Vol.2011-SE-172 No.8
Vol.2011-EMB-21 No.8
2011/5/18

# アーキテクチャ記述言語を用いた自己適応システム設計手法の検討

堀越永幸[†]　中川博之[†]　田原康之[†]　大須賀昭彦[†]

近年，環境の変化に対して，自身の構成を変え，柔軟に振る舞いを変える自己適応システム(Self-adaptive Systems)の実現に対する期待が高まっている．しかし，自己適応システム実現には多くの研究課題が存在しており，その中の一つとして構成変更の制御とその設計方法がある．自己適応システムは内部状態や外部環境などの変化に対して，システム自身が自己を評価し，その状況に即した構成変更を判断できる必要がある．そのため本研究では，自己適応システム特有の性質を既存のアーキテクチャ記述言語に反映させ，自己適応に特化したコンポーネント記述を提案をする．さらに，それを活用した構成変更アルゴリズムについて検討を行い，今後の研究の方針について述べる．

# Design Technique for Self-adaptive Systems Using Architecture Description Language

Hisayuki Horikoshi[†]　Hiroyuki Nakagawa[†]
Yasuyuki Tahara[†]　and　Akihiko Ohsuga[†]

Modern software systems are expected to adapt their environments at runtime. Such systems, called self-adaptive systems, have been studied intensively in recent years; however it is still difficult to adapt dynamically their environments. In particular, runtime configuration change is one of key issues. Self-adaptive systems have to not only deal with system's condition and external environmental changes but also determine the optimal configuration. In this paper, we propose an architectural description for self-adaptive systems by adding properties into the existing ADL (Architecture Description Language). Moreover, we propose two reconfiguration algorithm (partial alternation and global optimization algorithms). We also discuss our reconfiguration algorithms, and conclude the paper with future work.

## 1. はじめに

近年のソフトウェアは，その大規模・複雑化に伴い，環境の変化に対しても，人が介在することなく，動的に環境に適応する能力が求められるようになってきている．特に，システムの部分的な故障や突発的な負荷の増大に対処することや，システムに与えられる要求や実行時の環境の変化に追従しサービスを提供することが強く求められるようになった．こうした背景から，自己適応システム(Self-adaptive Systems)と呼ばれる，環境に適応し，自らの振る舞いや構成を変化させることのできるシステムの実現に対する期待が高まっている[1]．自己適応システムの実現のためには未だ多くの問題点が存在しており，その中の一つとして，システム実行時の構成変更をどう扱うかが重要な研究課題とされている．構成変更においては，変化が起こった際にシステムが自身の内部状態や外部環境の状況を理解し，その状況での最適な構成を考える必要がある．そこで本研究では自己適応システムの再構成に着目したアプローチとその設計手法について検討を行う．具体的には，アーキテクチャ記述言語を活用した自己適応システムの設計手法について着目し，システムが構成変更時に自身の持つ機能やその状態について理解できるようにする．またそれを活用した再構成手順とそのアルゴリズムについて検討を行う.そして本手法を活用した設計方法について考察を行い，その結果と今後の方向性について示す．

本稿の内容については，まず２章で，近年注目されている自己適応システムとその研究動向について述べる．続いて３章では，自己適応システムの構成変更に必要な要件について論じ，４章にてアプローチの概要を説明し，５章では Web アプリケーションを対象としたシナリオによる変化の想定を行う．６章で関連研究と本手法との比較について論じ，最後に本稿のまとめと今後の検討課題について述べる．

## 2. 自己適応システム

自己適応システムとは，システム自身が自身の機能と目標を管理し，状況変化に応じて柔軟な振る舞いができる能力を持つシステムのことである．情報システムの大規模・複雑化が進む中，部分的な機能の故障やバグ等による不具合，急激なアクセス数の増加による負荷の増大等に対しても，システム運用者などの人手を介さずに，シス

[†] 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications

Vol.2011-SE-172 No.8
Vol.2011-EMB-21 No.8
2011/5/18

テムが自律的に対処をすることが望まれている．近年ではデータベースやサーバなど一部の故障や障害によりシステム全体のサービスが停止する事例も発生しており，自律的に対処可能な自己適応システムの実現が期待されている．また，これらの事例以外でも，機能を追加・変更してのバージョンアップ，信頼性や効率性といったシステムに与えられる要求変化への対応においても期待されている．

自己適応システムに関する研究は近年様々なソフトウェア工学の技術により設計手法が提案されている[1,4,10]．自己適応システム研究の代表的論文[4]では自己適応システム実現のためには図１のような，システムの機能を管理する層と，実行時に動作し機能を提供する層の二つの部から構成されるシステムアーキテクチャが有効だと述べられている．[4,12]ではその参照モデルについて提案されているが，この論文の中でも多くの将来課題が残されていることが著者によって報告されている．その他にも様々な設計手法や参照モデルの提案はあるが，未だ確立された方法は存在していないと言える[8, 16,17]．

自己適応システムの実現にはコントロールループと呼ばれる，変化に対応するフィードバックプロセスが必要だと考えられている[1,2,4]．コントロールループは収集（Collect），分析（Analyze），決定（Decide），実行（Act）といったフェーズに分けられる[1,2]．また類似研究として，IBM が 2003 年に提唱したオートノミックコンピューティングの分野では MAPE ループと呼ばれるコントロールループがある[9]．こちらでは監視（Monitoring），分析（Analyze），計画（Plan），実行（Execute）と定義されている．自己適応システムの研究ではコントロールループの各フェーズについて，様々な問題がある．その中でも，変化イベントが発生した際には，状態を分析し，現在の状態から構成変更案を算出し，その中から１つの案を選択するという決定（計画）のフェーズに関する問題が重要課題として挙げられる．また，自己適応システムに必要とされるコントロールループでは各プロセスにおいて複雑な設計と実装が必要とされ，コントロールループ自体が，自己適応システム設計での複雑さによる問題を生みだしていると考えられている[1,2]．そのため，自己適応システムでは，実行時の構成変更やコントロールループを考慮しての設計が必要であり，さらに実行時にそれらを管理することが必要となる．

以上のことから本研究では，自己適応システムの特徴と構成変更，またそれを考慮した設計手法に焦点を当てる．本稿では自己適応システムの特徴的な性質に注目し，アーキテクチャ記述，実行時の構成変更手順について提案する．

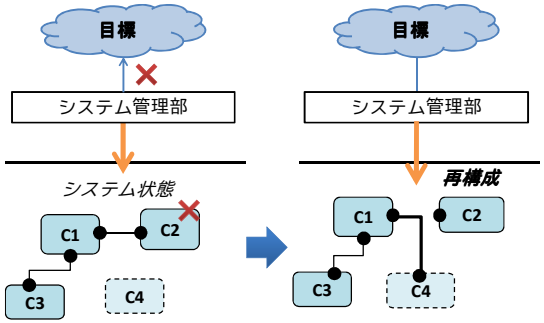


図 1　自己適応システムの振る舞い変更例

## 3.　自己適応システムの再構成に関する要件

本研究が提案する自己適応システムの再構成手法について，必要とされる要件を定義する．

【要件１】コンポーネントによるアーキテクチャ

システム実行時に構成を変え，振る舞いの変更を実現するためには，各振る舞いを提供するモジュールを動的に切り替えることのできるアーキテクチャである必要がある．そのため，自己適応システムの研究では，システム全体をコンポーネントによる機能の接続によって構成する手法が有効とされている[1,4]．図２はコンポーネント開発手法を用いたシステム例（Web ニュース配信システム）であり，複数のコンポーネントの接続によってアーキテクチャ全体が表現されている．

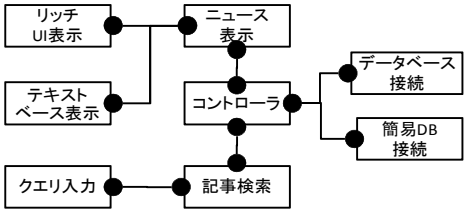


図 2　コンポーネント接続によるアーキテクチャの表現例

【要件２】自己評価による再構成計算メカニズム

自己適応システムは，状況にあった振る舞いを期待されていることから，異なる

Vol.2011-SE-172 No.8
Vol.2011-EMB-21 No.8
2011/5/18

振る舞いを提供するコンポーネントが複数用意され，その中から使用する機能の組合せを考える必要がある．先行研究では，それらのコンポーネント群の接続構成を全て事前に決定しておくことは容易ではなく，また，柔軟性に欠けるとの問題が指摘されている[1,14]．そのため，システム自身が実行時にコンポーネントの接続案を算出する必要がある．以上のことから自己適応システムでは，システム自身が，自身の状態や構成を参照できる自己参照性と，実行時にそれを評価するメカニズムが求められる．

【要件３】非機能特性を考慮した最適な再構成

自己適応システムはシステムをとりまく環境や状態の変化を考慮して再構成をする必要があることから，機能要求に加え，システムが関与する非機能要求（機能以外の性能や制約への要求）の特性を考慮して最適な構成案を割り出す必要がある．本研究での「最適」とは，その状況やシステム状態に合わせてもっとも望ましいとされる構成案のことである．さらに，部分的な交換（同等の機能，異なる非機能要求への貢献）に対応するだけでなく，スループットや CPU 使用率などシステム全体が関与する特性も考慮しなければならない．特に全体的な最適構成の場合には，複数箇所の変更や複数の構成候補案を算出できること，またその複数候補から１つの案を採用するアルゴリズムが必要になる．

## 4. 提案手法概要

前述の各要件を満たすために，以下の２つを軸とした手法を提案をする．

（１）自己適応システムに特化したコンポーネント仕様記述の提案を行う．要件１から，コンポーネント接続によるアーキテクチャとそれによる構成変更の制御が有効である．特に要件２，３から，自己適応システム特有の自己参照性や非機能特性に着目をする必要があると言える．

（２）（１）のコンポーネント仕様記述を実行時に解析して，複数の案から最適なコンフィグレーション（構成設定）を割り出す．その際に，アーキテクチャレベルの抽象度から構成変更を取り扱い，部分最適と全体最適を実現する．要件３より，構成変更の影響を考慮し，さらに複数案候補の中から選択するメカニズムが必要である．

### 4.1 ADL によるコンポーネント仕様記述

本研究では，ADL（アーキテクチャ記述言語, Architecture Description Language）[7]を活用した動的再構成に着目する．ADL とは，コンポーネント群からなるシステムアーキテクチャ全体や個々のコンポーネントの仕様を形式的に定めるための言語である．ADL は図２のような抽象度の高いレベルでの記述を行えることから，複雑なシステムを構造の観点から捉えることでシステムの詳細を隠蔽し操作できる利点がある．

さらに ADL は計算機による分析ができ，システムの非機能特性への影響や，システムの完全性・一貫性を検査することができる．これらの特徴から，自己適応システムのコンポーネント制御に有効とされている[4,12,17]．本研究では ADL Darwin [7,18]を参考としている． Darwin [4,12]はの参照モデルにおいても使用されている ADL である（図３）．図３では Darwin による簡単な記述例を示しており，個々のコンポーネント仕様，アーキテクチャ全体と個々のコンポーネントの接続関係が表現されている．

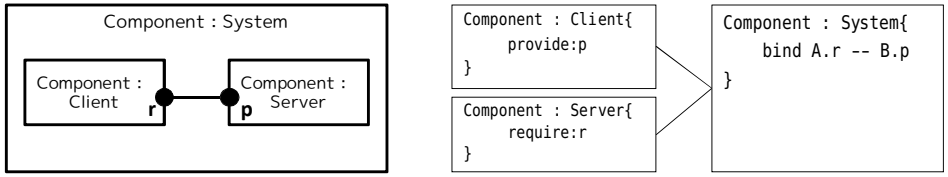


図 3　Darwin によるコンポーネント記述の例

### 4.2 自己適応 ADL 記述の提案（１）

ADL は様々な研究グループによって独自のものが定義されており[7], 自己適応システムの設計に使用されているが，未だ決め手となるようなものはない．特に近年の自己適応システムの研究では ADL を活用した手法も多々見受けられるが[4,14], ADL の記述自体を自己適応システムに特化させている研究は少ない．特に構成変更には，システム自身が各コンポーネントが持つ意味（機能・非機能特性への貢献）を評価し管理する必要がある（要件２）．しかし従来の ADL は各コンポーネントの依存関係とシステム構成を記述するのみであり，意味や状態をシステムが評価するには不十分である．そこで，自己適応に特化した ADL 記述として，新たに表１（右）の属性を導入する．

表 1　自己適応システム向け ADL の記述方式

| 【既存のADL記述例】 | 【追加する自己適応向けADL記述】 |
|---|---|
| Component　コンポーネント名{ | condition : コンポーネント状態 |
| require　: 要求コンポーネント | type　: 責務グループ |
| provide　: 提供コンポーネント | operation : 実行するメソッド |
| } | NFCont　: 非機能特性への貢献 |
| | NFBound　: 使用される条件 |

自己参照性を提供する *condition*，コンポーネントが提供・貢献する機能を意味する

Vol.2011-SE-172 No.8
Vol.2011-EMB-21 No.8
2011/5/18

*type*,*operations*,非機能要求への貢献 *NFCont* とそのコンポーネントが使われるべき状態を表す *NFBound* を既存の ADL 記述に加える．この提案により独自の ADL 記述となることから表１の属性を解析するためには独自パーサを導入する必要がある．

### 4.3 コンフィグレーションの決定（２）

本研究では 4.2 でのパーサを用いた，再構成計算アルゴリズムを導入する．以下の図４（左）に本手法の概観フローを示し，以下に提案をする再構成時のコンフィグレーション割り出し手順について述べる．

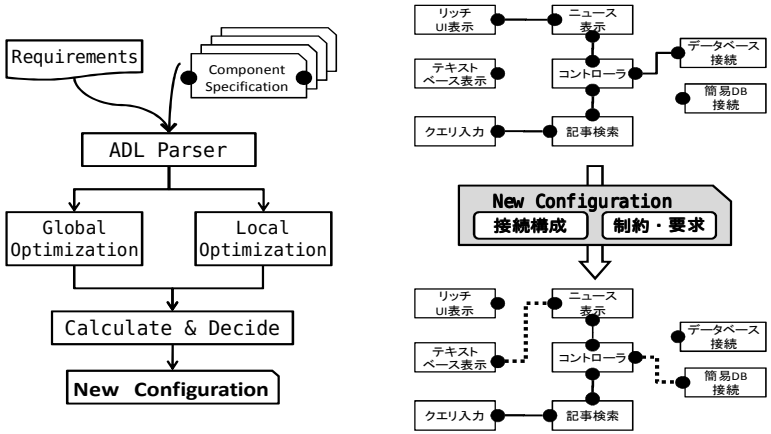


図 4 再構成手順概要（左） 再構成案の適用例（右）

【STEP１】: システムを再構成するイベントが生じた際には，システムに与えられる要求と全てのコンポーネント情報を集める．

【STEP２】: ADL 記述を読み取り解析をする．その後，与えられた要求をもとに計算をし，部分最適・全体最適案から最良の構成案・変更操作案を選び決定する．

【STEP３】: STEP２の結果を新たな構成とし，その設定に基づきシステムの構成変更を実行する(図４右).

部分・全体最適案の２つのアルゴリズムについては５章にて，シナリオ上でのイベントを想定して詳細を述べる．

### 4.4 提案手法の有効性の検討

本手法では，個々のコンポーネントが持つ機能や非機能要求への貢献をシステム自身が評価し，構成変更に必要な情報を管理する．さらに ADL 記述を活用し，各コンポーネントの接続案を実際の構成と対応付けて構成案の算出をする．これらにより，システムが実行時に，自身が持つコンポーネントの意味や情報を理解した上で，現在の状態に最適な構成と操作案を割り出すことができると考えられる．また一般に，非機能特性はソースコードレベルで取り扱うことが難しいとされており，アーキテクチャレベルでの操作をする ADL への記述は有効であると言える．

## 5. 再構成アルゴリズムの検討

本研究で提案する再構成手法とそのアルゴリズムについて,例題シナリオを想定し，検討する．例題として，位置情報を活用した意見共有 Web システムを想定する．

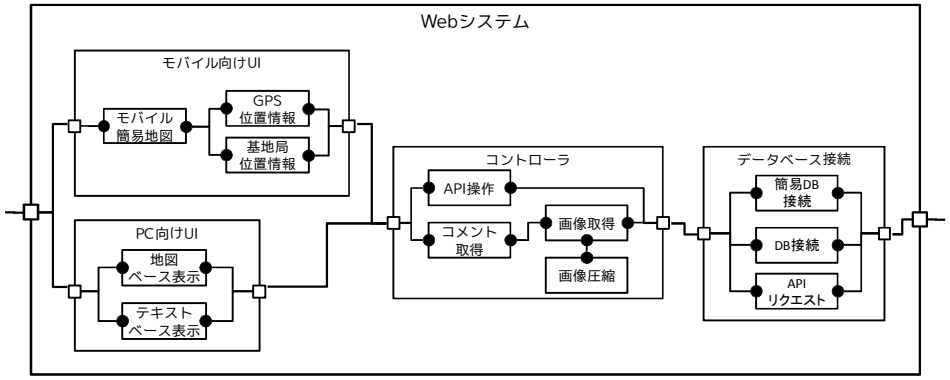


図 5 意見共有 Web システムのコンポーネント図

### 5.1 意見共有 Web システム

図５は対象とする例題シナリオの意見共有 Web システムをコンポーネント接続により表わしたものである．システムはユーザに対して，携帯端末等から閲覧するモバイル向けとフルブラウザにより閲覧をする PC 向けの２つのインタフェースを提供している．ユーザからアクセスがあった場合にはその位置情報を検出し，近隣の位置に投稿されているコメントや画像をデータベースから取得して表示を行う．

意見共有 Web システムは代表的な Web の３層クライアントサーバモデルにより構成され，インタフェース部，処理部，データベース接続部の３つのレイヤーがある．各レイヤーやサーバは正常稼働中をしているかに加え，CPU 使用状況，全体のパフォーマンスに関する値をモニタリングが可能と設定する．

### 5.2 コンポーネント記述例

4.2.で提案するコンポーネント記述に従って，図５例題シナリオの各コンポーネントとアーキテクチャ全体の記述を行う．

```
Component : PCMap{
   require:clientAccess
   provide:controller
   condition:dynamicparam
   type:pcInterface
   Operation:displayPCMap
   NFCont:Usability+,throghput-
   NFBound:CPU='Low',PCAccess
}
```

図 6　コンポーネント記述例（地図ベース表示コンポーネント）

図６は地図ベース表示コンポーネントを提案する ADL 記述で表現した例である．このコンポーネントはクライアント側からのアクセスに対して，地図インタフェース（GoogleMap を想定）によってデータベースの意見を表示する．リッチクライアントなインタフェースのためクライアントとサーバ間でのデータ通信量が多く，比較的処理が重い操作を行う．コンポーネントの接続関係は，クライアントからのアクセスとデータベースからコメント取得等を行うコンポーネントに接続されている(*requrie,provide*)．コンポーネントの状態に関しては，システム実行時に動的に決定されるため，コンポーネント設計時には dynamicparam と設定する．行うべき責務(*type*)は PC 向け画面のインタフェースを提供することであり，メソッド(*Operation*)，displayPCMap()が実装されると定める．このコンポーネントの非機能要求への貢献については，ユーザにとって使い易さを提供する反面，処理が重くなり全体のスループットに影響を与える(*NFCont*)．本コンポーネントが使われるべきタイミングとして，PC のフルブラウザからアクセスが発生し，かつ CPU 使用率が低い場合に適用されることが望ましい(*NFBound*)．

図７は意見共有システムのシステム全体のアーキテクチャの記述例である．このアーキテクチャ記述には，全レイヤーの処理手順（責務を実行する順序）について記述している．

```
arch1:mobileinterface->getPosition->controller->getComment->getGraphic->database
arch2:pcinterface->controller->getComment->getGraphic->contAPI->database->reqAPI
```

図 7　アーキテクチャ全体の記述例

arch1 はモバイル用であり，arch2 は PC からの接続を表わし，異なるコンポーネント接続の状態を表している．このアーキテクチャ全体の記述によって，再構成時に必要なコンポーネントを各責務に基づいて取得することが可能になると考えられる．現段階では図６のような記述であるが，最終的には形式的な手法を導入する予定である．

### 5.1 部分代替アルゴリズム

構成変更時に部分的なコンポーネント入れ替えを実現するアルゴリズムについて検討結果を述べる．部分的な構成変更が有効な場合として，部分的な故障や障害が想定される．

（１）部分代替シナリオ

意見共有 Web システムに起こりうるトラブルや障害としては，接続しているデータベースから突然応答がなくなることや，外部 API サーバとの間に何らかのトラブルが生じ，API リクエストが正常に行えない場合が考えられる．部分的な変更を正しく行うためには，行うべき責務，接続関係を管理し制御する必要がある．図６のコンポーネント記述の責務を表わす *type*，接続関係を表わす *requrie,provide* をもとに部分的な交換を実現する．図８は意見共有 Web システム内で部分的な交換が可能なコンポーネントを指し示している．交換可能なコンポーネントは共通する責務と接続関係を参照し，交換可能かを判断できる（図９）．メインで使用しているデータベースから応答がなく，「DB 接続」コンポーネントが正常に動作しない場合には，同じ責務と接続関係を持つコンポーネントである「簡易 DB 接続」コンポーネント（図５，図９参照）と交換できることが望ましい．

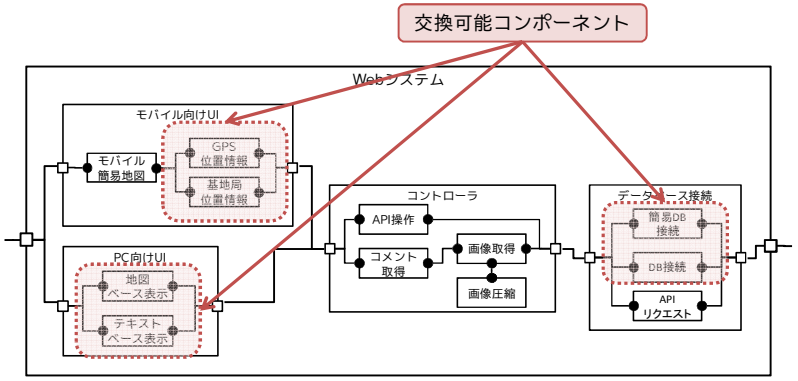


図 8　交換可能コンポーネント例

Vol.2011-SE-172 No.8
Vol.2011-EMB-21 No.8
2011/5/18

また，提案する ADL の *condition* 値については，図１０のように使用することを想定している．データベースからのデータ取得を行う「DB 接続」コンポーネントから応答がない場合には，以後 *condition* の値を unable とし，使用不可能コンポーネントへと変更し計算対象から除外をする．

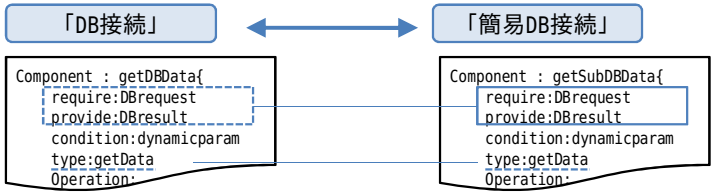


図 9　共通する責務・接続関係による部分代替の例

```
Component : getDBData{
   require:DBrequest
   provide:DBresult
   condition:dynamicparam
   type:AccessDB
   Operation:getData()
   NFCont:throghput-
   NFBound:
}
```

```
Component : getDBData{
   require:DBrequest
   provide:DBresult
   condition:unable
   type:AccessDB
   Operation:getData()
   NFCont:throghput-
   NFBound:
}
```

図 10　*condition* の変更例

### 5.2 全体構成アルゴリズム

システム全体に影響がでるイベントが発生した際の構成変更を実現する全体最適アルゴリズムについて検討結果を述べる．意見共有 Web システムシナリオにおいては，（2）システム全体に関するシナリオ，（3）機能追加シナリオ（要求変更に伴うシステム複数箇所に及ぶ変更等）が挙げられる．

（2）システム全体のシナリオ

システム全体が関与する構成変更のシナリオとしてはスループットや CPU 使用率が考えられる．システムにアクセスが集中し，CPU 使用率が上昇しスループットが低下するというシナリオを想定する．この場合の対応策として，意見共有システムでは，①重い処理を行う「地図ベース表示」コンポーネントでは対応しきれなくないため，軽量の「テキストベース表示」コンポーネントに切り替える，②「画像取得」コンポーネントに「画像圧縮」のコンポーネントをつけ，低画質・容量の画像の表示に切り替える，というシナリオを想定する．①の場合では「API 操作」と「API リクエスト」コンポーネントも影響を受けるため，アルゴリズムの中に構成変更の影響を考慮した処理を行う必要があると言える．さらに①と②の複数候補案が算出できた場合にはその中から一つに決定するメカニズムが必要となる．

（3）機能追加シナリオ

自己適応システムに期待される能力の一つとして，機能追加・削除等の拡張性が求められている[1,4,14]．実行時にできる限りシステムを止めずに機能性を変更することも自己適応システムにおいては重要な評価シナリオである．　意見共有 Web システムをセキュリティ面で強化するために，ログイン機能を持たせ，ログインページよりアクセスをし，ユーザ情報を管理する認証 DB によってパスワードを照合をするとする．そのため本シナリオでは図４のコンポーネントに加えて，図１１のコンポーネント群を追加すると考える．コンポーネント追加以後は，これらのコンポーネントがシステムを構成する要素の一部となることから，再構成の際にこれらを含め，計算する必要がある．その際にも，コンポーネントに記述した接続関係や責務を読み込み利用することで，追加するコンポーネントを正しい位置に追加し，構成計算を行うことが可能になると考えられる．

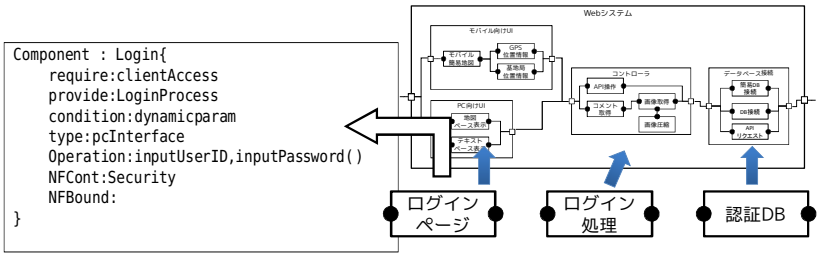


図 11　新規追加コンポーネント群

## 6. 関連研究

自己適応システムの構成変更メカニズムに関して，様々な手法を使用したものが提案されている．

モデル検査技術をプランニングに応用し，コンポーネントの接続構成プランを導出する研究 [4,11,12,13,14]がある．これらはアーキテクチャを ADL によって記述をし制御しているが，複数の代替案を計算することや，システムの非機能要求に基づいた

再構成を行うことについては未対応であり，要件３を満たしているとは言えない．特にプランニング技術に LTSA[5]や Model-Based-Planning [3]などのモデル検査技術を使用していることから，非機能要求を取り扱うことは難しいと言える．

ペトリネットと形式手法を使用し，外部環境の状態に合わせて行う処理のフローを変える手法がある[17]．この研究ではソースコードの処理をペトリネットと形式記述によってモデル化することで制御し，対象シナリオであるネットワーク環境の状態に応じて適応する手法，またそれを用いたモデル駆動開発手法を提案している．しかしこの手法では，プログラムロジックレベルで言及しているため，機能の追加・削除に対しては対応できない手法であるといえ，さらに Web システムのような多くのコンポーネントやサービスが連携するようなシステムへの適用は難しいと考えられる．

システムの要求と合致した再構成をするために，要求分析手法を用いて機能の取捨選択やトレードオフの関係にある非機能特性を制御する手法[15]がある．この手法は要求に対して部分的または全体的に機能の取捨選択をし，最良の構成を算出するが，要求分析上での提案にとどまり，実装するコンポーネント単位での操作を考慮していないため，システム実行中の動的な構成変更や機能追加には対応できない．よって要件１，２を満たすことができない手法だと言える．

[4,12]を発展させた近年の研究成果として，ADL 記述に非機能特性の記述を加え，効用関数値を加味し，接続構成案を計算をする手法[13]がある．しかし，この手法ではアーキテクチャレベルでコンポーネント接続構成を操作することやコンポーネント全体の最適構成計算には未対応であることが著者によって報告されている．

## 7. まとめと今後の方針

### 7.1 検討結果のまとめ

本研究では，自己適応システムに求められる性質に着目し，ADL 記述を活用した再構成手法の提案について方針を示した．

自己適応システムは自身の状態を実行時に参照し，イベントに対して構成変更を実現することから，自己適応システムに特化したアーキテクチャ記述が必要であると考えられる．コンポーネントへの意味づけを行うことで，構成変更時にシステムがコンポーネントの性質を評価した上で計算を行うことが可能になると考えられる．

さらに自己適応向けアーキテクチャ記述を活用した部分代替・全体構成アルゴリズムといった２種類のアルゴリズムについて提案をした．構成変更アルゴリズムの検討結果としては，部分代替アルゴリズムではコンポーネントの責務と実行時の状態を管理することが重要であると言え，全体構成アルゴリズムでは，構成変更時の影響分析を行うこと，複数候補案から１つの案を決定することが重要であると言える．

### 7.2 今後の方針について

今後は提案した ADL 記述を解析するための処理系であるパーサ，２つのアルゴリズムで実装をする予定である．その後，実際に稼働するアプリケーションに組み込むことで提案手法の有効性について評価をしたい．また，手法の発展性として，形式仕様記述等の導入を検討している．自己適応システムの構成変更時の問題の一つとして，構成変更後のシステムの整合性・一貫性を正しく保証しなければならない．さらに要求を形式的に記述し，パーサによって解釈をすることで実行時に変化しうる要求の変化を反映することができると考えられる．実際のアプリケーションでのケーススタディを重ね，実用性の高い手法に発展させていきたい．

## 8. 参考文献

1) B. H.C. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee ,et al.: *Software Engineering for Self-Adaptive Systems:A Research Road Map*, Dagstuhl Seminar 2008,pp1-26
2) S. Dobson, S. Denazis, Fernndez, Antonio, D. Gati, E.Gelenbe, Massacci, P. Nixon, F. Saffre, N. Schmidt and F. Zambonelli : *A survey of autonomic communications*,ACM Trans. Auton. Adapt. Syst., (2006), pp. 223-259.
3) F. Giunchiglia et al. : *Planning as Model Checking*. In 5th European Conference on Planning: Recent Advances in AI Planning, 1999.
4) J. Kramer ,J Magee: *Self-Managed Systems: an Architectural Challenge*, FOSE'07, pp.259-268, IEEE,2007
5) LTSA - Labelled Transition System Analyser, http://www.doc.ic.ac.uk/ltsa/
6) A. McVeigh, J. Kramer, and J. Magee. : *Using Resemblance to Support Component Reuse and Evolution,* In Proc. of SIGSOFT/FSE Workshop on Specification and Verification of Component-based Systems, (SAVCBS '06), ACM,2006
7) N. Medvidovic,R. Taylor: *A Classification and Comparison Framework for Software Architecture Description Languages*,Journal of IEEE Transaction on Software Engineering, vol. 26, no.1, IEEE,2000
8) M. Morandini, L. Penserini, and A. Perini.: *Towards goal-oriented development of self-adaptive systems*. In SEAMS '08: Proceedings of the 2008 international workshop on Software engineering, 2008.
9) IBM : *An architectural blueprint for autonomic computing*, April 2003. 8.
10) M. Salehie, L. Tahvildari: *Self-adaptive software: landscape and research challenges*, ACM Transactions on Autonomic and Autonomic Systems (TAAS), 4:2, pp. 1-42, May 2009.
11) D. Sykes, W. Heaven, J. Magee, and J. Kramer. : *Plan-directed architectural change for*

Vol.2011-SE-172 No.8
Vol.2011-EMB-21 No.8
2011/5/18

*autonomous systems,* In Proc. of SIGSOFT/FSE Workshop on Specification and Verification of Component-based Systems. (SAVCBS07)ACM,2007

12) D. Sykes, W. Heaven, J. Magee, and J. Kramer : *From Goals To Components : A Combined Approach To Self-Management*, Proceedings of ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (ICSE - SEAMS 2008)

13) D. Sykes, W. Heaven, J. Magee, J. Kramer: *Exploiting Non-Functional Preferences In Architectural Adaptation For Self-Managed Systems*, SAC 2010, March 2010, Sierre, Switzerland.

14) H. Tajalli, J. Garcia, G. Edwards, N. Medvidovic:PLASMA: *a Plan-based Layered Architecture for Software Model-driven Adaptation*, ASE2010, pp467-476, ACM,2010

15) Y. Wang, J. Mylopoulos: *Self-repair Through Reconfiguration: A Requirements Engineering Approach*,ASE 2009, pp257-268,IEEE,2009

16) D. Weyns, S. Malek, J. Andersson: *FORMS: a FOrmal Reference Model for Self-adaptation*, Proceedings of the 7th International Conference on Autonomic Computing and Communications, Washington, DC, USA, 7-11 June 2010

17) J. Zhang,B. H.C.Cheng: *Model-Based Development of Dynamically Adaptive Software*, ICSE2006, pp371-380, ACM, 2006

18) J. Magee, J. Kramer : *Dynamic structure in software architectures*, Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering,1996