情報家電ネットワークに向けた、ソフトウェア部品の開発

Development of Software Framework for Information Appliance

邑中 雅樹 [1)]
Masaki MURANAKA

1) 合資会社もなみソフトウェア (〒 229-1103 神奈川県相模原市橋本五丁目 17 番 23 号
E-mail: monaka@monami-software.com)

Abstract: Information Appliance (IA) is expected to be the next key device. IA is based on both the embedded computing and the distributed object computing. But there are some special issue to use these technology directly. In this paper we propose the light weight (means embedded) object bus specification for IA.

1. 背景

高速化/低価格化が進んだネットワークおよび計算機の有効活用という観点から、分散環境という概念が提唱され、その延長上に分散オブジェクト環境という概念が存在する。ビジネスアプリケーションを実現するための分散オブジェクト環境は実用の域にあり、いくつかの仕様に基づく製品が出荷されている。

一方、家庭内の家電製品にコンピュータを内蔵することは一般的になった。これらの家電製品を計算機ネットワークで接続することで、さらなる利便性を得ようとする動きがある。いわゆる家庭内ネットワークや地域ネットワークである。家庭内ネットワークや地域ネットワークは、専門家が厳密に管理することを期待できた従来の計算機ネットワークと異なる問題が存在する。存在するオブジェクトの数は格段に多くなり、信頼性の低い通信経路を想定せざるを得ず、セキュリティポリシーの一貫性も保てなくなる。

以降、このような環境を、従来の分散オブジェクト環境と区別するために、超分散環境と呼ぶ。

2. 目的

本事業は、既存の技術を超分散環境に流用した場合の問題点を明らかにし、解決するための軽量オブジェクトバス仕様を提唱した。また、新しい仕様に基づき構築したエミュレーションライブラリによって、仕様の妥当性を定性的に検証した。

3. 現状が抱える課題

超分散環境は、マイコン搭載家電製品を始めとする制御システム技術と、分散オブジェクト技術の融合と言える。それぞれの技術は成熟しているが、融合の結果としての超分散環境の要求に照らし合わせたとき、両者に解決すべき問題が発生する。以下に、発生が予想される問題点を整理する。

(1) 制御システム開発の問題点

a) アプリケーション開発の困難さ

従来の組込向け OS では、割り込みの制御やタスクスイッチなど、他の分野では許されない操作でもアプリケーション開発者が実行することができる。このような OS は、アプリケーション開発の困難さが増すことを承知で、ハードウェア資源

の節約と、実行速度を確保する目的で設計されている。

しかしながら、ことアプリケーション開発者から見た場合、超分散環境での率速は通信である。

ソフトウェア技術者に期待できる開発スキルが、低下の傾向にあることを併せて考えると、従来の組込向け OS を用いた場合、設計の困難さが浮き彫りになる、と予想される。

b) ソフトウェアの複雑化による品質の低下、テストコスト上昇

システムの多機能化と、ハードウェア資源の大規模化によって、ソフトウェアの開発規模も増大の傾向にある。開発規模の増大によって、一箇所の修正によって、他の個所で引き起こされる副作用を予期することが困難になりつつある。この事実は、ソフトウェアの品質低下に直結する。

副作用がおよぶ範囲を最小限にするためには、明確に規定されたインタフェースに基づいてモジュール間の通信を制御するソフトウェアバスの採用が効果的である。しかし、従来の組込向け OS は、このような機能を用意していない。

結果として、依存性の解らないモジュールの塊が生産される。一箇所の修正を行うごとに、システム全体副作用を検出するための、テストが必要になる。テストに係るコストの増大は、現在においても深刻な問題になりつつある。

さらに、超分散環境下での開発においては、各開発者間の連携が希薄となる。例えば、一度開発されたオブジェクト (モジュール) は、開発者の寿命を越えて存在する可能性がある。また、オブジェクトの種類も膨大で、アプリケーション開発時に、どのようなオブジェクトが互いに接続されるのか、予想もつかいない。開発者に、内部実装について問い合わせることは、事実上困難である。

c) "車輪の再発明"による生産性の悪化

ソフトウェアの複雑化という、上記の問題を引き金として、似て非なるソースコード片が、いくつも生産される。依存性が明らかでないソースコードから、必要なものを抽出するより、新規に生産することが、開発現場では好まれる。

しかし、新規に生産したモジュールも、従来のモジュールと同様に相互依存の排除を明確に打ち出していないため、数度の修正を経て、相互依存を内在してしまう。再利用性に乏しくなった段階で、また新規で類似のソースコードが生産される。

このような、いわゆる"車輪の再発明"によって、生産性がいつまでも向上しない。

(2) 分散オブジェクトフレームワークの問題点

a) メソッド束縛

分散オブジェクト環境の本質は、他のオブジェクトが提供する機能を、位置依存性を排除した形で利用することである。他のオブジェクトが提供する機能を利用するためには、公開しているメソッドやプロパティの一覧 (インタフェース) を取得し、ローカルな呼び出しに束縛する必要がある。束縛には、実行前に静的に定義する静的束縛と、実行時に動的に束縛する動的束縛に大別できる。このどちらにも、超分散環境での使用には、問題を含んでいる。

早期束縛は、実行効率がよく、実装も簡単だが、超分散環境においては、実行時に、どのようなオブジェクトが接続されるのか、設計時には、予想がつきにくい。このような環境下で、実行前に、インタフェース定義を取得することは不可能である。

動的束縛を用いてインタフェースを取得するには、静的束縛を用いた場合に比べ多くの計算機資源と処理時間が必要である。[1]超分散環境では、接続される機器には、従来よりも貧弱な計算機資源

[1]CORBA の場合、経験的に約 40 倍といわれている。

しか持たないと想定すべきである。

b) インタフェース爆発

分散オブジェクト環境のように、インタフェース定義に高い自由度を与えた場合、莫大な数のインタフェースの管理、という新しい問題が発生する。

例えば、Microsoft Windows2000 上の COM 実装では、Win32 サービスのために用意されたインタフェースだけで 300 を越え、アプリケーションを含めた場合の公開インタフェースの数は、4 桁以上のオーダで存在する。

このように、分散オブジェクト環境では、一つの OS を構成するためのインタフェースでさえ、膨大なものとなる。接続されるオブジェクトの種類を予測できない超分散環境において、インタフェースを管理することは、事実上不可能である。

c) 広域ディレクトリサーバ

分散オブジェクト環境では、他のオブジェクトが提供する機能や位置を知るために、ディレクトリサーバを持つ。典型的には、一台のディレクトリサーバは、複数のオブジェクトに関する情報の問い合わせに応じる。

分散環境において、ディレクトリサーバは生命線である。ディレクトリサーバが破綻した場合、ディレクトリサーバに依存する全てのオブジェクトは、利用したい他のオブジェクトの情報を取得することができずに破綻する。

超分散環境においては、環境の一部が破綻することは、避けることができない。そして、ディレクトリサーバも環境の一部でしかない。

4. Neko オブジェクトバス

上記で明らかにした問題を検討し、筆者は超分散環境の構築をサポートする Neko オブジェクトバス (Network embedded kernel object bus/ Neko-bus) を設計した。

Neko-bus は、TRON プロジェクトの研究成果である BTRON2 仕様の影響を強く受けている。BTRON2 もまた、分散環境用の OS であり、実身仮身モデルを基礎とする寿命管理機能を有する。

(1) 特徴

Neko-bus の特徴と、現在の問題をどのように解決するかを以下に挙げる。

a) 自律オブジェクトの集合体

Neko-bus におけるプログラミングスタイルは、典型的なコンポーネント指向である。具体的には、自律オブジェクトの関係を定義することと、アプリケーション開発を行うことが、等価である。自律オブジェクトの関係は、後述する超分散ディレクトリで用いられる。

コンポーネント指向が、アプリケーション開発の困難さを緩和し、ソフトウェアの複雑化を隠蔽し、生産性に寄与することは、JavaBeans や ActiveX の成功が示す通りである。

b) 参照カウント型の寿命管理

自律オブジェクトは、お互いに参照カウント型の寿命管理が行われる。

寿命管理の存在がアプリケーション開発の困難さを緩和し、ソフトウェア品質の向上に寄与することは、C 言語におけるメモリリーク問題が示すとおりである。

c) 軽量オブジェクトバス

全ての自律オブジェクトは、少数のメソッドを含む、単一のインタフェースを持つ。また、全てのオブジェクトは、1 つ、もしくは 2 つ以上の可変長レコードの集合体として定義される。各メソッ

ドは、ファイル操作に類似したものとなっている。

単一のインタフェースに限定することで、静的束縛が持つ、実行時の接続性について解決し、同時に、インタフェース爆発も抑制できる。一方で、複数レコードを持つことにより、少数のメソッドでも動的束縛と同様の動作をエミュレートすることもできる。

コンピュータの適用範囲が拡大するにつれて、技術者が持つ知識も細分化されつつある。しかしながら、ファイルへの入出力は、経験の有無、専門の種類に関わらない、共通の知識であるといえる。ファイル操作に類似したメソッドとすることで、学習コストを削減し、アプリケーション開発の困難さが緩和されることが期待できる。

インタフェースを簡潔にすることにより、将来、オブジェクトバスそのものをハードウェア化されることも期待できる。

d) 超分散ディレクトリ

ある種の自律オブジェクトは、自身が信頼関係を持つ自律オブジェクト (複数) へ接続するための、情報一覧を持つ。目的の自律オブジェクトへ接続するためには、飛び石伝い川を渡るように、複数の自律オブジェクトを渡り歩き、情報を得る。また、オブジェクトへのアクセス制御も、同時に行う。

つまり、すべての分散オブジェクトが、ディレクトリシステムの一部を担い得る。ディレクトリのサーバを究極的に分散させることで、環境の一部破綻による全体への影響を、最小限に抑えることが期待できる。

また、ディレクトリとオブジェクトの関連とアクセス制御を統合した結果、広域ネットワーク上に私的な空間を構築することが容易になる。

(2) Neko-bus の構造

Neko-bus の構造は、オブジェクトとバスマネージャだけで説明できる。

a) 自律オブジェクトと受動オブジェクト

オブジェクトは、内部処理によって自律オブジェクトと受動オブジェクトの 2 種類に分類される。ただし、両者ともに単一のインタフェースによって操作するため、アプリケーション開発者から見た場合、区別はつかない。

下層の OS から見たデバイスドライバやユーザタスクは、Neko-bus から見ると、自律オブジェクトとして一括りにされる。このように、自ら内部状態を変えるオブジェクトは全て自律オブジェクトとなる。

受動オブジェクトは、メモリ空間やディスクなど、自律オブジェクトからの操作によって内部状態が変わるオブジェクトである。

b) 実身、レコード、仮身

また、オブジェクトは、機能によって実身、レコード、仮身に分類される。

実身は、複数のレコードへのポインタを持っている。

レコードは、内部状態を持ち、外部からはバイトストリームに見えるようなインタフェースを持つ。ストリームの長さは、典型的には可変長であるが、種類によっては固定長の場合もある。

仮身は、一意な実身へのアクセス方法およびアクセス権限を保持している。特殊なレコードを読むことで、仮身の内容は変更される。自律オブジェクトは、内部状態として、0 個以上の仮身を持つ。

c) バスマネージャ

バスマネージャは、Neko-bus の最下層を受け持ち、オブジェクトの生存に必要な諸々の処理を行う。また、受動オブジェクトの操作を受け持つ。通常の場合、バスマネージャの直下には、OS が存在する。バスマネージャは、広域ネットワークとの接続を担うノードマネージャ、イベントを発生/伝達させる事象通知マネージャ、オブジェクトの管

理を行うオブジェクトマネージャなどを含む。

5. モデルケース

以下のようなアプリケーションを作成するというモデルケースによって、アプリケーション開発が実際に可能であることを検証した。

(1) 要件定義

GUI 指向のアプリケーションを作成する。

画面は、目的別に 3 つの領域を持つ。一つは領域にランダムなサイズの矩形を表示しつづける。他の 2 つは文字出力領域であり、キーボードからの入力を反映する。ユーザは、TAB キーを押下することで、入力を反映する文字出力領域を選択することができる。画面イメージは図 4 のようになる。

(2) モデル化

本アプリケーションを分析したものが図 5 である。図 5 で、実線の矢印は、仮身によるリンク、破線の矢印は事象通知を表している。

(3) 開発者の種別化/階層化

前述のように、超分散環境下での開発においては、各開発者間の連携が希薄となる。開発者を、バスコントローラ開発者、オブジェクト開発者、アプリケーション開発者、という種別で分け、各開発者がお互いに内部実装について知ることができない、という前提を立て開発を行った。

6. 評価

モデルケース開発において、以下のような利点、および課題が明らかになった。

(1) 利点

各開発者がお互いに内部実装について知ることができない、という前提を立てたが、このような環境においても、各開発者は自らの実装をおこなうことができた。

ドライバ開発時には、外部オブジェクトとのインタフェースを明確にすることで、単体テストの実施を容易に行なうことができた。

アプリケーション開発時には、事象通知の流れとオブジェクトへのアクセスのみの知識で、下層の RTOS に関する知識がなくても容易に開発できた。

超分散型ディレクトリによって、依存関係が明確になり、システム構成を簡潔に表現できた。

評価アプリケーションでの、矩形表示領域実身は、どのオブジェクトからの要求もバスコントローラによって拒否される。軽微なコストで、アクセス制御を実装できた。

(2) 課題

a) 消費リソースの削減

今回作成したエミュレーションライブラリは、IA-32 アーキテクチャ上の μ ITRON4.0 準拠仕様 OS を用いて、自律オブジェクト数 100 程度の環境を作成するために、総計 120KB 程度のメモリが必要となった。採用分野にも依るが、メモリ使用量に対する最適化が必要である。

b) リアルタイム性の確保

メッセージ通信を基礎とする通信では、応答性能の最悪値を求めることは困難であるが、測定環境に拘束条件をつけるなどして何らかの指標を示すことが必要である。しかしながら、今回の評価では、応答性能の計測は行なわなかった。

c) メモリ保護

CPU が持つメモリ保護と組にすることで、より堅牢なシステムへ発展させることが可能である。今回は単一のメモリ空間上で全てのオブジェクトが動作している。

7. おわりに

情報家電を含む次世代のネットワーク環境においては、基礎技術である組み込み制御技術と分散オブジェクト環境に、解決すべき問題が存在することを示した。

超分散環境に最適化したオブジェクトバス仕様 Neko-bus を提案し、エミュレーションライブラリの試験実装を行い、評価を行った。結果として、ソフトウェア部品プラットフォームとしての可能性、超分散ディレクトリの可能性を示した。

今回作成したエミュレーションライブラリを、実際の超分散環境フレームワークとして利用するためには、消費リソースの削減をはじめとする、各種の最適化が必要であり、今後も開発を継続する予定である。