情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-SE-169 No.2
2010/7/22

Alloy を用いた構成変更支援ツールと適用実験

谷 崎 裕 明[†1]　青 木 利 晃[†1]　片 山 卓 也[†1]

ソフトウェアの設定ファイルは，誤りの無いよう記述しなければならない．しかし，設定ファイルの内容が複雑な場合や規模が大きい場合，人手による管理も困難となる．我々は，設定ファイル等を体系的に管理するためのフレームワークを提案した．

フレームワークでは，設定ファイルの文法や記述内容の依存関係をフィーチャモデルで表現する．設定ファイルのフィーチャモデルと記述内容との整合性を判定することで，誤りの検出と誤りの原因と修正方法の特定を行う．整合性は，設定ファイルのフィーチャモデルと記述内容を Alloy で記述し，Alloy を用いて判定する．

本稿では，フレームワークに基づいて作成したツールを Apache の設定ファイルである httpd.conf へ適用した結果について述べる．

An Implementation of Configuration Change Support Tool Using Alloy and Application Test

Hiroaki Tanizaki,[†1] Toshiaki Aoki[†1]
and Takuya Katayama[†1]

Configuration files of software systems have to be written correctly. However, it is difficult for engineers to maintain configuration files when configuration files are complexity. We proposed a framework to maintain configuration of software systems systematically.

In the framework, syntax and dependency of description of configuration files are represented by feature-model. A consistency check of feature model of a configuration file and descriptions of the configuration file detects error of descriptions of the configuration file. The consistency is checked by using Alloy.

In this paper, we present a tool which is based on the framework and the result of application test of the tool.

†1 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

1. はじめに

ソフトウェアシステムに対して機能や実行環境の変更といった要求が変化する場合，ソフトウェアシステムの構成を決定付けるソースコードやコンポーネント，設定ファイルなどを変更する必要が生じる．設定ファイルを変更する際，記述に誤りが含まれてはいけない．誤りが含まれることにより，意図しない動作や動作しないなどの問題が生じる．また，意図通りの設定を行うには設定の依存関係などを把握しておく必要がある．しかし，設定の規模が大きく複雑な場合，開発者が全てを把握して運用することは困難である．

我々は，ソフトウェアシステムの構成を変更する際の整合性判定手法を提案した[6]．提案手法では，ソフトウェアシステムの構成をフィーチャ指向ドメイン分析 (FODA) のフィーチャモデル[9)10)] に基づいてモデル化して管理する．また，フィーチャモデルを形式化することで計算機により記述内容の整合性判定を実現する．整合性判定には，Alloy[1)5)] を用いる．記述に誤りがある場合，原因の特定と修正が必要になることから，整合性判定では誤りの検出に加え，誤りの原因を特定して修正方法を提示する仕組みを導入している．

本稿では，構成変更のフレームワークと作成したツールの概要を述べ，Apache の設定ファイルである httpd.conf を対象にしたツールの適用実験について述べる．

2. 構成の変更における整合性判定

我々の提案した整合性判定は，フィーチャモデルによる構成のモデル化とツールを利用した整合性判定により設定ファイル等の変更を支援するフレームワークである．本稿では，このフレームワークを構成変更フレームワークと呼ぶ．

2.1 構成変更フレームワーク

構成変更フレームワークの概要を図 1 に示す．

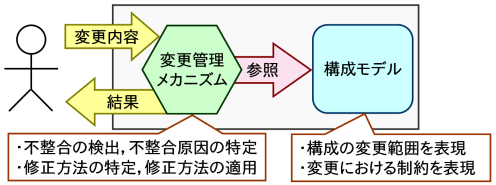

図 1　構成変更フレームワークの概要

構成変更フレームワークでは，ソフトウェアシステムの機能と実行環境をフィーチャモデルで表現する．機能と実行環境のフィーチャモデルで表現されるソフトウェアシステムのモデルを構成モデルと呼ぶ．フィーチャモデルを用いることで変更可能な構成の範囲と制約を表現し，変更の制約を整合性判定に利用する．整合性判定は，誤りの検出と誤りの原因と修正方法を特定することで，構成を変更する際の支援となる．整合性判定に関する一連の操作を変更管理メカニズムと呼ぶ．

2.2 構成モデル

構成モデルは，ソフトウェアシステムの構成を機能と実行環境のフィーチャモデルと機能と実行環境のフィーチャモデル間の対応関係を表現する．図 2 に構成モデルの例を示す．

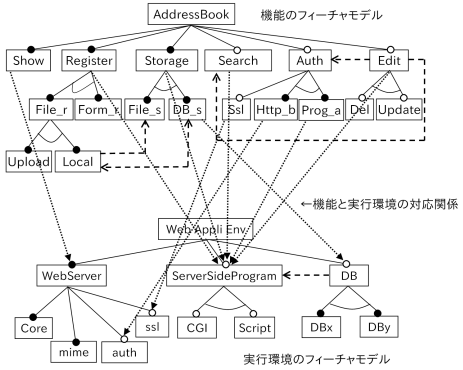

図 2 構成モデルの例

フィーチャモデルを用いることで，ある時点の構成だけではなく，変更可能な構成の範囲と変更における制約が表現可能となる．フィーチャモデルにおけるフィーチャとは，利用者が選択可能な機能や品質特性の単位である．フィーチャモデルはフィーチャの共通性と可変性を木構造で表現する．フィーチャモデルを用いたドメイン分析では，システムの具体的な機能や品質特性をフィーチャモデルから選択したフィーチャの集合で表現する．フィーチャ選択には，フィーチャの親子関係の選択に関する制約（Mandatory, Optional, Alternative, Or）とその他の制約（Requires, Excludes）が存在する．フィーチャモデルの記法は，これらの制約を表現する（図 3）． Mandatory は，親の選択に対して子の選択が必須であることを表す．Optional は，親の選択に対して子の選択が任意であることを表す．Alternative

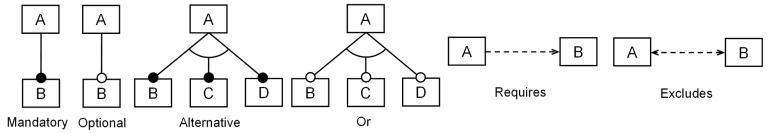

図 3 フィーチャモデルの記法（制約）

は，親の選択に対して一つの子の選択が必須であることを表す．Or は，親の選択に対して一つ以上の子の選択が必須であることを表す．Requires は，A の選択に対して B の選択が必須であることを表す．Excludes は，A と B の選択が排他的であることを表す．これらフィーチャモデルの表現するフィーチャ選択の制約を用いることで，フィーチャ選択の誤りを検出可能である．

機能と実行環境間の対応関係は，機能が実行に必要としている実行環境を明確にする．機能と実行環境の構成を任意に変更した場合，機能の実行に必要な環境が失われる可能性があるため，機能や実行環境の構成を変更する際には両者の構成を考慮する必要がある．機能と実行環境の対応関係を表現する事で，一方の構成の変更が他方の構成に与える影響を把握することが可能となる．

このように，構成モデルは，変更の範囲と制約を表現するモデルである．

2.3 変更管理メカニズム

構成変更フレームワークにおいてソフトウェアシステムの具体的な構成は，構成モデルから選択したフィーチャにより表現される．構成モデルの表現する制約とフィーチャ選択との整合性を判定することにより，誤りを検出する．また，誤りが生じる場合には修正が必要となるため，整合性判定手法は誤りの原因と修正方法の特定も対象とする．誤りの原因や修正方法は，構成モデルの表現する制約に基づく．

フィーチャモデルの制約に基づく整合性判定では，フィーチャ選択がフィーチャモデルの制約を満たさない場合が誤りとなる．また，フィーチャモデルの制約を充足しない選択方法が誤りの原因，誤りの原因となる選択方法から制約を満たす選択方法へ変更する方法が修正方法となる．フィーチャモデルの制約を充足しない選択方法は，フィーチャの選択の過不足であるため，修正方法はフィーチャの追加または削除となる．

計算機による変更管理メカニズムの実現のため，Alloy（Alloy Analyzer）を用いた．Alloy Analyzer は与えられた制約を充足する Alloy のモデルを探索するツールであり，フレームワークの整合性判定における誤りの原因や修正方法（修正候補）の探索に適している．

2.4 フレームワークに基づくツールの作成

今回，以下に示す三点の課題に対応するため，フレームワークに基づくツールを作成した．

- Alloy のモデルへの変換

構成変更フレームワークをソフトウェアの設定ファイルなどに適用するためには，適用対象の構成モデルが必要である．また，Alloy で実現した整合性判定の適用には Alloy のモデルを作成する必要がある．一度だけ判定すれば良いものであれば，判定の際に Alloy のモデルを作成することも問題にはならないが，設定ファイルのように変更の可能性があるものについては，判定のたびに Alloy のモデルを手作業で作成することは効率的ではない．

Alloy のモデルへの変換については，判定対象を Alloy のモデルへ自動的に変換する仕組みを導入することで対応する．

- 規模の問題

我々は，Alloy で実現した構成変更フレームワークの適用例として，Apache の設定ファイルである httpd.conf への適用実験を行った．しかし，実験に用いた計算機環境（OS：Windows XP Pro SP3, CPU：Core 2 Duo 3.16GHz, RAM：2GB）では，Apache の core モジュールの構成モデル（要素数：234，関係の数：168）を処理することが出来なかった．異なる計算機環境での実験は行っていないが，今回の整合性判定方法を Alloy にエンコードした方法では，要素数が数百のモデルを扱うことが出来ないことが分かった．

規模の問題には，構成モデルを部分木に分割し，Alloy に入力するモデルの規模を小さくすることで対応する．

- 誤りの修正

Alloy で実現した誤りの修正は，構成モデルの制約に基づいて Alloy が自動的に修正候補を探索する．従って，利用者の意図が反映された修正結果が得られるとは限らない．

誤りの修正に利用者の意図を反映させるため，利用者が対話的に誤りを修正する仕組みを導入する．具体的には，Alloy が特定する修正方法を利用者が選択し，選択された修正方法に基づいて判定用の Alloy のモデルを生成して再判定を行う．

3. 構成変更支援ツール

構成変更支援ツール[*1]（以降，本稿では単にツールと表記する．）の全体像を図 4 に示す．作成したツールは，後述の httpd.conf 専用ではなく汎用的に使うことを目的としているが，説明は httpd.conf への適用を基に行う．ツールは，適用対象 (httpd.conf) のパーサと Alloy のモデルを作成して整合性を判定する部分に分かれている．

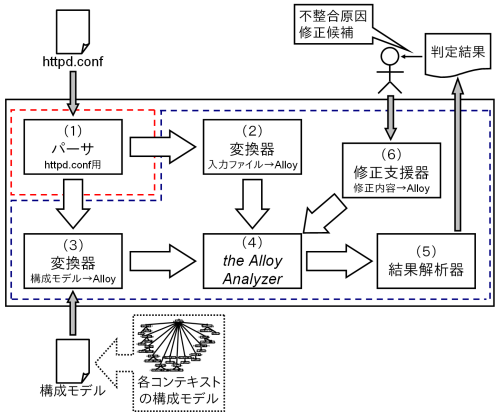

図 4 ツール全体像

ツールの概要は以下の通りである．適用の対象である httpd.conf と Alloy のモデルでは記述形式が異なるため，ツールは httpd.conf の記述内容を Alloy のモデルに変換できるよう処理し，httpd.conf の記述内容に対応した Alloy のモデルを生成する．httpd.conf の構成モデルから Alloy のモデルで記述した構成モデルを生成する．Alloy のモデルで記述された構成モデルは，規模の問題に対応するため，httpd.conf に記述されているディレクティブに関連する部分のみを抽出して生成される．Alloy Analyzer を用いて httpd.conf の記述内容に対応した Alloy のモデルの整合性を判定し，結果を得る．httpd.conf の記述内容に誤りが検出された場合，誤りの原因などを特定し，対話的に修正する．

3.1 ツールの各機能

ツールを構成する各機能について説明する．

(1) 適用対象 (httpd.conf) 用パーサ
整合性判定のため，httpd.conf の記述内容をフレームワークで定義した Alloy のモデルに変換する．Alloy のモデルに変換する方法は対象により異なる．httpd.conf の場合，詳細は 4 節で説明するが，ディレクティブと引数の関係を親と子の関係として扱うため，httpd.conf の記述内容を解析してディレクティブと引数を抽出する．

[*1] 本ツールは Java で実装されている．

(2) 判定対象モデル作成
判定対象モデルの記法は定義されているため，記法に合わせて適用対象を Alloy のモデルに変換する．httpd.conf の場合，httpd.conf 用パーサにより抽出されたディレクティブと引数が Alloy のモデルに変換される．

(3) 判定用構成モデル作成
整合性判定に利用するため，構成モデルを Alloy のモデルに変換する．ツールでは規模の問題に対応するため，各構成モデルから httpd.conf 用パーサにより得られたディレクティブに関連する部分を抽出し，構成モデルの部分木を作成する．ここで得られた構成モデルの部分木を判定用の構成モデルとして Alloy のモデルに変換する．

(4) Alloy Analyzer
Alloy Analyzer は，整合性判定のエンジンである．判定用構成モデルを基に，判定対象モデルの整合性を判定し，判定結果を XML 形式のファイルで出力する．従って，ツールは Alloy Analyzer とのインタフェースの役割を担っている．

(5) 整合性判定結果の解析
Alloy Analyzer の判定結果を解析し，利用者に提示する．提示される判定結果は，不整合の有無と不整合が生じる場合の原因と修正方法である．

3.2 ツールの使い方

利用者がツールを実行すると，ツールは httpd.conf の記述内容を解析して判定対象と判定用構成モデルの Alloy のモデルを生成する．ツールは生成された判定対象モデルの整合性を Alloy Analyzer で判定し，判定結果を解析して利用者に提示する．

整合性判定の結果，不整合が検出された場合，ツールは利用者に修正方法と修正に用いる要素を提示する．利用者が修正内容をツールに入力すると，ツールは修正内容を適用した判定対象のモデルを生成して Alloy Analyzer により整合性を判定する．

4. Apache(httpd.conf) への適用

4.1 Apache の設定変更

Web サーバソフトウェアである Apache は，Web サーバの運用目的に応じて利用者が設定を変更する．静的な Web サイトを利用する場合であっても，ログの記録方法，パフォーマンスの調整，バーチャルホストの利用など利用者の目的に応じた様々な設定が可能である．また，PHP 等の外部モジュールを導入して Web サーバ側で利用できるよう設定を変更することで，Pukiwiki[11] 等の Web アプリケーションの利用が可能となる．(図 5)．

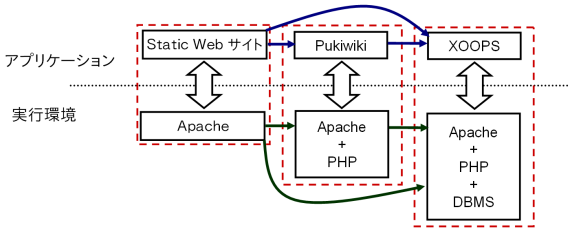

図 5 Apache 設定の変更例

図 5 に示すように，Apache は要求の変化に応じて構成（設定）を変更する．Apache の構成は，設定ファイル (以降，httpd.conf) に記述される設定項目 (以降，ディレクティブ) とディレクティブの引数の組み合わせにより決定される．httpd.conf の記述内容に誤りがある場合，動作しないなどの問題が生じるため，Apache にはシンタックスチェッカー（apachectl コマンドの configtest オプション．以降，configtest と表記）が提供されている．

4.2 httpd.conf

httpd.conf は，Apache の設定ファイルであり，ディレクティブとディレクティブの引数により設定を記述する．ディレクティブは Apache の動作への指示を表し，ディレクティブの引数は指示の具体的な内容を表す．ディレクティブと引数は，設定項目と設定値の関係である．ディレクティブの引数には，事前に定義されている定数や数値，いくつかの型がある．代表的な型には，URL，ファイルやディレクトリのパス (*file-path*, *directory-path*)，ファイル名 (*filename*)，ファイルの形式 (*MIME-type*) がある．

サーバが主に利用するディレクトリを設定する “ServerRoot” ディレクティブでは，ディレクトリパスを引数にとり， ServerRoot ”/usr/local/apache_1.3.41” のように記述する．

また，ディレクティブには記述可能な場所 (以降，コンテキスト) が存在する．コンテキストには，サーバ設定ファイル，バーチャルホスト，ディレクトリ，.htaccess がある．ディレクティブはコンテキストに記述された場合にのみ有効となり，誤ったコンテキストに記述された場合にはエラーの発生や Apache が起動しないなどの問題が生じる．

4.3 構成モデルの作成

KeepAlive ディレクティブを例に，httpd.conf の構成モデルの作成方法を説明する (図 6)． 構成モデルは，httpd.conf のコンテキスト（ServerConfig），コンテキストに記述されるディレクティブに関連する Apache のモジュール（core module），ディレクティブ（KeepAlive），引数（On, Off）を階層的に表現する．ServerConfig はサーバ全体の設定に

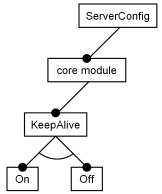

図 6 構成モデルの一部 (KeepAlive ディレクティブ)

関わるコンテキストである．ServerConfig に設定を記述する際，Apache のコアモジュール（以降，core）は必須であるため，core は ServerConfig を親とする Mandatory の子となる．KeepAlive ディレクティブは Apache の設定に必須であるため，KeepAlive は core を親とする Mandatory の子となる．KeepAlive の構文は，“ KeepAlive on | off ” であり，KeepAlive ディレクティブが “on” または “off” を引数にとることを意味する．引数の選択は択一的であるため，フィーチャモデルの制約では Alternative に相当する．

図 6 の構成モデルを Alloy のモデルに変換する際，関係に名前を付けて親と子を識別する名前を付ける．具体的には，関係の名前は一意に番号を振り，on や off といった他のディレクティブの引数としても存在するものについては親の名前を付けることで識別する．図 6 の例では，関係の番号を “001” とした場合，

```
CM.Alt = 001,    001.pE = KeepAlive,    001.cE = KeepAlive_on + KeepAlive_off
```

となる．ここで，CM.Alt は構成モデルにおける Alternative の集合，001.pE は関係 001 の親の集合，001.cE は関係 001 の子の集合を表す．

httpd.conf に記述する際，ディレクティブと引数は一対一で対応するため，複数記述されるディレクティブが存在する．モジュールを追加するディレクティブの LoadModule は，導入するモジュール (引数) の数だけ記述する．こういったディレクティブの場合，httpd.conf 上ではディレクティブが複数記述されるが，構成モデル上ではディレクティブを親として記述し，引数を Or の子として記述する．また，ディレクティブには定数以外に数値，ディレクトリやファイルのパスといった変数を引数にとるものがあるが，変数であるため事前に構成モデルで定義することは出来ない．従って，引数が変数である場合，数値やパスを引数にとることだけを構成モデルに記述し，引数に記述された変数が数値であるかパスの形式であるかといった型の判定を行う．

5. 適 用 実 験

作成したツールを用いて httpd.conf の記述内容を判定する実験を行った．実験ではツールの有効性を評価するため，configtest を比較対象に用いた．

5.1 判定の対象

httpd.conf の記述を変更する際，文法の誤りを防ぐことは必須である．また，意図した機能を実現するための記述を過不足なく記述する必要があるため，ディレクティブや引数の記述における依存関係や記述内容の一貫性なども考慮する必要がある．

httpd.conf の記述内容を変更する際に判定の対象となる性質を以下に示す．

- 文法
  - ディレクティブと引数の関係（ディレクティブに対して記述可能な引数であるか．引数の個数に誤りはないか.）
  - ディレクティブとコンテキストの関係（記述可能なコンテキストに記述されているか.）

文法的な制約が満たされていない場合，Apache が動作しないなど深刻な影響を与える恐れがあるため，判定は必須である．

- 他のディレクティブや引数の記述が必要なディレクティブや引数の記述
  - モジュールを追加する LoadModule ディレクティブとモジュールを使用するようにする AddModule ディレクティブでは，LoadModule されていないモジュールは AddModule が出来ない．
  - モジュールが提供するディレクティブは，モジュールが AddModule されていなければ利用出来ない．
    (ディレクティブが <IfModule> 内に記述されている場合，モジュールが AddModule されているかは判定されない．)
- 記述内容の一貫性
  - ディレクティブは文法的な誤りが無ければ複数記述することが可能である．(KeepAlive ディレクティブの場合，“KeepAlive On” と “KeepAlive Off” の両方を記述する事が可能) この場合，設定内容に矛盾が生じると共に，利用者の設定意図が不明確になってしまう．

これらは一例であるが，httpd.conf の記述を変更する際はディレクティブや引数間の関係を考慮する必要がある．

5.2 文法チェック

ディレクティブと引数の文法を判定する．ここでは，LogLevel ディレクティブの例を示す．LogLevel はエラーログへ記録するメッセージの冗長性を指定するディレクティブであり，引数には debug, info, notice, warn, error, crit, alert, emerg のうち一つを指定する．LogLevel の引数に “*criit*” を指定した場合，スペルミスであり誤りとなる．
configtest による判定結果を以下に示す．

```
Syntax error on line 509 of /usr/local/apache_1.3.41/conf/httpd.conf:
LogLevel requires level keyword:
        one of emerg/alert/crit/error/warn/notice/info/debug
```

“Syntax error” が検出され，引数の候補が表示されている．
ツールによる判定結果を以下に示す．

```
The cause of inconsistency :
Inconsistent relation : core146
    Parenet : LogLevel
    Child : info, alert, emerg, warn, debug, notice, error, crit
    cause : lack_of_child
    c_method : select_one
    c_element : info, alert, emerg, warn, debug, notice, error, crit
 Line : 508
```

ツールの判定は構成モデルに基づくため，引数の候補（Child）だけではなく，不整合の原因（cause）と修正方法（c_method）を提示可能である．cause の lack_of_child は，引数 (子) が存在していないことが不整合の原因であることを表し，c_method の select_one は，c_element の中から一つを選択する事が修正方法であることを表している．

5.2.1 誤りの修正

configtest の判定では，httpd.conf の誤りが生じた箇所 (行番号) と全ての誤りに対してではないが修正候補や修正方法が提示される．利用者は，configtest の結果を参照して httpd.conf の記述内容を修正する．
ツールでは，修正候補を選択可能な誤りについて，対話的に修正する仕組みを提供している．先に示した LogLevel の場合，修正内容を受け付けるために以下の内容が表示される．

```
inconsistent relation : core146
the cause of inconsistency : lack_of_child
select one element (number) from below :
1)info  2)alert  3)emerg  4)warn  5)debug  6)notice  7)error  8)crit
```

利用者が修正方法に従って修正に利用する番号を指定することで，修正方法を反映した判定用モデルが生成される．生成された判定用モデルを Alloy でチェックする事で，修正内容の判定が可能である．(修正内容が新たな誤りの原因となる可能性があるため，再度判定する必要がある．)

5.3 ディレクティブや引数の制約

ディレクティブや引数の記述には文法の制約だけではなく，意味的な制約も存在する．意味的な制約とは，ある機能の実現に複数のディレクティブや引数が関連する依存関係や，設定内容の一貫性や安定した動作を保証するためにディレクティブや引数の記述数を制限するものである．前者の制約が満たされていない場合，意図した機能を提供する Apache の設定 (構成) が実現されず，原因の究明が必要となる可能性がある．後者の制約が満たされていない場合，設定内容に矛盾が生じることで不具合の生じる恐れがある．また，唯一の設定が必要なディレクティブが複数記述されている場合，設定上の意図が不明確になる．

5.3.1 依 存 関 係

Web アプリケーションの一つである Pukiwiki を利用する場合の変更を例として扱う．Pukiwiki は PHP で実装されているため，Web サーバ側に PHP の実行環境が必要となる．多くの場合，Apache に PHP モジュール（以下，mod_php）を追加する事で対応する．
mod_php の導入に関する依存関係を抽出した構成モデルを図 7 に示す．

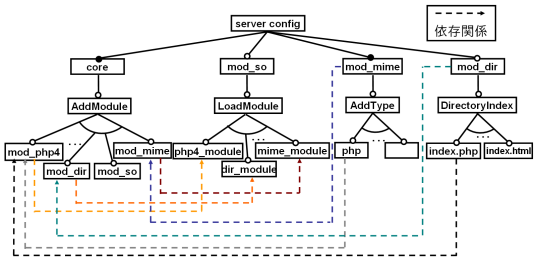

図 7 PHP の利用に関する構成モデル

PHP を Apache で有効にするため，httpd.conf に拡張子を指定されたコンテントタイプ

に対応付ける AddType ディレクティブ[⋆1]を追記する．AddType は mod_mime モジュールのディレクティブであるため，AddType を有効にするためには mod_mime が有効になっている必要がある．このようにディレクティブの利用には前提となるディレクティブが存在する．PHP 導入に関する記述を追加した httpd.conf を configtest とツールにより判定した．

configtest の判定結果は “*SyntaxOK*” であったが，提案手法のツールでは誤りを検出した．ツールで検出された誤りは，図 7 において，AddType の PHP が必要とする要素である AddModule の子の mod_php が存在しないことを表すものである．(httpd.conf の記述では，mod_php が AddModule されていないことを表す.）この依存関係は文法的に必要な記述ではないが，PHP を利用するという目的からは必要な設定である．

5.3.2 ディレクティブの記述数

ディレクティブの記述数に関する例を用いる．各ディレクティブの記述数には制限がある．サーバの動作を決定付けるようなディレクティブは唯一の設定が必要であるため，複数記述する事は不具合の原因となる恐れがある．しかし，ディレクティブと引数の文法を守りさえすれば，ディレクティブは複数記述することが可能である．また，複数記述した場合，先に記述された内容が有効になるといった場合であっても，複数記述した中でどの記述が利用者の意図した内容であるかは明確ではない．よって，ディレクティブの記述数に関する判定も必要となる．

httpd.conf に “KeepAlive On” と “KeepAlive Off” を記述した．configtest の判定結果は，“*SyntaxOK*” であった．提案手法のツールでは，KeepAlive を親とする Alternative の誤りが検出された．このように，提案手法のツールでは，ディレクティブの記述可能な数を構成モデルで定義しているため，構成モデルの制約に反する記述を検出可能である．

6. 評　　価

ツールの適用実験では，httpd.conf の文法チェックと意図した性質を設定に反映する際に必要となる制約の判定を行った．configtest とツールを用いた実験内容及び結果を**表 1** に示す（○は判定や提示が可能であることを示す．）．

表 1 より，configtest とツールでは，依存関係とその他の制約の判定において能力が異なることが分かる．この違いは，configtest が文法チェックに重点を置いていることに対し，

⋆1 PHP の場合，“AddType application/x-httpd-php .php” と記述する．

表 1　実験結果

	configtest	提案手法
文法	○	○
定数の修正候補	○	○
変数（数値，パス）の修正候補	×	×
モジュールに依存するディレクティブの記述（<IfModule> 無し	○	○
モジュールに依存するディレクティブの記述（<IfModule> 有り）	×	○
Apahce に付属しないモジュールのディレクティブ	×	○
ディレクティブの記述数	×	○

提案手法のツールは構成モデルに基づき，構成の変更を誤りなく行うための判定に重点を置いていることにより生じる．また，configtest は 標準的な Apache のモジュール構成を対象にしているため，サードパーティから提供されるモジュール（今回の例では PHP モジュール）には対応していない．構成モデルは，構成変更フレームワークを適用する対象に合わせて作成するため，柔軟な対応が可能である．

作成したツールは，構成変更フレームワークを Apache(httpd.conf) に適用したものである．評価実験の結果から，図 1 に示した構成変更フレームワークの一連の流れを具体的な事例に適用し，変更内容の判定に一定の有効性があることを確認した．

構成変更のフレームワークは，構成モデルと変更管理メカニズムによる整合性判定が核となる．構成モデルを作成しフレームワークを用いることで，設定 (構成) の変更における依存関係などの制約を意識せずに変更内容を作成し，判定することが可能となる．

7. 議　　論

適用実験の結果より，構成モデルに基づく提案手法のツールを利用することで，configtest では検出することの出来ない依存や排他関係の誤りを検出することが可能であることを確認した．configtest は標準的な Apache のモジュールが対象であり mod_php のように外部のモジュールには対応していないが，外部モジュールを追加する際にも httpd.conf の記述内容は configtest で判定するため，何らかの判定方法は必要である．

設定ファイルの変更によりソフトウェアシステムの構成を変更させる場合，設定の記述内容に誤りのないことを確認することは不可欠であるが，記述内容が意図した内容を満足するものであるかを判定する事も重要である．我々の構成変更のフレームワークは，ソフトウェアシステムに対する要求の変化を構成の変更として捉えるものであり，構成モデルに基づく

整合性判定により構成の変更内容における誤りの有無を判定する．Apache のように設定の記述内容により構成を変更するソフトウェアシステムにおいて，構成の変更における制約を表現するモデルに基づいたツールの利用は有効である．

8. 関連研究

Apache の設定ファイルをチェックするツールには，configtest を GUI で操作する ApacheConf[2] が提供されているが，基本的な判定能力は configtest と同じである．

フィーチャモデルは変更のバリエーションを表現可能であるため，構成の変更をフィーチャモデルで表現する手法が Cetina らにより提案されている[4]．Cetina らの手法におけるフィーチャ選択は，システムを運用する際のシナリオに基づいたものであり，修正方法に相当する選択もシナリオに基づいたものである．

構成変更フレームワークでは，ソフトウェアシステムの構成をフィーチャモデルで表現し，構成の変更内容を Alloy で判定する．フィーチャモデルの形式化及び整合性判定に Alloy を用いる研究は多い[8][12]．Alloy はフィーチャモデルからフィーチャを選択する際の整合性を判定するツールに用いられるが，我々の提案手法では，不整合の原因や修正方法の特定を対象とした構成モデルの形式化を行った．

要求の変化をソフトウェアシステム自身が解決することを目的とした自己管理システムが提案されている[7]．要求の変化に構成を適応させる際，変更が要求の変化に即したものであるかを判定すること，変更内容に誤りがあってはいけないなど，解決すべき問題が挙げられている[3]．我々の提案手法は，構成の変更内容をモデル化し，変更における整合性を判定するものであるため，自己管理システムの実現に向けた一つの方法論である．

9. おわりに

本稿では，我々の提案手法である構成変更のフレームワークに基づいて作成した構成変更支援ツールの概要を述べ，httpd.conf へ適用実験では configtest との比較を行った．ツールは，構成モデルを基に，Alloy を利用して変更内容の整合性を判定する．Alloy による整合性判定は，不整合の検出に加え，不整合原因を特定して修正方法を提示することで，構成を変更する際の支援に利用可能である．また，整合性判定には構成モデルを用いているため，文法的な判定だけではなく，設定の記述における依存関係や記述内容の一貫性を保つための制約を対象とした判定も可能である．

今回は適用対象に Apache の設定ファイルである httpd.conf を選択したが，設定の変更により構成を変更するソフトウェアシステムは Apache に限らないため，他の事例についても適用し，提案手法の有効性を評価する必要がある．

ツールは httpd.conf の整合性を判定して修正候補を得ることは出来ているため，誤りを修正した httpd.conf を出力する機能が追加されることで，構成変更の支援となる．

また，本稿の適用実験構成モデルにおける実行環境に相当する部分を対象としている．Apache を含む実行環境上で動作する Web アプリケーションなどの構成モデルを作成し，機能と実行環境，機能と実行環境間の対応関係を扱う用いたツールへの拡張も必要である．

参考文献

1) Alloy website : http://alloy.mit.edu/
2) ApacheConf : http://www.apache-gui.com/apacheconf/
3) Betty H. C. Cheng, et al. Research Directions in Requirements Engineering. Proceedings of IEEE ICSE Future of Software Engineering, eds., Minneapolis, Minnesota, pp.285-303, 2007.
4) C. Cetina, et al. Using Variability Models for Developing Self-configuring Pervasive Systems. Workshop on Autonomic and SELF-adaptive systems (WASELF*), Gijon, Spain, pp.10-20, 2008.
5) Daniel Jackson. Software Abstractions: Logic, Language, and Analysis. MIT Press. Cambridge, MA. March 2006. ISBN 0-262-10114-9
6) Hiroaki Tanizaki and Takuya Katayama. Formalization and Consistency Checking the Changes of Software System Confgurations Using Alloy, APSEC '08, Beijing, China, pp.343-350, 2008.
7) J.Kramer and J.Magge. Self-Managed Systems : an Architectural Challenge. In Future of Software Engineering, FOSE07, pp.259-268, 2007.
8) Jing Sun, et al. Formal Semantics and Verification for Feature Modeling, ICECCS '05, IEEE Press, Shanghai, China, pp. 303-312, 2005.
9) Kang, Kyo C., et al, Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, November 1990.
10) Krzysztof Czarnecki and Ulrich Eisenecker. Generative Programming : Methods, Tools, and Applications, Addison-Wesley Pub (Sd), 2000, ISBN 978-0201309775
11) Pukiwiki : http://pukiwiki.sourceforge.jp/
12) Rohit Gheyi, et al. A Theory for Feature Models in Alloy, First Alloy Workshop, Portland, United States, pp. 71-80, 2006.
13) The Apache Software Foundation : http://www.apache.org/