コンポーネントシステムを用いた組込みシステム向けアクセス制御機構*

安積　卓也[†,††]　山田　晋平[†††]　大山　博司[††††]　中本　幸一[†††]
高田　広章[†]

Access Control Mechanism Based on Component System for Embedded System*

Takuya AZUMI[†,††], Shimpei YAMADA[†††], Hiroshi OYAMA[††††],
Yukikazu NAKAMOTO[†††], and Hiroaki TAKADA[†]

あらまし　μITRON 仕様や OSEK/VDX 仕様のリアルタイム OS 上で資源を保護することを目的とし，組込みシステム向けコンポーネントシステムである TECS を利用したアクセス制御機構を提案する．ソフトウェアの大規模化に伴うバグによる障害への対処，サードパーティのコンポーネント（ソフトウェア部品）の安全な利用，ネットワーク接続に伴う外部からのアクセス侵害の理由から，組込みシステムにおいても，資源を保護するためのアクセス制御が必要になってきた．しかし，リアルタイム OS が管理しているハードウェア資源は，プロセッサ，メモリ，タイマのみであり，ソフトウェア資源もタスクなどに限られている．そのため，ファイルなどの資源を OS で管理している汎用システムのように，OS 内部でアクセス制御機構を行っても不十分である．そこで，本研究では，アクセス制御対象の資源を管理するデバイスドライバやミドルウェアを，コンポーネントシステムにおけるコンポーネントとして実現されるものとし，コンポーネントに対するアクセス制御機構の導入方法を提案し，アクセス制御機構の導入によるオーバヘッドを評価する．

キーワード　アクセス制御，コンポーネントベース開発，組込みシステム，リアルタイムシステム

1. まえがき

ソフトウェアの大規模化に伴うバグによる障害への対処，サードパーティのコンポーネント（ソフトウェア部品）の安全な利用，ネットワーク接続に伴う外部からのアクセス侵害の理由から，組込みシステムにおいても，資源を保護するためのアクセス制御(注1)が必要になってきた [1], [2].

本研究の目的は，μITRON4.0 仕様 [3] や OSEK/VDX 仕様 [4] のリアルタイム OS（図 1 (a)）の上でアクセス制御機構を実現し，資源を保護することである．ここでの保護する資源とは，タスク，ファイル，ソケット，デバイスドライバなどのソフトウェアのことである．Linux などの汎用 OS（図 1 (b)）では，資源へのアクセスは OS を介すため，アクセス制御は OS で実現できる．

一方，リアルタイム OS が管理するハードウェア資源は，プロセッサ，メモリ，タイマのみであり，ソフトウェア資源もタスクなどのカーネルオブジェクトに限定される．そのため，アクセス制御をリアルタイム OS 内部で行ったとしても十分でない．したがって，デバイスドライバやファイルシステムなどの資源にかかわるアクセス制御は，リアルタイム OS より上のデバイスドライバやミドルウェアに管理がゆだねられてい

† 名古屋大学大学院情報科学研究科，名古屋市
Graduate School of Information Science, Nagoya University, Nagoya-shi, 464–8603 Japan
†† 現在，立命館大学情報理工学部，草津市
College of Information Science and Engineering, Ritsumeikan University, Kusatsu-shi, 525–8577 Japan
††† 兵庫県立大学大学院応用情報科学研究科，神戸市
Graduate School of Applied Informatics, University of Hyogo, Kobe-shi, 650–0044 Japan
†††† オークマ株式会社，愛知県
OKUMA Corporation, Aichi-ken, 480–0193 Japan
* 本論文はシステム開発論文である．

(注1)：アクセス制御とは，アクセス主体がアクセス対象への操作を，アクセスルールに従って制御することである．

る．よって，図 1 (a) に示すとおりアクセス制御機構は，各種デバイスドライバ・ミドルウェアに分散する．

そこで，本研究では，アクセス制御対象の資源を管理するデバイスドライバやミドルウェアを，コンポーネントシステムにおけるコンポーネントとして実現されるものとし，コンポーネントに対するアクセス制御機構の導入方法を提案し，アクセス制御機構の導入によるオーバヘッドを評価する．アクセス制御を導入するコンポーネントシステムとして，組込みシステムに適したオーバヘッドの小さいコンポーネントシステムとして提案されている TOPPERS Embedded Component System（TECS）[5], [6] を用いる．

アクセス制御機構を追加することによる貢献は，μITRON4.0 仕様などの資源管理機構を有さないリアルタイム OS 上で組込みシステム向けのコンポーネントシステムである TECS にアクセス制御機構を取り込むことで次の二点を達成することである．(1) 各種資源に対して統一したアクセス制御機構を提供する．(2) ソフトウェア部品に変更を加えずアクセス制御機構を利用する．(1) は，各種デバイスドライバやミドルウェアをコンポーネントとして扱い，保護すべき資源を扱うコンポーネントに対してアクセス制御を行うことで実現される．(2) は，保護すべき資源を所有するコンポーネントへの呼出しに対し，新たにアクセス制御用のコンポーネントを追加することで実現される．アクセス制御機構は，プログラム本体と独立しているため，元々のコンポーネント（ソフトウェア部品）に変更を加えずに利用できる．

本論文の構成は以下のとおりである．まず，**2.** で本アクセス制御機構のベースとなる TECS について述べる．次に，**3.** で提案するアクセス制御機構とその実装について述べ，**4.** で適用例・評価について示す．**5.** で関連研究について述べ，最後に **6.** で本論文をまとめる．

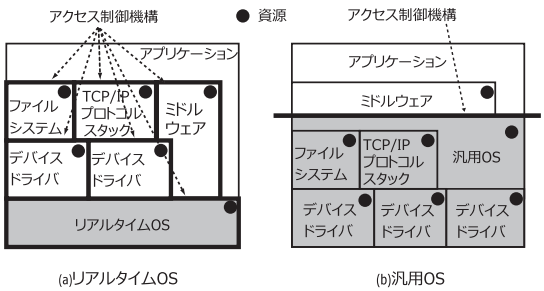

図 1　資源管理の違い
Fig. 1　Difference of resource management.

2.　TECS の概要

本章では，提案アクセス制御機構のベースとなる TECS の概要を述べる．

2.1　コンポーネントモデル

TECS では，インスタンス化されたコンポーネントをセルと呼ぶ(注2)．セルは，自身の機能を提供するインタフェースである受け口，他のセルの機能を利用するためのインタフェースである呼び口，セルの付随情報（定数）を表す属性，セルの内部状態を表す変数で構成される．受け口は，セルの機能を提供する関数の集合であり，呼び口は，他のセルの機能を利用するための関数呼出しの集合である．一つのセルは，複数の呼び口や受け口をもつことができる．

受け口と呼び口の型は，シグニチャと呼ばれる関数ヘッダの集合で定義される．セルの呼び口は，同一のシグニチャをもつ他のセルの受け口と結合できる．これにより，前者のセルから後者のセルの関数群を呼び出すことが可能になる．図 2 は，セル App の呼び口 cFile とセル File の受け口 eFile を結合したことを表す．

セルの型，すなわち，セルがもつ受け口，呼び口，属性，変数の組を，セルタイプと呼ぶ．一つのセルタイプから，複数のセルをインスタンス化できる．

2.2　コンポーネント記述

コンポーネントの要素は，TECS の仕様で定義されたコンポーネント記述を用いて表される．コンポーネント記述は，シグニチャ記述，セルタイプ記述，組上げ記述に分類される．下記で図 2 を例として，それぞれのコンポーネント記述を説明する．

2.2.1　シグニチャ記述

シグニチャ記述は，セルのインタフェースを定義するために用いられる．signature キーワードに続けて

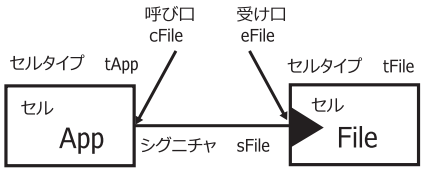

図 2　コンポーネントモデル
Fig. 2　Component model.

(注2)：オブジェクト指向の用語との対比では，後述のセルタイプがクラスに，セルがインスタンスに対応する．

```
 1 /* シグニチャsFile の定義 */
 2 signature sFile{
 3   /* ファイルを開く */
 4   ER open([in,string]const char_t*
           fileName,[in]uint8_t modo );
 6   /* ファイルを閉じる */
 7   ER close(void);
 8   /* ファイル読込み */
 9   ER read( /* 読込み用のバッファ */
         [out,size_is(size)]void* buffer,
10       [in]uint16_t size,/* 読込みサイズ */
          /* 読み込まれたサイズ */
11       [out]uint16_t* readSize );
12   /* ファイル書込み */
13   ER write(/* 書込みデータ */
         [in,size_is(size)]const void* buffer,
14       [in]uint16_t size, /* 書込みサイズ */
         /* 書き込まれたサイズ */
15       [out]uint16_t* writtenSize );
16 };
```

図 3　シグニチャ記述
Fig. 3　Signature description.

シグニチャ名（通例，接頭辞 “s” を付ける）を記述し，そのシグニチャがもつ関数ヘッダを中括弧内に列挙する．図 3 の例では，関数 open，close，read，write をもつシグニチャsFile を定義している．

TECS では，インタフェースの定義を明確にするために，図 3 の 4 行目の open で見られるような，引数の指定子を用意している．指定子は引数が，in が入力，out が出力，inout が入出力，size_is(size) が大きさ size の配列であることをそれぞれ示す．

2.2.2　セルタイプ記述

セルタイプ記述は，呼び口，受け口，属性，変数を用いてセルタイプを定義するために用いられる．celltype キーワードに続けてセルタイプ名（通例，接頭辞 “t” を付ける）を記述し，セルタイプがもつ要素を列挙する．呼び口は，call キーワード，シグニチャ名，呼び口名（通例，接頭辞 “c” を付ける）の順に記述する．受け口は，entry キーワード，シグニチャ名，受け口名（通例，接頭辞 “e” を付ける）の順に記述する．属性は attr，変数は var キーワードを用いてそれぞれ列挙する．属性，変数が存在しない場合は省略できる．

図 4 の例では，シグニチャが sFile の呼び口 cFile をもつセルタイプ tApp と，シグニチャが sFile の受け口 eFile，属性，変数をもつセルタイプ tFile を定義している．

2.2.3　組上げ記述

組上げ記述は，セルタイプをインスタンス化してセルを生成し，セル同士の結合関係を定義してアプリケーションを構築するために用いられる．cell キーワードに続けてセルタイプ名，セル名を記述し，中括弧内には，自身の呼び口と他のセルの受け口との結合を列挙する．結合は，呼び口名，=，結合先の受け口の順に記述する．なお，呼び口のないセルには結合の記述は必要ない．

```
 1 /* セルタイプ tApp の定義 */
 2 celltype tApp{
 3 /* シグニチャsFile の呼び口の定義 */
 4   call  sFile cFile;
 5 };
 6 /* セルタイプ tFile の定義 */
 7 celltype tFile{
 8 /* シグニチャsFile の受け口の定義 */
 9   entry sFile eFile;
10   attr{
11       /* 属性の定義 */
12   };
13   var{
14       /* 変数の定義 */
15       char_t * fileName;
16   };
17 };
```

図 4　セルタイプ記述
Fig. 4　Cell type description.

```
1 /* File のインスタンス化 */
2 cell tFile File{
3 };
4 /* App のインスタンス化 */
5 cell tApp App{
6 /* File の受け口と結合 */
7   cFile = File.eFile;
8 };
```

図 5　組上げ記述
Fig. 5　Build description.

図 5 の例では，セルタイプ tFile のセル File（2〜3 行目）とセルタイプ tApp のセル App（5〜8 行目）が，それぞれインスタンス化されている．またセル App の呼び口 cFile は，セル File の受け口 eFile と結合している（7 行目）．

2.3　TECS における開発の流れ

TECS における開発の流れを図 6 に示す．TECS を用いたシステムの開発者は，コンポーネント設計者，コンポーネント開発者，アプリケーション開発者（コンポーネント利用者）の三つに分けられる．コンポーネント設計者は，シグニチャ記述によりセル間のインタフェースを定義し，定義したインタフェースを利用し，セルの呼び口，受け口，属性，変数をセルタ

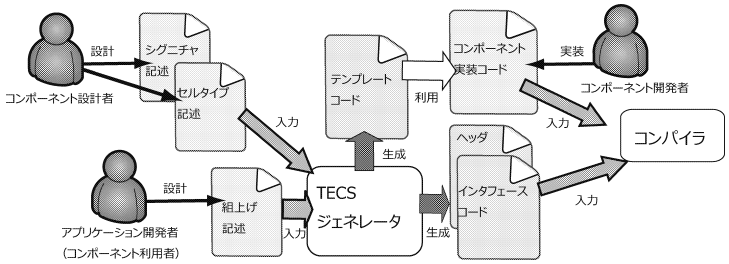

図 6　TECS における開発の流れ
Fig. 6　Development flow in TECS.

イプ記述に記述する．コンポーネント開発者は，シグニチャ記述とセルタイプ記述の情報から TECS ジェネレータよって自動生成されたテンプレートコードを利用し，コンポーネント実装コードにコンポーネントの振舞いを C 言語で実装する．テンプレートコードは，セルタイプ記述で定義された受け口の関数に対してそれぞれ自動生成される．アプリケーション開発者は定義されたセルタイプを利用し，組上げ記述により組上げを行う．

TECS ジェネレータのもつ機能は，上記で説明したテンプレートコードの生成や，組上げ記述の情報から，セル間を結合するためのグルーコードを生成することが挙げられる．グルーコードは，セルの結合先の関数の呼出しを行うマクロなどを含むヘッダ，受け口側の関数テーブルなどを含むインタフェースコードから構成されている．TECS では静的なコンポーネントモデルを採用しているため，構成によっては結合を最適化 [5] することもできる．TECS ジェネレータが自動生成したヘッダ・インタフェースコードとコンポーネント開発者が実装したコンポーネントの実装コードをコンパイル・リンクして製品に組み込む．

3. アクセス制御機構

本章では，まず，組込みシステムにおけるアクセス制御を考察し，提案アクセス制御機構，アクセス制御ポリシについて述べた後，提案アクセス制御機構の実装について述べる．

3.1 組込みシステムにおけるアクセス制御

アクセス制御とは，アクセス主体がアクセス対象への操作を，アクセスルールに従って制御することである．組込みシステムは，一般にサーバや PC とは異なり，機器に組み込まれたコンピュータであり，携帯端末，家庭電化製品，制御システムなど形態や応用領域が様々である．以下で，組込みシステムにおける主体と対象について考察する．

主体に関しては，組込みシステムでは，携帯電話や情報家電のように利用者が明確に存在するものから，ルータなどのネットワーク機器のように利用者が明確でないものまで多様である．利用者以外にタスクや機器も資源をアクセスすることから，これらも，主体として考えることが妥当である．なお，機器の識別には，IP アドレスなどが用いられる．

次に，アクセス制御の対象について考える．汎用システムで利用されている Linux では，アクセス制御の対象がファイルなどに抽象化され，アクセス操作（読込み，書込み，実行など）が規定されている．これに対して，組込みシステムでは，対象が多様であり，すべての対象に対して操作を規定することは難しい．また，対象をメモリアクセスという形態で直接アクセスするため，アクセス操作を規定できない場合もある．

3.2 提案アクセス制御機構

上記で述べたように，主体は利用者，タスクなどが考えられる．提案アクセス制御機構が主体に依存しないようにするため，主体をアクセス制御コンテクスト（以下，コンテクストと呼ぶ）という形で一般化する．コンテクストの指定方法は，**3.3** で述べる．コンポーネントベースの組込みシステムにおいて，保護対象である資源はすべてコンポーネントによって扱われる．そのため，資源を扱うコンポーネントへのアクセスを制御することで，資源そのものを保護できる．つまり，対象をコンポーネントという形で一般化し利用する．

TECS におけるアクセス制御機構を用いた例を図 7 に示す．保護対象とは，保護すべき資源を扱うコンポーネントのことであり，図 7 では File が資源を扱うコンポーネントである．アクセス制御は，保護対象にアクセスする部分（図 7 では App と File の結合部分）に対して，アクセス制御機構を追加することで行う．アクセス制御機構はアクセス制御コンポーネントとリ

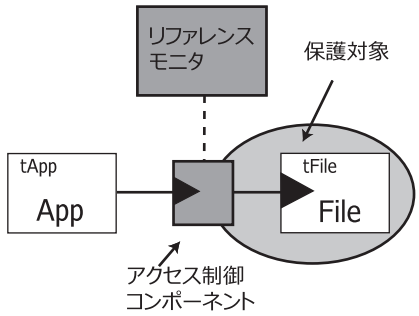

図 7　アクセス制御機構
Fig. 7　Access control mechanism.

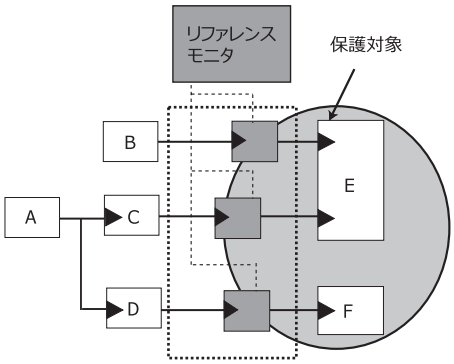

図 8　アクセス制御エンハンサ
Fig. 8　Access control enhancer.

ファレンスモニタからなる．アクセス制御コンポーネントは，コンテクストから保護対象に対するアクセスが発生したときに，実際にアクセス制御を行うものである．リファレンスモニタは，アクセス制御の際にアクセスの可否を判定するためのアクセスルールをもっており，アクセスの状況に対する結果を返す．アクセス制御コンポーネントがリファレンスモニタに問い合わせた結果，アクセスが許可された場合には，保護対象のコンポーネント呼出しを行う．図 7 では File の関数が呼び出される．

コンテクストから保護対象にアクセスするすべてのコンポーネント呼出しに対し，アクセス制御コンポーネントを追加する作業は，アクセス制御エンハンサが行う．アクセス制御エンハンサは，アプリケーション開発者が指定した保護対象の情報，コンポーネント記述からコンポーネントの結合関係を読み込み，自動的にアクセス制御コンポーネントを追加する．図 8 にアクセス制御エンハンサを利用した例を示す．図 8 の円で囲まれた部分に含まれる E と F が保護対象である．

```
1 /* コンテクストの宣言 */
2 type コンテクスト名;
3 /* グループの定義 */
4 group グループ名 { コンテクスト名のリスト };
5 /* 許可操作の定義 */
6 allow グループ名 セルタイプ名．受け口名.
                    { 許可操作リスト }[条件文];
7 allow グループ名 セル名．受け口名.
                    { 許可操作リスト }[条件文];
```

図 9　アクセス制御ポリシ
Fig. 9　Access control policy.

この場合，図 8 の B-D から保護対象にアクセスする部分に対して，アクセス制御エンハンサが自動的にアクセス制御コンポーネントを追加する（図 8 の点線で囲まれた部分）．なお，図 8 の A は，保護対象に直接結合していないため，A から呼び出す部分（A から C 及び，A から D）には，アクセス制御コンポーネントは追加されない．

3.3　アクセス制御ポリシ

提案アクセス制御機構における，アクセス制御ポリシ(注3)を図 9 に示す．

type 文（2 行目）では，アクセス制御にかかわるコンテクスト情報を宣言する．コンテクスト情報としては，タスク ID，利用者情報，IP アドレスなどアクセス制御時に区別を要するものが含まれる．group 文（4 行目）では，上記で宣言したコンテクストの所属するグループを定義する．なお，グループは，Role-Based Access Control [7] の役割（Role）と同等の概念である．

allow 文（6〜7 行目）では，グループに対して許可される操作を定義する．allow 文で許可された操作は，条件文と一致する場合のみ適用される．なお，条件が省略された場合には，すべての場合に適用される．条件としてはセルの属性や変数を利用できる．6 行目のようにセルタイプ名の受け口に対する allow 文は，そのセルタイプからインスタンス化されたすべてのセルに対して適用される．7 行目のようにセル名の受け口に対する allow 文は，そのセルに対してのみ適用される．なお，セルタイプ名とセル名の両方の allow 文が記述された場合には，セル名の allow 文を優先して適用する．保護すべき資源を所有するセルへのアクセスは，allow 文で指定された操作のみを許可する．allow 文で指定されていないコンテクストに対しては，すべてアクセスを禁止する．そのため，アクセス制御ポ

(注3)：アクセス制御ポリシの具体例は，**4. 2** で紹介する．

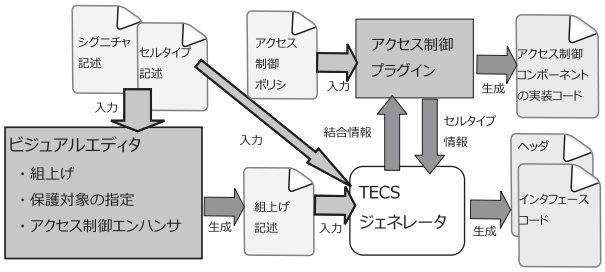

図 10　アクセス制御機構追加の流れ
Fig. 10　Flow of adding access control mechanism.

リシを利用することで，必要最小限のアクセスを許す最小特権の原則 [8]（Principle of Least Privilege）に基づいた強制アクセス制御 [9]（MAC：Mandatory Access Control）が実現できる．強制アクセス制御は，任意アクセス制御と比較するとポリシ記述が複雑になる可能性があるが，アクセス制御ポリシを徹底でき，外部からポリシを容易に変更できない利点がある．更に，組込みシステムでは，一度ポリシを設定すれば，ポリシの変更は多くないため，運用上はポリシの複雑さは問題にならないと考えられる．

3.4　アクセス制御機構の追加による効果

アクセス制御機構を追加することで，以下に示す効果が期待できる．

- アクセス制御の抜けがなくなる．

保護対象が決まれば，アクセス制御エンハンサによって一括して自動的にアクセス制御コンポーネントが追加されるため，アクセス制御の抜けがなくなる．

- コンポーネントが安全に再利用できる．

アクセス制御機構を導入することで，安全にサードパーティのコンポーネントを再利用できる．

- アクセス制御の動作確認が容易になる．

従来はデバイスドライバやミドルウェアのコード中にアクセス制御のコードが埋め込まれており，アクセス制御の確認が容易ではなかった．アプリケーションのプログラム本体とアクセス制御機構が分離しているため，アクセス制御が正しく行われているかの確認は容易である．更に，デバイスドライバやミドルウェアのコードにアクセス制御のコードを記述する必要がなくなり，開発効率が向上する．

- コンポーネントベースのアプリケーションのテスト，デバッグに利用できる．

コンポーネントの呼出しの確認ができるため，コンポーネントベースのアプリケーション開発のテスト・デバッグ用途で使用できる．

3.5　アクセス制御機構の実装

本節では，アクセス制御機構の実装及び，TECS にアクセス制御機構を取り入れる仕組みについて説明する．本研究で新規に開発したソフトウェアは，後述の through 記述を処理するためのプラグイン機構（TECS ジェネレータの拡張），アクセス制御プラグイン，リファレンスモニタ（check 関数）及び，ビジュアルエディタである．

アクセス制御機構追加の流れを図 10 に示す．ビジュアルエディタは，GUI 上でセルの配置や定義を行い，組上げを行うエディタである．アクセス制御プラグインは，**3.3** で述べたアクセス制御ポリシの情報を読み込み，コンポーネントの結合関係の情報を TECS ジェネレータから受け取り，アクセス制御コンポーネントのセルタイプコードやその C 言語の実装コードを生成するプログラムである．アクセス制御プラグインの指定には through 記述を用いている．下記でそれぞれの詳細を述べる．

3.5.1　through 記述

through 記述とはコンポーネント同士の結合部分に影響を与えずにコンポーネントを追加する仕組みのことである．すなわち，既存のコンポーネントの変更は必要ない．アクセス制御コンポーネントの追加は through 記述を用いて行われる．through 記述はアクセス制御用のコンポーネントの追加以外にトレースや RPC チャネルの追加 [10] などの用途で用いられる．図 11 の左側は元の組上げ記述とコンポーネント図を示す．図 11 の右側は through を用いた組上げ記述の例を示す．組上げ記述の呼び口と受け口を結合する記述（図 11 左 4 行目）に対して through 記述（図 11 右 4 行目）を追加する．through 記述の第一引数には，追加されるセルを生成するためのプラグイン名，それ以降はプラグインに渡す引数を記述する．なお，through 記述により追加されるセルの受け口・呼び口


```
1 cell tFile File{
2 };
3 cell tApp App{
4  cFile=File.eFile;
5 };
```

```
1 cell tFile File{
2 };
3 cell tApp App {
4  [through(ACPlugin,"arg1")]
5  cFile=B.eFile;
6 };
```

図 11　through 記述の例
Fig. 11　Example of through description.

```
1 /* 自動生成されるのセルタイプ */
2 celltype tACPlugin_tFile_sFile{
3   /* 受け口 */
4   entry sFile eThroughEntry;
5   /* 呼び口 */
6   call sFile cCall;
7 };
```

図 12　アクセス制御コンポーネントのセルタイプコード
Fig. 12　Cell type code of access control component.

のシグニチャは，追加される前のシグニチャと同じである．

3.5.2　アクセス制御コンポーネントとリファレンスモニタ

アクセス制御コンポーネントは through 記述で指定されたアクセス制御プラグインにより自動生成される．リファレンスモニタは，check 関数で実現される．check 関数は，コンテクスト情報や，アクセス制御リストの情報を利用して，アクセスの可否を判定する．アクセス制御リストは，**3.3** で説明したアクセス制御ポリシから変換された C 言語の配列で，セルの属性として保持される．アクセス制御リストは，基本的に ROM 領域に置かれるため，アクセス制御ポリシ自体の改ざんのおそれはない．RAM 領域に置かれる場合にも，メモリ保護機構を利用することでアクセス制御ポリシの改ざんを防ぐことができる．アクセス制御コンポーネントは，check 関数を利用することでリファレンスモニタにアクセスできる．

図 12 にアクセス制御コンポーネントのセルタイプコード，図 13 にアクセス制御コンポーネントの実装コードの例を示す．

アクセス制御コンポーネントのプラグインの種類により，生成される実装コードの内容を変えることもできる．例えば，ファイル名の違いによりアクセス制御内容を変更したい場合などに有効である．デバイスのアクセス制御はセルと資源が 1 対 1 で結び付いている

```
1 /* アクセス制御プラグインが自動生成する C 言語 */
2 ER eThroughEntry_open(CELLIDX idx,
      const char_t* filename, uint8_t modo)
3 {
4     /* セルの情報の取得省略 */
5
6     /* リファレンスモニタに問い合わせ */
7      if(check((void*)ATTR_accessMatrix,
            get_context(), FUNC_OPEN)){
8         /* アクセス許可 */
9         return cCall_open(filename, modo);
10    }else{
11        /* アクセス拒否 */
12        return E_AC;
13    }
14 }
```

図 13　アクセス制御コンポーネントの実装コード
Fig. 13　Access control component code.

場合やファイル名が固定の場合には，デフォルトのアクセス制御プラグインで対応できる．

図 13 は，シグニチャsFile の関数 open に対してデフォルトのアクセス制御プラグインを適用した場合の例である．7 行目の check 関数で，リファレンスモニタに問い合わせを行う．check 関数に渡す情報は，アクセス制御リスト，コンテクスト情報，関数情報である．セルの属性は，7 行目の ATTR_accessMatrix のように ATTR_属性名で定義された属性マクロを利用することでアクセスできる．属性マクロは TECS ジェネレータによって自動生成される．現在のコンテクスト情報は，7 行目の get_context() を呼び出すことで取得できる．コンテクスト情報は，コンポーネントシステムが管理しており，セルの呼出し時に，自動的にコンテクスト情報が受け渡される．関数情報は，アクセス制御プラグインにより 7 行目の FUNC_OPEN のように，アクセス制御コンポーネント内で一意に決まる値にマクロ定義される．

アクセスが許可された場合には，保護対象の関数を呼び出す（9 行目）．一方，アクセスが拒否された場合には，アクセスエラー（E_AC）を返す（12 行目）．

3.5.3　ビジュアルエディタ

図 14 にビジュアルエディタの概観図を示す．ビジュアルエディタは，インタフェース情報であるシグニチャ記述やセルタイプ記述を読み込み，GUI 上でセルの配置や定義を行い，組上げを行うエディタである．組上げ記述は，ビジュアルエディタにより自動生成される．ビジュアルエディタは組上げの機能に加えて，アクセス制御機構における，保護対象の指定及び，アクセス

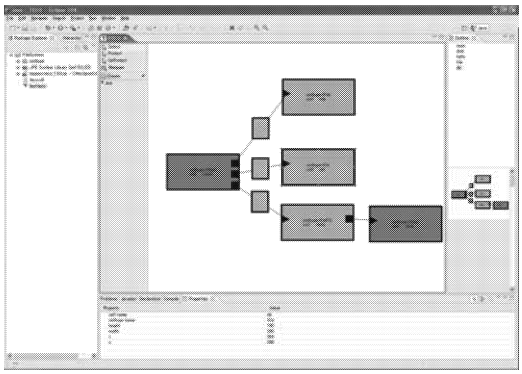

図 14　ビジュアルエディタ
Fig. 14　Visual editor.

制御エンハンサの機能をもつ．アクセス制御エンハンサの機能は，コンテクストから保護対象にアクセスするすべてのコンポーネント呼出しに対し，through 記述を自動的に追加することで実現される．

4. アクセス制御機構の適用例

本章では，ファイルシステムを利用したアプリケーションに本アクセス制御機構を用いた例を示す．

4.1 ファイルを用いたアプリケーションの構成

利用するファイルシステムは，組込みシステム向け FAT ファイルシステムの FatFs [11] である．FatFs のセルタイプは，ファイル操作を行う tFile，ディレクトリ操作を行う tDir，マウントやディスク空き領域の管理などを行う tFS の 3 種類がある．本来は，tDir と tFS へのアクセス制御も行っているが，本章では，簡単のため，セルタイプ tFile からインスタンス化されたセルのアクセス制御のみ考える．

図 15 にファイルを用いたアプリケーションの構成図を示す．アクセス制御コンテクストとしては，利用者の，su，usr1，usr2 とログを出力するタスクの logtask である．保護対象は，ファイルを管理するセルタイプ tFile からインスタンス化された ConfFile と LogFile である．ConfFile は設定ファイルを，LogFile はログファイルを扱う．アクセス制御機構は，ユーザに対してコマンドラインのインタフェースを提供する ConsoleApp とログの出力を行う LogApp から，ConfFile と LogFile へのアクセスを制御する．なお，ConsoleApp には，コンテクストを区別するための認証機構を備えている．アクセス制御コンポーネントは，保護対象へアクセスする部分に対してアクセス制御エンハンサが自動的に追加する．

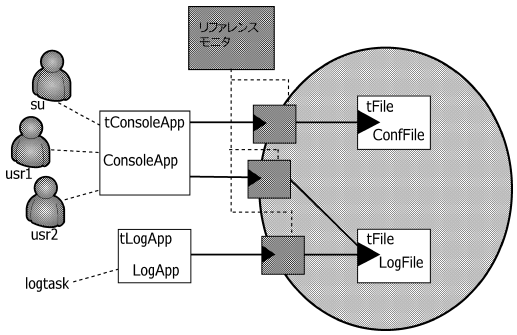

図 15　ファイルを用いたアプリケーションの構成図
Fig. 15　Configuration of application using files.

```
 1 /* コンテクストの宣言 */
 2 type su;
 3 type usr1;
 4 type usr2;
 5 type logtask;
 6
 7 /* グループの定義 */
 8 group SuGroup{su};
 9 group UsrGroup{usr1, usr2};
10 group LogGroup{logtask};
11
12 /* 許可操作の定義 */
13 allow SuGroup tFile.eFile.{*}
              [tFile.filename = "/setting/*"];
14 allow SuGroup tFile.eFile.{open,close,read}
              [tFile.filename = "/log/*"];
15 allow UsrGruop tFile.eFile.
              {open,close,read}
              [tFile.filename = "/log/*"];
16 allow LogGruop LogFile.eFile.
                  {open,close,write};
```

図 16　アクセス制御ポリシの例
Fig. 16　Example of access control policy.

4.2 アクセス制御ポリシ

図 16 に **4.1** で説明したアプリケーションをアクセス制御するためのアクセス制御ポリシの例を示す．type 文（2〜5 行目）では，アクセス制御にかかわるコンテクスト情報を宣言する．図 16 の例では，su，usr1，usr2，logtask というそれぞれ異なるコンテクストを定義している．group 文（8〜10 行目）では，上記で宣言したコンテクストの所属するグループを定義する．su は SuGroup に，usr1 と usr2 は UsrGroup に，logtask は LogGroup に属することをそれぞれ定義している．

allow 文（13〜16 行目）では，グループに対して許可する操作を定義する．13〜15 行目の allow 文は，セルタイプに対しての定義であるため，セルタイプ File

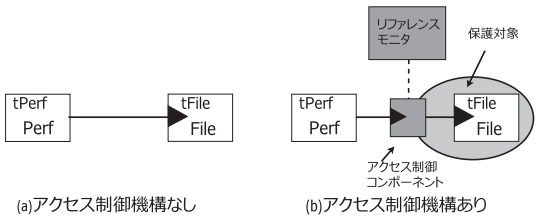

図 17　評価アプリケーションの構成図
Fig. 17　Configuration of evaluation application.

からインスタンス化されたすべてのセルに対して適用される．この例では，セル ConfFile 及び LogFile に適用される．13 行目では，ディレクトリ setting 以下にあるファイルに対して，グループ SuGroup は，セルタイプ tFile の受け口 eFile のすべての操作が許可される．* を指定することで，すべての操作対象を指定できる．14〜15 行目では，ディレクトリ log 以下にあるファイルに対して，グループ SuGroup と UsrGroup は，セルタイプ tFile の受け口 eFile の open，close，read の操作が許可される．

16 行目の allow 文は，セルに対しての定義である．グループ LogGroup は，セル LogFile の受け口 eFile の open，close，write の操作が許可される．

4.3　アクセス制御機構の性能評価

本節では，提案アクセス制御機構を利用することによる実行時間と，実行形式のファイルサイズのオーバヘッドを評価する．図 17 に評価で用いるアプリケーションの構成を示す．図 17 中の Perf は，実行時間を測定するコンポーネントである．File が提供する open，close，read，write の四つの関数についてアクセス制御機構を追加しない場合（図 17 の (a)）と追加した場合（図 17 の (b)）のそれぞれの実行時間の測定を行った．アクセス制御機構なしの場合は，Perf から直接 File の関数を呼び出し，処理が Perf に戻ってくるまでの時間を測定する．一方，アクセス制御機構ありの場合は，Perf からアクセス制御コンポーネントを呼び出し，アクセスの可否の判定後，アクセス制御コンポーネントが File の関数を呼び出し，処理が Perf に戻ってくるまでの時間を測定する．なお，読込み，書込みで使用したファイルのサイズは 1 kByte である．

実験環境は表 1 の示すとおりである．実行時間の測定結果を表 2 に示す．表 2 中の実行時間は，10,000 回測定した平均値であり，キャッシュは測定のたびにパージした．表 2 の結果が示すとおり，実行時間の増加は 2 μs から最大で 7 μs である．実行時間オーバヘッドは数 μ 秒で一定であり，実行時間が予測可能であることを考慮すると，リアルタイム性への影響は少ないと考えられる．

表 1　実験環境
Table 1　Experiment environment.

Processor	SH3 (SH7727)
Board	MS7727CP02
Clock	96 MHz
PC Card Controller	MR-SHPC-01 V2T-F
PC Card Adaptor	SDAD-38-J60
Media (Compact Flash)	1 GB SDCFH-1024-903
Compiler	gcc (3.3)
Compiler Option	-O2

表 2　関数呼出しの測定結果
Table 2　Experimental results of calling functions.

	アクセス制御機構なし	アクセス制御機構あり	オーバヘッド	オーバヘッド
open	27 μs	29 μs	2 μs	7.41%
close	7 μs	9 μs	2 μs	28.57%
read	2,193 μs	2,196 μs	3 μs	0.14%
write	5,927 μs	5,934 μs	7 μs	0.12%

表 3　実行形式のファイルサイズ（Byte）
Table 3　File sizes of execution formats.

	アクセス制御機構なし	**アクセス制御機構あり**	**リファレンスモニタ**	**オーバヘッド**
Text	**48,491**	**49,867**	**280**	**2.84%**
Data	**685**	**809**	**122**	**18.1%**
Bss	**9840**	**9840**	**0**	**0%**
Total	**59,016**	**60,516**	**402**	**2.55%**

実行形式のファイルサイズについて，アクセス制御機構がない場合とある場合及び，リファレンスモニタ（check 関数，アクセス制御リスト）の測定結果を表 3 に示す．表 3 の Text の増加は，アクセス制御コンポーネントの実装コード，リファレンスモニタの check 関数及び，アクセス制御コンポーネントの結合に伴うグルーコードによるものである．表 3 の Data の増加は，アクセス制御コンポーネントの関数テーブルなどの情報や，リファレンスモニタの実体であるアクセス制御リストの配列によるものである．アクセス制御リストのサイズは，保護対象の数，保護対象が提供する関数の数，保護対象にアクセスするグループの数に比例して増加する．表 3 の結果が示すとおり，実行形式のファイルサイズのオーバヘッドは全体で 2%程度と小さい．

5.　関 連 研 究

本章では，汎用 OS でのアクセス制御機構，既存の

コンポーネントシステムにおけるアクセス制御機構，既存のリアルタイム OS での保護機能と，提案アクセス制御機構の比較を行う．更に，through 記述を用いたプログラミングとアスペクト指向プログラミング比較を行う．

5.1 アクセス制御

SELinux [12] などに代表される既存のアクセス制御機構は，最小特権の原則 [8] に基づき必要最低限の権限しか与えない．SELinux は，RBAC [7]，強制アクセス制御 [9] の機能を提供している．**3.3** で示したとおり，提案アクセス制御機構も RBAC，強制アクセス制御の機能を提供している．一方，SELinux などの既存のアクセス制御機構は，ファイルシステムをベースとしてアクセス制御を行っているため，ファイルシステムを OS で管理していないリアルタイム OS では，既存のアクセス制御機構をそのまま適用することはできない．

COM [13] では，IAccessControl インタフェースのメソッドを利用することでアクセス制御リストを設定し，コンポーネントへのアクセス制御を実現する．しかし，提案アクセス制御機構のように，アクセス制御機構とプログラム本体が分離していないため，設定を各コンポーネントに対して行う必要がある．

資源の保護を目的として，μITRON4.0/PX 仕様 [14]（μITRON4.0 仕様の保護機能拡張）のリアルタイム OS が提案されている．しかし，**1.** で述べたようにリアルタイム OS が管理しているソフトウェア資源はカーネルオブジェクトのみである．したがって，カーネルオブジェクト以外のファイルなどのミドルウェアやデバイスドライバの資源に対して保護を行うものではない．提案アクセス制御機構では，コンポーネントシステムを利用することで，ミドルウェアやデバイスドライバに依存しない統一されたアクセス制御機構を実現している．

5.2 アスペクト指向プログラミング

アスペクト指向プログラミングの用語との対比では，コンポーネントを挟み込む箇所の指定がポイントカット（pointcut），追加されるコンポーネントがアドバイス（advice）に相当する．

コンポーネント技術にアスペクト指向の概念を持ち込んだ例として，AspectCCM [15] が提案されている．AspectCCM は，CORBA Component Model（CCM）に AspectCCM のインタフェースの定義で，use_aspect という指定子を利用することで，結合点を指定する．AspectCCM は，アスペクト（追加される機能）を利用するには結合点を指定する必要があり，追加されるコンポーネントは，提案機構とは違い自動生成されない．

一方，TECS では，他のコンポーネント技術と比べ，セマフォなど粒度が小さいものまでコンポーネントとして扱える．粒度が小さいことにより，様々なコンポーネントの結合箇所に定型的なコンポーネントを挿入でき，あらかじめ結合点（join point）を指定せずに利用できる．

6. むすび

本論文では，μITRON4.0 仕様や OSEK/VDX 仕様などのリアルタイム OS 上で資源を保護することを目的とし，組込みシステム向けコンポーネントシステムを利用したアクセス制御機構を提案した．提案アクセス制御機構では，アクセス制御対象の資源を管理するデバイスドライバやミドルウェアを，コンポーネントシステムにおけるコンポーネントとして扱うため，アクセス制御機構は各種デバイスドライバやミドルウェアに依存しない．そのため，従来リアルタイム OS 上の各種デバイスドライバやミドルウェアに分散していたアクセス制御機構を，統一されたアクセス制御機構の枠組みで利用できる．更に，アクセス制御機構はデバイスドライバ・ミドルウェアのプログラム本体と分離しており，既存のソフトウェア部品に変更を加えずにアクセス制御機構を利用できる．

また，アクセス制御機構の導入によるオーバヘッドを，組込みシステム向けの FAT ファイルシステムである FatFs にアクセス制御機構を適用し評価した．評価の結果，実行時間のオーバヘッドは 2 μs から 7 μs であり，実行形式のファイルサイズのオーバヘッドはプログラム全体で 2.5%に抑えられることを示した．

謝辞 本研究の一部は（独）情報処理推進機構（IPA）が実施した 2006 年度下期未踏ソフトウェア創造事業の支援を受けたものである．

文　献

[1] D.D. Hwang, P. Schaumont, K. Tiri, and Verbauwhede, “Securing embedded systems,” Security & Privacy, vol.4, no.2, pp.40–49, March 2006.

[2] K.-D. Kan and S.H. Son, “Systematic security and timeliness tradeoffs in real-time embedded systems,” Proc. 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA’06), pp.183–189, Aug. 2006.

[3] H. Takada and K. Sakamura, “μITRON for small-scale embedded systems,” IEEE Micro, vol.15, no.6, pp.46–54, Dec. 1995.
[4] OSEK/VDX, Operating system, version 2.3.3, OSEK/VDX organisation, 2005.
[5] 安積卓也, 山本将也, 小南靖雄, 高木信尚, 鵜飼敬幸, 大山博司, 高田広章, “組込みシステムに適したコンポーネントシステムの実現と評価,” コンピュータソフトウェア, vol.26, no.4, pp.39–55, Nov. 2009.
[6] TECS. http://www.toppers.jp/tecs.html
[7] D.F. Ferraiolo, R. Sandhu, and S. Gavrila, “Proposed NIST standard for role-based access control,” ACM Trans. Information and System Security, vol.4, no.3, pp.224–274, Aug. 2001.
[8] DoD Computer Security Center, “Department of Defense Trusted Computer System Evaluation Criteria,” 1985. http://csrc.nist.gov/publications/history/dod85.pdf
[9] P. Loscocco, S. Smalley, P. Muckelbauer, R. Tayler, S. Turner, and J. Farrell, “The inevitability of failure: The flawed assumption of security in modern computing environments,” Proc. 21st National Information Systems Security Conference, pp.303–314, Oct. 1998.
[10] 安積卓也, 大山博司, 高田広章, “メモリ共有を考慮した RPC システム,” 情処学論（プログラミング）, vol.2, no.2, pp.37–53, March 2009.
[11] T. Akamatu, “FatFs.” http://elm-chan.org/fsw/ff/00index_e.html
[12] P. Loscocco and S. Smalley, “Integrating flexible support for security policies into the linux operating system,” Proc. FREENIX Track: 2001 USENIX Annual Technical Conference, pp.29–42, Jan. 2001.
[13] Microsoft Corporation, “Microsoft Component Object Model.” http://www.microsoft.com/com/
[14] 高田広章（編）, μITRON4.0 仕様 保護機能拡張（μITRON4.0/PX 仕様）, トロン協会, 2002.
[15] P.J. Clemente, J. Hernandez, J.M. Murillo, M.A. Perez, and F. Sanchez, “Aspectccm: An aspect-oriented extension of the corba component mode,” Proc. 28th Euromicro Conference, pp.10–16, Sept. 2002.

（平成 22 年 1 月 6 日受付，5 月 3 日再受付）

安積　卓也　（正員）

立命館大・情報理工学部助教．2009 名古屋大学大学院情報科学研究科情報システム学専攻博士後期課程了．2008〜2010 日本学術振興会特別研究員．組込みシステム向けのコンポネントシステムの研究に従事．博士（情報科学）．IEEE，情報処理学会，日本ソフトウェア科学会各会員．

山田　晋平

2006 明石高専専攻科了．2008 兵庫県立大学大学院応用情報科学研究科博士前期課程了．現在，同研究科博士後期課程在学中．組込みソフトウェア，オペレーティングシステムに興味をもつ．

大山　博司　（正員）

オークマ（株）主管技師．1984 岐阜大・工・電気卒．同年（株）大隈鐵工所（現オークマ（株））に入社．2002 岐阜大学大学院工学研究科電子情報システム工学専攻後期課程了．2004 より TOPPERS プロジェクト・コンポーネント仕様 WG 主査を兼務．数値制御装置の開発に従事し，リアルタイム制御，組込みシステムのプログラミング言語に関する研究等を行う．博士（工学）．計測自動制御学会会員．

中本　幸一　（正員）

1982 大阪大学大学院基礎工学研究科前期課程了．同年 NEC 入社．1990〜1991 Cornell 大学計算機科学科客員研究員．1997 大阪大学大学院情報数理系専攻博士課程入学．2000 単位取得退学．博士（工学）．2003〜2004 電気通信大学客員教授．2004 より現職．2006 より名古屋大学大学院情報科学研究科附属組込みシステム研究センター特任教授，組込みソフトウェア，モバイルシステムソフトウェア，ソフトウェア開発環境に興味をもつ．情報処理学会，日本ソフトウェア科学会，システム制御情報学会，IEEE Computer Society，ACM，USENIX 各会員．

高田　広章　（正員）

名古屋大学大学院情報科学研究科情報システム学専攻教授．1988 東京大学大学院理学系研究科情報科学専攻修士課程了．同専攻助手，豊橋技術科学大学情報工学系助教授等を経て，2003 より現職．2006 より大学院情報科学研究科附属組込みシステム研究センター長を兼務．リアルタイム OS，リアルタイムスケジューリング理論，組込みシステム開発技術等の研究に従事．オープンソースの ITRON 仕様 OS 等を開発する TOPPERS プロジェクトを主宰．博士（理学）．IEEE，ACM，情報処理学会，日本ソフトウェア科学会各会員．