情報処理学会論文誌 Vol.53 No.10 2328–2344 (Oct. 2012)

ゴール指向要求記述の整形に基づいたソフトウェアシステム進化手法

中川 博之[1,a)] 大須賀 昭彦[1,b)] 本位田 真一[2,3,c)]

受付日 2011年12月19日, 採録日 2012年7月2日

概要：近年，ソフトウェアシステムを取り巻く要求や環境の変化に対応するために，ソフトウェア進化（Software Evolution）の効率的な実現が求められるようになっている．そこで本研究では，進化を考慮したソフトウェアシステム開発プロセスとして，制御モデルとして知られる Control loop をシステムの構成要素とした，機能単位での拡張が容易なソフトウェア開発プロセスを提案する．本開発プロセスにおいては，要求が記述されたゴールモデルに対して整形プロセスを定義することにより，独立した機能提供が可能な Control loop を抽出し，Control loop により構成されるシステム構成を決定する．本論文では，ゴールモデルの整形プロセスを中心に提案する開発プロセスについて論じ，KAOS モデリングツールである k-tool を対象としたソフトウェアシステム進化実験の結果を示すことで，提案手法の有効性を評価する．

キーワード：ソフトウェア進化，ゴール指向要求分析，Control loop

A Software Evolution Method Based on Goal-oriented Requirements Description Forming

Hiroyuki Nakagawa[1,a)] Akihiko Ohsuga[1,b)] Shinichi Honiden[2,3,c)]

Received: December 19, 2011, Accepted: July 2, 2012

Abstract: Software evolution has recently attracted attention in order to adapt the changes in their environments or requirements changes. This paper describes our approach to extracting control loops, which constitute extensible systems. Our method is based on the goal-oriented requirements description, and it provides a technique for elaborating the goal model in order to identify the control loops in the target system. We evaluate our method experimentally and show that it helps to evolve software systems through the development process.

Keywords: software evolution, goal-oriented requirements modeling, control loops

1. はじめに

近年，ソフトウェアの活躍する場面が広がり，オンラインバンキングシステムや電子商取引システムなどの Web 上のシステムだけでなく，携帯端末や自動車の制御システムなど，我々が生活するあらゆる局面において，ソフトウェアシステムが動作している．ソフトウェアにはハードウェアのような摩耗や劣化，損傷がない一方で，運用開始後もエラーの修正が求められるとともに，環境や要求の変化にともなう継続的な変更が期待されている．

Lientz ら [1] は，ソフトウェアシステムの変更をともなう活動を，環境変化への適応（Adaptive），新しい要求に対する拡張（Perfective），エラーの修正（Corrective），将来発生しうる問題の予防（Preventive）の 4 種類に分類して

[1] 電気通信大学
The University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[2] 国立情報学研究所
National Institute of Infomatics, Chiyoda, Tokyo 101–8430, Japan
[3] 東京大学
The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
[a)] nakagawa@uec.ac.jp
[b)] ohsuga@uec.ac.jp
[c)] honiden@nii.ac.jp

いる．これらの活動の約 75% を適応，拡張が，約 21% を修正が占めているという調査結果もある [2] が，近年のソフトウェアシステムにおいては，システムの長寿命化により，要求や環境の変化による機能の追加や変更，つまり適応や拡張という**ソフトウェア進化（Software Evolution）**の側面がますます重要となってきている．特に，Parnas [3] が “Software aging” と呼び，Lehman ら [4] が “Laws of software evolution” の 1 つとして定義しているように，ソフトウェアシステムの品質は徐々に低下するという性質を持ち，進化はソフトウェアシステムのライフサイクルにおいて必須のアクティビティと考えなければならない．また，長寿化の一方で，ソフトウェアシステムを取り巻く環境の変化も激化していることからも，要求や環境の変化に対応するためのソフトウェア進化を確実かつ効率的に実現するための開発プロセスが求められるようになってきている．

そこで本研究では，ソフトウェア進化に対応するための効果的な開発法として，ゴール指向要求記述（以降，ゴールモデル）を活用した開発プロセスを提案する．本研究では特に，制御モデルとして知られる Control loop をシステムの構成要素とした，機能単位での拡張が容易なソフトウェアシステムを実現するために，ゴールモデルの整形プロセスを導入し，Control loop の同定法と進化時の設計・実装支援のための各種情報の抽出法を検討する．

本論文の構成は以下のとおりである．まず 2 章で，ソフトウェア進化の観点から開発プロセスに求められる要件を定義し，既存研究の問題点を指摘する．また，2 章では本研究のアプローチについても述べる．続く 3 章では，提案する開発プロセスで用いるゴールモデルの整形プロセスについて説明し，4 章で提案手法を利用した進化実験の実験結果を示す．5 章で提案手法の有効性と適用範囲について議論し，最後に 6 章で結論を述べる．

2. ソフトウェア進化を考慮した開発プロセス

2.1 開発プロセスに求められる要件

はじめに，ソフトウェア進化の観点からシステム開発プロセスに求められる要件を議論する．まず，Lindvall ら [5] が実証実験により示しているように，経験を積んだソフトウェア開発者であっても，変化によってソフトウェアシステム上にもたらされる影響を分析するのは容易ではない．したがって，ソフトウェア進化を許容する開発プロセスにおいては，進化により生じる影響の分析を支援する必要がある．また，ソフトウェア進化には，小さな機能変更から初期開発時には想定していなかった大がかりな機能追加まで様々なものがあるが，既存のシステム構成要素の予期しない変更を避けるためには，不適切なアーキテクチャの変更を避ける必要がある．同様に，ソフトウェアシステムを進化させる場合，システム内での変更箇所が他の構成要素に副作用として悪い影響を与えないような変更が必要となる．

そこで本研究では，ISO9126 [6] に示された品質特性や Breivold ら [7] の進化性に関する議論をもとに，ソフトウェア進化に対してシステム開発手法に求められる要件として，以下の 3 つの要件を定義し，これらを満たすソフトウェアシステム開発プロセスの実現を目指す．

定義 2.1　ソフトウェア進化の観点から開発プロセスに求められる要件

- **要件 1：分析可能性（Analyzability）**変化によって生じる影響を分析することができる．
- **要件 2：整合性（Integrity）**進化時に，現在のシステム構成に矛盾するような過剰な変更を防ぐことができる．
- **要件 3：変更容易性（Changeability）**変更の範囲を効率的に限定することができる．

2.2 関連研究

分析可能性を追求するためには，進化の要因となる要求の変化を分析し，その結果から設計モデル上での変更箇所を判断する手段が有効であると考えられる．van Lamsweerde [8] は，ゴール指向要求分析 [9] の結果を設計モデルに反映する手段として，ゴールモデルからシステム構造を表現するアーキテクチャモデルの構築指針を提案している．この手法では，各ゴール達成の責務をコンポーネントに割り当て，ゴール間のデータフローからコンポーネント間接続を同定することにより，システムの構造を決定する．ただし，このような構築法ではゴール達成の責務を割り当てるコンポーネントを決定する必要があり，この判断は容易ではない．過度に多くのゴールを割り当てると，コンポーネントの単位が大きくなり，変化の影響範囲を明確に分析できないし，逆にゴール達成の責務を細分化しすぎると，コンポーネント数が多くなり，コンポーネント間に必要以上の依存関係が混入することとなる．

同じくゴールモデルを利用したシステム構成決定法として，Yu ら [10] の研究がある．Yu らの手法では，ゴールモデル中のゴールをコンポーネントに変換し，ゴール間の接続関係からコンポーネント間接続を決定する．この手法は，ゴールモデルに 1 対 1 対応するシステム構成を決定可能であるという点で，設計モデル上での変更箇所を同定できるという観点から一見有効であるように思われる．しかし，Yu らの手法においては，ゴールモデルの条件や変換対象とするゴールの範囲については言及していないため，コンポーネントの独立性や責務が十分に明確化されないことになる．その結果，ゴールモデルに新たなゴール群を追加したとしても，ゴールモデル上での変更をシステムの設計・実装上でどう扱うべきかは明らかではない．また，ゴールに 1 対 1 対応してコンポーネントを変更するため，類似のゴールが複数あった場合に，それらが共通の変数をアクセ

スすることで競合発生の可能性が生じることとなる．

ここであげた 2 つの手法の問題は，分析可能性や変更容易性を低減させる要因となるものであり，これらはゴールモデル上で機能実現や変数に対する責務を適切に分離する難しさに起因している．したがって，進化を扱う場合には，進化の要因となる要求変化と，機能実現，変数に対する責務とが同時にゴールモデル上で分析・記述できることが求められる．

2.3 本研究のアプローチ

ゴールモデルにおける責務分析を考えた場合，ゴールはシステムの振舞いにより達成される状態を記述したものであることから，ゴールモデル上においては，振舞いの単位での責務割当てが適切であると考えられる．そこで，本研究では，ゴールモデル上で Control loop を同定する手法に基づいた開発プロセスを導入する．

Control loop とは，システムの振舞いに着目した**プロセスコントロールモデル** [11] における制御プロセスを表現したものである．プロセスコントロールは，古くから監視制御システムにおいて用いられてきた概念であり，処理を実現する処理部と処理部を制御する制御部の 2 つの構成要素を持ち，目標の範囲内の出力を維持するために，外部への出力を変化させる処理部に対して，制御部が状態を監視しながら制御を加えるものである．同モデルにおいては，処理部に入力される入力変数（Input variable），プロセスコントロールにより制御したい対象を示す制御変数（Controlled variable），制御のために変更可能な操作変数（Manipulated variable）の 3 種類の**プロセス変数**が定義され，**Control loop** [11], [12], [13] と呼ばれる制御プロセスが形成される．この制御プロセスは，以下の 4 つのアクティビティがこの順序で繰り返し実行されるプロセスである．

- **収集（Collect）**：外部から情報を収集する．
- **分析（Analyze）**：情報の収集結果から，現在の状態を分析する．
- **決定（Decide）**：環境に適応するためのシステム動作（振舞い）を決定する．
- **実行（Act）**：決定結果に基づいた振舞いを実行する．

本研究では，ゴールが表現する達成状態を満足するシステムの振舞いとして，収集，分析，決定，実行といった Control loop の観点からシステムをモデリングすることで，入力から出力までの一連のアクティビティを独立して提供可能な範囲をゴールモデル上で決定する．ただし，Control loop の観点から，進化への対応を考慮したシステムの構成要素を同定するが，これは各構成要素の責務を決定するためのものであり，入力変数の値に応じてアクションを決定するという一連のプロセスを実現できれば，各構成要素の設計・実装をプロセスコントロールモデルに束縛するものではない．

3. ゴールモデル整形による Control loop の同定

本手法ではまず，システムに対する要求をゴールモデル上で分析し，得られたゴールモデルを整形することで，進化による変更が容易なシステム構成を生成するためのゴールモデルへと洗練化する．本章では，まず，本研究で要求記述として用いる KAOS について説明し，その後，整形プロセスと整形後ゴールモデルの活用方法について述べる．

3.1 KAOS

KAOS [14], [15] は Lamsweerde らによって提案されたゴール指向要求分析法であり，システムに対する要求（ゴール）を明示的に記述し，それを詳細化することで，要求のどの範囲をシステム化すべきかの判断を助けるとともに，詳細化の段階で要求間の矛盾の発見や解消を実現するなど，ゴールを達成するための要件を系統的に導出する手法である．

図 1 は KAOS ゴールモデルの記述例である．KAOS 分析ではまずゴールモデルにおいて，システムに対する要求，つまりゴールを単一アクタが実現できる大きさにまで細分化する．細分化には，すべての達成が必要なサブゴールへと詳細化する AND-refinement と，いずれかの達成が必要なサブゴールへと詳細化する OR-refinement の 2 種類があり，十分に細分化されたゴールは，KAOS におけるアクタであるエージェント（Agent）に割り当てられる．

KAOS では，これらの要求を分析・記述するモデル要素のほかに，ゴールとエンティティとの関連（Concerns）を定義するための関係線などを提供している．このように，KAOS においては，ゴールモデルをもとに分析を進めることで，ゴール間の関係だけでなく，設計モデルに該当するモデル要素との関係も分析，定義することができる．

3.2 ゴールモデル整形プロセス

本研究では，進化に有効なシステム構成をゴールモデルから決定することを目的として，ゴールモデル上で Control loop を抽出・同定するために，定義 3.1 に示すゴールモデルの整形プロセスを導入する．

定義 3.1 ゴールモデル整形プロセス

(1) エンティティの追加：エンティティを追加し，ゴールとの関連を定義する．

(2) 共通ゴールの集約：共通ゴールの記述を分離・集約化し，依存関係を記述する．

(3) 主要ゴールの同定：ガイドラインに従って，主要ゴールを同定する．

(4) 充足性判定：判定条件を満たす主要ゴール群を決定する．

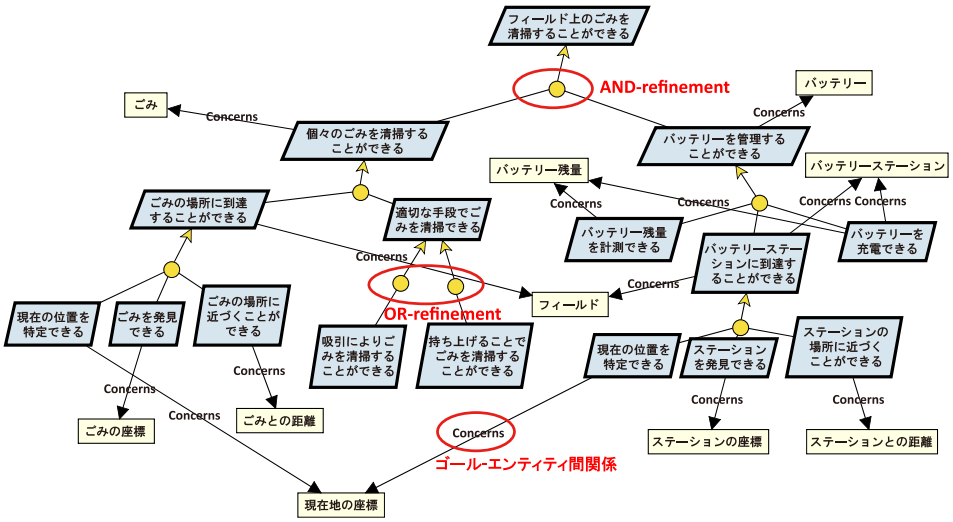

図 1 清掃ロボットに対する KAOS ゴールモデル
Fig. 1 Goal model for cleaning robot.

(5) Control loop の構築：各主要ゴールが Control loop を形成するように構造化する．

(6) 責務の割当て：各 Control loop を責務としてシステムに割り当てる．

以降，整形プロセスの内容を，清掃ロボットに対するゴールモデルを例に説明する．

3.2.1 エンティティの追加

整形プロセスでは，まず，ゴールモデル上に各ゴールに関連するエンティティを追加し，ゴールとエンティティとを Concerns 関係により関連付ける．従来の KAOS 分析では，システムに関連するオブジェクトを導出するために各ゴールに関与するエンティティを記述するが，本研究では，ゴールの集約や Control loop の同定，Control loop 間の競合検出のためにも利用する．図 1 はエンティティ追加後の清掃ロボットに対するゴールモデルである．ゴール「現在の位置を特定できる」については，エンティティ「現在地の座標」が，ゴール「バッテリー残量を計測できる」に対しては，計測対象となる「バッテリー残量」が関連エンティティとして考えられるため，これらのエンティティを定義し，各ゴールと Concerns 関係により関連付けている．

なお，本研究では，すべてのサブゴールとその親ゴールが関与すると判断したエンティティについては，親ゴールに Concerns 関係を集約する．たとえば，「ごみ」は，ごみ清掃に関するすべてのゴールに関与するため，上位ゴールである「個々のごみを清掃することができる」との関係に集約して記述している．

3.2.2 共通ゴールの集約

ソフトウェアシステムの進化を考えた場合，追加される機能や変更が必要な各機能に対して，それぞれ独立に変更要求を記述すると，同様の機能や共通の機能に関する要求記述が分散化，冗長化する可能性がある．したがって，提案する整形プロセスでは，複数箇所に出現する可能性のあるゴールの冗長な定義を避けるために，それらのゴールを集約して共通化する．抽出された共通ゴールへの依存関係は，本研究で導入する “Uses” ラベルにより記述する．たとえばゴール A の達成にゴール B の達成が必要である場合は，ゴール A に対して “Uses B” ラベルを付与する．共通ゴールとして抽出すべきかどうかの判断には，達成すべきゴール，つまり実現すべき機能の類似性や，エンティティとの関係の類似性を用いる．

たとえば，清掃ロボットの例では，ごみ清掃機能やバッテリー管理機能においては，ごみやバッテリーステーションなどの対象に向かって移動する機能が共通で必要となるため，図 2 に示すように，共通ゴール「目標物に到達することができる」を抽出し，共通ゴールへの依存関係を “Uses” ラベルにより表現する．

3.2.3 主要ゴールの同定

共通ゴールを集約すると，続いて，ゴールモデル上に記述されたゴール群から**主要ゴール（Prime goals）**となりうるゴール群を同定する．主要ゴールとは，ゴールの中で，特にシステムが持つべき機能により達成が明示的に期待されているゴールであり，本研究においては，ソフトウェアの進化・変更を考慮して，固有の Control loop を割り当てる対象とするゴールを指す．主要ゴール単位で Control loop を割り当て，これを動作の単位とすることで，進化時に他の主要ゴール，つまり他 Control loop により構成される他のシステム構成要素との依存関係を Control loop 間の関係のみに限定することが可能になる．本研究では，以下のとおり，主要ゴール同定のガイドラインを定義する．

定義 3.2 主要ゴール同定のガイドライン：以下のいずれかの指針を満たすゴール g_i を主要ゴール候補とする．ここで，$Child_{g_i}$ はゴール g_i を親とするゴールの集合であり，$uses(g_x, g_y)$ はゴール g_x 上に “Uses g_y” が定義されていることを，$independent(g_x, g_y)$ はゴール g_x, g_y 双方が自身の

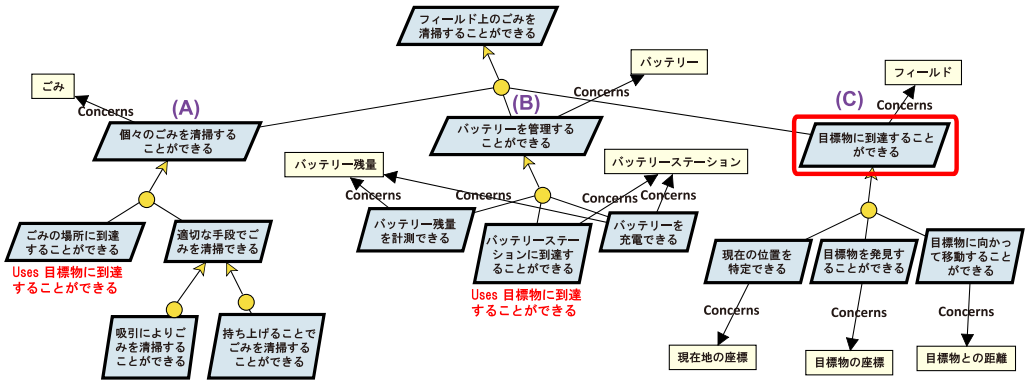

図 2 共通ゴール集約後のゴールモデル
Fig. 2 Goal model after aggregating common goals.

ゴール達成に他方のゴール達成を必要としないことを示す述語である．

- **指針 1**：Uses ラベルにより参照されているゴールである．

$$\exists g_i\ (\exists g_j\ uses(g_j, g_i))$$

- **指針 2**：Divide-and-conquer パターンで分解されたゴールである．

$$\forall g_{ik}\ \forall g_{il}(g_{ik}, g_{il} \in Child_{g_i} \wedge g_{ik} \neq g_{il} \wedge independent(g_{ik}, g_{il}))$$

- **指針 3**：複数のサブゴールが同一エンティティと Concerns 関係にある．

$$\exists g_{ik} \exists g_{il}(g_{ik}, g_{il} \in Child_{g_i} \wedge \exists ent(ent \in Entity \wedge concerns(g_{ik}, ent) \wedge concerns(g_{il}, ent))) \square$$

定義 3.2 で示した各指針は，主要ゴール決定の判断を支援するものである．まず，指針 1 は，共通ゴールを主要ゴールとして同定するためのものである．共通ゴールは共通の機能を集約したゴールであり，システムが提供すべき複数の機能の実現に不可避のゴールと判断できることから，主要ゴールとして抽出すべきゴールといえる．指針 2，指針 3 は，ソフトウェアの変更に対する影響を限定化するために，他の主要ゴールとの依存関係を限定させるような主要ゴールを，ゴールの意味的な観点とゴールモデルの構造的な観点から同定するための指針である．指針 2 は，Divide-and-conquer パターン [16] が分割統治法に基づいた問題細分化を実現するパターンであり，詳細化されたゴールは兄弟ゴールとの関連が薄く，独立性が高いと判断できることから，意味的な機能の単位として主要ゴールの候補を同定するものである．一方の指針 3 は，ゴールモデルの構造的な観点から主要ゴールを同定するものであり，Control loop の単位で関心事や操作変数を集約可能なゴール，つまり，エンティティとの関連を包含できるようなゴールを主要ゴールの候補として同定する．このような同定は，複数の Control loop が同一の操作変数を扱うことを避ける効果がある．

図 2 のゴールモデルでは，まず，ゴール (C) は他ゴールから Uses 関係によって利用されているゴールであるため，指針 1 を満たす．また，AND-refinement リンクに着目すると，ゴール「フィールド上のごみを清掃することができる」に関しては，ごみ清掃とバッテリー管理，対象物への移動という独立した目的に分解されているために，Divide-and-conquer パターンが適用されていると考えられ，(A)〜(C) が指針 2 に合致するゴールであることが分かる．指針 3 については，ゴール (A) がエンティティ「ごみ」に対して合致し，ゴール (B) がエンティティ「バッテリー」，「バッテリーステーション」に対して合致し，ゴール (C) がエンティティ「フィールド」に対して合致していることが分かる．以上の判断から，図 2 中の (A)〜(C) が主要ゴール候補として同定される．本例では，指針 2 と指針 3 に該当するゴールが同一となったが，指針 2 と指針 3 に該当するゴールが異なる深さのゴールとなる場合もある．この場合は，それぞれの指針に該当するゴールを主要ゴール候補として同定する．

3.2.4 充足性判定

本研究では，ゴールモデル上の主要ゴールに Control loop を割り当てることで，ソフトウェアの構成要素を決定する．提案する整形プロセスでは，Control loop を割り当てる主要ゴールの集合がソフトウェアの構成要素として十分であるかを判断するために，定義 3.3 に示す主要ゴール集合の決定（同定）に対する充足性判定条件を導入する．

定義 3.3 充足性判定条件：主要ゴール候補のいくつかを選択し，下記の条件 1〜条件 3 がすべて満たされるとき，ゴールモデルが主要ゴールの同定に関して充足されているという．また，このとき，選択された主要ゴール候補を主要ゴールとする．ここで，*FuncReq* は機能要求[*1]の集合を，*PGoals* は選択した主要ゴール候補の集合を，G_i はゴー

[*1] 本研究では，ゴールモデル末端の葉ゴールに記述される，システムが達成すべき状態を機能要求と呼ぶ．

ル g_i を根とするツリーに含まれるゴールの集合を，述語 $conflict(g_1, g_2, ent)$ はゴール g_1 とゴール g_2 間においてエンティティ ent に関する競合が発生するときに真となる述語を示したものである．

- **条件 1**：すべての機能要求がいずれかの主要ゴールを根とするツリーに包含されている．

$$\forall g(g \in FuncReq \wedge \exists i(g_i \in PGoals \wedge g \in G_i))$$

- **条件 2**：いずれの主要ゴールも，他の主要ゴールを根とするツリーに包含されていない．

$$\forall i \forall j \ (g_i, g_j \in PGoals \wedge i \neq j \Rightarrow g_i \notin G_j \wedge g_j \notin G_i)$$

- **条件 3**：主要ゴール間でエンティティに関する競合が発生していない．

$$\forall i \ \forall j(g_i, g_j \in PGoals \Rightarrow \neg \exists ent(ent \in Entities \ \wedge \ conflict(g_i, g_j, ent))) \quad \square$$

ここで，主要ゴールは Control loop を割り当てられるゴールであることから，条件 1 はすべての機能要求がいずれかの Control loop に包含されていることを，条件 2 は Control loop が他の Control loop を包含していないことを示すものである．条件 3 の競合の判定については，まずゴールモデルの構造から形式的に競合の可能性のある箇所を検出し，検出結果から考慮すべき競合であるかどうかを判定する．考慮すべき競合と判断した場合は競合の解消を試みる．競合が解消されない場合は，前プロセスに戻り，ゴールの構造を再整形する．以降，条件 3 の判定法と，競合が発生する場合の解消法について説明する．

エンティティに関する競合

複数の Control loop をシステムに配置した場合，Control loop 間で**競合**（conflict）が引き起こされる可能性がある．本研究では，このような競合をゴールモデル上で検出することを目的として，Entity-conflict パターンを定義する．

定義 3.4　Entity-conflict パターン：以下の条件を満たすエンティティ ent を，競合の可能性のあるエンティティとする．ここで，g_i および g_j は主要ゴール候補である．

$$\exists i \ \exists j(g_i, g_j \in PGoals \wedge \exists ent(ent \in Entity \ \wedge \ \exists k \ \exists l(g_{ik} \in G_i \wedge g_{jl} \in G_j \wedge concerns(g_{ik}, ent) \wedge concerns(g_{jl}, ent)))) \ \square$$

図 3 は Entity-conflict パターンを図示したものである．Entity-conflict パターンは，複数の主要ゴール候補を根とするツリーから `Concerns` 関係が定義されているエンティティを検出するものであり，各主要ゴール候補に Control loop を割り当てた場合に，複数の Control loop から変数へのアクセスが生じる可能性があることを示したものである．Entity-conflict パターンに合致する場合，エンティティと

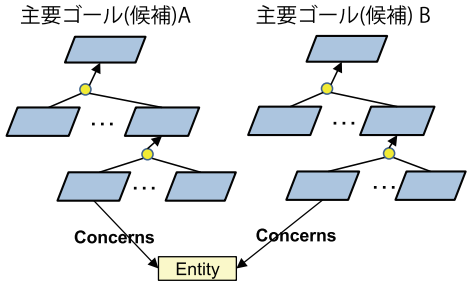

図 3　Entity-conflict パターンのイメージ
Fig. 3　An example of Entity-conflict pattern.

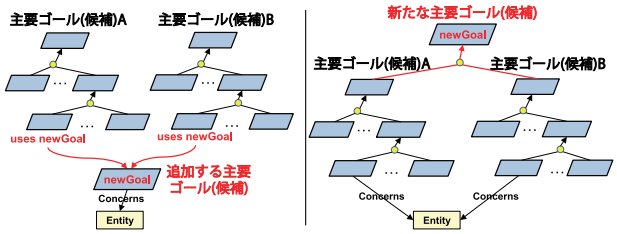

図 4　競合の解消法（左：解消法 1（共通ゴールの導入），右：解消法 2（ゴールの集約））
Fig. 4　Conflict resolution: (Left) Common goals introduction. (Right) Goal aggregation.

主要ゴールの関係から，実際に考慮すべき競合であるかどうかを判断する．図 2 のゴールモデルには Entity-conflict パターンに該当するエンティティは存在しないが，たとえば，図 1 の共通ゴール集約前のゴールモデルにおいては，エンティティ「現在地の座標」が競合の可能性のあるエンティティとして検出される．

競合の解消法

Entity-conflict パターンにより検出された競合をゴールモデル上で解消するには，エンティティとの関係を唯一の主要ゴール候補からの関係に集約する必要がある．そこで，本研究では，Entity-conflict パターンにより検出された競合のうち解消すべきものに関して，以下のいずれかの手段によりゴールモデルを整形する．

- **解消法 1（共通ゴールの導入）**：エンティティに唯一関係するゴールを共通ゴールとして新たに定義する．
- **解消法 2（ゴールの集約）**：競合する主要ゴール候補を 1 つのゴールツリーとして集約し，集約したゴールを新たな主要ゴール候補とする．

解消法のイメージをそれぞれ**図 4** に示す．解消法 1 は，競合するエンティティに対して，アクセスの責務を持つ唯一のゴールを共通ゴールとして新たに導入するものであり，主要ゴール候補が新たに定義された共通ゴールを利用するように修正する．これは，共通ゴールの導入と同様の整形に該当し，共通のエンティティに対する関与を集約することで，Control loop からエンティティへのアクセスの責務集約を目的としたものである．もう一方の解消法 2 は，競合が発生している主要ゴール候補群を集約して，これらを

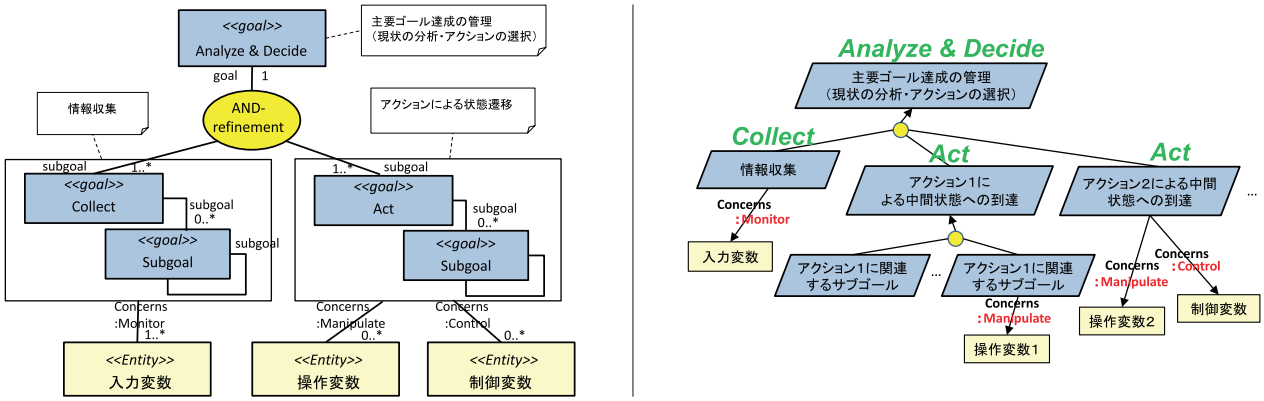

図 5 Control loop パターン（左：一般記述，右：パターンの一例）

Fig. 5 Control loop pattern: (Left) General description. (Right) An example of the pattern.

包含する新たな主要ゴール候補を定義するというものである．この修正は，競合 Control loop 群を統合することで，共通エンティティへのアクセスを制御するものであり，新たな主要ゴール候補，つまり統合した Control loop がエンティティに対してのアクセスの責務を持つことになる．

いずれの解消法を利用するかは，各主要ゴール候補の意味を考慮する必要がある．解消法 1 は，独立した複数の主要ゴール候補が競合している場合の解消法であるのに対して，解消法 2 は，同様の目的を持つ主要ゴール候補を統合する場合に有効である．

3.2.5 Control loop の構築

充足性判定の全条件を満たす主要ゴールが同定されると，各主要ゴールに対して Control loop を割り当てる．本ステップにおいては，Control loop 内において主要ゴールを達成するためのサブゴールを抽出し，記述を整理することで，Control loop 設計のための情報を抽出する．本研究では，このサブゴールの抽出，つまり主要ゴールにおける Control loop に関するアクティビティの抽出のために，ゴール記述パターンを導入する．

Control loop パターン

図 5 は本研究で導入する Control loop パターンを図示したものである．本パターンにおいては，Control loop のアクティビティに対応する 3 種類のゴールタイプを用いる．Control loop パターンの根に該当するゴールは Analyze & Decide タイプに該当し，現在の状態を分析しながら，次にとるべき行動を判断し，主要ゴールを達成することを示している．サブゴールのうち，入力変数に関するゴールは Collect タイプに該当する．Collect タイプ以外のサブゴールは Act タイプに該当し，入力変数の値と現在の目標達成状態，選択された振舞いによって次の状態に遷移することを表すゴールである．

ここで，Collect と Act は環境に関与するアクティビティであるのに対し，Analyze，Decide はシステム内部で閉じたアクティビティである．また，Analyze，Decide がゴールモデル上で状態としての明示的な表現が難しいアクティビティであるのに対して，Collect，Act が明示的に示されていれば，Control loop の割当て場所を同定することができると考えられる．したがって，Control loop 同定の観点からは，Collect タイプと Act タイプのゴールを集約，つまりサブゴールとして持つゴールに Control loop を割り当てる．このとき，同ゴールは Collect，Act タイプゴールの達成状態を利用することで達成可能なゴールと判断することができる．また，プロセスコントロールモデルにおける制御部と処理部の役割を考えると，Analyze と Decide は制御部の役割となり，システム上では同一コンポーネントとして実装される場合も多い．以上の観点から，本研究では設計モデルとの連携も考慮して，Analyze と Decide の役割を同ゴールに割り当てる．

開発者は，Control loop パターンの根のゴールに各主要ゴールを対応付け，Control loop パターンのゴール分割に従って，主要ゴールのサブゴールを整理する．この際，2.3 節で述べたプロセス変数も明確化する．したがって，Control loop パターンに従ったゴール整形においては，プロセス変数を同定しながら，Control loop の各アクティビティに該当するゴールを同定し，それらの関係を Concerns 関係を用いて定義する．本研究では，**表 1** に示すラベルを用いて Concerns 関係を詳細化する．もし，新たなプロセス変数が同定され，複数の Control loop から操作変数・制御変数としての関係が定義されていれば，図 4 に示す解消法に従って，プロセス変数の割当てを再検討する．

3.2.6 責務割当て

通常の KAOS では，ゴールモデルの下層に位置する十分に分解されたゴールが機能要求として抽出され，システムに対する責務として，各アクタに割り当てられる．本研究では，KAOS における責務割当てを拡張し，システムに主要ゴールを責務として割り当てる．これは，Control loop を包含するサブツリーを責務として割り当てることを意味し，したがって，システムに Control loop 単位での責務を

表 1 Concerns 関係詳細化ラベル
Table 1 Labels specifying relationship with process variables.

ラベル	ゴールタイプ	エンティティ	Concerns 関係の意味付け
:Monitor	Collect	入力変数	入力変数の値を決定する情報収集
:Manipulate	Act	操作変数	操作変数の変更に関与するアクション
:Control	Act	制御変数	実行結果により，制御変数を変更させる可能性のあるアクション

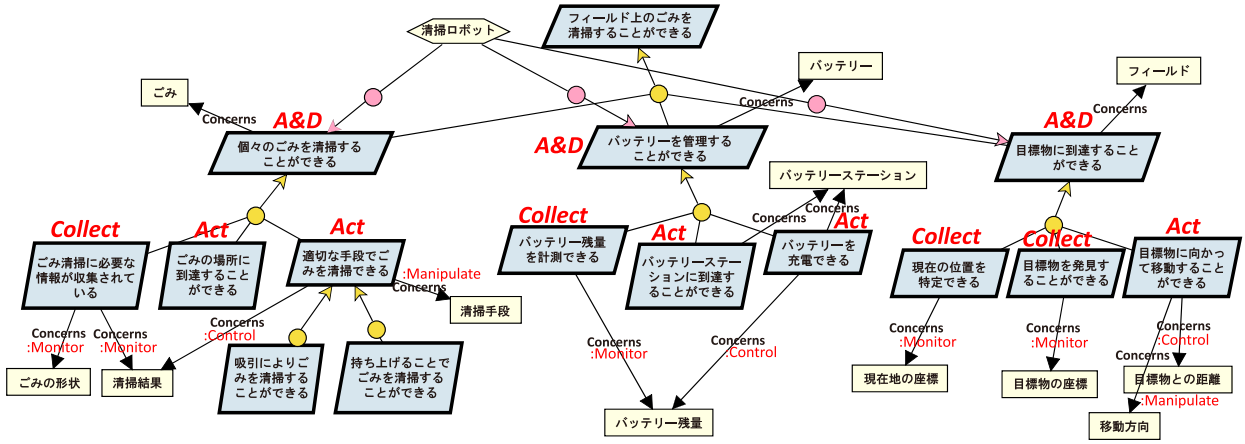

図 6 責務の割当て
Fig. 6 Responsibility assignment.

割り当てることに該当する．たとえば図 6 のゴールモデルでは，同定した各主要ゴール（図中で A&D（Analyze & Decide）と記された 3 つのゴール）を清掃ロボットを表現するエージェントに割り当てている．

3.3 ソフトウェア進化を考慮した開発プロセス

2.1 節で定義した開発プロセスに求められる要件に考慮し，本研究では，ゴールモデル整形プロセスを用いた以下のようなソフトウェア開発プロセスを定義する．

(1) ゴールモデル構築，整形：システム開発者はまず，開発対象となるシステムに対する要求をゴールモデル上で分析する．このフェーズの目的は，システムに対する要求の分析・定義と，Control loop を抽出するための要求記述の構造化にある．要求分析法や要求，つまりゴールの記述は通常の KAOS における分析法，記述法に従うが，本開発プロセスでは，得られた KAOS のゴールモデルに対して，本研究で導入する整形プロセスに従った整形，詳細化を施すことにより，システム構成を決定可能なモデルへと詳細化する．ソフトウェア進化時においても，機能追加などの進化に関する要求をゴールモデル上に記述し，追加した要求に対して，整形プロセスを再適用する．

(2) Control loop 設計：続いて，前フェーズで整形した KAOS ゴールモデルから，システム構成（コンフィギュレーション）を決定する．文献 [17] の手法を用いることで，整形されたゴールモデルをもとに，Darwin コンポーネント表記 [18] によるコンフィギュレーションを生成することができる．この手法では，まず，整形プロセスにより責務割当てをしたゴール群をコンポーネントに変換し，AND/OR-refinement リンクをコンポーネント間のとりうる接続関係に変換する．その後，Uses ラベルに従って，利用関係にあるコンポーネント間を接続する．このような変換により，Control loop パターンに従ったゴールモデルの階層構造と Control loop 間の依存関係をもとに，整形された KAOS モデルから，1 つ以上の Control loop を含んだコンフィギュレーションの決定が可能である．各 Control loop の振舞いは，該当する主要ゴール以下のゴールおよびエンティティとの関係情報をもとに設計する．各 Control loop の設計・実装は，入力変数の値に応じてアクションを決定するという一連のプロセスを実現できれば，プロセスコントロールモデルに束縛するものではない．ゴール記述から振舞いモデルを構築する手段としては，たとえば，文献 [19] などの手法が参考になる．この手法を用いる場合は，ゴールを時相論理記述に形式化し，振舞いモデルとして LTS（Labeled Transition System）を構築することとなる．また，プロセスコントロールモデルに従った設計をする場合は文献 [11] が参考になる．Uses ラベルにより被利用側となる Control loop については，必要に応じて Control loop の利用に関する競合を避けるために，優先度によるサービス提供や依頼順序に従ったサービス提供など，利用側 Control loop を決定するための手段やインタフェースを決定させる．

進化時においては，進化前後のゴールモデルや生成されるコンフィギュレーションの差分をとることで，進化に対する要求の変化やコンフィギュレーションの変化の情報を

明確化することができるため，これらを実現するための設計・実装モデル上の変更箇所を同定する．

(3) Control loop 実装：開発者は各 Control loop の動作を決定し，Control loop を実装する．また，実装した Control loop を実行環境に配備することで，システムを動作させる．進化時には，同定された変更箇所を実装し，変更に関与する Control loop のみを変更，つまり該当 Control loop の追加，削除，あるいは修正をする．

4. k-tool 進化実験

提案手法の実現可能性と有効性を検証するために，本研究では KAOS のモデリングツールとして実利用されている k-tool を対象アプリケーションとした進化実験を実施した．

4.1 実験概要

本実験では，大きく 2 種類の実験を実施した．まず，Control loop モデリングに基づいてソフトウェアシステムの構築・進化が可能であることを確認するために，整形プロセスに従ってゴールモデルを整形し，得られたコンフィギュレーションから Control loop を構成要素とするソフトウェアシステムを実際に構築し，3 種類の進化要求に対して，構築した k-tool を進化させた（実験 1）．また，提案プロセスの分析可能性を評価するために，情報系の大学院生計 3 名を被験者として，実験 1 で用いた 3 つの進化に対して，進化の影響範囲分析実験を実施した（実験 2）．

本実験においては，まず，k-tool の開発仕様書から k-tool に対する要求を抽出し，初期ゴールモデルを構築した．この初期ゴールモデルにおいては，後述する進化 1，進化 2 に該当する機能などを除外した状態の要求が記述されている．本論文では以降，この k-tool から一部の機能を除外したバージョンのソフトウェアを便宜上，**k'-tool** と呼び，k'-tool のゴールモデルに対して整形プロセスを適用し，Control loop 単位で構築したバージョンのソフトウェアを **c-tool** と呼ぶ．

4.2 本実験で扱う進化

k-tool はゴールモデルなどの図形表現を描画可能なメインエディタや，記述された各要素がツリー構造で表示されるツリービューなど複数のビューを持つモデリングツールである．k-tool は Java 言語により実装され，GUI ベースのアプリケーションにおいて広く利用されている MVC（Model View Controller）モデル [20] に従って構築されている．本実験では，k-tool に対して以下の 3 種類の機能を追加する．ここで進化 1，進化 2 により追加される機能は，従来の k-tool において提供されている機能であり，本実験では，初期の k'-tool および c-tool を進化 1，進化 2 に該当する機能を除外した状態で構築する．

進化 1（コンソールの追加）：KAOS モデルの編集過程を示すログ出力機能をツールに付与する．画面右下に新たにログを出力するためのコンソールを追加し，ポップアップメニューにより，コンソールの隠蔽，ログの保存，削除（クリア）機能が実行できるようにする．

進化 2（プロパティビューの追加）：KAOS モデルの各要素に対して属性（プロパティ）を定義，編集できるよう，画面左側のツリービューの下側に，KAOS モデルの各要素に対して属性を定義，編集可能なプロパティビューを追加する．

進化 3（Uses 関係定義タブの追加）：本研究で導入する利用関係（Uses）の概念が記述できるよう拡張する．メインエディタ上で選択したゴールに対して，プロパティビュー上の「Uses」タブを選択することで，利用するゴールの情報を設定できるように拡張する．

4.3 実験 1：c-tool 構築・進化実験

実験 1 では，初期ゴールモデルをもとに，本研究で導入する整形プロセスに基づいて c-tool を実装した．**図 7** は c-tool のゴールモデル，つまり k'-tool のゴールモデルに整形プロセスを適用したゴールモデルから抽出される Control loop と Control loop 間の階層構造を示したものである．図 7 に表示されているコンポーネントは各 Control loop を表現しており，メインエディタなどの各ビューを構成するものや，共通操作に関するものなど，合計 10 個の Control loop が同定された．また，エンティティに関する競合は検出されないことを確認することができた．その後，整形後ゴールモデルから得られた情報をもとに c-tool を実装した．本実験では，図 7 に示した各コンポーネント内に，“Manage” + ⟨コンポーネント名⟩ の名称を持つクラスを Control loop を制御するクラスとしてそれぞれ実装し，これらのクラスが必要に応じて k-tool の構成部品クラスを再利用し，構成部品クラスを呼び出すことで各 Control loop を実装した．k-tool の構成部品クラスを再利用し，構成部品クラスを呼び出すことで各 Control loop を実装した．これらの各 Control loop を結合することで，c-tool が問題なく動作し，要求された機能を提供していることが確認できた．

続いて，k-tool に対して機能を限定した k'-tool と，構築した c-tool とをそれぞれ進化 1〜進化 3 の順で進化させた．k'-tool に対する進化については，従来の開発プロセスを想定し，ゴールモデル上に進化に対する要求を追加した後，設計モデル上で変更箇所を同定したうえで，実装モデル上で同定した変更を反映させた．一方，c-tool における進化への対応手順は 3.3 節で述べたとおりである．

進化実施後に両開発プロセスを比較した．k'-tool と c-tool のゴールモデルの相違として，まず，c-tool のゴールモデルでは各進化に対する変更箇所が分散されることが確

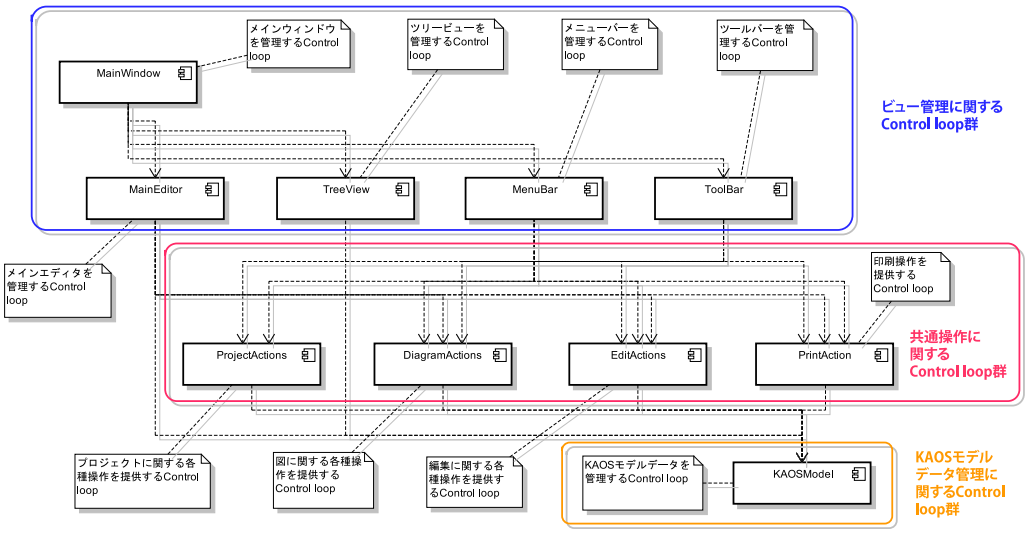

図 7 c-tool の Control loop 間階層構造（破線矢印は Uses 関係を表す）
Fig. 7 Control loop hierarchy of c-tool. Dashed arrows represent Uses dependencies.

認された．これは，整形プロセスの適用により，共通化されているゴールに関連する要求が分散して配置されることによるものである．また，進化 1 と進化 2 において，システムを構成する Control loop が 1 つずつ追加されていることも確認できた．

続いて，各進化に対するコード上の変更箇所を計測した．計測結果を付録 A.1 の**表 A·1**，**表 A·2**，**表 A·3** に，システム構成図上での各進化における影響を**図 A·1**，**図 A·2** に示す．変更箇所を比較すると，まず，k'-tool においては，各進化において MVC モデルのすべての要素（Model，View，Controller）に変更影響が及んでいることが分かる．MVC モデルは，新たなビューの追加などに対しては，View を追加するだけでよいという利点があるが，Model の変更をともなうような進化に対しては，変更影響がすべての要素に及ぶ可能性が高い．今回の実験においても，各進化に対して，MVC モデルのすべての要素において，変更すべき箇所や変更により影響が及ぶ可能性のある箇所を特定する必要があった．一方，c-tool においては，実装上での追加・変更箇所に該当する Control loop と，ゴールモデルの変更箇所を包含する Control loop とが一致していることが確認できた．従来の開発法に従った k'-tool においては，ゴールモデルと MVC モデルとを関連付けることは難しく，ゴールモデルの変化から実装モデル上での変更個所を特定することは困難であるが，提案する開発プロセスでは，整形後ゴールモデルからシステム構成が決定されるため，結果として，要求モデル上の変化から実装モデル上の変更すべき箇所を特定することが可能となる．

また，表 A·1〜表 A·3 からは，c-tool において進化に必要な追加・変更コード行数は，k'-tool の場合とほぼ同程度であるものの，修正が必要なクラス数が k'-tool の場合よりも少ないことが確認できる．これは，k'-tool においては MVC 各要素内で共通利用するクラスにも変更影響が及んだことによるものである．このようなクラス修正に対しては，他の機能に対しての変更影響も考慮した修正が必要となるため，進化の実現はより困難なものとなる可能性がある．一方で，c-tool においては，変更が各 Control loop 内に閉じているため，変更が与える影響は限定されることになる．

4.4 実験 2：影響範囲分析実験

実験 2 では，提案手法の分析可能性を評価するために，情報科学系の大学院生 3 名を被験者として，k'-tool と c-tool それぞれのソフトウェア進化における設計モデル上の影響範囲分析実験を実施した．本実験では，進化における影響範囲を分析するための開発用文書として，要求が記述されたゴールモデルと，主要クラス群に関するクラス図により構成されるクラス設計書を用意した．k'-tool のゴールモデルとしては実験 1 で用いた初期ゴールモデルを用い，c-tool のゴールモデルとしては実験 1 で整形した整形後ゴールモデルを用いた．本分析実験では，進化範囲の分析は被験者のソフトウェア開発スキルにも大きく依存すると考え，k'-tool と c-tool の分析実験を同一の被験者 3 名により実施した．ただし，k'-tool 分析の前に c-tool 分析においてゴールモデルを再整形すると，k'-tool 分析での進化影響範囲同定においても提案手法の効果を与えてしまうと考えられるため，k'-tool，c-tool の順で分析を実施した．

k'-tool と c-tool いずれの進化分析においても，各被験者には，まずゴールモデル上で進化に対する要求を分析し，その後，クラス設計書上での追加・変更箇所を同定してもらった．c-tool における分析では，3.3 節で示したプロセスに従って，進化に対する要求分析後にゴールモデルを再整形し，その後，クラス設計書上での追加・変更箇所を同

表 2 実験 2 における正解率
Table 2 Accuracy in experiment 2.

■ k'-tool

	追加・変更すべきクラス数	被験者 A		被験者 B		被験者 C		平均	
		指摘数	正解率	指摘数	正解率	指摘数	正解率	指摘数	正解率
進化 1	6	2	0.33	3	0.50	3	0.50	2.67	0.44
進化 2	34	5	0.15	6	0.18	8	0.24	6.33	0.19
進化 3	8	3	0.38	4	0.50	3	0.38	3.33	0.42

■ c-tool

	追加・変更すべきクラス数	被験者 A		被験者 B		被験者 C		平均	
		指摘数	正解率	指摘数	正解率	指摘数	正解率	指摘数	正解率
進化 1	2	2	1.00	2	1.00	2	1.00	2	1.00
進化 2	31	24	0.77	27	0.87	27	0.87	26	0.84
進化 3	6	4	0.67	6	1.00	6	1.00	5.3	0.89

定してもらった．得られた分析結果は，各被験者ごとに正解率を算出し，それらを k'-tool と c-tool とで比較することにより評価した．正解率には再現率（recall）を用い，以下の数式により算出した．ここで，C は追加・変更すべきクラスの集合であり，P は被験者により同定されたクラスの集合である．

$$\text{正解率} = \frac{|C \cap P|}{|C|}$$

各被験者の各進化ごとの正解率を**表 2** に示す．表 2 の実験結果からは，進化 2 と進化 3 において，各被験者とも c-tool の方が高い正解率であることが分かる．これは，MVC モデルにおける Model に関する変更影響を分析できたことによるところが大きい．k'-tool に対する分析については，被験者 3 名とも，ビューの更新と追加すべき操作（View と Controller）については，追加・修正が必要なクラスや変更内容の多くを指摘することができたが，Model の変更については，ほとんど指摘することができなかった．一方，c-tool に対する分析においては，実験 1 で示したように，ゴールモデル上での変更箇所から変更の可能性のある Control loop を特定することが可能であり，Model に関する Control loop（KAOS モデル管理部）にも変更の可能性があることを被験者が認識することができた．ただし，c-tool においても，すべての変更箇所を同定できるわけではなかった．たとえば進化 2，進化 3 においては属性情報が追加されるため，KAOS モデルの属性追加だけでなく，永続化に関するクラスも修正する必要があるが，このような同一 Control loop 内のクラス修正を指摘できない被験者もいた．正解率の違いのもう 1 つの要因は，共通変数を定義しているクラスの存在である．実験 1 で示したように，k'-tool の進化においては共通変数を扱うクラスに対しても変更影響が及ぶが，共通変数の変更をクラス設計書上で分析するのは容易ではなく，変更の影響範囲の把握を難しくしているといえる．

5. 考察

実験結果に基づいて，2.1 節で定義した各要件に対して提案手法を評価し，その後，提案手法の適用範囲について議論する．

5.1 分析可能性

実験 2 の影響範囲分析実験の結果が示すように，提案する開発プロセスでは，進化時に要求の変化によってシステム上に生じる影響の範囲をゴールモデル上で分析，限定化することが可能であった．これは，提案する開発プロセスにおいては，ゴールモデルの整形により，ゴールモデル上でソフトウェアシステムに対する要求を Control loop という制御単位で分割し，システムの各機能を独立化させるとともに，システム構成要素の責務を明確化させることによるものである．

本開発プロセスにおいては，一般に，主要ゴール自体の追加・削除は，Control loop の追加・削除に該当し，主要ゴール内の変更については，Control loop の振舞いの変更に該当する．しかし，Uses 関係における被利用側の Control loop の変更においては，利用側の Control loop において変更が必要かどうかを確認する必要がある．ただし，この点においては，被利用側の提供サービスに変化があるかどうかで，変更の影響が利用側に及ぶかどうかを判断することができる．また，Uses 関係を明確に定義しておくことにより，ゴールモデル上において Control loop 間の影響範囲を特定することができる．

一方で，実験 2 でも示されたように，Control loop 内での影響範囲分析に関しては，ゴール記述の差分情報から設計者が判断する必要がある．この点においては，各 Control loop が過度に大きくならないように主要ゴールを同定する工夫も必要と考えられる．

5.2 整合性

実験 1 で示した 3 つの進化では，いずれもゴールモデル上での整形結果から，Control loop の追加・変更の範囲内で進化が実現できることを確認した．進化 1，進化 2 は既存の Control loop の変更と新規 Control loop の追加により，また，進化 3 は既存 Control loop の変更により，進化を実現することができた．

整合性を満足するには，現在のシステム構成に矛盾するような過剰な変更を防ぐような開発手法が求められる．提案手法では，1 つ以上の Control loop により構成されるシステム構成をアーキテクチャとして想定し，文献 [17] の変換法を用いることで，1 つ以上の主要ゴールが存在するゴールモデルから，各主要ゴールを根とするサブツリーにに割り当てられた Control loop をシステム構成要素として対応付けている．ゴール，サブツリーの追加・削除は該当する Control loop の追加・削除，あるいは変更に対応し，エンティティや関連の追加・削除は，関連ゴールを包含する Control loop 内の処理変更，あるいは競合に対処するための新たな Control loop の追加や Control loop の統合に該当する．したがって，本研究で導入したシステム構成決定法においては，ゴールモデル上の変更のみがシステム構成に影響を与え，その影響も，変更されたゴールに関与する Control loop の追加，削除，変更に限定されるため，進化によって過剰にシステム構成が変更されることを防いでいるといえる．ただし，5.4 節で言及するアーキテクチャに関与する非機能要求に対して変更があった場合には，本研究で想定する Control loop により構成されるアーキテクチャからの変更も検討する必要がある．

5.3 変更容易性

実験 1 では，従来の MVC モデルに従った場合，Model の変更をともなうような進化に対して，変更影響が Model，View，Control のすべての要素に及んだことに対して，提案手法では変更箇所がゴールモデル上で同定される Control loop 内に限定されることが確認された．

5.1 節で述べたように，本手法においては，ゴールモデル上の変更箇所に応じて，変更の影響範囲を特定の Control loop に限定することができる．これは，システム構成要素に Control loop を割り当てることにより，各構成要素が情報収集から処理の実行までの一連のプロセスを独立実行できることを目指したものであり，構成要素間の依存関係の最小限化が期待できる．ただし，変更容易性を追求するためには，Control loop の粒度に注意を払う必要があるといえる．もし複数の機能を 1 つの Control loop として構築した場合は，変更の範囲が Control loop に閉じられるとしても，Control loop 内の他の機能実装部分に影響を与える可能性がある．

ここで，ゴールモデルの変更と Control loop の対応関係について議論する．提案手法では，進化の要因となる要求の変化をゴールモデル上で抽出し，ゴールモデルの整形により，進化要求を固有の Control loop に対応付ける．整形により，進化要求は複数の Control loop に分配される場合もあるが，定義 3.3 の条件 1 により，進化要求のうち機能要求に関するものは Control loop に対応する主要ゴール以下に含まれる．非機能要求に関しては，固有の Control loop に関連するものは該当する主要ゴール以下に含め，そうでないものに関しては，5.4 節で言及する本研究で想定するシステム構成の適用範囲を確認し，合致しないものについては必要に応じてアーキテクチャを再検討することとなる．

5.4 適用範囲

最後に，提案手法の適用範囲について議論する．本研究ではエンティティの責務を Control loop ごとに分離することで，各 Control loop におけるプロセス変数を同定している．したがって，Control loop 間で生じる競合や干渉については，これらのプロセス変数がエンティティとして同定されていることと，他の Control loop に影響を与えないようなプロセス変数の割当てが必要となる．本研究では，Entity-conflict パターンによるエンティティ競合の回避を経て Control loop を分離することで，Control loop に関与するプロセス変数に対する競合の解消を図っている．したがって，もしこれらのプロセス変数をうまく単一の Control loop に割り当てられない場合は，該当 Control loop の責務の範囲が大きくなり，結果として，システムを構成する Control loop の数が減少することとなる．

本手法では，整形プロセスにより整形されたゴールモデル上で，システムの構成要素，つまり進化の単位となる Control loop を同定するため，Control loop が少数しか抽出されない場合は適用の効果が小さくなり，複数の Control loop が同定される場合に本開発プロセスの効果が期待できる．複数 Control loop の抽出が期待できるシステムの 1 つとして，多種にわたる入力変数と入力変数ごとに独立したアクションが定義されるシステムがあげられる．このようなシステムとしては，たとえば豊富な GUI を持つシステムや Web アプリケーションが該当する．これらのシステムは，部品（Widget）群ごとに入力変数やアクションが独立していることから，Control loop が複数構築されることが期待できる．また，アクションが独立していることは，Control loop 単位での変数の集約化も可能であることを示している．他に，多種多様な情報（入力変数）を扱い，状況に応じてとるべきアクションが異なるシステムとして，ユビキタスアプリケーションがあげられる．ユビキタスアプリケーションに対しても，入力情報収集とアクション実行に関する責務が集約され，進化の影響範囲を分析しやすいという特徴は同様である．また，Control loop をシステ

ム構成要素とすることは，状況変化への対応が必要であるユビキタスアプリケーションにおいて，状況に応じた制御の切替えという観点からも有益であると考えられる．

次に，非機能要求の観点から提案手法の適用範囲について議論する．ゴールモデルにおいて各主要ゴールに包含される非機能要求については，基本的には該当する Control loop に割り当てられる責務として考えることができる．各主要ゴールに包含されない非機能要求や，システム全体に影響すると考えられる非機能要求については，本手法の想定するアーキテクチャの特徴により満足されるかどうかを判断する必要がある．ここでは，POSA のソフトウェアアーキテクチャ非機能特性 [21] に沿って，本手法で想定するアーキテクチャの特徴について議論する．変更容易性（Changeability）についてはすでに述べたとおりであるが，他システムとの相互運用性（Interoperability）の観点からは，他システムからの入力を入力変数に，他システムへの出力を制御変数，あるいは操作変数としてモデリングすることで，他システムとの相互運用を意識したシステム開発が可能であるといえる．一方，効率性（Efficiency）の観点からは，Control loop として同定された各構成要素を並行に動作させる場合には，プロセス実行のオーバヘッドを考慮する必要がある．信頼性（Reliability）については，各 Control loop が独立した制御プロセスを形成することから，特定の Control loop 内に閉じた障害に対しては一定の頑強性を期待することができる．一方で，テスト容易性（Testability）については，各 Control loop の統合に対するコストを考慮する必要がある．特に，複数の Control loop を並行動作させる場合には，テストが難しくなるといえる．最後に再利用性（Reusability）については，提案手法が Control loop の観点からシステムを細分化することで各構成要素間の依存関係を減少させることを目指したものであることから，各 Control loop 単位での再利用により，ソフトウェア開発におけるモジュールの再利用が期待できる．また，ゴールモデルの段階でモジュール化を進めるという特徴は，再利用時における既存モジュールの要求把握を支援しているともいえる．もし，これらの特徴に矛盾する非機能要求が存在する場合は，システム設計段階において本研究で示したアーキテクチャが利用できないなど，本開発プロセスの適用が限定的なものとなる．

6. まとめ

本研究では，ソフトウェアの進化を考慮した効果的なソフトウェアシステム開発手法として，ゴール指向要求記述を利用した Control loop の同定法を提案した．この Control loop の同定のために，本研究では，ゴール指向要求記述上での整形プロセスを定義した．整形されたゴールモデルは，既存の変換手法を利用することで，システム構成や，進化時に利用する差分情報を生成することに利用することができる．このような開発環境の提供は，要求分析結果に合致したソフトウェアシステム進化の実現を支援するものであり，本研究により，要求変化により生じる影響を分析でき，変更に対する影響が限定的であるシステム進化法が提供されると考える．確実なシステム進化の実現に関しては，文献 [22] に代表されるモデル変換技術や形式仕様記述による設計モデルと実装モデルのさらなる連携手段や，進化後のシステムに対する効率的な回帰テストの実現方法 [23], [24]，さらには進化時の矛盾のない要求記述の更新手段 [25] など，まだまだ解決すべき課題が多く残されているが，本研究の試みが実世界に適応するソフトウェアシステムの構築に対する 1 つの有効な手段となれば幸いである．

謝辞 本論文の実験では，KAOS ツールとして国立情報学研究所 GRACE センター所有の k-tool を使用させていただきました．当ツールの利用に関してご協力を賜りました国立情報学研究所 GRACE センターの方々に深く感謝いたします．

参考文献

[1] Lientz, B.P. and Swanson, E.B.: *Software Maintenance Management*, Addison-Wesley Longman Publishing Co., Inc. (1980).
[2] Bennett, K.H. and Rajlich, V.T.: Software maintenance and evolution: A roadmap, *Proc. Conference on The Future of Software Engineering* (*FOSE '00*) *in ICSE '00*, pp.73–87, ACM (2000).
[3] Parnas, D.L.: Software aging, *Proc. 16th International Conference on Software Engineering* (*ICSE '94*), pp.279–287, IEEE Computer Society Press (1994).
[4] Lehman, M.M. and Ramil, J.F.: Rules and Tools for Software Evolution Planning and Management, *Annals of Software Engineering*, Vol.11, No.1, pp.15–44 (2001).
[5] Lindvall, M. and Sandahl, K.: How well do experienced software developers predict software change?, *Journal of Systems and Software*, Vol.43, pp.19–27 (1998).
[6] ISO/IEC: Software engineering – Product quality – Part 1: Quality model (2001).
[7] Breivold, H.P., Crnkovic, I. and Eriksson, P.J.: Analyzing Software Evolvability, *Proc. 32nd Annual IEEE International Computer Software and Applications Conference* (*COMPSAC '08*), pp.327–330, IEEE Computer Society (2008).
[8] van Lamsweerde, A.: From System Goals to Software Architecture, *Proc. Formal Methods for Software Architectures*, Vol.LNCS 2804, pp.25–43, Springer (2003).
[9] van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour, *Proc. 5th IEEE International Symposium on Requirements Engineering* (*RE'01*), Toronto, Canada, pp.249–262 (2001).
[10] Yu, Y., Lapouchnian, A., Liaskos, S., Mylopoulos, J. and Leite, J.C.S.P.: From goals to high-variability software design, *Proc. 17th International Conference on Foundations of Intelligent Systems* (*ISMIS'08*), pp.1–16, Springer-Verlag (2008).
[11] Shaw, M.: Beyond objects: A software design paradigm based on process control, *SIGSOFT Software Engineering Notes*, Vol.20, pp.27–38 (1995).

[12] Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N. and Zambonelli, F.: A survey of autonomic communications, *ACM Trans. Autonomous and Adaptive Systems* (*TAAS*), Vol.1, No.2, pp.223–259 (2006).

[13] Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S. and Wolf, A.L.: An Architecture-Based Approach to Self-Adaptive Software, *IEEE Intelligent Systems*, Vol.14, No.3, pp.54–62 (1999).

[14] Dardenne, A., van Lamsweerde, A. and Fickas, S.: Goal-Directed Requirements Acquisition, *Science of Computer Programming*, Vol.20, No.1-2, pp.3–50 (1993).

[15] Letier, E.: Reasoning about Agents in Goal-Oriented Requirements Engineering, Ph.D. Thesis, Universite Catholique de Louvain (2001).

[16] van Lamsweerde, A.: *Requirements Engineering – From System Goals to UML Models to Software Specifications*, Wiley (2009).

[17] Nakagawa, H., Ohsuga, A. and Honiden, S.: gocc: A Configuration Compiler for Self-adaptive Systems Using Goal-oriented Requirements Description, *Proc. 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (*SEAMS '11*), pp.40–49, ACM (2011).

[18] Magee, J., Dulay, N., Eisenbach, S. and Kramer, J.: Specifying Distributed Software Architectures, *Proc. 5th European Software Engineering Conference* (*ESEC '95*), Sitges, Spain, pp.137–153, Springer-Verlag (1995).

[19] Damas, C., Lambeau, B. and van Lamsweerde, A.: Scenarios, goals, and state machines: A win-win partnership for model synthesis, *Proc. 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (*SIGSOFT '06/FSE-14*), pp.197–207, ACM (2006).

[20] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M.: *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*, John Wiley & Sons, Hoboken, New Jersey, USA (1996).

[21] Buschmann, F., Rohnert, H., Stal, M., Meunier, R., Sommerlad, P.（著），金沢典子，水野貴之，桜井麻里，関富登志，千葉寛之（訳）：ソフトウェアアーキテクチャ：ソフトウェア開発のためのパターン体系，近代科学社 (2000).

[22] Gomaa, H.: Towards Feature-Based Evolutionary Software Modeling, *Preliminary Proc. International Workshop on Models and Evolution* (*ME'11*), pp.64–73 (2011).

[23] Huang, S., Li, Z.J., Zhu, J., Xiao, Y. and Wang, W.: A novel approach to regression test selection for J2EE applications, *Proc. IEEE 27th International Conference on Software Maintenance* (*ICSM'11*), pp.13–22, IEEE (2011).

[24] Taneja, K., Xie, T., Tillmann, N. and de Halleux, J.: eXpress: Guided Path Exploration for Efficient Regression Test Generation, *Proc. 2011 International Symposium on Software Testing and Analysis* (*ISSTA'11*), pp.1–11, ACM (2011).

[25] Sampath, P., Arora, S. and Ramesh, S.: Evolving specifications formally, *Proc. 19th IEEE International Requirements Engineering Conference* (*RE'11*), pp.5–14, IEEE (2011).

付　　録

A.1 実験 1 におけるコード上での変更箇所

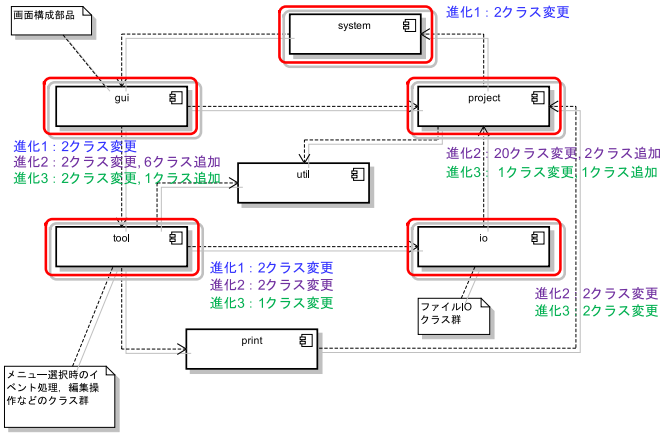

図 A·1 進化 1〜3 に対する k'-tool における変更箇所
Fig. A·1 Changes for three evolution scenarios in k'-tool.

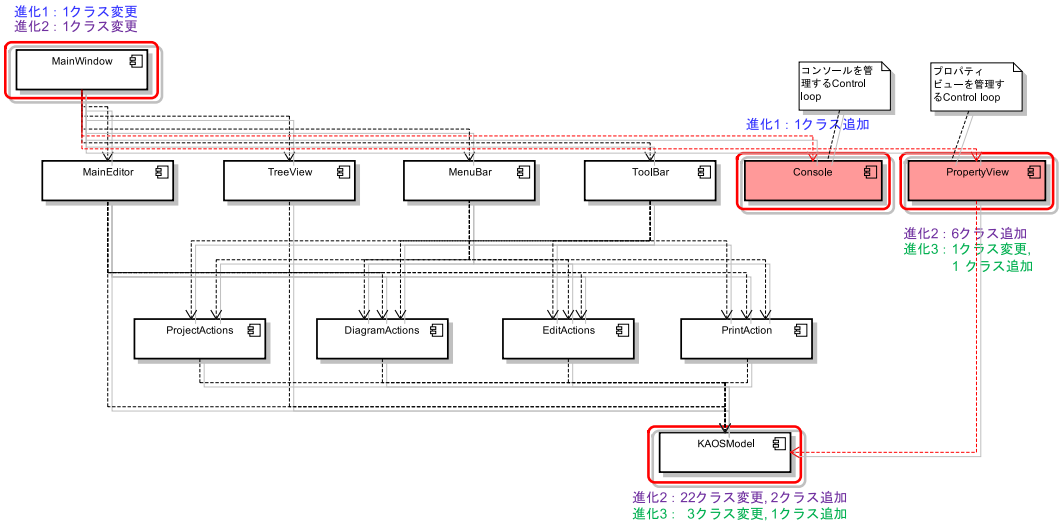

図 A·2 進化 1〜3 に対する c-tool における変更箇所
Fig. A·2 Changes for three evolution scenarios in c-tool.

表 A·1 進化 1（コンソール画面の追加）に関する実装レベルでの変更内容
Table A·1 Code changes in evolution 1.

■ **k'-tool**

パッケージ	変更クラス	変更種別	変更内容	LOC
gui（View）	MainFrame	クラス修正	Console の配置，Console を隠す動作の追加，Observer の動作追加	142
	Resource	クラス修正	コンソール関係の定数宣言	1
tool（Control）	ViewActionTool	クラス修正	ログの保存，クリア機能を定義	82
	ToolResource	クラス修正	コンソール関係の定数宣言	5
system（Model）	SystemManager	クラス修正	保存機能の実装，メッセージ追加メソッドの提供	31
	Resource	クラス修正	コンソール関係の定数宣言	1
合計（新規作成：**0** クラス，クラス修正：**6** クラス）				262

■ **c-tool**

Control loop	変更クラス	変更種別	変更内容	LOC
コンソール管理部	ManageConsole	新規作成	Console の定義，動作（保存，クリア，メッセージ追加機能）の実装，Observer の動作追加	261
メインウィンドウ管理部	MainFrame	クラス修正	Console の配置，Console を隠す動作の追加	16
合計（新規作成：**1** クラス，クラス修正：**1** クラス）				277

表 A·2 進化 2（プロパティビューの追加）に関する実装レベルでの変更内容
Table A·2 Code changes in evolution 2.

■ **k'-tool**

パッケージ	変更クラス	変更種別	変更内容	LOC
gui（View）	PropertyPanel	新規作成	プロパティ部のパネル	265
	PropertyPane	新規作成	表示処理およびビュー上でのイベントハンドラを実装	2,710
	MainFrame	クラス修正	プロパティビュー関連部定義，各種メソッド追加	59
	Resource	クラス修正	プロパティ関係の定数宣言	14
	PropertyTableModel など 4 クラス	新規作成	プロパティ部編集テーブルのモデルなど	768
tool（Control）	PopupActionTool	クラス修正	プロパティ関連アクションの宣言，プロパティ部の Popup 用削除アクション	24
	ToolResource	クラス修正	プロパティ関連の定数宣言	2
project（Model）	要素・関係線関連クラス（Agent, Concern など 19 クラス）	クラス修正	プロパティ変数の追加	2,408
	ElementAttribute	新規作成	属性情報を保持するクラス	126
	Role	新規作成	関係のロールクラス	105
	ElementTypeChangeSupport	クラス修正	型変更時のプロパティ変数の扱い	85
io（Model）	ProjectXMLWriter	クラス修正	プロパティ関連部の XML 記述生成部を追加	868
	ProjectHandler1_0	クラス修正	Uses 関係情報を XML 記述から読み込めるよう拡張	49
合計（新規作成：**8** クラス，クラス修正：**26** クラス）				7,873

■ **c-tool**

Control loop	変更クラス	変更種別	変更内容	LOC
プロパティビュー管理部	ManagePropertyView	新規作成	プロパティビューの動作，Observer を定義	282
	PropertyPane	新規作成	表示処理およびビュー上でのイベントハンドラを実装	2,766
	PropertyTableModel など 4 クラス	新規作成	プロパティ部編集テーブルのモデルなど	768
メインウィンドウ管理部	MainFrame	クラス修正	プロパティビューの配置，フォント変更，Observer の追加など	17
KAOS モデル管理部	要素・関係線関連クラス（Agent, Concern など 19 クラス）	クラス修正	プロパティ変数の追加	2,408
	ElementAttribute	新規作成	属性情報を保持するクラス	126
	Role	新規作成	関係のロールクラス	105
	ElementTypeChangeSupport	クラス修正	型変更時のプロパティ変数の扱い	85
	ProjectXMLWriter	クラス修正	プロパティ関連部の XML 記述生成部を追加	868
	ProjectHandler1_0	クラス修正	Uses 関係情報を XML 記述から読み込めるよう拡張	439
合計（新規作成：**8** クラス，クラス修正：**23** クラス）				7,864

表 A·3 進化 3（Uses 関係定義タブの追加）に関する実装レベルでの変更内容
Table A·3 Code changes in evolution 3.

■ **k'-tool**

パッケージ	変更クラス	変更種別	変更内容	LOC
gui（View）	PropertyPane	クラス修正	Uses 定義用タブの追加	141
	EditUsesTableModel	新規作成	Uses の情報を編集するテーブルのモデルを定義	74
	Resource	クラス修正	定数宣言の追加	5
tool（Control）	PopupActionTool	クラス修正	Uses 編集テーブルでの削除アクションを定義	36
project（Model）	UseGoal	新規作成	Uses 情報を管理するクラスを定義	121
	Objective	クラス修正	UseGoal クラスをメンバ変数に持つよう拡張	67
io（Model）	ProjectXMLWriter	クラス修正	Uses 関係情報を XML ファイルに保存するよう拡張	42
	ProjectHandler1_0	クラス修正	Uses 関係情報を XML ファイルから読み込めるよう拡張	49
合計（新規作成：2 クラス，クラス修正：6 クラス）				535

■ **c-tool**

Control loop	変更クラス	変更種別	変更内容	LOC
プロパティビュー管理部	PropertyPane	クラス修正	Uses 定義用タブの追加	178
	EditUsesTableModel	新規作成	Uses の情報を編集するテーブルのモデルを定義	74
KAOS モデル管理部	UseGoal	新規作成	Uses 情報を管理するクラスを定義	121
	Objective	クラス修正	UseGoal クラスをメンバ変数に持つよう拡張	67
	ProjectXMLWriter	クラス修正	Uses 関係情報を XML ファイルに保存するよう拡張	42
	ProjectHandler1_0	クラス修正	Uses 関係情報を XML ファイルから読み込めるよう拡張	49
合計（新規作成：2 クラス，クラス修正：4 クラス）				531

中川 博之 （正会員）

1974 年生．1997 年大阪大学基礎工学部情報工学科卒業．同年鹿島建設（株）に入社．2007 年東京大学大学院情報理工学系研究科修士課程修了，2008 年同大学院博士課程中退．同年より電気通信大学助教，現在に至る．要求分析，形式手法，エージェントおよび自己適応システム開発手法の研究に従事．電子情報通信学会，IEEE CS 各会員．

大須賀 昭彦 （正会員）

1981 年上智大学理工学部数学科卒業．同年（株）東芝入社．同社研究開発センター，ソフトウェア技術センター等に所属．1985〜1989 年（財）新世代コンピュータ技術開発機構（ICOT）出向．2007 年より，電気通信大学大学院情報システム学研究科教授．工学博士（早稲田大学）．主としてソフトウェアのためのフォーマルメソッド，エージェント技術の研究に従事．1986 年度情報処理学会論文賞受賞．現在，IEEE Computer Society Japan Chapter Chair，人工知能学会理事．電子情報通信学会，人工知能学会，日本ソフトウェア科学会，IEEE CS 各会員．

本位田 真一 （フェロー）

1953 年生．1978 年早稲田大学大学院理工学研究科修士課程修了．（株）東芝を経て 2000 年より国立情報学研究所教授，2012 年より同研究所副所長を併任，現在に至る．2001 年より東京大学大学院情報理工学系研究科教授を兼任，現在に至る．現在，電気通信大学，英国 UCL 等の客員教授を兼任．2005 年度パリ第 6 大学招聘教授．工学博士（早稲田大学）．1986 年度情報処理学会論文賞受賞．日本ソフトウェア科学会理事，情報処理学会理事，日本ソフトウェア科学会編集委員長を歴任．ACM 日本支部会計幹事，日本学術会議連携会員．