety) 検証は LTS を探索し、出力遷移アークを持たないデッドロック状態がプロセスの停止状態であることや無限ループがないことを確認する。ところが、停止しないシステムの LTS はループを持つ。そのため、Promela 記述上の適切な位置に end ラベル (文字列 “end” で始まるラベル) を挿入することで、検出したループが違反でないことを表す手段を提供する。

進行性 (progress) に関する性質は線形時相論理 (LTL : Linear Temporal Logic) で表現し、SPIN が LTL 式から自動生成する Büchi オートマトンを用いる。SPIN は「成立してはならない」性質を表現した never オートマトンを用いることで効率の良い検証方式を実現している。検証対象システムと合成した never オートマトンが受理言語を持たないことを確認する。never オートマトンが受理 (accept) 状態になる場合は、検証したい性質が満たされないことを示す。

なお、本稿で用いる LTL 式では、通常の命題論理演算子に加えて、[] (always)、<> (eventually)、U (strong until) の時相オペレータを用いる。ここで、 $p \text{ U } q$ は命題 q が成り立つまで命題 p が成り立つことを表す。

3.2 Promela 記述

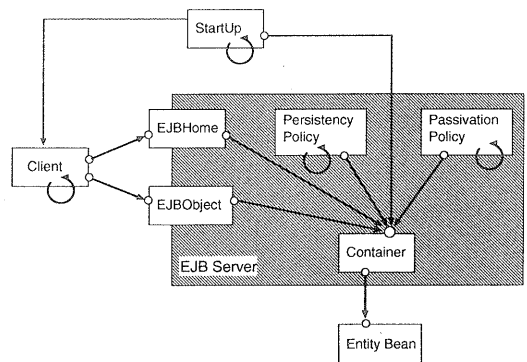


図 5: Promela プロセス構成

本稿の振舞い検証のための形式モデルは、従来のモデル検査手法の主な適用分野であった通信プロトコルマシン [4] とは異なるいくつかの特徴を持つ。すなわち、(1) 正常処理と例外条件が定義された API 中心の仕様であること [2]、(2) Client が起動する API の使い方や起動順序に依存してシステムの振舞いが変わること、(3) 構成要素のいくつかは停止が正しい終了状態であること、である。

Promela 記述の作成に当たっては、これらの 3 点に加えて、LTL 式により検証性質を表現するための工夫が必要となる。一般に、LTL 式の構成命題は「状態」に対して定義される。 σ を状態の列 $(s_0, s_1, \dots, s_j, \dots)$ 、 j を状態インデックスとする時、

$$(\sigma, j) \models p$$

は、命題 p が j 番目の状態 s_j で成り立つことを示す。ところが、第 2.2 節で述べたように、本稿の振舞い検証では、特定イベントの発生を命題とするような性質を LTL 式で表現したい。そのために、当該プロセスの表現方法を工夫し、特定のイベントを受け付け可能な状態が明確に判別可能であるようなプロセスとして作成する。以下、具体的な記述の断片を示す。

```
proctype EntityBean ()
{
  endLoop:

```

表 1: Client モデルと安全性解析

	状態数	遷移数	深さ	起動する API とその順序
Case-1	7,384	15,674	100	create, remove
Case-2	12,273	27,852	719	create, BusinessMethod, remove
Case-3	12,033	27,052	100	{create or find}, remove
Case-4	18,207	42,902	719	{create or find}, BusinessMethod, remove

```

do
  :: mthd?ejbCreate ->
    if :: retv!Value :: excp!Error fi
  :: mthd?ejbPostCreate ->
    if :: retv!Void :: excp!Error fi
  :: mthd?BusinessMethod ->
    if :: retv!Value :: excp!Error fi
  ...
od
}

```

do ... od は繰返しブロックを示す。Entity Bean プロセスが mthd チャンネルを通して受信したメッセージをメソッド起動イベントと解釈する。ここで、チャンネル受信の式 (mthd?ejbCreate 等) は、当該チャンネル (mthd) の先頭に指定メッセージ (ejbCreate)¹ が存在する時に、本プロセスの LTS が指定メッセージを受け付け可能な状態になることを示す。Promela が提供するチャンネル内イベント監視構文 (mthd?[ejbCreate]) を用いることで、対象状態が ejbCreate を受け付け可能か否かを判定する命題を表現することができる。さらに、繰返しブロックに endLoop ラベルを付加し、Entity Bean プロセスがメソッド起動待ちの状態にいる場合を正当な終了状態と見做すことを表した。

図 5 に Entity Bean 振舞い仕様を表現する Promela プロセス構成を示す。(1) Client (クライアント)、(2) EJBHome (Home インタフェースを実現するオブジェクト)、(3) EJBObject (Remote インタフェースを実現するオブジェクト)、(4) Entity Bean (Entity Beans の実体オブジェクト)、(5) Persistency Policy (永続化処理を行なう ejbStore/ejbLoad の起動制御)、(6) Passivation Policy (退避処理を行なう ejbPassivate の

¹ 図 4 の例と同様にイベントを定数で表現した。

表 2: Client 要求の進行性

M	M'
create	ejbCreate
remove	ejbRemove
BusinessMethod	BusinessMethod

起動制御)、(7) Container (Entity Bean 実行の制御)、(8) Start Up (全体を作動させる初期化プロセス)、からなる。Container プロセスが振舞い仕様の中心をなし、第 2.2 節で示したような仕様書記載の具体例を実現するように Entity Bean の状態を管理する。

3.3 解析例

本節では安全性と進行性に関する解析を行なう。安全性については第 3.1 節で述べた方式が基本であるが、解析するためには Client の振舞いを決める必要がある。表 1 に Client モデルの要約を示す。各モデルの複雑さは状態数ならびに遷移数として表れる。また、「深さ」欄より、BusinessMethod を考慮するか否かで探索の深さが異なるが、find の有無は影響しないことがわかる。いずれも検証は成功 (デッドロックがないこと) した。ただし、構成プロセスごとに決めた「終了状態」の条件を満たすことを確認したにすぎない。

次に進行性について、(1) Client 要求の実行、(2) Container が満たすべき個別の性質、に関する解析内容と結果の例を報告する。なお、以下の解析では表 1 の Case-2 の Client を用いた。

(1) Client 要求の実行とは、Client が起動する全てのメソッドについて、Entity Bean の対

応するメソッドが起動されることである。LTL 式は次の形をとる。

$\Box (M@Client \rightarrow \Diamond M'@EntityBean)$

表 2 のメソッドの組 (M ならびに M') に対する検証は失敗した。 $M@Client$ の後、Container の自律的な内部イベントによる livelock のために $M'@EntityBean$ イベント発生に至らない。上記の安全性解析ではわからなかった問題点である。また、内部イベントに起因する例外発生によって Container が操作不能状態に遷移する場合があることもわかった。そこで、上記の可能性を考慮した以下の LTL 式を用いると検証に成功した。

```
#define p1 remote?[BusinessMethod]
#define p2 mthd?[BusinessMethod]
#define p3 Container[3]@endDoesNotExist
#define p4 Passivation[1]@progressLoop
#define p5 Persistency[2]@progressLoop
```

$\Box (p1 \rightarrow (<p2 \parallel <p3 \parallel <p4 \parallel <p5))$

一方、満たすべき性質の否定を与え意図的に検証を失敗させて、失敗を導いた系列を抽出することができる。この系列が、想定する状況を作り出すイベント系列に対応する。例えば、

$<p1 \ \&\& \ <p2$

の否定を与えると、Client が起動した BusinessMethod を EntityBean に委譲するイベント系列の一例を得ることができる。特に最短の長さのイベント系列は自明なものである。

(2) Container が満たすべき個別の性質とは、図 3 の中ほどに示したようなイベント系列を満たすことである。すなわち、BusinessMethod (B-M) の起動対象 EntityBean が passivate されている場合、Container はビジネスメソッドを委譲する前に、ejbActivate ならびに ejbLoad の 2 イベントを発生する、等である。この性質は $\mathcal{U}$ オペレータを用いて表現することができる。

$\Box (ejbActivate \rightarrow (\neg BM \ \mathcal{U} \ ejbLoad))$

この式の検証は失敗する。Entity Bean が ejbActivate を実行する際に例外を発生し、その結果、Container が処理継続不能と判断する場合に失

敗する。そこで、 $\mathcal{W}$ (weak until) オペレータ [9] を用いると検証に成功した。

$p \ \mathcal{W} \ q \approx (p \ \mathcal{U} \ q) \vee \Box p$

具体的には以下の LTL 式を用いた。

```
#define q1 mthd?[ejbActivate]
#define q2 mthd?[ejbLoad]
#define q3 mthd?[BusinessMethod]
```

$\Box (q1 \rightarrow ((\neg q3 \ \mathcal{U} \ q2) \parallel \Box (\neg q3)))$

さらに、Container が処理継続不能状態であることを確認するために次の式も検証した。

```
#define p3 Container[3]@endDoesNotExist
```

$\Box (q1 \rightarrow ((\neg q3 \ \mathcal{U} \ q2) \parallel <p3))$

最後に、先の例と同様に、期待するイベント系列の一例を求めるために、次の式の否定を与えて検証を失敗させる。

$<p1 \ \&\& \ <p2 \ \&\& \ <p3$

先ほどと同様に最短の長さのものは図 3 にある自明なイベント系列である。

以上、いくつかの例を通して述べたように、例外発生を考慮した仕様を扱っているため、検証すべき性質の洗い出しに工夫が必要であった。表 2 の性質のように自明と思われる LTL 式の検証が失敗することから、例外を考慮した検証条件を求めた。しかし、個々の例外事象を区別して取り扱っているわけではないため、意図しない例外が起こっても切り分けが難しい。そこで、成立すべき状態の否定を与えることでイベント系列を抽出して、少なくとも 1 つの正しい系列が存在することを確認した。

もうひとつの問題は、Container 内部で自律的に作動する永続化や退避処理に関係する内部イベントが livelock を起こすことである。これは、今回の Promela 記述の問題点である。一方、EJB 仕様書は livelock に絡む制御には言及しない。具体的なインプリメンテーションで解決すべき事項である。逆に、本 Promela 記述の解析は、インプリメンテーション時に考慮すべき項目を明示する効果があったと考えることもできる。な

お、livelockの問題は解析時の公平性 (fairness) の取扱いに絡む。SPINの解析アルゴリズムが持つ公平性の扱いを考慮することで、今回問題となったlivelockを避けるようにPromela記述を作成することも可能である²。

4 議論

形式化ならびに解析過程で得た知見をもとにEJBサーバ仕様の振舞い検証の意義について考察する。

形式化は、従来手法で作成されたインフォーマルな仕様書を解説し、振舞い検証可能なモデルを作成する作業である。仕様書は種々の視点から断片的に書かれているため、ひとつの統合モデルを構築することが難しい。さらに、検証すべき性質の抽出では、第2.2節に示したような振舞いの例から一般的な規則に抽象化する作業を行なう。この種のモデル構築ならびに検証性質抽出作業の難しさは形式手法を用いる場合に遭遇する一般的なことである。しかし、モデル検査手法を用いる場合は特に問題が大きい。形式化の過程で自動検証を可能にするための高度な決断を伴うからである [5][11]。

形式化の過程で、断片記述間の整合性を確認ならびに文書化されていない仮定を明確にすることができる [3]。例えば、図3のような種々の状況に対処するためには、Entity Beansのready状態を細分化した状態をContainerが管理する必要がある、等がわかった。導入した状態管理の方法で意図通りのEntity Beans管理ができるか否かは進行性検証で確認した。

解析の過程では、前節で述べたように、検証する性質の定式化と検証作業を試行錯誤的に行なった。当初、例外やlivelock発生を考慮しなかったことが大きな理由である。しかし、この過程を通して対象システムへの理解を深めることができる。すなわち、検証が完了したシステム記述だけではなく、解析の過程で得た中間的な生成物ならびにその関連の全体 (性質Aの検

証失敗により次に性質Bを検証した等) が成果物である。SPIN/Promelaは、このような段階的な作業サイクルを可能にする「軽い」デザインカリキュレータ [11] として有用である。

本稿の検証手法では、インフォーマルな仕様書から形式化を行ない、形式化されたモデルに対して検証を行なうことで性質を確認する。この時、「正しく」形式化しているのであろうかという疑問が常につきまとう。入力仕様書から対象システムと検証性質を抽出し形式化する際に、両者の切り分けが曖昧になる恐れがある。検証対象システムのモデル中に明示的に書き込まれた内容を検証手段で確認しているだけかもしれない。入力文書と形式モデルとの間のトレーサビリティを明確にする系統的なモデリング技術を確立する必要がある。

5 関連研究

Sullivan 他 [13] は、COMの仕様をZ記法で形式化し、COM仕様の問題点を指摘した。コンポーネント技術への形式手法適用の最も早い成功例である。その後、JacksonとSullivan[6]は、Alloyで再度形式化を行ない、Alcoaツールを用いて検証を自動化した。

Allen 他 [2] ならびにMagee 他 [8] は、分散アーキテクチャ検証にモデル検査技術を適用して自動検証の有効性を示した。Allen 他 [2] はComponent-Connectorモデルで対象アーキテクチャを表現するアーキテクチャ記述言語 (ADL) Wrightを用いた。Wrightは本質的にはCSPに変換され、モデル検査ツールFDR [10] を用いて検証を行なう。Magee 他 [8] はダイアグラムベースのADLであるDarwinを用いてシステムの構造的な側面を表現し、振舞いの側面をFSPと呼ぶCSP系の並行プロセス言語で表し独自のモデル検査ツールで解析を行なった。

Sousa 他 [12] はWrightによる形式化と検証技法をEJBサーバ仕様の振舞い検証に適用した。EJBサーバをConnectorとしてモデル化しているが、オブジェクトの集まりで構成されると見做す方が仕様書とのトレーサビリティが良

²形式仕様はわかりやすさが大切なので「ハッキング」にならないように気をつけなければならない。

い。Wright では対象モデルを CSP プロセス式として表現するため、仕様書記載の仮想的なオブジェクトが持つ状態をアクションの引数としてコード化する等、表現方法が繁雑となる。また、検証性質の表現方法は LTL 式が使えないために間接的になる。「正しい」ことが判別可能な単純な CSP プロセスと検証対象との詳細化関係を確認することで検証を行なうため、検証手順が繁雑となる。さらに、公平性を考慮した進行性検証を行なっているか不明である。公平性の問題は解析のキーとなることが多い。

本稿では SPIN を用いて EJB サーバ仕様を対象とする分散アーキテクチャの振舞い検証を行なった。SPIN の代表的な応用事例としては、通信プロトコルや分散アルゴリズムの検証 [4]、C 言語で書かれたプログラムの検証 [5] があり、分散アーキテクチャへの適用は本稿が初めてである。この過程で、SPIN は分散アーキテクチャの振舞い検証に有用な機能を持つことがわかった。検証対象の構成オブジェクトを Promela プロセスに対応させるトレーサビリティの良さ、容易に例外検知を表現する構文 (unless)、チャネル内のイベント監視構文 (mthd?[ejbCreate]), 等が有用であった。また、LTL 式を用いることで検証する性質を簡便に表現できるという点が、第 4 節で述べたような段階的な作業サイクルを容易にしたと考えられる。

6 おわりに

本稿では、EJB 仕様書 [1] 記載の EJB サーバ振舞い仕様を Entity Beans の場合について、SPIN/Promela を用いて形式化し自動検証を行なった。Session Beans に対しても形式化と検証を行なうことで仕様全体に対する検証を完成させる。その際に第 4 節で述べた「トレーサビリティ」の問題を考慮してモデリングを行なう。

今回の報告では既存の仕様書から振舞い仕様を抽出して検証することを試みた。今後、特定のソフトウェア開発過程での効果的な自動検証技術の適用方法を具体的に検討していきたいと考えている。

参考文献

- [1] Sun Microsystems, Inc. : Enterprise JavaBeansTM Specification, v1.1 (1999).
- [2] Allen, R., Garlan, D., and Ivers, J. : Formal Modeling and Analysis of the HLA Component Integration Standard, Proc. ACM SIGSOFT FSE'98, pp.70-79 (1998).
- [3] Easterbrook, S., Lutz, R., Covington, R., Kelly, J., Ampo, Y., and Hamilton, D. : Experiences Using Lightweight Formal Methods for Requirements Modeling, IEEE trans. SE, vol.24, no.1, pp.4-14 (1998).
- [4] Holzmann, G.J. : The Model Checker SPIN, IEEE trans. SE, vol.23, no.5, pp.279-295 (1997).
- [5] Holzmann, G.J. and Smith, M.H. : Software Model Checking: Extracting Verification Models from Source Code, Proc. FORTE/PSTV (1999).
- [6] Jackson, D. and Sullivan, K. : COM Revisited: Tool-Assisted Modelling and Analysis of Complex Software Structures, Proc. ACM SIGSOFT FSE'00 (2000).
- [7] Leavens, G. and Sitaraman, M. (ed.) : *Foundations of Component-based Systems*, Cambridge University Press 2000.
- [8] Magee, J., Kramer, J., and Giannakopoulou, D. : Analysing the Behaviour of Distributed Software Architectures: a Case Study, Proc. IEEE FTDCS'97 (1997).
- [9] Manna, Z. and Pnueli, A. : *Temporal Verification of Reactive Systems : safety*, Springer-Verlag 1995.
- [10] Roscoe, A.W. : *The Theory and Practice of Concurrency*, Prentice Hall 1998.
- [11] Rushby, J. : Mechanized Formal Methods: Where Next?, Proc. FM'99, pp.48-51 (1999).
- [12] Sousa, J. and Garlan, D. : Formal Modeling of the Enterprise JavaBeansTM Component Integration Framework, Proc. FM'99, pp.1281-1300 (1999).
- [13] Sullivan, K., Marchukov, M., and Socha, J. : Analysis of a Conflict Between Aggregation and Interface Negotiation in Microsoft's Component Object Model, IEEE trans. SE, vol.25, no.4, pp.584-599 (1999).
- [14] Szyperski, C. : Components and the Way Ahead, in [7], pp.1-20 (2000).