

オープンソースソフトウェア活用基盤 整備事業

共通コンポーネント基盤と
サービス連携基盤開発のための技術調査

技術仕様書

2006 年 12 月

独立行政法人 情報処理推進機構

登録商標等について

- Microsoft, MS, Windows, などは、米国 Microsoft Corporation の米国およびその他の国における登録商標または商標である。
- Java, JDK などは、米国 Sun Microsystems の米国およびその他の国における登録商標または商標である。
- Linux は、Linus Torvalds の米国およびその他の国における登録商標または商標である。
- OpenOffice.org は Team OpenOffice.org e.V. の登録商標である。
- Mozilla および Mozilla のロゴは Mozilla Foundation の商標である。
- 「UNIX」は X/Open Co., Ltd. 社の独占ライセンスで米国及びその他の国での登録商標である。
- Qt は、ノルウェー TrollTech 社の登録商標である。
- その他、本文章に記載されている会社名、商品名、製品名などは、一般に各社の商標または登録商標である。
- 本書では、™などを記載しない。

—目次—

1 概要	5
1.1 名称	5
1.2 背景	5
1.3 目的	5
1.4 本技術仕様書の位置づけ	6
1.5 本技術仕様書の範囲	7
2 システム要件	8
3 条件	9
4 各既存技術と共通デスクトップ基盤の対応およびメッセージ通信の仕様	10
4.1 DCOP 及び QtDBus	10
4.1.1 オブジェクト管理	10
4.1.2 インタフェース (IDL)	10
4.1.3 抽象コンポーネント	10
4.1.4 通信	11
4.2 ORBIT2	11
4.2.1 オブジェクト管理	11
4.2.2 インタフェース	11
4.2.3 抽象コンポーネント	11
4.2.4 通信	12
4.3 XPCOM	12
4.3.1 オブジェクト管理	12
4.3.2 インタフェース	12
4.3.3 抽象コンポーネント	12
4.3.4 通信	12
4.4 UNO	13
4.4.1 オブジェクト管理	13
4.4.2 インタフェース	13
4.4.3 抽象コンポーネント	13
4.4.4 通信	13
4.5 D-BUS	14
4.5.1 ネーミングサービス	14
4.5.2 インタフェース	14
4.5.3 通信	14
4.6 .NET FRAMEWORK	14
4.6.1 ライフサイクル管理	15
4.6.2 アクティベーション	15
4.6.3 スレッド対応	15
4.7 WSDL	15
4.7.1 インタフェース	15
4.8 SOAP	16
4.8.1 インタフェース	16
4.9 JAX-WS	18
4.9.1 SOAP へのマッピング	18
4.9.2 同期・非同期	18
4.10 AXIS2	19
4.10.1 WSDL からのコード生成	19

技術仕様書

4.10.2 SOAP へのマッピング.....	19
4.11 XFIRE	19
4.11.1 SOAP へのマッピング.....	19
4.12 JBI	20
4.12.1 通信.....	20
4.12.2 コンポーネント管理とコンポーネントライフサイクル.....	20
4.13 SERVICEMIX.....	20
4.13.1 複数メッセージ通信プロトコル間のマッピング.....	20
4.14 SCA.....	21
4.14.1 サービスアセンブリモデル.....	21
4.14.2 複数のメッセージプロトコル間のマッピング.....	21
4.15 TUSCANY	21
4.15.1 複数メッセージ通信プロトコル間のマッピング.....	21
5 システム構成	22
5.1 本構成の特徴.....	22
5.2 論理構成	22
6 IDL の仕様	25
6.1 インタフェース名の規則.....	25
6.2 メソッド名の規則.....	26
6.3 引数名の規則.....	26
6.4 アノテーションの規則	26
6.5 IDL で指定可能な型名.....	27
7 機能仕様	28
7.1 抽象コンポーネント定義機能.....	28
7.2 抽象コンポーネントの生成機能.....	29
7.2.1 抽象コンポーネント生成のユースケース.....	35
7.2.2 QtDBus 版シーケンス.....	40
7.2.3 ORBit2 版シーケンス.....	41
7.2.4 XPCOM 版シーケンス.....	42
7.2.5 UNO 版シーケンス.....	43
7.3 抽象コンポーネントの登録機能.....	44
7.3.1 シーケンス.....	46
7.4 WEB サービスの登録機能	47
7.5 抽象コンポーネントの検索機能.....	49
7.5.1 シーケンス.....	52
7.6 抽象コンポーネントの参照機能.....	53
7.6.1 同期通信処理.....	53
7.6.2 非同期通信処理.....	56
7.6.3 抽象コンポーネント参照のユースケース.....	59
7.7 抽象コンポーネントの削除機能.....	61
7.8 WEB サービス呼び出し機能.....	63
8 今後の課題.....	66
8.1 技術的課題.....	66
8.2 コミュニティへの働きかけ	67

1 概要

1.1 名称

本書は、「技術仕様書」とする。

1.2 背景

ISV、SI 事業者、ソフトウェアハウス等は、Linux 等のオープンソースデスクトップ環境をビジネス用途のアプリケーションやシステムを開発するためのプラットフォームとして選択していない。Windows に比べて業務アプリケーションの開発が難しいことがその理由である。その原因としては、以下の機能の実現が不十分となっていることが考えられる。

- (1) デスクトップアプリケーション間の連携機能
 - アプリケーション間でのデータ移動
 - オブジェクトのリンクや埋め込み
 - オブジェクトの結合
- (2) ネットワーク上のサービスとデスクトップアプリケーションの連携機能

(1)の機能については、オープンソースソフトウェア(以下 OSS)におけるデスクトップコンポーネント技術である DCOP 及び QtDBus(KDE)、ORBit2(GNOME)、XPCOM(Mozilla)、UNO(OpenOffice.org)のそれぞれに閉じた環境であれば、ある程度は実現可能である。しかし、オープンソースデスクトップ環境における規格やシステムが十分に整備されていないため、上記のコンポーネント技術を横断して相互に利用することは不可能となっている。そのため、デスクトップのアプリケーションの種類としては、相互に補完することでニーズを満たすにも関わらず、それらを連携して使うことができない。また、(2)については OSS においては実現されていない。

1.3 目的

本システムでは、上記の問題を解決するために、以下のユースケースを満たす共通コンポーネント基盤、及びサービス連携基盤(以下、共通デスクトップ基盤という)を実現することを目的とする。

- (1) ワードプロセッサ、スプレッドシート、メール、スケジューラなどの一般的(抽象的)なコンポーネントを組み合わせることでリッチな UI のアプリケーションを容易に開発可能にする。
 - ISV、SI 事業者、ソフトウェアハウス等のプロジェクトマネージャ、開発者、CIO がシステムやアプリケーションの開発コストを削減するために本システムの選択を促進する。
- (2) サービス連携技術
 - 各コンポーネントを組み合わせることで、Web サービスをキックした結果を表示することができる。複数の Web サービスを組み合わせることで動作するシステム(証券取引システム、旅券発行システム等)をデスクトップアプリケーションとして作成することができる。
- (3) (1)と(2)の機能を Ruby や Python のようなスクリプト言語から利用して組み合わせることで、業務アプリケーションを簡単に作成可能にする。

1.4 本技術仕様書の位置づけ

本プロジェクトは、1.3 目的に述べた内容を達成するために、DCOP 及び QtDBus (KDE)、ORBit2 (GNOME)、XPCOM (Mozilla)、UNO (OpenOffice.org) を横断して相互に利用できるシステムを漸進的に実現するための第一歩である。

本プロジェクトでは、1.3 目的を達成するための技術として、デスクトップの各コンポーネント技術をつなぐ基盤には D-BUS、Web サービスのプロトコルには SOAP を選択している。D-BUS の選択理由は、freedesktop.org にてデスクトップの共通バスとして開発されており、KDE ver.4、GNOME の一部の機能、HAL (Hardware Abstraction Layer)、Skype API など、オープンソースのデスクトップにおける利用が拡大する傾向にあるためである。また、SOAP の選択理由は、最も広範囲で利用されている Web サービスのプロトコルであるためである。

本プロジェクトの成果物は、既存技術の調査結果をまとめた「調査報告書」および、その調査結果を元に導き出した技術仕様を記した「技術仕様書」(本書)の2冊で構成される。

本技術仕様書は、前述した「実現すべきシステム」の外部仕様の設計を記述した。4 章で調査結果に基づく共通デスクトップ基盤の仕様、特にメッセージ定義およびメッセージ通信機構の仕様、および SOAP への変換を行うためのメッセージ変更の仕様を記述した。5 章では、システムの論理構成について記述した。6 章では、システムの IDL の仕様について記述した。7 章では、システムの機能仕様について記述した。なお 4 章は調査報告書に記載した考察を基にしているため、詳細は調査報告書を参照のこと。

本プロジェクトでは、2006 年 7 月から 2006 年 12 月までを調査期間としている。また、調査対象とした各技術のバージョンや調査時点は以下の通り。

- KDE : ver. 3.3.5
- Qt : ver. 4.2
- ORBit2 : ver. 2.14.3
- Bonobo : ver. 1.0.22
- XPCOM : 2006 年 9 月時点
- UNO : 2006 年 9 月時点
- D-Bus : ver. 0.9.1 (Specification 0.11)
- .NET Framework 3.0 : 2006 年 9 月時点
- WSDL : ver. 2.0
- SOAP : ver. 1.2
- JAX-WS : ver. 2.0
- Axis2 : ver. 1.0
- XFire : ver. 1.2 RC1
- JBI : ver. 1.0
- ServiceMix : ver. 3.0
- SCA : ver. 0.9.6
- Tuscany : ver. M1

1.5 本技術仕様書の範囲

本技術仕様書では、上記の目的を満たすための基本的な機能として、以下の内容についての仕様を記述する。

- (1) IDL の仕様
 - プリミティブなデータ型と一般的（抽象的）なコンポーネントが持つ振る舞いを定義するための IDL の仕様
- (2) 抽象コンポーネント定義機能
 - 一般的（抽象的）なコンポーネントと各コンポーネント技術における具象コンポーネントの継承関係を記述可能にする機能
- (3) 抽象コンポーネントの生成機能
 - 一般的（抽象的）なコンポーネントを継承した具象コンポーネントの作成を行う機能
- (4) 抽象コンポーネントの登録機能
 - 一般的（抽象的）なコンポーネントとそれを継承した具象コンポーネントへのリファレンスを検索可能にする機能
- (5) Web サービスの登録機能
 - Web サービスへのリファレンスを検索可能にする機能
- (6) 抽象コンポーネントの検索機能
 - 一般的（抽象的）なコンポーネントを継承した具象コンポーネントを検索する機能
- (7) 抽象コンポーネントの参照機能
 - 各コンポーネント技術をまたがって一般的（抽象的）なコンポーネントをデータとして参照可能にする機能
- (8) 抽象コンポーネントの削除機能
 - 一般的（抽象的）なコンポーネントを継承した具象コンポーネントの削除を行う機能
- (9) Web サービス呼び出し機能
 - ネットワーク上のサービスに対する呼び出し、ネットワーク上のコンポーネントに対する参照を可能にする機能

2 システム要件

本システム（フレームワーク）の要件は以下の通りとする。

- (1) 本システム（フレームワーク）を利用するツールキット等は、D-BUS ライブラリおよび、本書に記述した仕様を実装した新規開発のライブラリが提供する機能を使うことでサービスを受けることができる。
 - D-BUS の機能が不足している場合は、D-BUS コミュニティに機能追加を提案する。
- (2) エンタープライズのデスクトップ、ワークステーションをターゲットとする。
 - ただし、D-BUS が動作することを条件とする。

3 条件

本システム（フレームワーク）が満たすべき設計上の条件は以下の通りとする。

- (1) 複数コンポーネント技術間での相互参照が可能
 - 呼び出し、被呼び出しの両方を考える
 - セキュリティ的に「呼び出されない」ことも可能とする
- (2) オブジェクト（データ）を抽象的に扱える
 - ここで言うオブジェクトとは、スプレッドシートやエディタとする
- (3) 各技術における呼び出しは極力変更しない
- (4) ライトウェイトな仕組みとする
- (5) コンポーネントの組み合わせを簡単に行える
 - スクリプト言語で実現可能とする
- (6) メモリリークをしない
- (7) ローカルの処理とネットワーク上のサービスを透過的に扱える
- (8) ネットワークセキュリティを考慮すること

4 各既存技術と共通デスクトップ基盤の対応およびメッセージ通信の仕様

本プロジェクトでは、デスクトップコンポーネント技術である DCOP 及び QtDBus (KDE)、ORBit2 (GNOME)、XPCOM (Mozilla)、UNO (OpenOffice.org) を連携させ、Web サービスとも連携させる基盤的技術の設計を行うために、複数の技術を調査した。本章では、各調査結果に基づき以下に列挙した点についての各既存技術と共通デスクトップ基盤の対応およびメッセージ通信の仕様について記述する。

- (1) オブジェクトの管理
- (2) インタフェース (IDL)
- (3) 抽象コンポーネント
- (4) メッセージ定義およびメッセージ通信機構の仕様
- (5) 共通コンポーネントのメッセージから SOAP への変換を行うためのメッセージ変更の仕様

なお QtDBus, ORBit2 については、その上位のコンポーネント技術 (QtDBus に対する KParts、ORBit2 に対する Bonobo) も合わせて調査を行った。

4.1 DCOP 及び QtDBus

オブジェクト管理や通信の方式、共通コンポーネント基盤として抽出すべきメッセージ定義・機構と IDL の分析・検討のため、KDE の現在安定リリースバージョンである ver.3.x までのコンポーネント技術である DCOP (Desktop COmmunication Protocol) と、次期リリースバージョンである ver. 4.0 以降の QtDBus モジュール (D-BUS のラッパー)、オブジェクト技術である KParts に関して、DCOP のメッセージや IDL に関する仕様、QtDBus に関連する Qt や KDE4 の実装と開発動向を調査した。その結果導き出された、「共通コンポーネント基盤およびサービス連携基盤」にて実現すべき仕様を以下に示す。

4.1.1 オブジェクト管理

オブジェクトのライフサイクル管理は、Qt のガーベジコレクタ (GC) 機能が使われているため、オブジェクト生成のみを考慮すればよく、簡単になっている。共通コンポーネント基盤でもオブジェクトの管理には GC を組み込むことが望ましい。そのためには、正しくオブジェクトのライフサイクル管理が行われるためには、共通コンポーネント基盤と Qt の GC は同期を取る必要がある。詳細は 7 章を参照のこと。

オブジェクトの検索・参照については、DCOP および QtDBus の機能としては外部から利用できるかたちでは提供されていない。また、オブジェクトのアクティベーションの仕組みも存在しない。これらの機能は、共通コンポーネント基盤で提供する抽象コンポーネントで提供する必要がある。例えば、KDE のオブジェクトから UNO のオブジェクトを参照すること、その逆に UNO から KDE のオブジェクトを参照することが実現できなければいけない。その実現のために、QtDBus を拡張し、オブジェクトの検索、参照、アクティベーションを外部から実行できるようにする。実現するためには、KDE および Qt の開発コミュニティとの連携が必要となるため、今後の課題とする。

4.1.2 インタフェース (IDL)

DCOP の IDL は、ほぼ C++ に準拠しているため、表現の柔軟性は高い。共通コンポーネント基盤で作成する IDL から容易に変換可能である。また、QtDBus の IDL は、「D-BUS 調査報告書」の 4.1 に書いたイントロスペクションと同じ DTD に基づいている。そのため、共通コンポーネント基盤で作成する IDL としても利用する。

4.1.3 抽象コンポーネント

KParts で提供されているコンポーネントの種類は、そのまま共通コンポーネントで提供する抽象コンポーネントとして使用する。

技術仕様書

また、オリジナルの KParts は DCOP や QtDBus とは独立している。一方 KOffice ではこれを拡張し、DCOP や QtDBus と連携動作するようにしている。共通コンポーネント基盤で抽象コンポーネントを実現するには、コンポーネント技術の通信、検索、参照、アクティベーションなどの機能と KParts が連携する必要がある。その実現のためには、KDE および Qt の開発コミュニティとの連携が必要となるため、今後の課題とする。

4.1.4 通信

QtDBus は、基本的に同期呼び出しを意味する MethodCallMessage、非同期通信（シグナルの通知）を意味する SignalMessage など必要な種類のメッセージをサポートしている。データ型については、DCOP は全ての C++ の型、QtDBus も D-BUS の拡張可能な型（ARRAY、STRUCT）を利用して任意の型を送信可能なように設計されている。

メッセージの種類については、ORBit2、XPCOM、UNO と比較しても不足は無い。そのため、共通コンポーネント基盤を実現するためにメッセージの種類を追加する必要は無い。

データ型については、ORBit2、XPCOM、UNO と比較しても不足は無い。そのため、抽象コンポーネントを実現する際に特に型は追加する必要が無い。また、新たな記述との連携の可能性を考えると、共通コンポーネント基盤にも拡張可能な仕組みを提供すべきである。

4.2 ORBit2

共通コンポーネント基盤として抽出すべきオブジェクト管理や通信の方式、メッセージ定義・機構と IDL の分析・検討のため、GNOME に使われているコンポーネントオブジェクトモデルである Bonobo および通信基盤である ORBit2 に関して、メッセージや IDL 仕様、実装の調査を行った。その結果導き出した、「共通コンポーネント基盤およびサービス連携基盤」にて実現すべき仕様を以下に示す。

4.2.1 オブジェクト管理

Bonobo では、CORBA レベルのライフサイクル管理とは別に参照カウンタによるライフサイクル管理を行っている。共通コンポーネント基盤でもオブジェクトの管理には GC を組み込むことが望ましい。そのため、Bonobo にも GC を実装する必要がある。また、共通コンポーネント基盤と Bonobo で生存しているオブジェクトの参照に齟齬があっては問題となる。齟齬が発生しないためには、共通コンポーネント基盤と Bonobo の GC は同期を取る必要がある。詳細は 7 章を参照のこと。

オブジェクトの検索、参照、アクティベーションについては「ORBit2 調査報告書」の 3.2 検索・参照と 3.3 アクティベーションに述べたように Bonobo に実装されている。そのため、共通コンポーネント基盤として提供すべき基本機能を実現するために必須となる機能は、既の実装されている。しかし、実際に共通コンポーネント基盤との連携をとるためには、検索や参照に使われる名前の指定方法と共通コンポーネント基盤の検索用のキーとの対応付けが必要になる。DCOP 及び QtDBus、ORBit2、XPCOM、UNO それぞれの名前とオブジェクトの対応付け方法が異なるため、この対応付けを新たに定めることとする。詳細は 7.2 (6) を参照のこと。

4.2.2 インタフェース

「ORBit2 調査報告書」の 4.1.1 データ型に述べたように、IDL として記述できる型は D-BUS の型とも対応しており、型の追加は必要ではない。ただし、同期・非同期の区別ができないため、ORBit2 に拡張の必要があると考えられる。また、Oneway オペレーションは D-BUS のシグナルに対応させる。

4.2.3 抽象コンポーネント

Bonobo の抽象コンポーネントのリストは、そのまま共通コンポーネントで提供する抽象コンポーネ

ントとして使用する。

4.2.4 通信

「ORBit2 調査報告書」の 6.1 および 6.3 で述べたように、ORBit2 では通常の同期呼び出し、Oneway を使うことができる。これは、ちょうど D-BUS のメソッド呼び出し、シグナルに対応する。また、コールバックによる非同期呼び出しが CORBA3.0 以降で導入されており、共通デスクトップ基盤でも必要であると考えため、ORBit2 にも機能追加が必要である。その実現のためには、GNOME の開発コミュニティとの連携が必要となるため、今後の課題とする。

4.3 XPCOM

共通コンポーネント基盤として抽出すべきオブジェクト管理や通信の方式、メッセージ定義・機構と IDL の分析・検討のため、Mozilla に使われているコンポーネントオブジェクトモデルである XPCOM (Cross-Platform Component Object Model) に関して、メッセージや IDL 仕様、実装の調査を行った。その結果導き出された、「共通コンポーネント基盤およびサービス連携基盤」にて実現すべき仕様を以下に示す。

4.3.1 オブジェクト管理

「XPCOM 調査報告書」の 3.1 ライフサイクル管理に述べたように XPCOM のオブジェクト管理は、現在 GC の開発が進められているため、実装が完了すればオブジェクトの管理は単純化されることになる。共通コンポーネント基盤でもオブジェクトの管理には GC を組み込むことが望ましい。しかし、共通コンポーネント基盤と XPCOM で生存しているオブジェクトの参照に齟齬があっては問題となる。齟齬が発生しないためには、共通コンポーネント基盤と XPCOM の GC は同期を取る必要がある。詳細は 7 章を参照のこと。

「XPCOM 調査報告書」の 3.2 検索・参照と 3.3 アクティベーションに述べたようにオブジェクトの検索、参照、アクティベーションについては実装されている。そのため、共通コンポーネント基盤として提供すべき基本機能を実現するために必須となる機能は、既に実装されている。しかし、実際に共通コンポーネント基盤との連携をとるためには、「XPCOM 調査報告書」の 3.2 および 6.4 で述べた検索や参照に使われる Class ID と Contract ID を共通コンポーネント基盤の検索用のキーとの対応付けが必要になる。DCOP 及び QtDBus、ORBit2、XPCOM、UNO それぞれの名前とオブジェクトの対応付け方法が異なるため、この対応付けを新たに定めることとする。詳細は 7.2 (6) を参照のこと。

4.3.2 インタフェース

「XPCOM 調査報告書」の 4.1.1 データタイプに述べたように IDL として記述できる型に任意の型もしくは拡張可能な型が記述できないため、型の追加が必要である。また、同期・非同期の区別ができないため、拡張の必要がある。その実現のためには、Mozilla の開発コミュニティとの連携が必要となるため、今後の課題とする。

4.3.3 抽象コンポーネント

XPCOM の抽象コンポーネントのリストは、そのまま共通コンポーネントで提供するコンポーネントのリストとして使用する。

4.3.4 通信

「XPCOM 調査報告書」の 6.1 メッセージ定義に述べたように XPCOM ではメッセージパッシングのみをサポートしている。これは、「DCOP 及び QtDBus 調査報告書」や「D-BUS の調査報告書」で述べたシグナルと対応付けられる。また、DCOP 及び QtDBus、D-BUS、UNO では、それぞれの調査報告書で述べたとおりメソッド呼び出しがサポートされている。そのため、共通コンポーネント基盤として一貫性を取るためには、メソッド呼び出し機能の追加が必要である。その実現のためには、

技術仕様書

Mozilla の開発コミュニティとの連携が必要となるため、今後の課題とする。

同様に、通信で利用できる型についても PRUnichar*のみであるため、共通コンポーネント基盤として一貫性を取るためには、IDL と同様の型が使えるように機能追加が必要である。型の追加についても Mozilla の開発コミュニティとの連携が必要となるため、今後の課題とする。

4.4 UNO

共通コンポーネント基盤として抽出すべきオブジェクト管理や通信の方式、メッセージ定義・機構と IDL の分析・検討のため、OpenOffice に使われているコンポーネントオブジェクトモデルである UNO (Universal Network Objects) に関して、メッセージや IDL 仕様、実装の調査を行った。その結果導き出された、「共通コンポーネント基盤およびサービス連携基盤」にて実現すべき仕様を以下に示す。

4.4.1 オブジェクト管理

UNO のオブジェクト管理は、C++の実装と Java の実装で異なっている。「UNO 調査報告書」の 8.1 分散コンポーネントに述べたように C++の場合には、参照カウンタによる管理が行われ、Java においては GC によって管理が行われる。共通コンポーネント基盤でもオブジェクトの管理には GC を組み込むことが望ましい。そのため、C++の実装の場合にも GC を実装する必要がある。また、共通コンポーネント基盤と UNO で生存しているオブジェクトの参照に齟齬があっては問題となる。そのため、正しくオブジェクトのライフサイクル管理が行われるためには、両者の GC は同期を取る必要がある。詳細は 7 章を参照のこと。

オブジェクトの検索、参照、アクティベーションについては「UNO 調査報告書」の 3.2 検索・参照と 3.3 アクティベーションに述べたように UNO に実装されている。そのため、共通コンポーネント基盤として提供すべき基本機能を実現するために必須となる機能は、既に実装されている。しかし、実際に共通コンポーネント基盤との連携をとるためには、検索や参照に使われる名前の指定方法と共通コンポーネント基盤の検索用のキーとの対応付けが必要になる。DCOP 及び QtDBus、ORBit2、XPCOM、UNO それぞれの名前とオブジェクトの対応付け方法が異なるため、この対応付けを新たに定めることとする。詳細は 7.2 (6)を参照のこと。

4.4.2 インタフェース

「UNO 調査報告書」の 4.1.1(1)データタイプに述べたように UNOIDL に記述できる情報が一番豊富である。そのため、共通コンポーネント基盤の IDL の記述できる情報は、UNOIDL で記述できる情報をベースとして仕様を決定する。ただし、同期・非同期を記述する方法がないため、その点は機能追加が必要である。その実現のためには、OpenOffice.org の開発コミュニティとの連携が必要となるため、今後の課題とする。

「UNO 調査報告書」の 4.1.1(4)Service に述べたように複数のインタフェースを継承して抽象コンポーネントの IDL を記述している点をベースとして仕様を決定する。詳細は 6 章を参照のこと。

4.4.3 抽象コンポーネント

「UNO 調査報告書」の 5.1 抽象コンポーネント定義に述べたように調査対象ドキュメントには抽象コンポーネントが列挙されていない。今後、デスクトップ基盤の普及を考えた場合、開発者が利用可能な抽象コンポーネントの列挙が必要であると考えられる。

4.4.4 通信

UNO では、同期および非同期の通信をサポートしている。共通コンポーネント基盤を実現するためには十分な機能を備えている。

同様に、通信で利用できる型についても IDL と同様の型が使えるため、共通コンポーネントを実現する基盤としては十分である。

4.5 D-BUS

共通コンポーネント基盤として抽出すべきメッセージ定義・機構と IDL の分析・検討のため、freedesktop.org にてデスクトップの共通バスとして開発されている D-BUS に関して、メッセージや IDL 仕様、実装の調査を行った。その結果導き出された、「共通コンポーネント基盤およびサービス連携基盤」にて実現すべき仕様を以下に示す。

4.5.1 ネーミングサービス

D-BUS には「D-BUS 調査報告書」の 3 章に述べたようにネーミングサービスの仕組みがあり、アプリケーションごとに名前を割り当てて管理することができる。この名前を「D-BUS 調査報告書」5.1 で述べたメッセージのヘッダフィールドの DESTINATION に指定することによって、アプリケーション間の通信が実現可能である。また、「D-BUS 調査報告書」3.2 で述べたように Bus が所有している名前のリストを取得することができるため、D-BUS における送信先のアプリケーション名を検索、参照することができる。同様に、「D-BUS 調査報告書」3.3 で述べたように、アプリケーションをアクティベーションすることができる。

このネーミングサービスの仕組みは、上記のように共通コンポーネント基盤のオブジェクト管理に必要な機能は備えている。しかし、あくまでアプリケーションを対象としたものであり、オブジェクトを検索、参照、アクティベーションできない。オブジェクトバスを対象にこれらの機能が使えるようになれば、共通コンポーネント基盤の基本となる機能を実現可能となるため、機能追加が必要になる。その実現のためには、D-BUS の開発コミュニティとの連携が必要となるため、今後の課題とする。

4.5.2 インタフェース

D-BUS のイントロスペクションの仕組みとして提供されているインタフェース情報は、「D-BUS 調査報告書」4.1 に述べたように、現状でも IDL として利用可能なだけの情報を記述可能である。

また、このインタフェース情報は XML 形式であるため、DTD を追加することで拡張が容易である。よって、DCOP 及び QtDBus、ORBit2、XPCOM、UNO を調査した結果として表現できる情報に不足があるため、DTD を拡張することで、本システム(フレームワーク)の IDL として利用する。詳細は 6 章を参照のこと。

4.5.3 通信

D-BUS が対応している通信の種類は、「D-BUS 調査報告書」の 5.1 に記述したとおり、メソッド呼び出し、シグナル(返値なし)の 2 種類である。また、送信に利用する関数として見ると、メソッド呼び出しには同期通信と非同期通信(コールバック)の 2 種類あることが分かる。DCOP 及び QtDBus、ORBit2、XPCOM、UNO がサポートしている通信は、以上の合計 3 種類の通信で実現可能であるため、現状では機能追加は必要ない。

通信可能なデータ型については、拡張可能な ARRAY 型、STRUCT 型、DICT_ENTRY 型があることを考えると、共通コンポーネント基盤として必要なデータ型はサポートされている。

4.6 .NET Framework

共通コンポーネント基盤として抽出すべきオブジェクト管理や通信の方式、メッセージ定義・機構と

技術仕様書

IDL の分析・検討のため、Microsoft Windows の .NET Framework 3.0 に関して、.NET におけるコンポーネント技術である COM/COM+ についての調査を行った。その結果導き出された、「共通コンポーネント基盤およびサービス連携基盤」にて実現すべき仕様を以下に示す。

4.6.1 ライフサイクル管理

.NET Framework 3.0 の CLR および他のコンポーネント技術では GC の導入が実施、および検討されているため、共通コンポーネント基盤では実際には GC の導入を実現することが望ましい。また、他のコンポーネント技術でもそれぞれライフサイクル管理を行っているため、共通コンポーネント基盤ではそれらを統合して、管理する技術が求められる。詳細は 7 章を参照のこと。

4.6.2 アクティベーション

COM はオブジェクトのアクティベーションについてもサービスを提供しているが、共通コンポーネント基盤ではこの部分は各コンポーネント技術において実装されるため、仕様の参考となる点はない。

今後の課題として、共通コンポーネント基盤が各コンポーネント技術の開始、終了を直接制御することになった場合には、共通コンポーネント基盤にアクティベーションの機能が必要になる。COM でのアクティベーションの方法は、本技術仕様をもとにプロトタイプを作成する場合に、内部仕様の決定にあたって参照する。

4.6.3 スレッド対応

共通コンポーネント基盤上ではスレッドの管理を行わないため、仕様の参考となる点はない。

今後の課題として、共通コンポーネント基盤が各コンポーネント技術のスレッド対応の有無を隠蔽するような機能を提供する場合には、このスレッド対応の機能が必要になる。そのため、COM でのスレッド対応の方法は、本技術仕様をもとにプロトタイプを作成する場合に、内部仕様の決定にあたって参照する。

4.7 WSDL

サービス連携基盤の設計における通信方式とインタフェース記述の方法の分析・検討のために、WSDL について調査を行った。その結果導き出された、「共通コンポーネント基盤およびサービス連携基盤」にて実現すべき仕様を以下に示す。

4.7.1 インタフェース

WSDL では、「抽象的サービスモデル」と「具象的サービスモデル」の 2 段階の定義を設けることでトランスポートプロトコルや開発言語といった前提を避けたインタフェース定義を実現している。D-BUS およびデスクトップ基盤のインタフェースは、このうち具象的サービスモデルにあたりと考えられる。

D-BUS にはインタフェースを記述するイントロスペクション（詳細は「D-BUS 調査報告書」4 章を参照のこと）と呼ばれる仕組みがある。イントロスペクションで記述したインタフェース表現の例を図 4-1 に示す。

```
<!DOCTYPE node PUBLIC "-//freedesktop//DTD D-BUS Object Introspection 1.0//EN"
"http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">
<node name="/org/freedesktop/sample_object">
  <interface name="org.freedesktop.SampleInterface">
    ...省略...
    <method name="Bazify">
      <arg name="bar" type="(iiu)" direction="in"/>
      <arg name="bar" type="v" direction="out"/>
    </method>
    ...省略...
  </interface>
</node>
```

図 4-1イントロスペクションで記述したインタフェース表現の例

このイントロスペクションで記述できるインタフェース情報は、表 4-1に示すように、「WSDL 調査報告書」の 3.1.6 節の binding 要素や 3.1.7 節の service 要素に対応付け可能な構造を持っている。よって、D-BUS のイントロスペクションによるインタフェース定義は、WSDL を使った Web インタフェース定義に対して表 4-1の通りに対応付ける。

表 4-1WSDL の要素と D-BUS イントロスペクションの対応関係

WSDL	D-BUS イントロスペクション
service 要素内の endpoint 要素	node 要素
binding 要素	interface 要素
operation 要素	method 要素
input/output/infault/outfault 要素	arg 要素 (in/out は direction 属性で指定)

4.8 SOAP

サービス連携基盤の設計における通信方式とインタフェース記述の方法の分析・検討のために、SOAP 技術におけるメッセージ定義の方法、メッセージ変換について調査を行った。その結果導き出された、「共通コンポーネント基盤およびサービス連携基盤」にて実現すべき仕様を以下に示す。

4.8.1 インタフェース

本調査では SOAP 技術におけるメッセージ定義の方法、メッセージ変換について主に調査を行った。JAX-WS、Axis2、XFire などの仕様や実装のアーキテクチャを参考に D-BUS およびデスクトップ基盤を Web サービスに対応付けるとすると、プロトコルに SOAP を選択する場合は、D-BUS のメッセージと SOAP ボディとの対応付けが基本になる。D-BUS の ErrorMessage は、SOAP フォルトと対応付ける。

その際、D-BUS メッセージと SOAP ボディとのマッピングでは、型の対応付けがポイントとなる。「D-BUS 調査報告書」5.2 節に記述している D-BUS の型のうち、OBJECT_PATH、ARRAY、DICT_ENTRY が直接対応付ける対象が無い。一方、SOAP 側の型には表 4-3に示す基本型のいくつかと列挙型が D-BUS の型に対応付けられない。これらについてはマッピングのルールによって対応付け可能とする。詳細は 7.8 (3)を参照のこと。

表 4-2D-BUS の型と SOAP の型の対応

No.	D-BUS 型名	SOAP の型への対応
1	BYTE	Byte
2	BOOLEAN	Boolean
3	INT16	Short
4	UINT16	unsignedShort
5	INT32	Int
6	UINT32	unsignedInt
7	INT64	Long
8	UINT64	unsignedLong
9	DOUBLE	Double
10	STRING	String
11	OBJECT_PATH	直接対応付ける対象が無い
12	ARRAY	直接対応付ける対象が無い
13	STRUCT	構造体
14	DICT_ENTRY	直接対応付ける対象が無い

表 4-3 D-BUS の型に対応付け不可能な SOAP 単純型

No.	SOAP 単純型
1	normalizedString
2	Token
3	base64Binary
4	hexBinary
5	Integer
6	positiveInteger
7	negativeInteger
8	nonNegativeInteger
9	nonPositiveInteger
10	unsignedByte
11	Decimal
12	Float
13	Duration
14	dateTime
15	Date
16	Time
17	gYear
18	gYearMonth
19	gMonth
20	gMonthDay
21	gDay
22	Name
23	QName
24	NCName
25	anyURI
26	Language
27	ID
28	IDREF

29	IDREFS
30	ENTITY
31	ENTITIES
32	NOTATION
33	NMTOKEN
34	NMTOKENS

4.9 JAX-WS

サービス連携基盤の設計における通信方式とインタフェース記述の方法の分析・検討のために、JAX-WS における Java と SOAP のマッピング仕様について調査を行った。その結果導き出された「共通コンポーネント基盤およびサービス連携基盤」にて実現すべき仕様を以下に示す。

4.9.1 SOAP へのマッピング

共通コンポーネント基盤およびサービス連携基盤では、D-BUS ベースの通信方式を採用する方針であるが、このとき各コンポーネントがサービス連携を行おうとした場合、D-BUS メッセージと SOAP メッセージのマッピングが必要となる。JAX-WS における Java と SOAP とのマッピングは、D-BUS メッセージを SOAP メッセージに対応付ける仕様の参考にできる。具体的には、「D-BUS 調査報告書」の 5.1 に記述したメッセージのヘッダフィールドを Java と対応付けると、表 4-4 のようになる。

表 4-4 D-BUS のヘッダフィールド

No.	D-BUS フィールド種別	型	対応する Java の要素
1	PATH	OBJECT_PATH	Package
2	INTERFACE	STRING	Interface
3	MEMBER	STRING	Method
4	ERROR_NAME	STRING	Exception
5	REPLY_SERIAL	UINT32	特になし
6	DESTINATION	STRING	Package
7	SENDER	STRING	特になし

この構造は、「D-BUS 調査報告書」の 4.1 に記述した、D-BUS のイントロスペクションともほぼ同じである。

また、JAX-WS にて導入されている Annotation によるマッピング関連メタデータの付与については、「D-BUS 調査報告書」の 4.1 に記述した D-BUS のイントロスペクションにも同様の付加情報を記述できる Annotation が存在するため、それに対応付けることで同じアーキテクチャを実現する。

4.9.2 同期・非同期

各コンポーネントがサービス連携する場合、「JAX-WS 調査報告書」の 8.1 に述べたマッピング以外に、メッセージングの同期性も検討する必要がある。D-BUS は「D-BUS 調査報告書」の 5.1 に記述したとおり、同期呼び出しと、非同期呼び出しのコールバックのみサポートしている。非同期呼び出しのポーリングは、Web サービスの仕組みとして一般的にサポートされているため、D-BUS にもポーリングの仕組みの実装を検討する必要がある。

しかし、ポーリングはコールバックでラッピングすることが可能である。また、ポーリングの仕組みをデスクトップアプリケーションが利用することを想定すると、定期的な結果チェックはリアルタイム性に影響が出る可能性がある。よって、ポーリングをサポートするのか、コールバックでラッピングするのかは、今後のユーザからの要望に合わせて検討する課題とする。

4.10 Axis2

サービス連携基盤の設計における通信方式とインタフェース記述の方法の分析・検討のために、Axis2 における Java と SOAP のマッピング仕様およびそれを実現する実装方式について調査を行った。その結果導き出された、「共通コンポーネント基盤およびサービス連携基盤」にて実現すべき仕様を以下に示す。

4.10.1 WSDL からのコード生成

共通コンポーネント基盤およびサービス連携基盤では、D-BUS ベースの通信方式を採用する方針であるが、このとき各コンポーネントがサービス連携を行おうとした場合、D-BUS メッセージと SOAP メッセージのマッピングが必要となる。本節では、そのマッピングを実現する為の実装上の方式について記述する。

Web サービスのインタフェース定義 (WSDL) は当然ながら Web サービス毎に異なる。マッピング処理を行う際にはインタフェースに即したマッピングを行う必要がある。Axis2 では WSDL からスタブやスケルトンコードを生成することによりこれを実現しており、この Axis2 のアーキテクチャを参考に仕様を策定できる。ただし、Axis2 が生成するのは Java や C++ のコードである (つまり Java/C++ からダイレクトに SOAP へ変換する) のに対して、共通コンポーネント基盤上のコンポーネントは D-BUS で通信を行うため、対応する D-BUS メッセージの定義と SOAP との変換ルールを生成することになる。そして、実際のプログラムは、その D-BUS メッセージを送信することで Web サービスの呼び出しを実現する。詳細は 7.8 を参照のこと。

問題は、D-BUS を利用するサービスからは、SOAP への変換方法が見えないために、どのような D-BUS メッセージを送ればよいか分からないことである。それを補うためには、対応する D-BUS メッセージを生成するような C や C++ のコードを生成することとする。

4.10.2 SOAP へのマッピング

Axis2 では、「Axis2 調査報告書」5.1 で述べた通りインタフェースに即したマッピング処理は WSDL から生成したコードが行っているが、インタフェースに依らない処理、例えば SOAP ヘッダの解析、対応するサービスオブジェクト (アプリケーション実装) の特定、リクエストのディスパッチ等は、「Axis2 調査報告書」3.7 に示すとおり Handler 機構で実現している。サービス連携のためのモジュールを作成し、DCOP、ORBit2、XPCOM、UNO などと同列の技術として共通コンポーネント基盤に乗せることを考えた場合、Axis2 と同様に Handler によって SOAP メッセージを構成するアーキテクチャを採用したモジュールとすることで、D-BUS メッセージから SOAP への変換を実現する。詳細は 7.8 を参照のこと。

4.11 XFire

サービス連携基盤の設計における通信方式とインタフェース記述の方法の分析・検討のために、XFire における Java と SOAP のマッピング仕様およびそれを実現する実装方式について調査を行った。その結果導き出した、「共通コンポーネント基盤およびサービス連携基盤」にて実現すべき仕様を以下に示す。

4.11.1 SOAP へのマッピング

XFire では、インタフェースに即したマッピング処理は WSDL から生成したコードが行っている («XFire 調査報告書」7.2 を参照のこと) が、インタフェースに依らない処理、例えば SOAP ヘッダの解析、対応するサービスオブジェクト (アプリケーション実装) の特定、リクエストのディスパッチ等は、「XFire 調査報告書」の 3 章に示すとおり Handler 機構で実現している。サービス連携のためのモジュールを作成し、DCOP、ORBit2、XPCOM、UNO などと同列の技術として共通コンポーネント基盤に乗せることを考える。その場合、同様に Handler によって SOAP メッセージを構成するアーキテクチャを採用したモジュールとすることで、D-BUS メッセージから SOAP への変換を実現することとする。詳細は 7.8 を参照のこと。

4.12 JBI

共通コンポーネント基盤として抽出すべきオブジェクト管理や通信の方式、メッセージ定義・機構の分析・検討のため、Java による共通サービスコンポーネント基盤である JBI(Java Business Integration)に関して、サービス統合手法の調査を行った。その結果導き出された、「共通コンポーネント基盤およびサービス連携基盤」にて実現すべき仕様を以下に示す。

4.12.1 通信

JBI の Normalized Message(NM)と Message Exchange(ME)は、「D-BUS 調査報告書」の 5 章に記述している D-BUS メッセージのボディとヘッダの関係と似ていると考えることができる。実際、ME の送信先に相当する EndPoint は D-BUS のヘッダフィールドの Path に相当する。

そのように見ると、NM に含まれているコンテキスト情報が D-BUS のメッセージには不足していることが分かる。また、ルーティング時の送信先の名前解決に使用できる EndPoint Reference (EPR) のような仕組みも無い。これらをヘッダフィールドに追加するのか、ボディに追加するのかは検討の余地があるが、Web サービス機能を実現するためには必要である。D-BUS のメッセージ拡張のためには、D-BUS の開発コミュニティとの連携が必要となるため、今後の課題とする。

4.12.2 コンポーネント管理とコンポーネントライフサイクル

JBI では、コンポーネント管理をするためのインタフェースが決まっている。このインタフェースが持つメソッド呼び出しによって、コンポーネントのライフサイクルが制御されている。コンポーネントを管理するためのインタフェースを決めるというアイデアは、共通コンポーネント基盤と各コンポーネント技術とのコンポーネントライフサイクル管理の同期を取る際に参考に仕様を策定できる。具体的には、共通コンポーネント基盤と各コンポーネント技術における GC の連携を実現するために、インタフェースを共通コンポーネント基盤側に登録し、共通基盤でコントロールする仕様とする。しかし、共通コンポーネントでの GC の実現は難しいため、プロトタイプを作成し検証した上で決定する今後の課題とする。詳細は、7 章を参照のこと。

4.13 ServiceMix

共通コンポーネント基盤として抽出すべきオブジェクト管理や通信の方式、メッセージ定義・機構の分析・検討のため、Java による共通サービスコンポーネント基盤である JBI(Java Business Integration)のオープンソース実装である ServiceMix に関して、ServiceMix における JBI の通信方法、コンポーネントの追加・削除方法の調査を行った。その結果導き出された、「共通コンポーネント基盤およびサービス連携基盤」にて実現すべき仕様を以下に示す。

4.13.1 複数メッセージ通信プロトコル間のマッピング

ServiceMix は JBI の実装であるため、インタフェースと構成のみが定義されていた Normalized Message(NM)と Message Exchange(ME)が実際に実装されている。NM と ME については「JBI 調査報告書」3.1 にてその詳細を述べている。

それらの実装の中で特に着目すべきは、これらのメッセージと外部プロトコル(HTTP、SOAP、JMS 等)のメッセージとの相互変換である。この点は、JBI には特に記述されていなかった。実際の処理の中身を見てみると、各プロトコルの内容に合わせて、データを NM に一つ一つ当てはめたり、その逆が行われたりしているのが分かる。XFire のような外部の変換モジュールを利用する場合も、委譲しているだけで、本質は同じである。

Web サービスとの連携を考えるとときには、同様に共通のインタフェースを決めた上で、実際の変換は各プロトコルで実装する。その実現は、内部仕様の検討が必要となるため今後の課題とする。

4.14 SCA

共通コンポーネント基盤として抽出すべきオブジェクト管理や通信の方式、メッセージ定義・機構の分析・検討のため、サービス指向アーキテクチャに基づいた実装のオープンな仕様である SCA(Service Component Architecture)に関して、SCA におけるサービス統合手法の調査を行った。その結果導き出された、「共通コンポーネント基盤およびサービス連携基盤」にて実現すべき仕様を以下に示す。

4.14.1 サービスアセンブリモデル

SCA のサービスアセンブリモデルでは、特定の技術や言語に依存せずにサービスの結合関係を記述することができる。粒度は異なるが、共通コンポーネント基盤でも抽象コンポーネントは、特定の技術や言語に依存しないコンポーネントの結合として記述する必要がある。そして、その組み合わせとしてアプリケーションを記述できると、コンポーネントの再利用性が高まり、開発効率も向上する。

具体的には、複数のコンポーネント記述をまたがったコンポーネントの組み合わせ(例: OpenOffice の Calc 内のスプレッドシートと KDE の KOrganizer 内のアドレス帳)としてあるアプリケーションを作成する場合、アプリケーションの機能やインタフェースは特定のコンポーネントに依存せずに記述できるため、各コンポーネントの結合度は低下し、効率のよいアプリケーション構築が可能となる。

そのためには、SCA のサービスアセンブリモデルが仕様策定の参考になる。具体的には、interface 要素をコンポーネントの機能に、Component 要素をコンポーネントに、Composite 要素をアプリケーションに対応させ、アプリケーションを記述することとする。その実現方法は、プロトタイプを作成し検証した上での決定が必要となるため、今後の課題とする。

4.14.2 複数のメッセージプロトコル間のマッピング

SCA では、複数プロトコル間のマッピングは Binding 要素として記述することになっている。この Binding 要素は各プロトコルやインタフェース記述によって拡張可能になっている。しかし、Binding 要素における各プロトコル、インタフェース記述の処理方法については、SCA では定義されておらず、SCA 仕様の実装に依存する。

同様に共通コンポーネント基盤で Web サービスを含めた複数のプロトコル間の通信を実現する場合には、通信関係の定義のみを共通コンポーネント基盤の仕様として決め、複数プロトコル間のマッピングはプロトコルに依存した形で実装することとする。

4.15 Tuscany

共通コンポーネント基盤として抽出すべきオブジェクト管理や通信の方式、メッセージ定義・機構の分析・検討のため、Apache のインキュベーションとして承認されたプロジェクト Tuscany に関して、サービス統合手法、複数メッセージ通信プロトコル間のマッピングの仕様の調査を行った。その結果導き出された、「共通コンポーネント基盤およびサービス連携基盤」にて実現すべき仕様を以下に示す。

4.15.1 複数メッセージ通信プロトコル間のマッピング

Tuscany では SCA 仕様に基づいた SCA の Binding 要素を参照し、実行環境が Java インタフェースを自動生成、複数プロトコル間の通信を可能とすることがわかった。例えば Web サービスに対応した Binding 要素の場合、Tuscany ではこの記述から対応する WSDL などのインタフェース記述を参照し、Java インタフェースを自動生成することで、他技術との通信を可能としている。

同様に共通コンポーネント基盤で Web サービスを含めた複数のプロトコル間の通信を実現する場合には、共通コンポーネント基盤が外部に対して、どのようにインタフェースを公開し、どのように外部インタフェースへアクセスするかの記述が問題となる。この問題を解決するのに、上記の SCA Binding 要素とその拡張モデルを参考に仕様を策定する。その実現方法は、プロトタイプを作成し検証した上での決定が必要となるため、今後の課題とする。

5 システム構成

5.1 本構成の特徴

本論理構成は、1.2 で示した背景を踏まえ、さらに 1.3 に示した目的を達成するために、以下のような特徴を持つ設計となっている。

- (1) D-BUS のプリミティブなメッセージ通信バスとしての機能を活かして、コンポーネント技術間の連携を実現している。
D-BUS 上に新たにコンポーネント技術を構築することは行わない。
抽象コンポーネントという概念を導入することで、モデルの共通化を行う。
- (2) カスタムアプリケーションを作成するために利用するコンポーネントを抽象化することで、アプリケーション開発を容易にする。
- (3) 既存コンポーネント技術との整合性を考慮した漸進的な設計である。
現状のオープンソースのデスクトップ環境におけるコンポーネント技術は、歴史的にすぐに一本化することは難しい。
コミュニティに受け入れてもらいやすくするために、各技術のインターフェース変更を最小限にする。

5.2 論理構成

本プロジェクトで実現すべきシステムの論理構成は以下の通りとする。

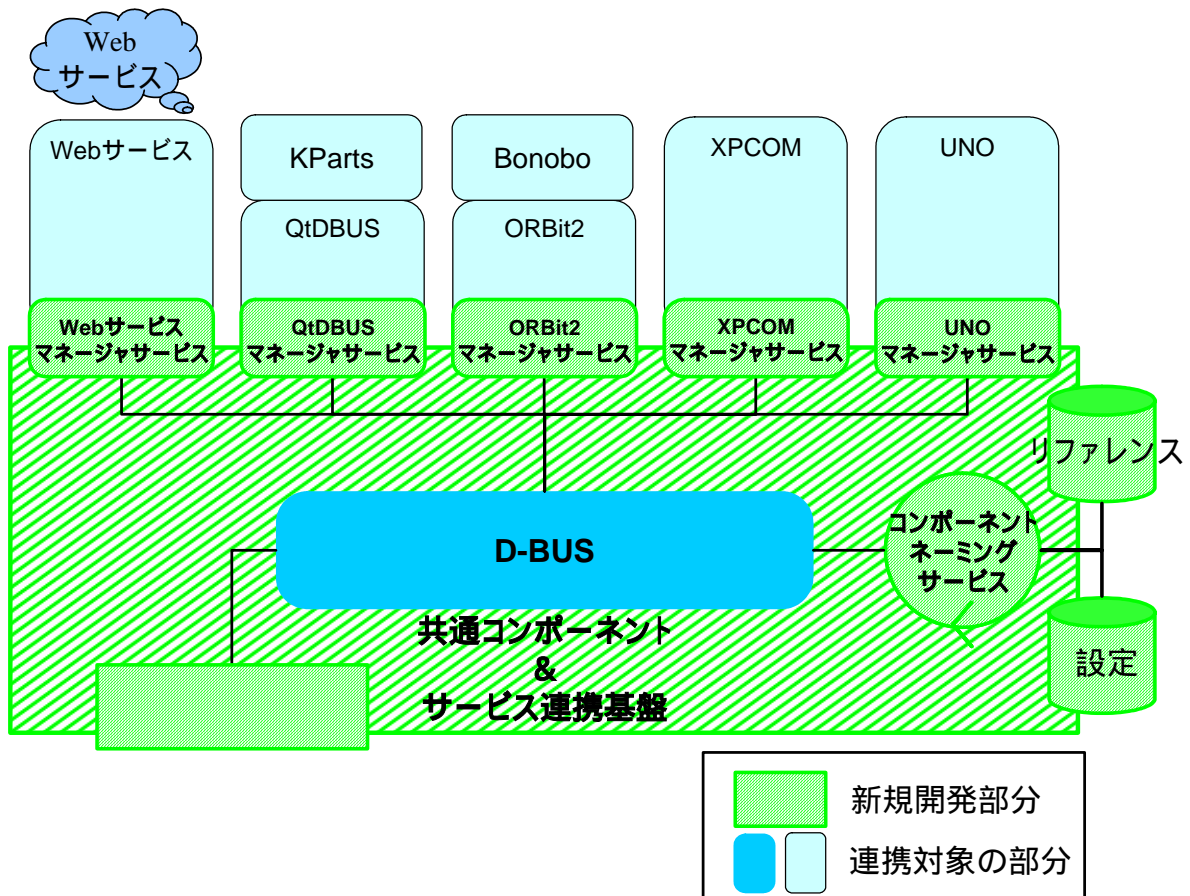


図 5-1 システムアーキテクチャ

技術仕様書

実装するものは、緑色の斜線で記述した部分になる。具体的には、以下の通り。

なおこれ以降、デスクトップ技術のうち通信技術とコンポーネント技術が分かれているもの（Bonobo と ORBit2、KParts と QtDBUS）は、通信技術の名称（ORBit2、QtDBUS）をその代表として表現する。

(1) コンポーネントネーミングサービス

- コンポーネントサービスを起動する。
 - D-BUS を起動する。
 - D-BUS の起動に伴い、Session Bus も起動する。
 - Session BUS 起動直後にアプリケーション名「:component_namingservice」でコンポーネントネーミングサービスが起動する。
- 抽象コンポーネントの生成、削除、検索を実行する。
 - カスタムアプリから生成を要求された抽象コンポーネントについて、設定を参照して生成する。生成したオブジェクトのオブジェクトパス、抽象コンポーネント名、抽象コンポーネントの継承関係をリファレンスに登録する。詳細は7.2に記述。
 - カスタムアプリで削除された抽象コンポーネントを削除する。詳細は7.7に記述。
 - カスタムアプリから検索を要求された抽象コンポーネントに対応する具象コンポーネントをリファレンスから検索する。詳細は7.5に記述。

(2) QtDBUS マネージャサービス

- D-BUS 上のアプリケーションとして、コンポーネントネーミングサービスの起動直後から動作する（アプリケーション名「:qtdbus_mgr」）。
- 具象コンポーネントとそれに対するアダプタを生成し、アダプタを保持する。
- D-BUS を経由して呼び出されたメソッド呼び出しやシグナルを QtDBUS 上のオブジェクトに転送する。

(3) ORBit2 マネージャサービス

- D-BUS 上のアプリケーションとして、コンポーネントネーミングサービスの起動直後から動作する（アプリケーション名「:orbit2_mgr」）。
- 具象コンポーネントとそれに対するアダプタを生成し、アダプタを保持する。
- D-BUS を経由して呼び出されたメソッド呼び出しやシグナルを Bonobo/ORBit2 の通信に変換する。

(4) XPCOM マネージャサービス

- D-BUS 上のアプリケーションとして、コンポーネントネーミングサービスの起動直後から動作する（アプリケーション名「:xpcom_mgr」）。
- XPCOM と D-BUS の proxy として動作する。
- 具象コンポーネントとそれに対するアダプタを生成し、アダプタを保持する。
 - 具象コンポーネントの生成時に、Firefox、Thunderbird などの対応するアプリケーションを起動する。
 - アプリケーションが起動済みの場合は新たに起動を行わない。
- D-BUS を経由して呼び出されたメソッド呼び出しやシグナルを XPCOM 上の通信に変換する。

(5) UNO マネージャサービス

- D-BUS 上のアプリケーションとして、コンポーネントネーミングサービスの起動直後から動作する（アプリケーション名「:uno_mgr」）。
- 具象コンポーネントとそれに対するアダプタを生成し、アダプタを保持する。
 - 具象コンポーネントの生成時に OpenOffice.org を起動する。
 - 起動済みの場合は新たに起動を行わない。
- D-BUS を経由して呼び出されたメソッド呼び出しやシグナルを UNO 上の通信に変換する。

技術仕様書

- (6) Web サービスマネージャサービス
 - D-BUS 上のアプリケーションとして、コンポーネントネーミングサービスの起動直後から動作する（アプリケーション名「:webservice_mgr」）。
 - Web サービスの WSDL を事前に読み込んで生成したインタフェースを保持する。
 - 起動時にインタフェースに対応するアダプタを生成し、コンポーネントネーミングサービスに登録する。
 - D-BUS を経由して呼び出されたメソッド呼び出しやシグナルを Web サービスのメッセージとして転送する。

- (7) カスタムアプリ作成ライブラリ
 - 「共通コンポーネント基盤とサービス連携基盤」を使ったカスタムアプリケーションを開発する際に利用するライブラリ。
 - 抽象コンポーネントの生成要求をコンポーネントネーミングサービスに転送する。
 - 生成した抽象コンポーネントへの proxy を生成する。

6 IDL の仕様

「共通コンポーネント基盤とサービス連携基盤」における IDL は、D-BUS のイントロスペクション（「D-BUS 調査報告書」4 章参照）を以下のように変更し、XML 形式で記述することとする。

- 抽象コンポーネントを表す属性として interface 要素を用いる。
- interface 要素に継承関係を示す extends 属性を追加できるものとする。
 - 複数の interface を継承している場合には、カンマ区切りで列挙する。
- method 要素に非同期通信であることを明示する org.freedesktop.DBus.Method.Async アノテーション付加できるものとする。
- signal 要素と property 要素は使用できないものとする。

DTD は以下の通り。

```
<!ELEMENT interface (annotation*,method*)>
<!ATTLIST interface name CDATA #REQUIRED>
<!ATTLIST interface extends CDATA #IMPLIED >

<!ELEMENT method (annotation*,arg*)>
<!ATTLIST method name CDATA #REQUIRED>

<!ELEMENT arg EMPTY>
<!ATTLIST arg name CDATA #IMPLIED>
<!ATTLIST arg type CDATA #REQUIRED>
<!-- Method arguments SHOULD include "direction",
      while signal and error arguments SHOULD not (since there's no point).
      The DTD format can't express that subtlety. -->
<!ATTLIST arg direction (in|out) "in">

<!ELEMENT annotation EMPTY> <!-- Generic metadata -->
<!ATTLIST annotation name CDATA #REQUIRED>
<!ATTLIST annotation value CDATA #REQUIRED>
```

図 6-1 IDL の DTD

IDL を記述する際の XML タグ構造以外の仕様については以下にまとめる。

6.1 インタフェース名の規則

インタフェース(タグ名:interface)は有効な UTF8 である STRING 型の名前(属性名:name)を持つ。インタフェース名の制限としては以下のものがある。

- インタフェース名は 1 つ以上のピリオドで区切られた要素で構成される。全ての要素は 1 文字以上になる。
- 全ての要素は ASCII 文字("[A-Z][a-z][0-9]_")のみが使用され、数字以外で始まる必要がある。
- インタフェース名は一つ以上のピリオドを含む必要がある。つまり必ず 2 要素以上で構成する。
- インタフェース名はピリオドで始まってはいけない。
- インタフェース名は名前の最大長 (255 文字) を超えてはいけない。

6.2 メソッド名の規則

メソッド(タグ名:method)は有効な UTF8 である STRING 型の名前(属性名:name)を持つ。インタフェース名の制限としては以下のものがある。

- ASCII 文字("[A-Z][a-z][0-9]_")のみが使用され、数字以外で始まる必要がある。
- ピリオド文字は含めてはいけない。
- 名前の最大長 (255 文字) を超えてはいけない。
- 最低でも 1 文字以上の長さであること。

6.3 引数名の規則

メソッド名と同じ規則になる。

6.4 アノテーションの規則

アノテーション(タグ名 : annotation)は、名前(属性名:name)と値(属性名:value)を持つ。名前と値は以下の組み合わせのみが指定可能となる。

表 6-2 アノテーションで指定できる名前と値の組み合わせ

No.	名前	値	デフォルト	値
1	org.freedesktop.DBus.Method.NoReply	true false	false	true ならばメソッド呼び出しの返答を期待しない。
2	org.freedesktop.DBus.Method.Async	true false	false	true ならば関数呼び出しは非同期呼び出しとなる。

6.5 IDL で指定可能な型名

型として IDL に指定可能な型名は以下の通りとなる。

表 6-3 IDL で指定可能な型名

No.	型名	説明
1	INVALID	仕様に一致しない型。
2	BYTE	8 ビット符号なし整数。
3	BOOLEAN	0 が FALSE、1 が TRUE を表す boolean 型。
4	INT16	16 ビット符号つき整数。
5	UINT16	16 ビット符号なし整数。
6	INT32	32 ビット符号つき整数。
7	UINT32	32 ビット符号なし整数。
8	INT64	64 ビット符号つき整数。
9	UINT64	64 ビット符号なし整数。
10	DOUBLE	IEEE754 で規定されている double 型。
11	STRING	UTF8 の文字列型。終端文字として nul が含まれていること。
12	OBJECT_PATH	オブジェクトインスタンスの名前。長さ制限なし。 例：/org/freedesktop/sample_object
13	SIGNATURE	型シグニチャ。最大 255 バイト。再帰構造の深さ制限は、ARRAY が 32、STRUCT が 32 となっている。
14	ARRAY	配列。例えば INT32 を二つ含む配列は”aii”となる。
15	STRUCT	構造体。INT32 と UINT32 を一つずつ含む構造体は、”r(iu)”となる。
16	VARIANT	バリエーション型。
17	DICTIONARY_ENTRY	キーと値のペアの配列で表される辞書型。

7 機能仕様

7.1 抽象コンポーネント定義機能

抽象コンポーネントを定義するために6章のIDL の仕様に示した IDL を利用する。各具象コンポーネントから抽象コンポーネントを利用するため、抽象コンポーネントの IDL から具象コンポーネントの IDL に変換する。変換先の具象コンポーネントの IDL は QtDBus IDL、CORBA IDL、XPIDL、UNOIDL である。

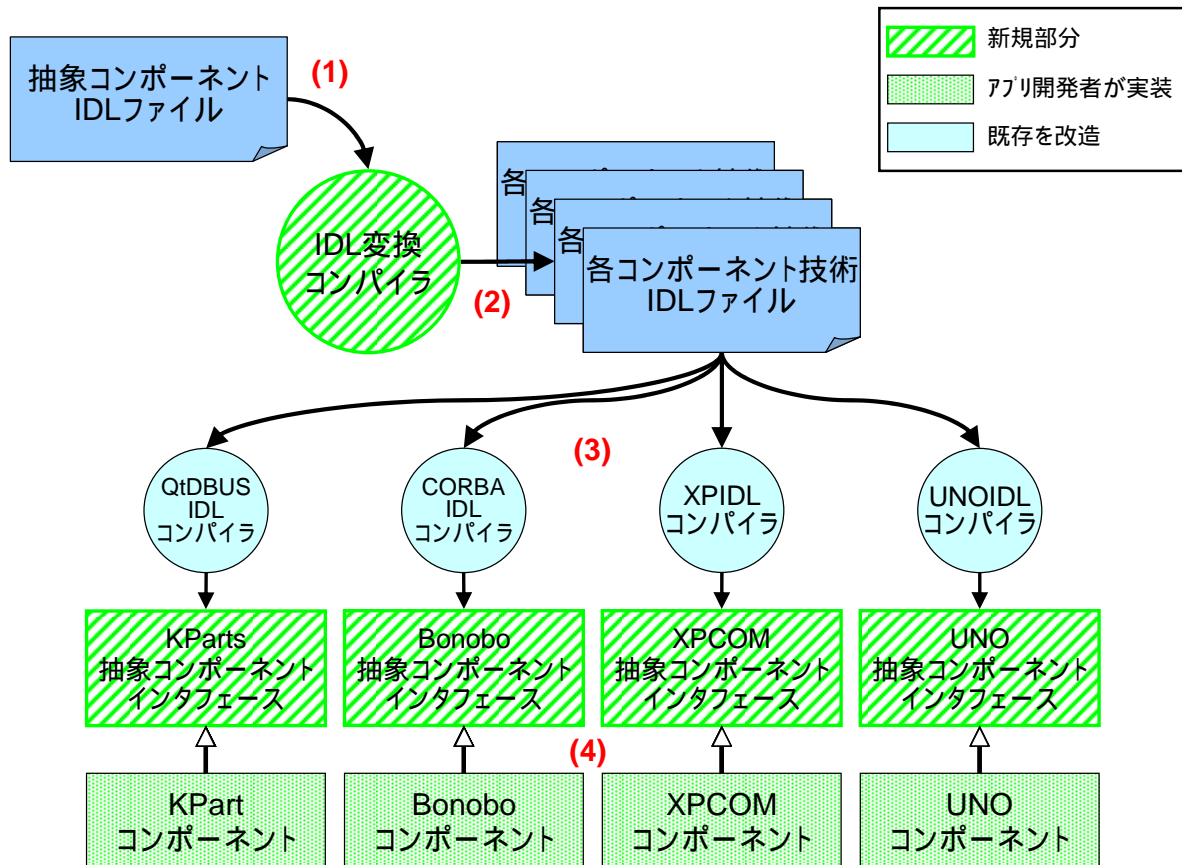


図 7-1 抽象コンポーネントの定義

- (1) 抽象コンポーネントの IDL 作成 (図 7-1)
 - 6 章に記述した IDL を用いて抽象コンポーネントの IDL を作成する。
- (2) IDL の変換 (図 7-1)
 - 抽象コンポーネントの IDL を本システム (フレームワーク) の IDL 変換コンパイラにより、各コンポーネント技術の IDL に変換する。
 - 4 章の各既存技術と共通デスクトップ基盤の対応およびメッセージ通信の仕様に記述したように、既存の各コンポーネント技術には、抽象化された IDL からマッピングできない部分があるため、その部分について拡張する必要がある
- (3) IDL のコンパイル (図 7-1)
 - 各コンポーネント技術の IDL コンパイラを用いて、抽象コンポーネントのインタフェースを生成する。
- (4) 具象コンポーネントの実装 (図 7-1)
 - 生成した抽象コンポーネントのインタフェースを継承して、具象コンポーネントを実装する。

7.2 抽象コンポーネントの生成機能

抽象コンポーネントの生成処理の流れを、指定する抽象コンポーネントに対応する具象コンポーネントが UNO のコンポーネントである場合を例に図 7-2～図 7-8で示す。それに合わせて、通信、設定、リファレンスに利用および保存するデータ形式を記述する。なおこれ以降は各コンポーネント技術の上位の技術については省略する。また図中の(1)～(10)、～等の番号は、説明文の番号と対応している。

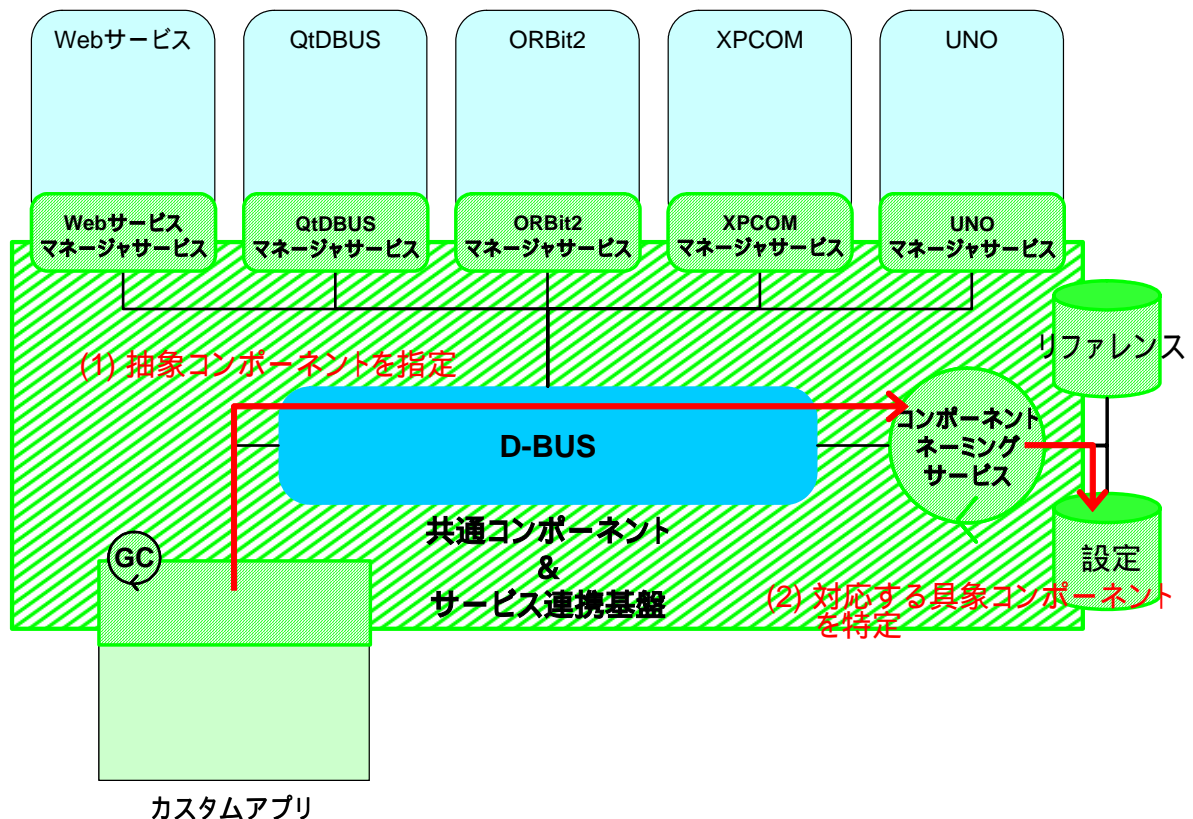


図 7-2 抽象コンポーネントの生成 1

(1) 抽象コンポーネントの指定 (図 7-2)

- カスタムアプリケーションは、カスタムアプリ作成ライブラリを利用して、コンポーネントネーミングサービス (:component_namingservice) に対して抽象コンポーネント (例: Spreadsheet) の生成を要求する。

(2) 対応する具象コンポーネントを特定 (図 7-2)

- コンポーネントネーミングサービス (:component_namingservice) は、生成を要求された抽象コンポーネント (例: spreadsheet) に対応する具象コンポーネントの設定を読み込み、生成する具象コンポーネントを特定する。
- 抽象コンポーネントに対応する具象コンポーネントの設定は、XML 形式とする。フォーマットは、以下の通り。
 - abstractcomponent 要素: 抽象コンポーネントの設定エン트리
 - name 属性: 抽象コンポーネントの名前
 - mgrservice 属性: 具象コンポーネントの生成を要求する (ライフサイクルを管理する) マネージャサービスの名前
 - component 属性: 具象コンポーネントの名前

```
<abstractcomponents>
  <abstractcomponent name="spreadsheet" mgrservice=":uno_mgr" component="com.sun.star.sheet.Spreadsheet"/>
  <abstractcomponent name="wordprocessor" mgrservice=":qtdbus_mgr" component="KWDDocument"/>
</abstractcomponents>
```

図 7-3 抽象コンポーネントの設定例

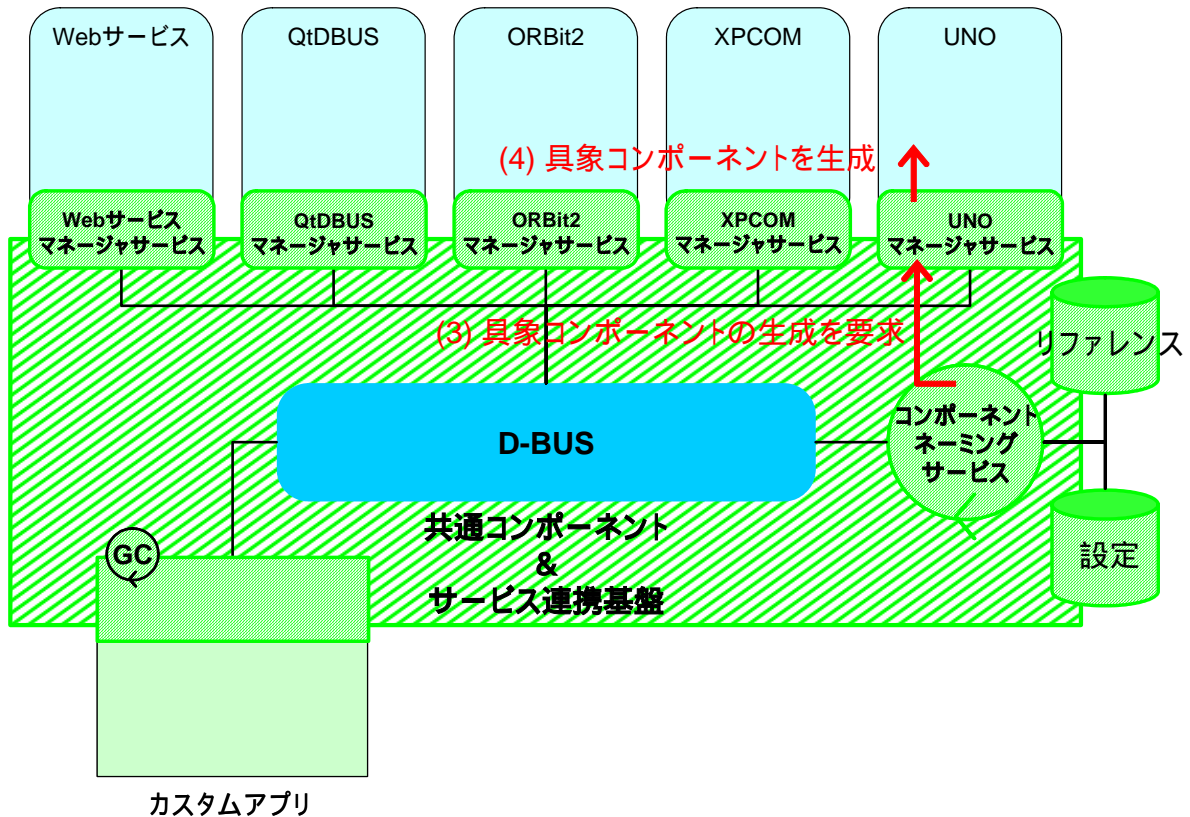


図 7-4 抽象コンポーネントの生成 2

- (3) 具象コンポーネントの生成を要求 (図 7-4)
- コンポーネントネーミングサービスは、設定から読み込んだ各マネージャサービス (例えば UNO の場合は:uno_mgr) に対して、具象コンポーネントの生成を要求する。
 - 生成の要求時には、具象コンポーネント名を指定する。
- (4) 具象コンポーネントを生成 (図 7-4)
- 各マネージャサービスが担当しているコンポーネント技術の仕組みに従って、具象コンポーネントを生成する。
 - 例えば UNO の場合は、以下のような手順を踏んで具象コンポーネントを生成する。
 - OpenOffice.org を外部からアクセス可能なリスニングモードとして起動する (図 7-5)。
 - OpenOffice.org に対して要求した具象コンポーネントの生成を要求する (図 7-5)。

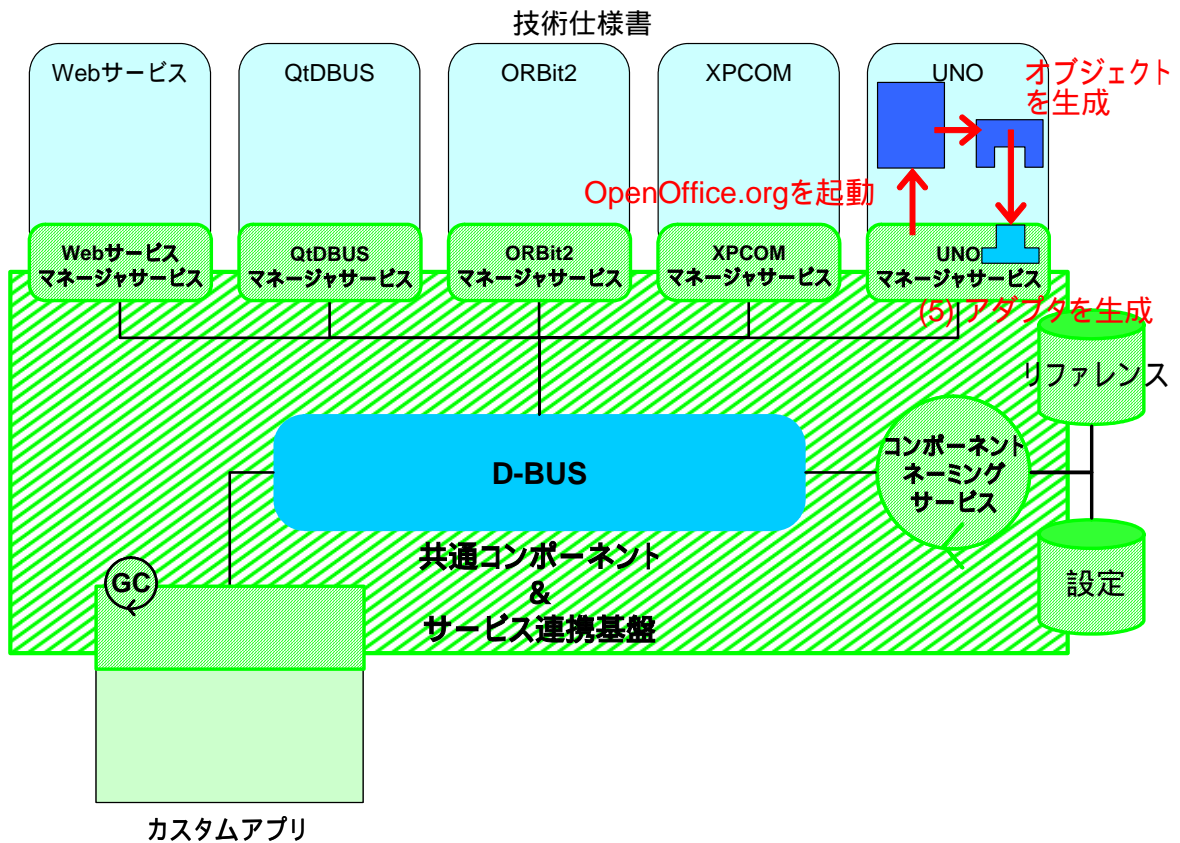


図 7-5 抽象コンポーネントの生成 3

(5) アダプタを生成 (図 7-5)

- 生成した具象コンポーネントに対する D-BUS メッセージを各コンポーネント技術における通信方式に変換するアダプタを生成する。

技術仕様書

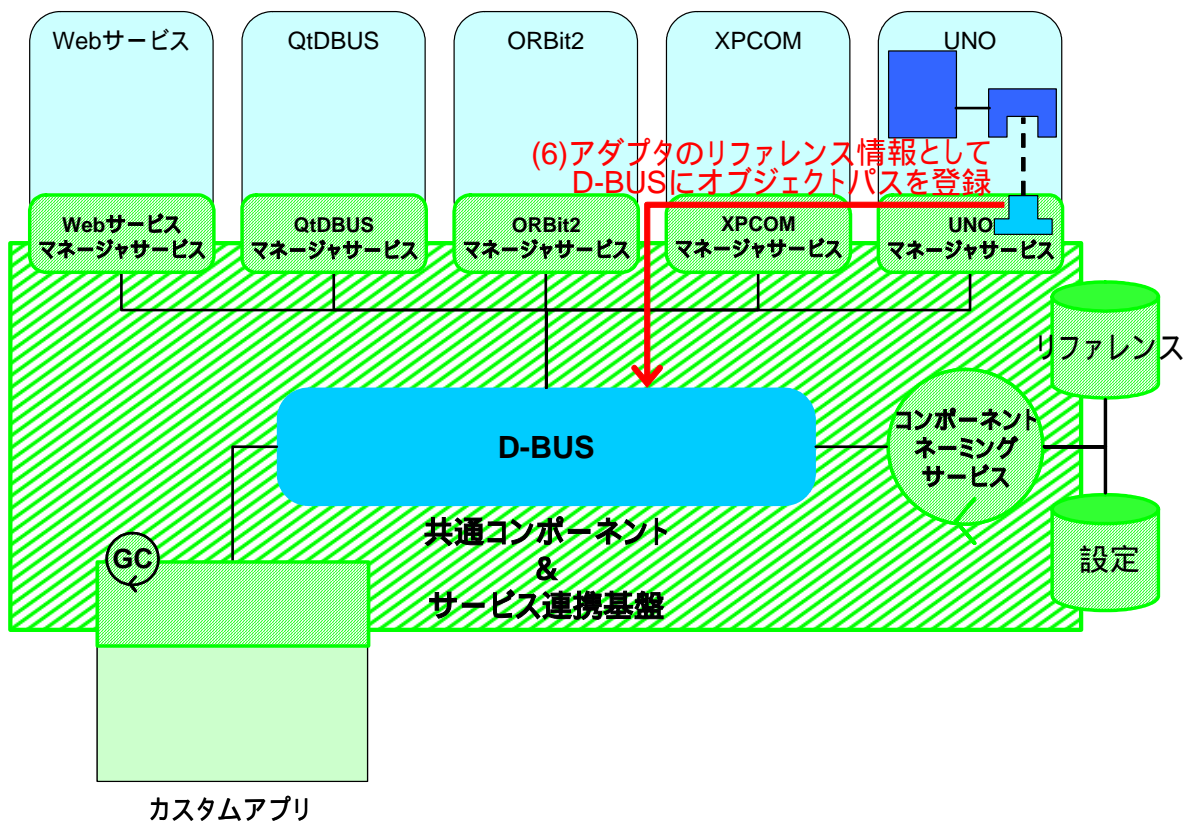
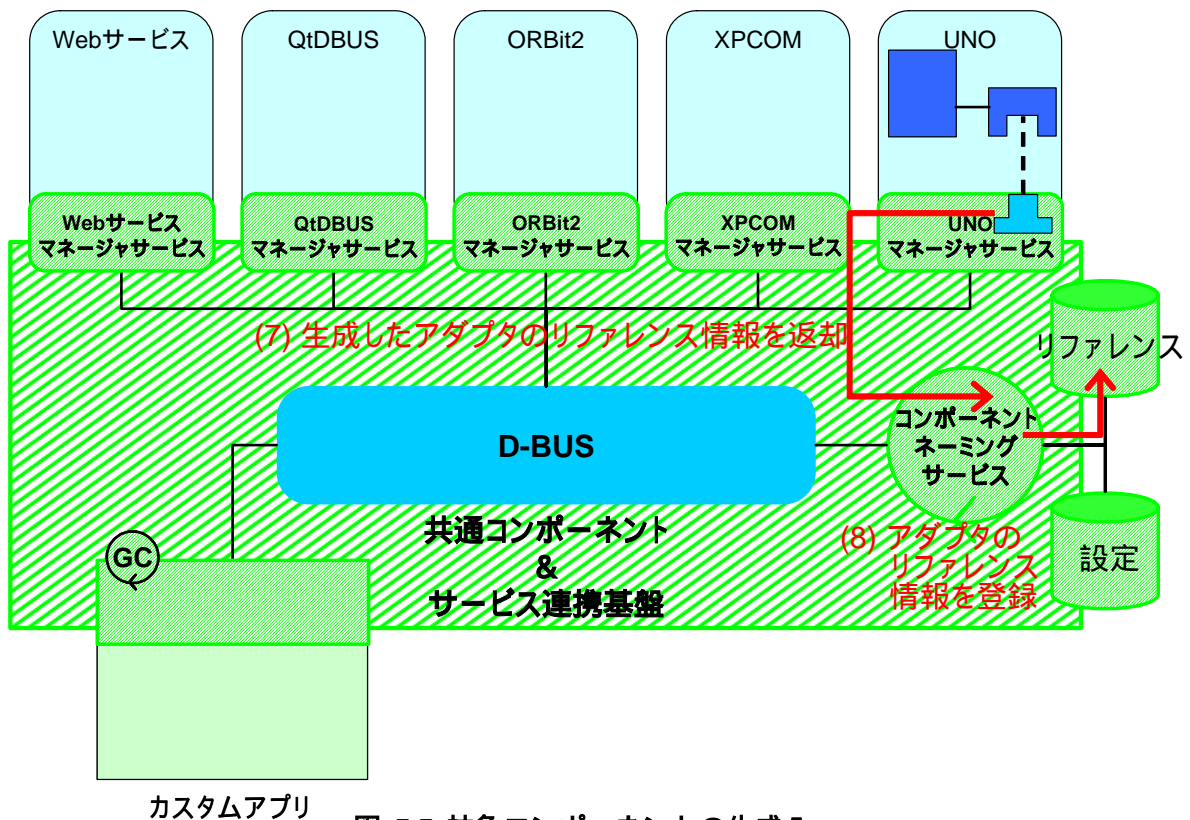


図 7-6 抽象コンポーネントの生成 4

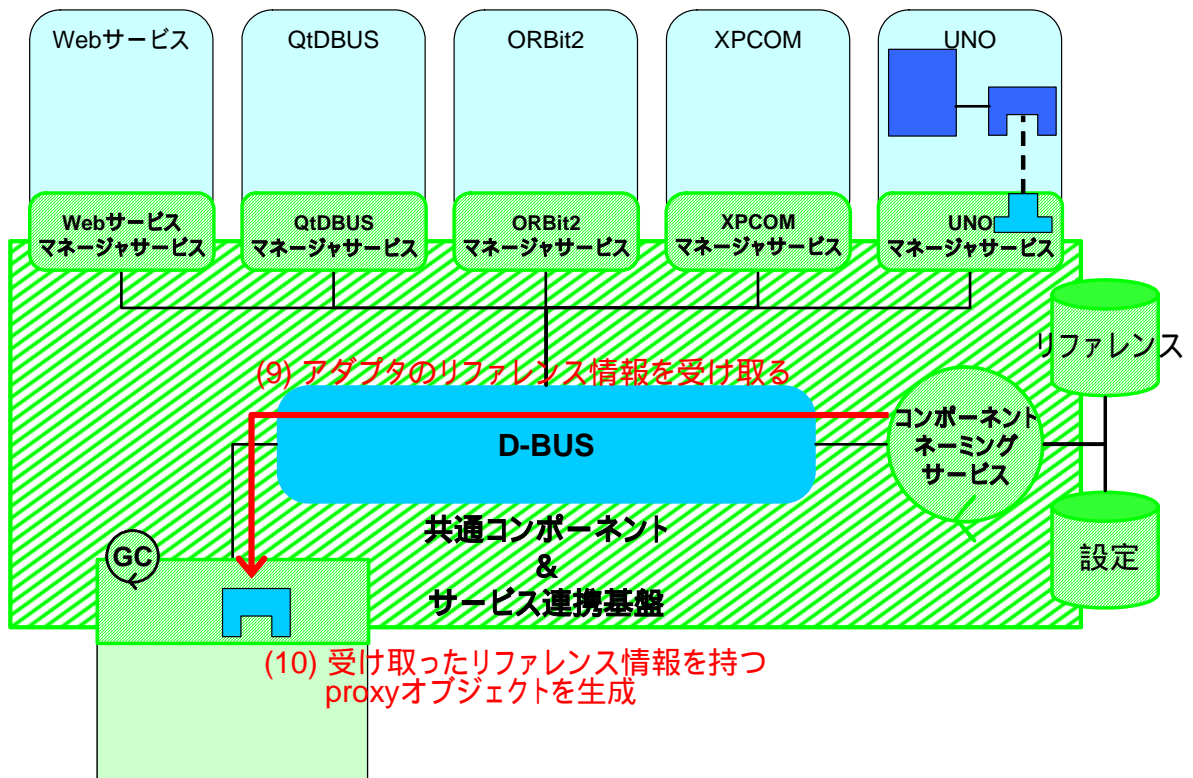
- (6) アダプタのオブジェクトパスを D-BUS に登録 (図 7-6)
- 各マネージャサービス上のオブジェクトとしてアダプタのオブジェクトパスを登録する。
 - /[アプリケーション名]/[クラス ID]/[オブジェクト ID]
 - 例 : /OpenOffice.org/uno:socket,host=localhost,port=2002;urp;StarOffice.ServiceManager/1
 - 例 : /Mozilla/@mozilla.org/file/directory_service;1/2

技術仕様書



- (7) 生成したアダプタのリファレンス情報を返却 (図 7-7)
- 各マネージャサービスで生成したアダプタのリファレンス情報をコンポーネントネーミングサービスに返す。
 - リファレンス情報は、D-BUS の METHOD_RETURN メッセージで返す。
- (8) アダプタのリファレンス情報を登録 (図 7-7)
- 取得したアダプタのリファレンス情報をコンポーネントネーミングサービスの所有するリファレンス情報に登録する。
 - 登録するリファレンス情報の項目は以下の通り。フォーマットは、Web サービスも含めて検索性やパフォーマンスの検討が必要となるため、プロトタイプにて使い勝手や検索性能を検証した上で決定する。
 - コンポーネント技術名 (例: UNO)
 - 生成したマネージャサービス名 (例: :uno_mgr)
 - オブジェクトパス (例: /Calc/com.sun.star.sheet.Spreadsheet/sheet1)
 - 抽象コンポーネント名 (例: spreadsheet)
 - 抽象コンポーネントの継承関係
 - メソッドのシグネチャ (同期・非同期の情報を含む)

技術仕様書



カスタムアプリ 図 7-8 抽象コンポーネントの生成 6

- (9) 生成したアダプタのリファレンス情報を受領 (図 7-8)
- コンポーネントネーミングサービスから登録したアダプタのリファレンス情報をカスタムアプリ作成ライブラリに渡す。
 - リファレンス情報の受け渡しは、D-BUS の METHOD_RETURN メッセージで行う。
 - 受け渡すリファレンス情報は以下の通り。
 - 生成したマネージャサービス名 (例: :uno_mgr)
 - オブジェクトパス (例: /Calc/com.sun.star.sheet.Spreadsheet/sheet1)
 - 抽象コンポーネント名 (例: spreadsheet)
 - 抽象コンポーネントの継承関係
 - メソッドのシグネチャ (同期・非同期の情報を含む)
- (10) 受け取ったリファレンス情報を元に proxy オブジェクトを生成 (図 7-8)
- カスタムアプリ作成ライブラリは受け取ったリファレンス情報を持つ proxy オブジェクトを生成する。

7.2.1 抽象コンポーネント生成のユースケース

【1】 メールボックス

カスタムアプリケーションから、メールクライアントの抽象コンポーネントを生成し、メールボックスをオープンさせるユースケースを以下に示す。メールクライアントの具象コンポーネントとして Thunderbird を例に説明する。

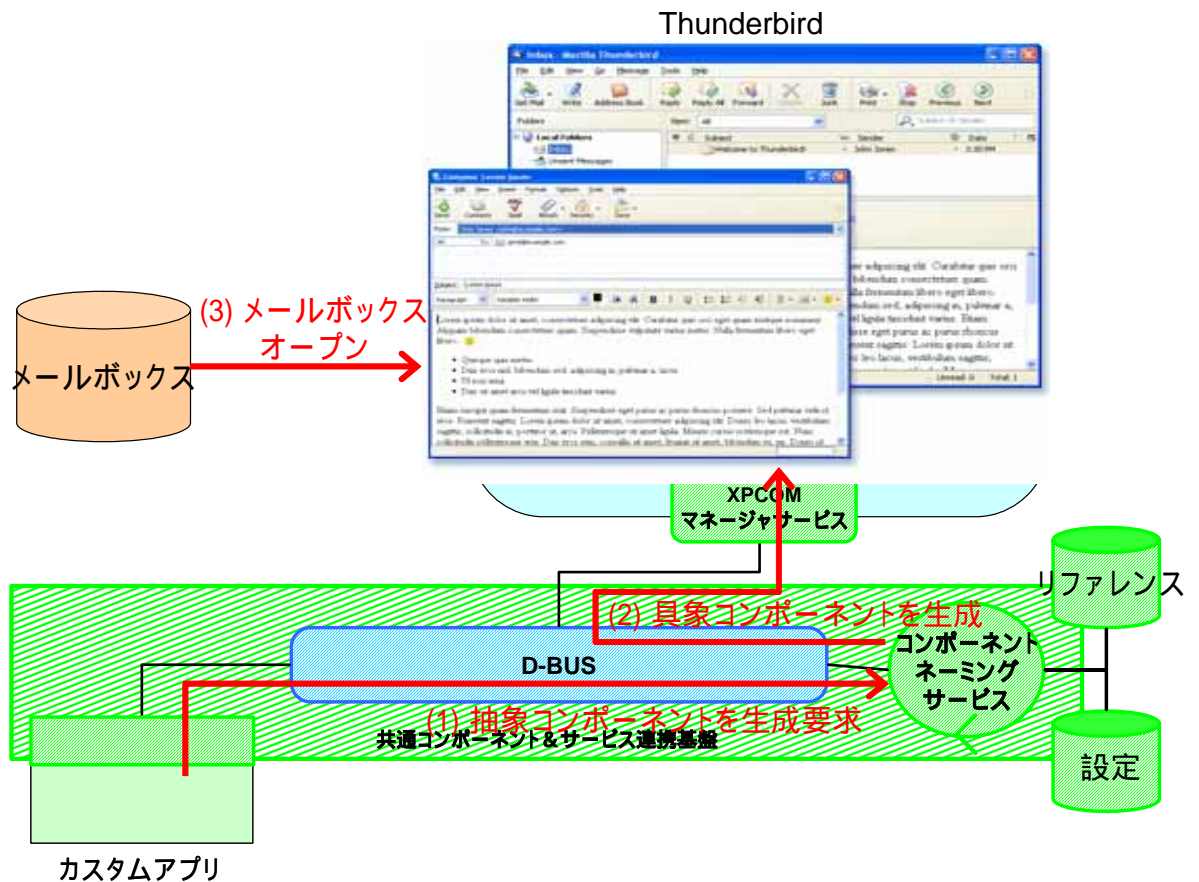


図 7-9 抽象コンポーネント生成 (メールボックス) のユースケース

- (1) カスタムアプリケーションはコンポーネントネーミングサービスに、メールクライアントの抽象コンポーネントの生成を要求する。
- (2) コンポーネントネーミングサービスは、XPCOM マネージャサービスを通じて具象コンポーネント Thunderbird を生成する。
- (3) Thunderbird は予め設定された内容に従い、ユーザのメールボックスをオープンする。

【2】 レンダリング (HTML/XML+CSS)

カスタムアプリケーションから HTML/XML+CSS ファイルのレンダリングを行う抽象コンポーネントを生成し、HTML/XML+CSS ファイルをレンダリングするユースケースを以下に示す。具象コンポーネントとして FireFox を例に説明する。

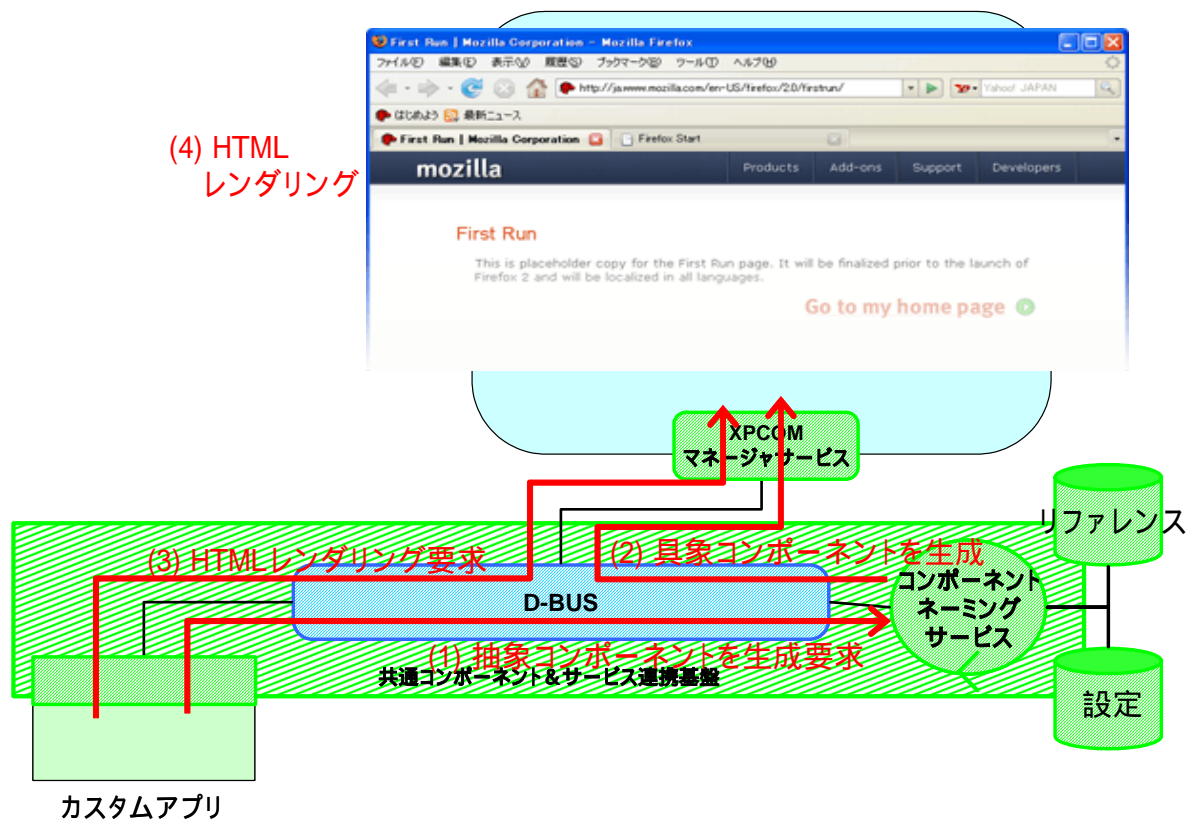


図 7-10 抽象コンポーネント生成 (レンダリング (HTML/XML+CSS)) のユースケース

- (1) カスタムアプリケーションはコンポーネントネーミングサービスに、HTML や XML+CSS ファイルのレンダリングを行う抽象コンポーネントの生成を要求する。
- (2) コンポーネントネーミングサービスは、XPCOM マネージャサービスを通じて具象コンポーネント FireFox を生成する。
- (3) カスタムアプリケーションは抽象コンポーネントに対して HTML データを渡し、HTML レンダリングを要求する。
- (4) FireFox は HTML データをレンダリングする。

【3】 レンダリング (SVG)

カスタムアプリケーションから SVG レンダラの抽象コンポーネントを生成し、SVG ファイルをレンダリングするユースケースを以下に示す。SVG レンダラの具象コンポーネントとして KDE の KSVG を例に説明する。

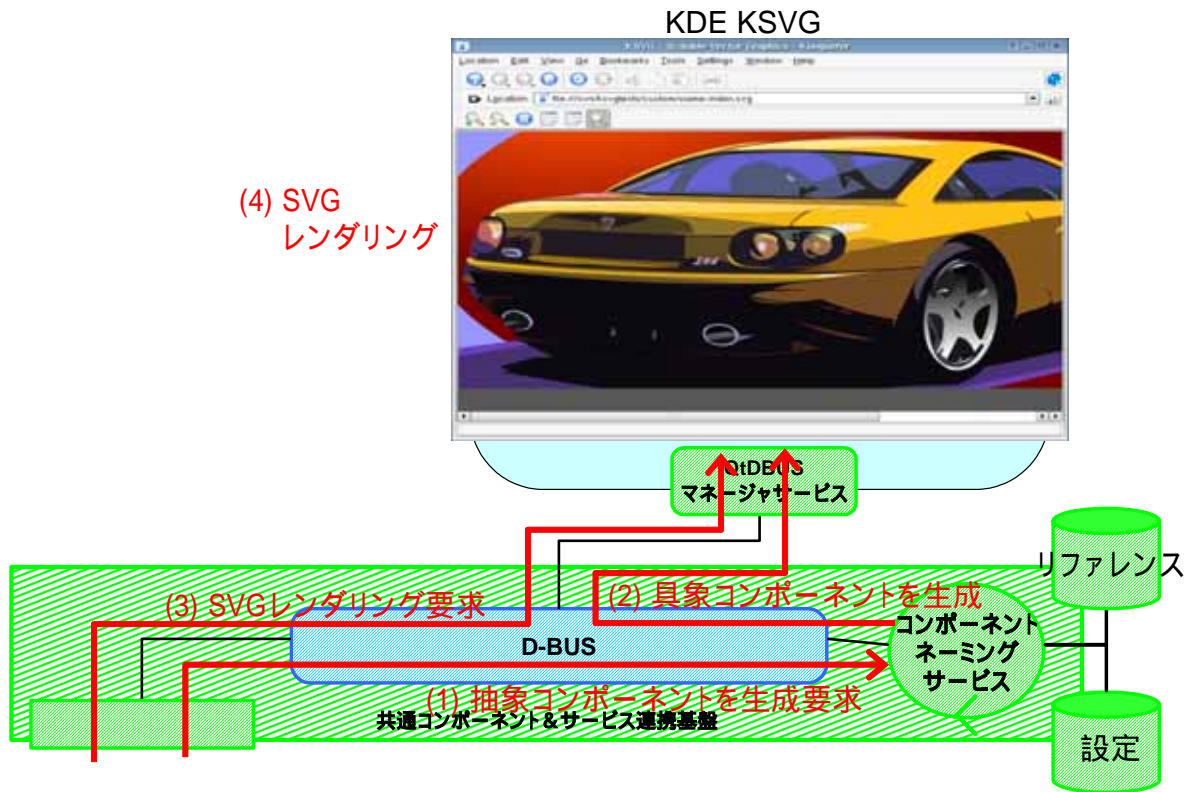


図 7-11 抽象コンポーネント生成 (レンダリング (SVG)) のユースケース

- (1) カスタムアプリケーションはコンポーネントネーミングサービスに、SVG ファイルのレンダリングを行う抽象コンポーネントの生成を要求する。
- (2) コンポーネントネーミングサービスは、QtDBus マネージャサービスを通じて具象コンポーネント KDE の KSVG を生成する。
- (3) カスタムアプリケーションは、抽象コンポーネントに対して SVG データを渡し、SVG レンダリングを要求する。
- (4) KSVG は SVG データをレンダリングする。

【4】 レンダリング (PDF)

カスタムアプリケーションから PDF レンダラの抽象コンポーネントを生成し、PDF ファイルをレンダリングするユースケースを以下に示す。PDF レンダラの具象コンポーネントとして KDE の KPDF を例に説明する。

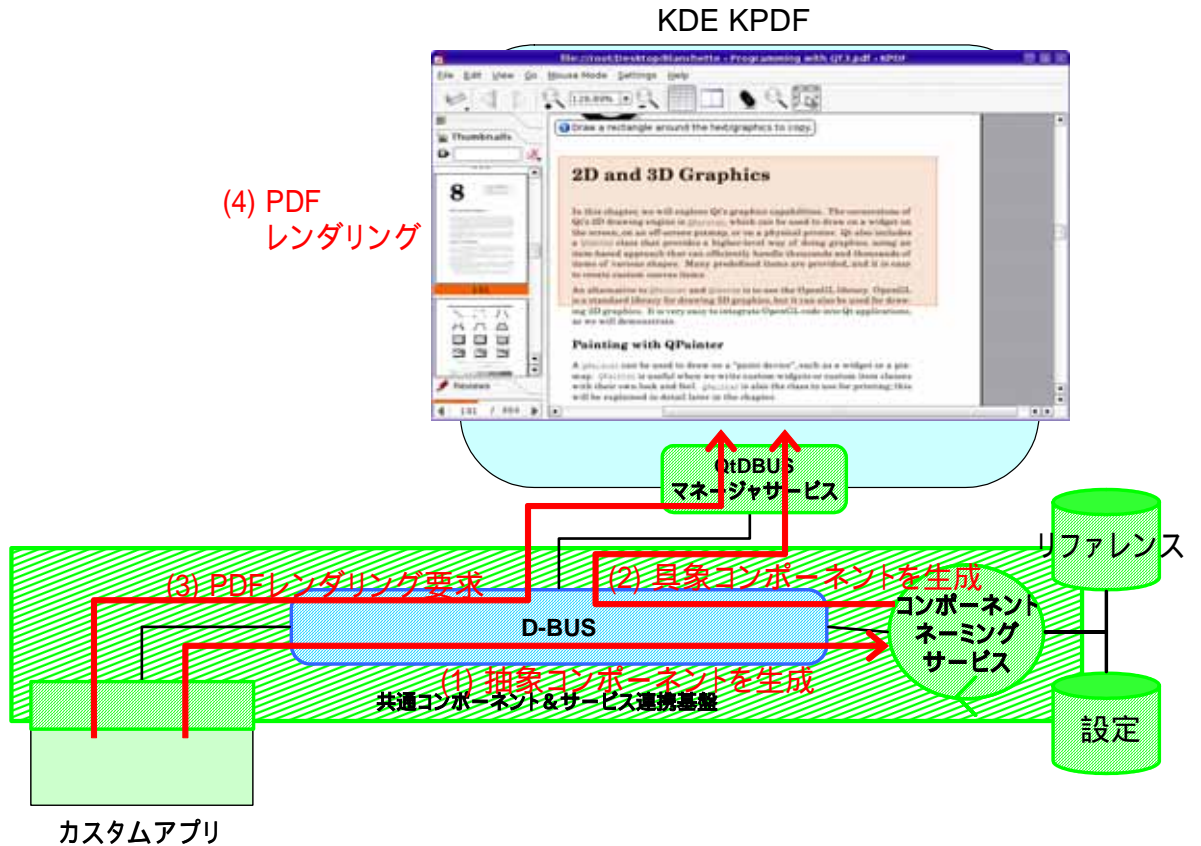


図 7-12 抽象コンポーネント生成 (レンダリング (PDF)) のユースケース

- (1) カスタムアプリケーションはコンポーネントネーミングサービスに、PDF ファイルのレンダリングを行う抽象コンポーネントの生成を要求する。
- (2) コンポーネントネーミングサービスは、QtDBus マネージャサービスを通じて具象コンポーネント KDE の KPDF を生成する。
- (3) カスタムアプリケーションは、抽象コンポーネントに対して PDF データを渡し、PDF レンダリングを要求する。
- (4) KPDF は PDF データをレンダリングする。

【5】 データベース

カスタムアプリケーションからデータベースの抽象コンポーネントを生成し、データベースをレンダリングするユースケースを以下に示す。データベースの具象コンポーネントとして OpenOffice.org の Base を例に説明する。

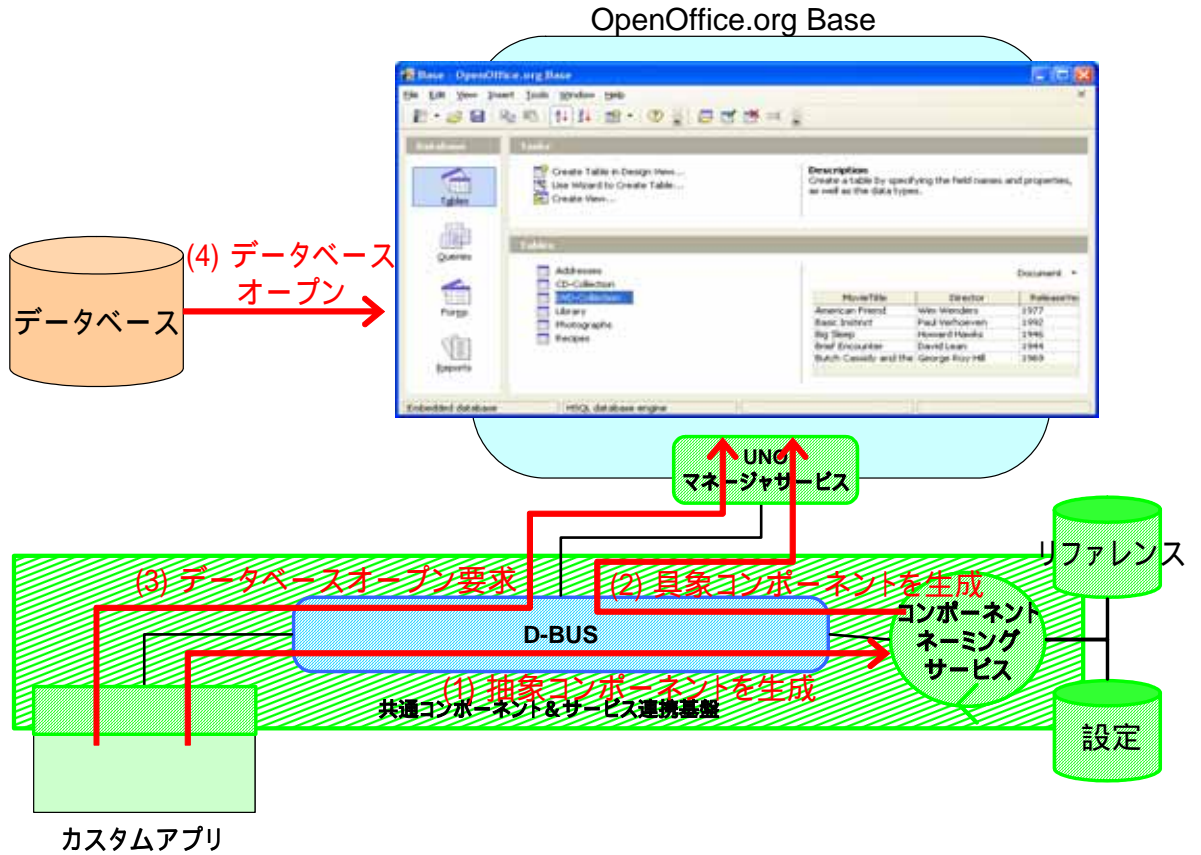


図 7-13 抽象コンポーネント生成 (データベース) のユースケース

- (1) カスタムアプリケーションはコンポーネントネーミングサービスに、データベースの抽象コンポーネントの生成を要求する。
- (2) コンポーネントネーミングサービスは、UNO マネージャサービスを通じて具象コンポーネント OpenOffice.org Base を生成する。
- (3) カスタムアプリケーションは、抽象コンポーネントに対して指定するデータベースのオープンを要求する。
- (4) OpenOffice.org Base は指定されたデータベースをオープンする。

7.2.2 QtDBus 版シーケンス

QtDBus 上の具象コンポーネントを生成する場合のシーケンスを以下に示す。

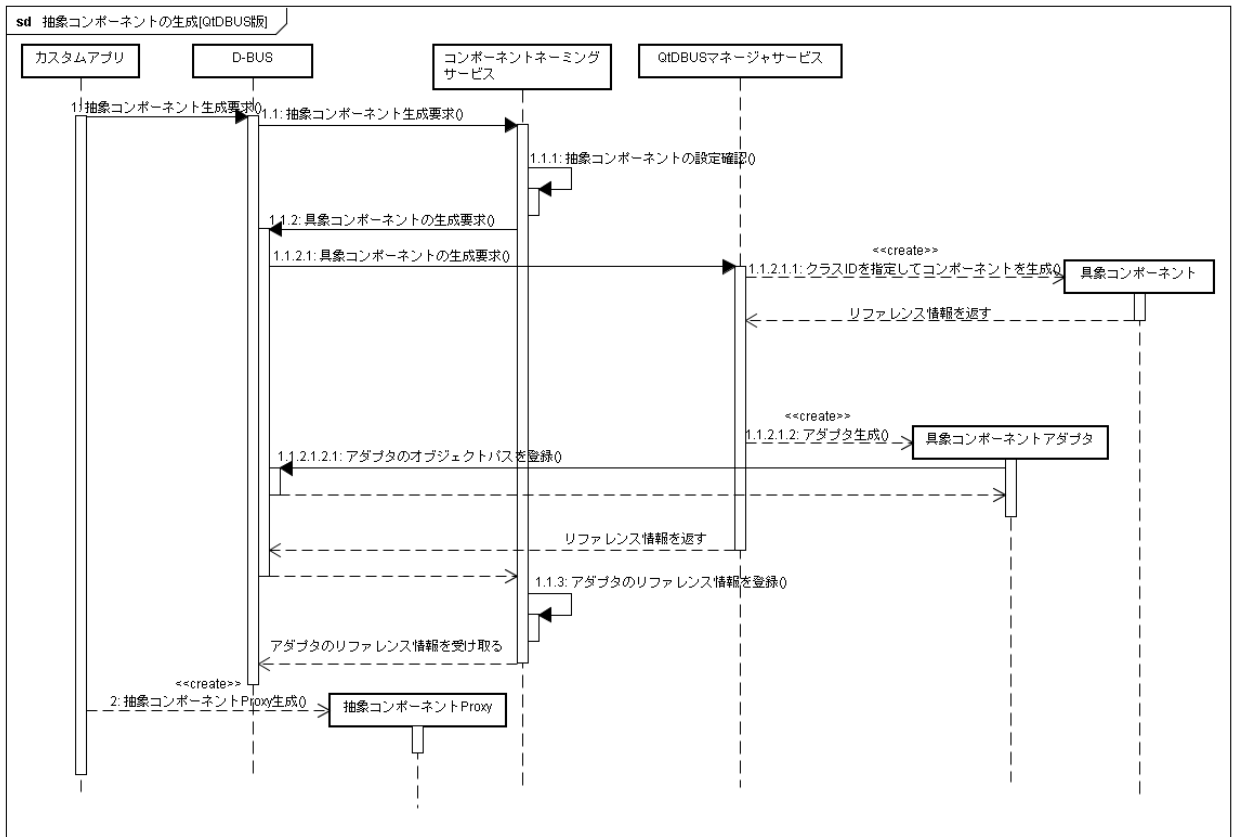


図 7-14 QtDBus における抽象コンポーネントの生成シーケンス

7.2.3 ORBit2 版シーケンス

ORBit2 上の具象コンポーネントを生成する場合のシーケンスを以下に示す。

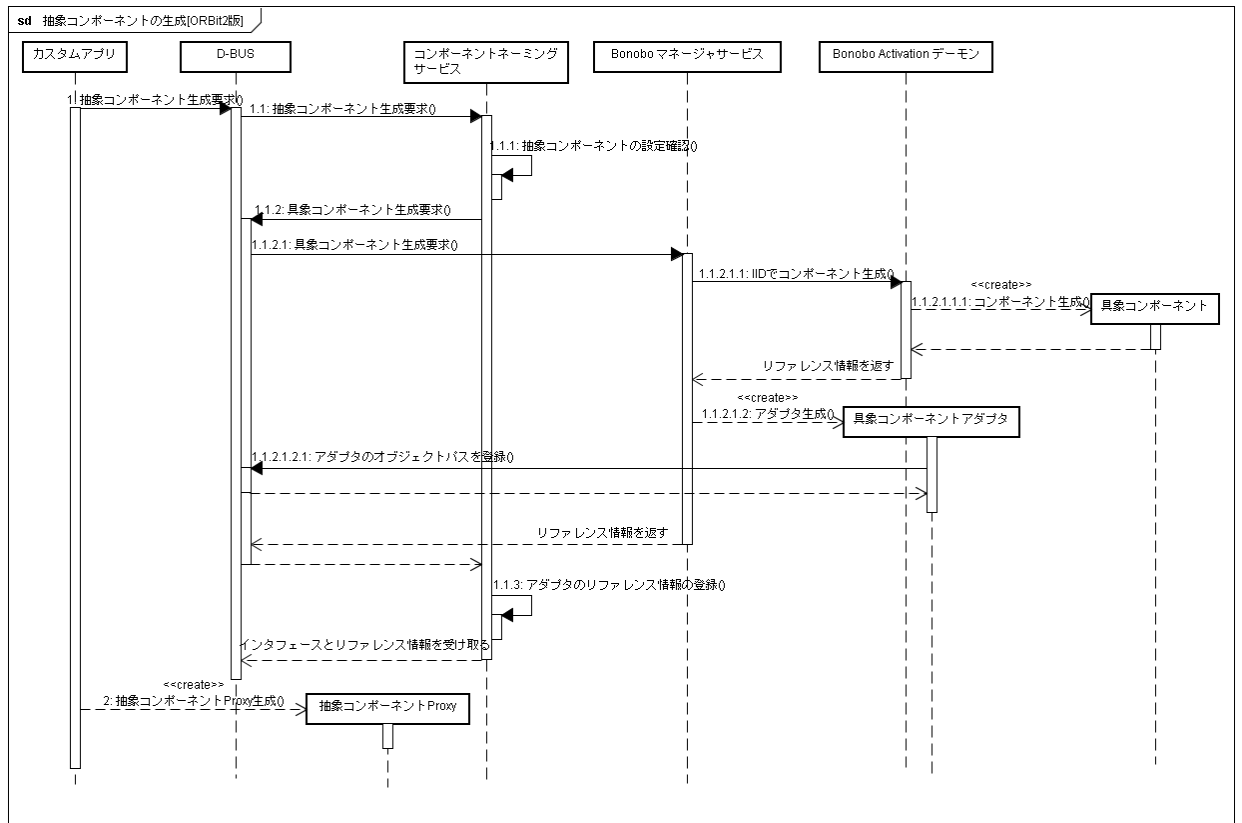


図 7-15 ORBit2 における抽象コンポーネントの生成シーケンス

7.2.4 XPCOM 版シーケンス

XPCOM 上の具象コンポーネントを生成する場合のシーケンスを以下に示す。

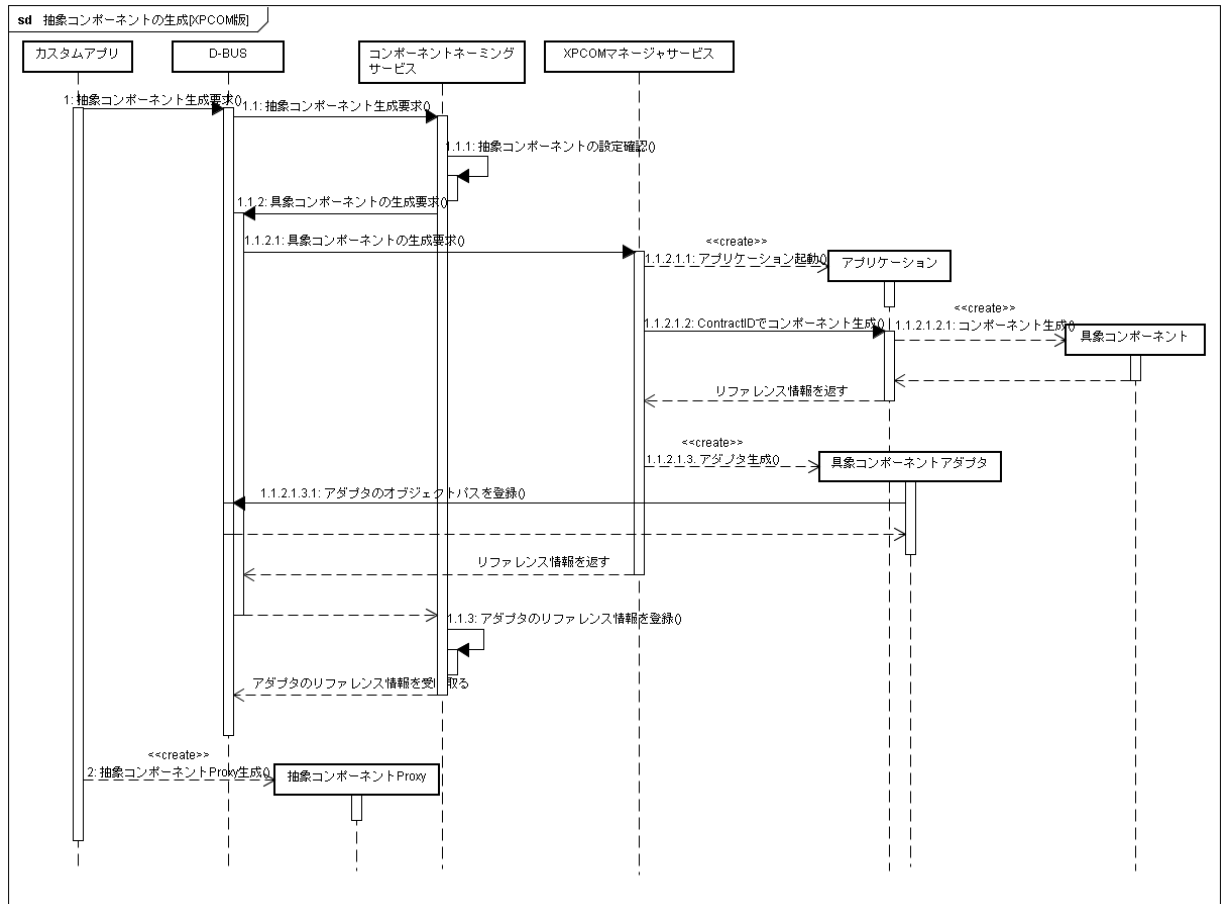


図 7-16 XPCOM における抽象コンポーネントの生成シーケンス

7.2.5 UNO 版シーケンス

UNO 上の具象コンポーネントを生成する場合のシーケンスを以下に示す。

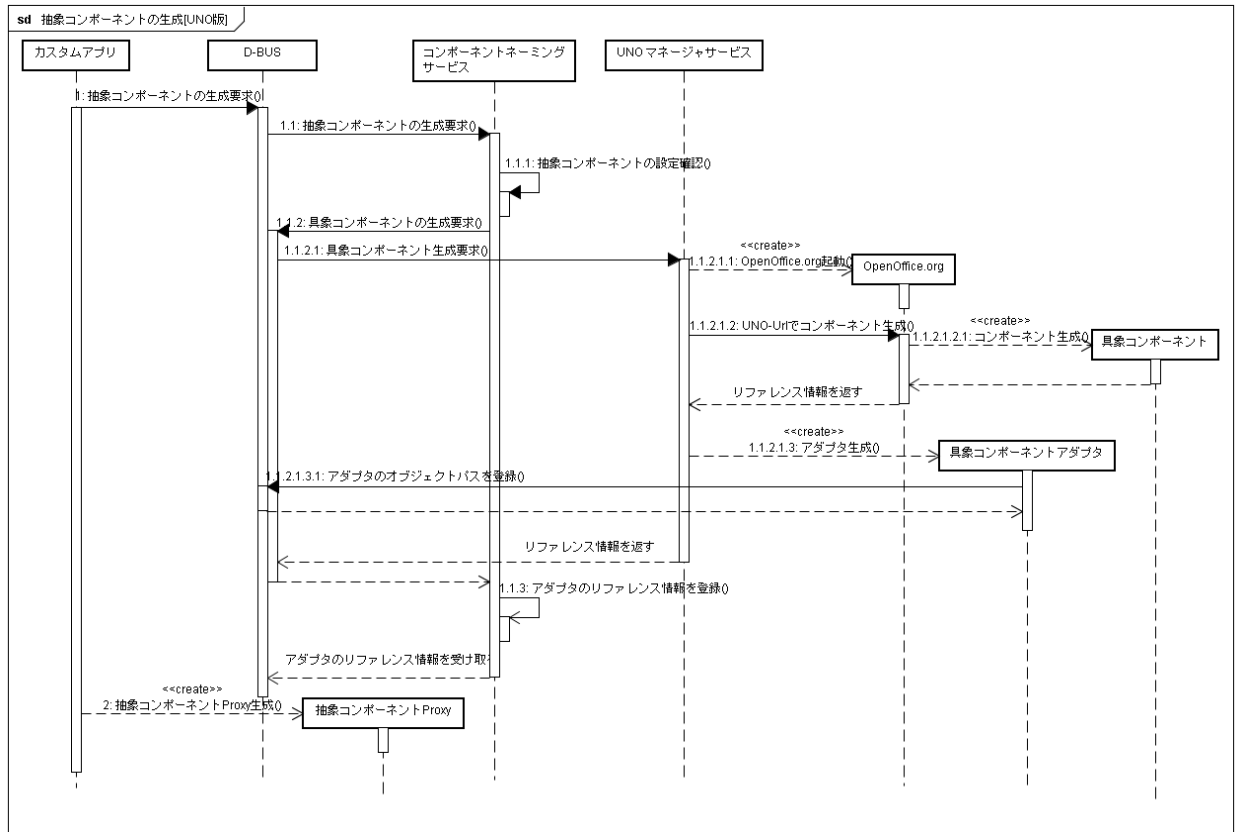


図 7-17 UNO における抽象コンポーネントの生成シーケンス

7.3 抽象コンポーネントの登録機能

抽象コンポーネントのリファレンス情報の登録の流れを図 7-18～図 7-19で示す。この処理は KDE や GNOME のようにシステムが起動する際に複数の具象コンポーネントが生成される場合に実行される。

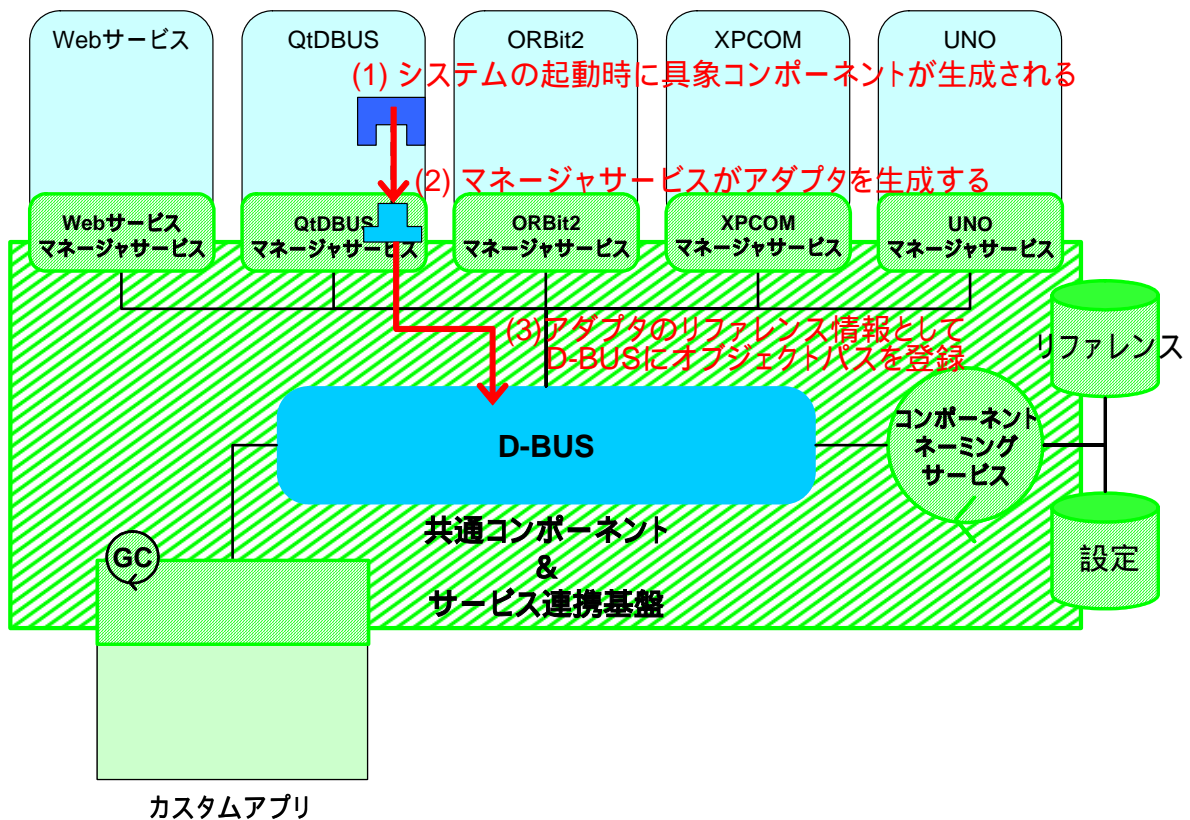


図 7-18 抽象コンポーネントの登録 1

- (1) システムの起動時に具象コンポーネントが生成される (図 7-1-18)
 - システム (KDE や GNOME) の起動時に具象コンポーネントが生成される。
- (2) マネージャサービスがアダプタを生成する (図 7-1-18)
 - 生成された具象コンポーネントが各マネージャサービスにアダプタの生成を要求する。
 - 具象コンポーネント生成時に、マネージャサービスに対してアダプタの生成を要求する機能を KParts、Bonobo に追加する必要がある。
- (3) アダプタのオブジェクトパスを D-BUS に登録 (図 7-1-18)
 - 各マネージャサービス上のオブジェクトとしてアダプタのオブジェクトパスを登録する。



図 7-19 抽象コンポーネントの登録 2

- (4) 生成したアダプタのリファレンス情報を渡す (図 7-1-19)
- 各マネージャサービスで生成したアダプタのリファレンス情報をコンポーネントネーミングサービスに渡す。
 - リファレンス情報は、D-BUS の METHOD_CALL メッセージで返す。
- (5) アダプタのリファレンス情報を登録 (図 7-1-19)
- 取得したアダプタのリファレンス情報をコンポーネントネーミングサービスの所有するリファレンス情報に登録する。

7.3.1 シーケンス

抽象コンポーネントのリファレンス情報登録処理のシーケンスを以下に示す。

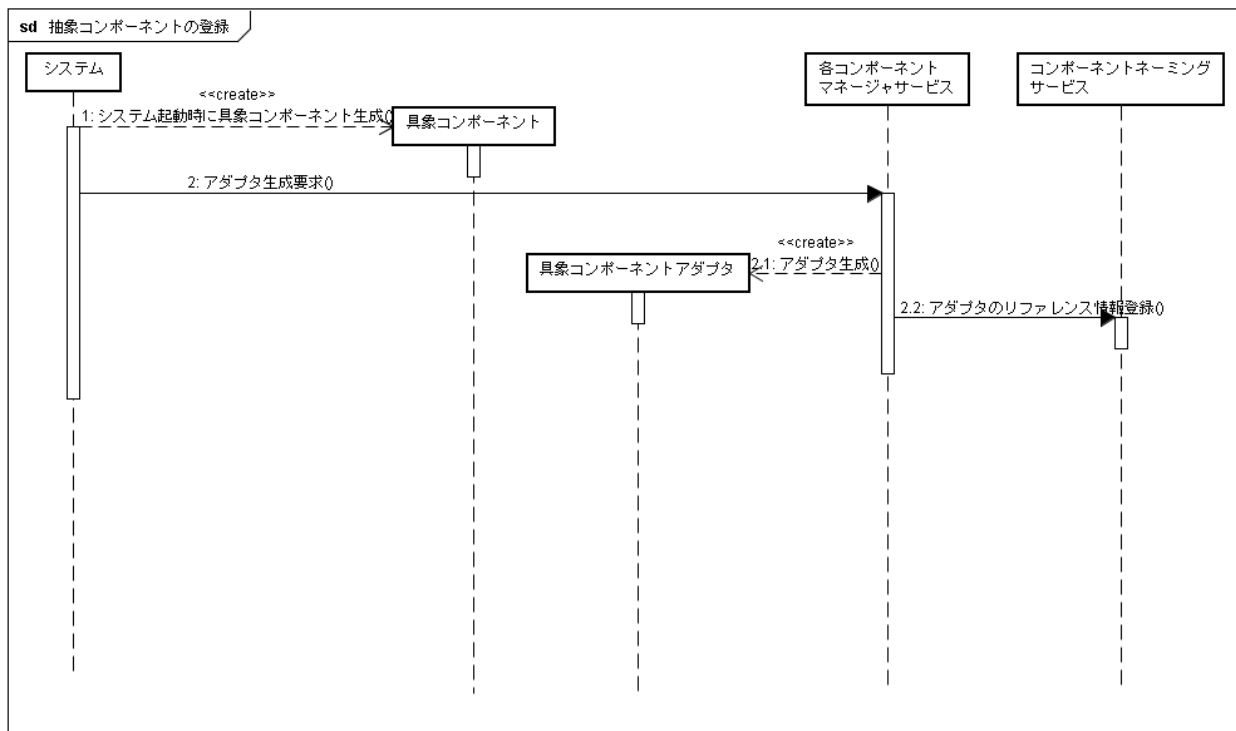


図 7-20 抽象コンポーネントのリファレンス情報登録シーケンス

7.4 Web サービスの登録機能

Web サービスのインタフェース情報の登録の流れを図 7-22～図 7-23で示す。図 7-21の WSDL 変換の処理は事前に行う必要がある。また、Web サービスのインタフェースのコンポーネントネーミングサービスへの登録は、Web サービスマネージャサービスの起動時に実行される。

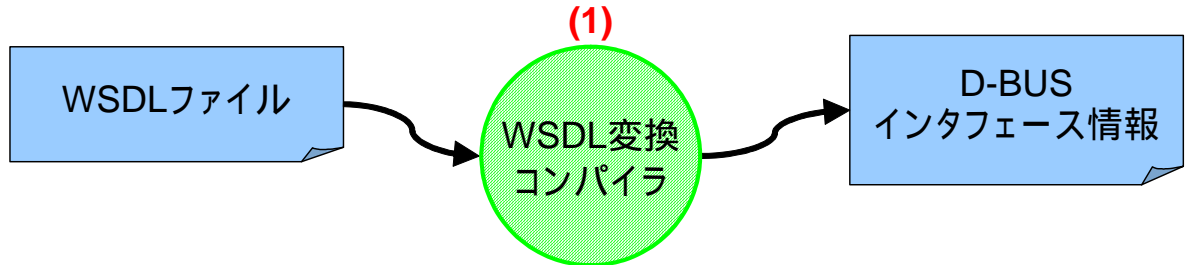


図 7-21 WSDL から D-BUS インタフェースの生成

(1) WSDL の変換 (図 7-1-21)

- 利用したい Web サービスの WSDL ファイルを本システム (フレームワーク) の WSDL 変換コンパイラにより、Web サービスに対応する D-BUS のインタフェース情報 (イントロスペクション) に変換する。
- **4章に記述したように、WSDL ファイルは D-BUS のインタフェース情報であるイントロスペクションに対応付ける。**

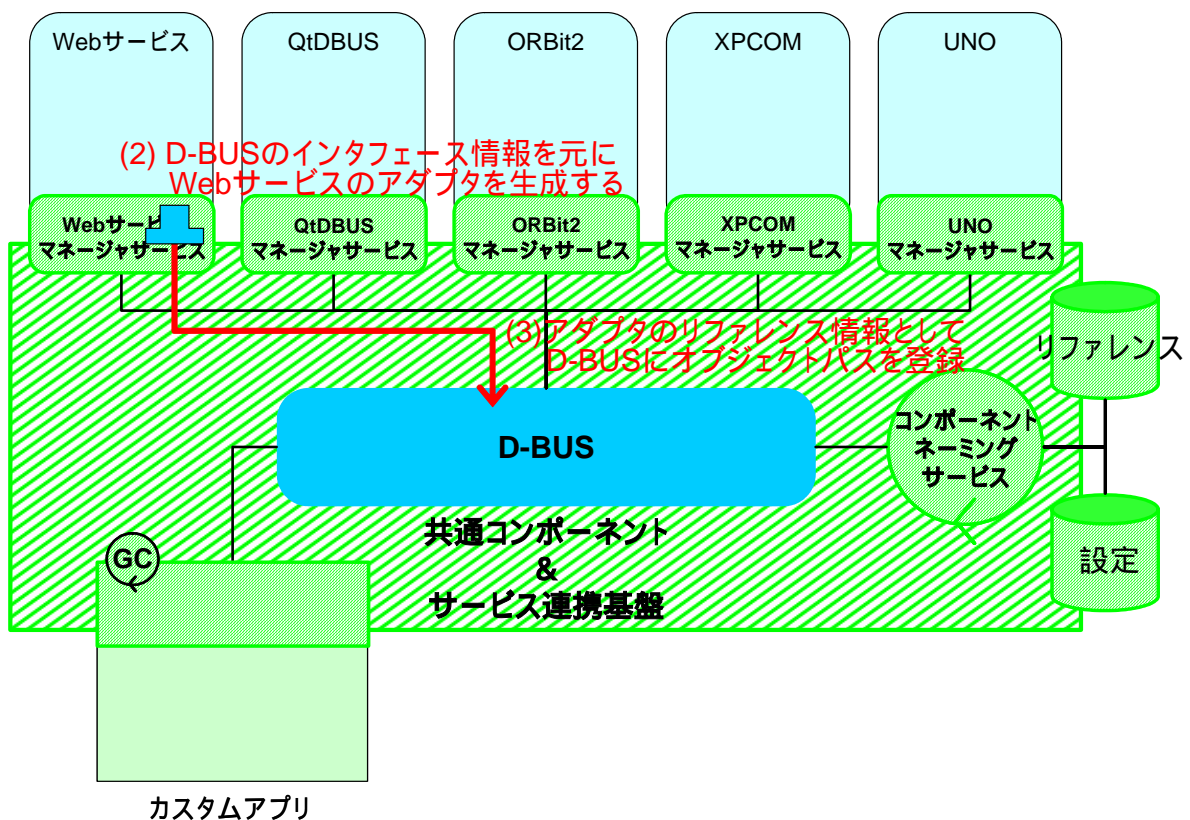


図 7-22 Web サービスインタフェース情報の登録 1

(2) Web サービスのアダプタを生成 (図 7-1-22)

- (1)で生成した D-BUS のインタフェース情報を元に Web サービスのアダプタを生成する。

技術仕様書

(3) アダプタのオブジェクトパスを D-BUS に登録 (図 7-1-22)

- Web サービスマネージャサービス上のオブジェクトとしてアダプタのオブジェクトパスを D-BUS に登録する。
 - オブジェクトパスは、Web サービスの URL とする。

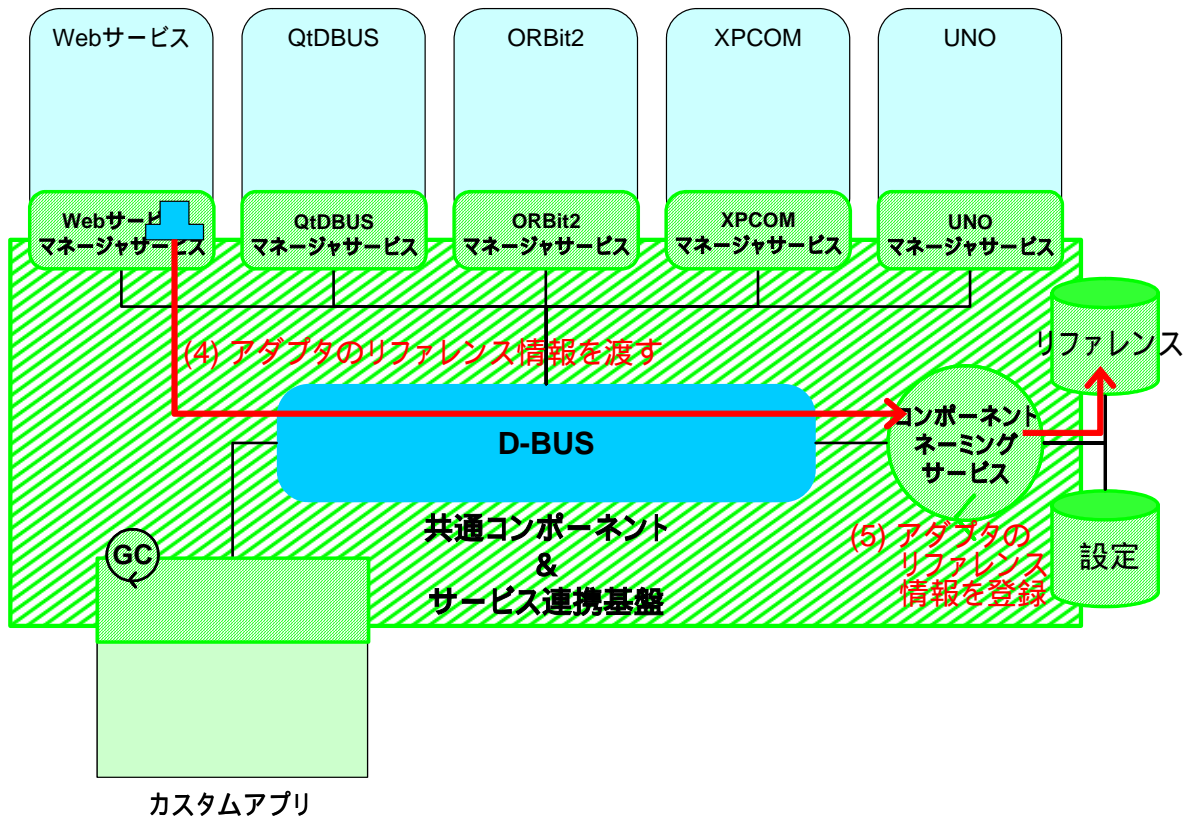


図 7-23 Web サービスインタフェース情報の登録 2

(4) アダプタのリファレンス情報を渡す (図 7-1-23)

- 各アダプタのリファレンス情報をコンポーネントネーミングサービスに登録する。

(5) アダプタのリファレンス情報を登録 (図 7-1-23)

- 各アダプタのリファレンス情報をコンポーネントネーミングサービスが所有しているリファレンス情報に登録する。

7.5 抽象コンポーネントの検索機能

抽象コンポーネントの検索処理の流れを、指定する抽象コンポーネントに対応する具象コンポーネントが UNO のコンポーネントである場合を例に図 7-24 ~ 図 7-26 で示す。

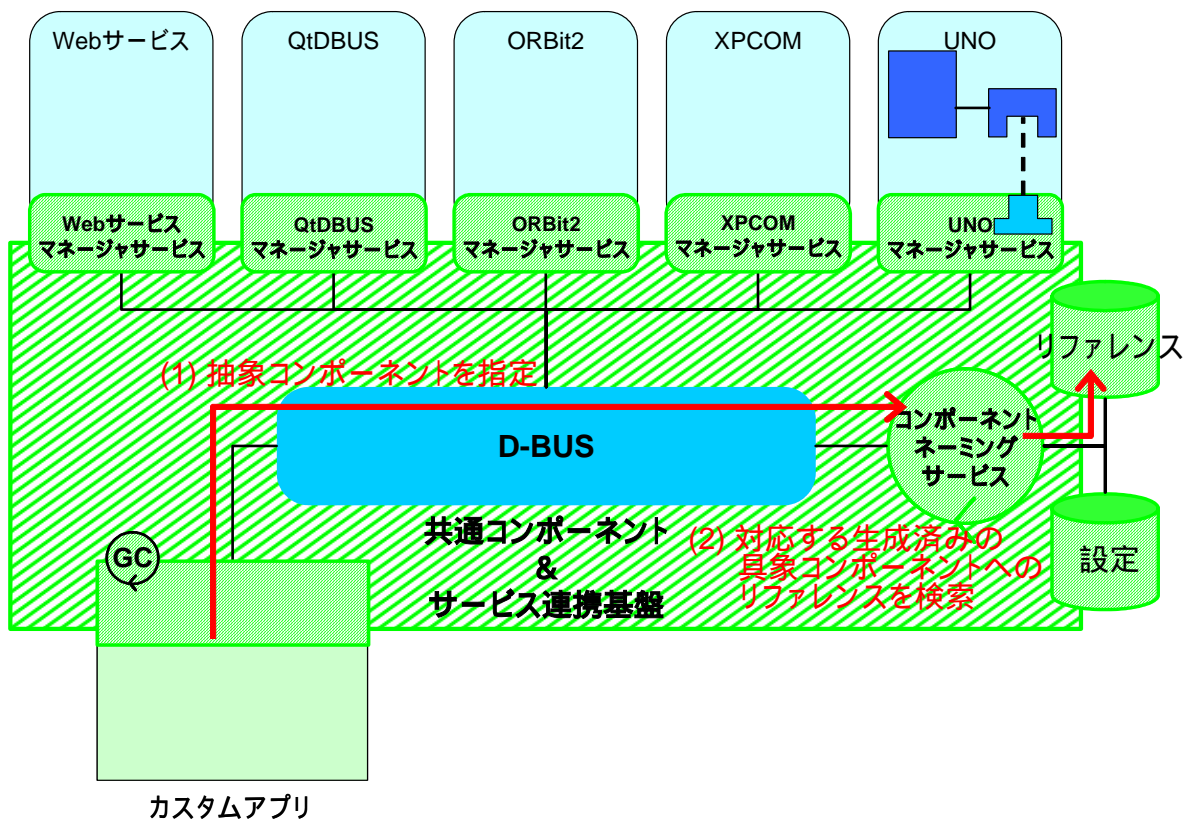


図 7-24 抽象コンポーネントの検索 1

- (1) 抽象コンポーネントの指定 (図 7-1-24)
 - カスタムアプリケーションは、カスタムアプリ作成ライブラリを利用して、コンポーネントネーミングサービス (:component_mgr) に対して取得したい抽象コンポーネント (例 : Spreadsheet) の検索を要求する。
- (2) 対応する生成済みの具象コンポーネントへのリファレンスを検索 (図 7-1-24)
 - コンポーネントネーミングサービス (:component_mgr) は、所有しているリファレンス情報から、取得したい抽象コンポーネント (例 : spreadsheet) に対応する具象コンポーネントのリファレンスを検索する。

技術仕様書

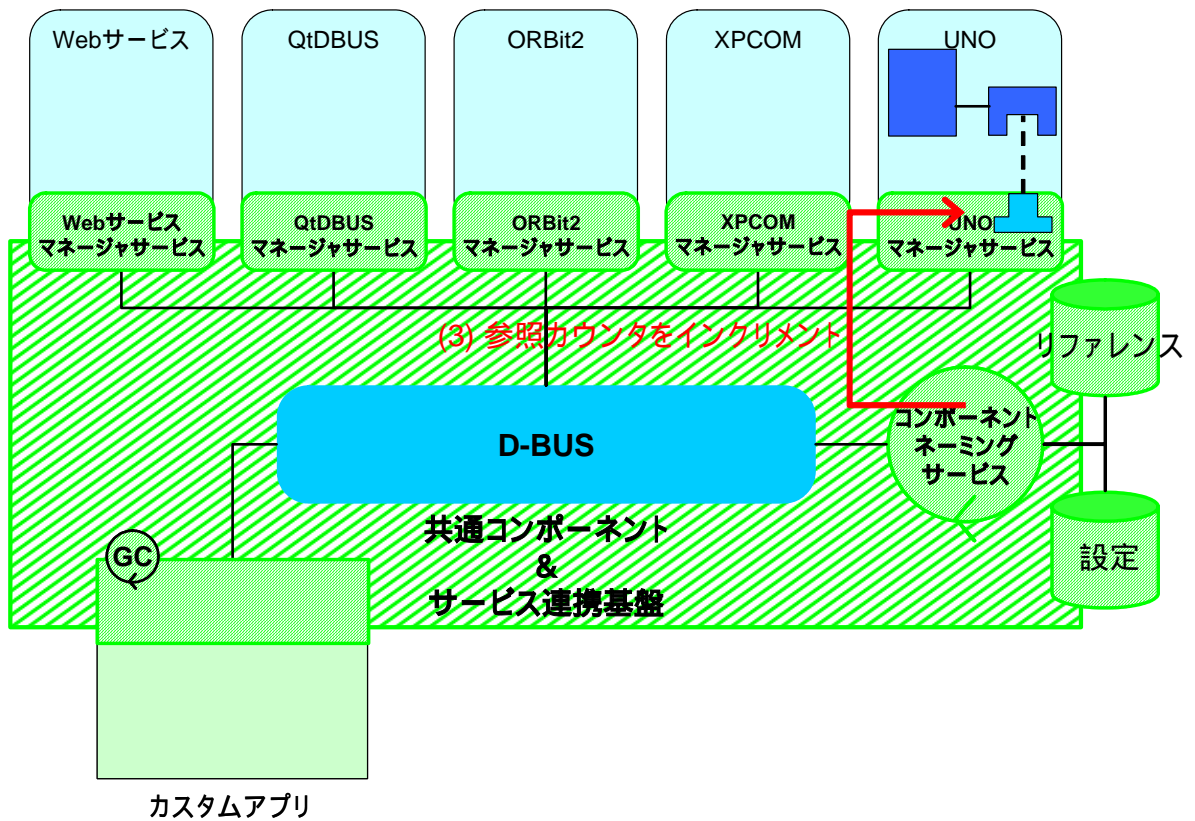


図 7-25 抽象コンポーネントの検索 2

(3) 参照カウンタをインクリメント (図 7-1-25)

- コンポーネントネーミングサービスから各マネージャサービスに対して、参照する具象コンポーネントの参照カウンタのインクリメント、もしくは参照ポインタの生成を要求する。
- **各コンポーネント技術が GC を用いている場合には、GC 実行の基準になる参照カウンタのインクリメント、もしくは参照ポインタの生成が必要になる。**
 - **共通コンポーネント基盤における GC の利用 (および GC の同期) は望ましいが実現が難しいため、プロトタイプ等を作成し検証した上で決定する課題とする。**

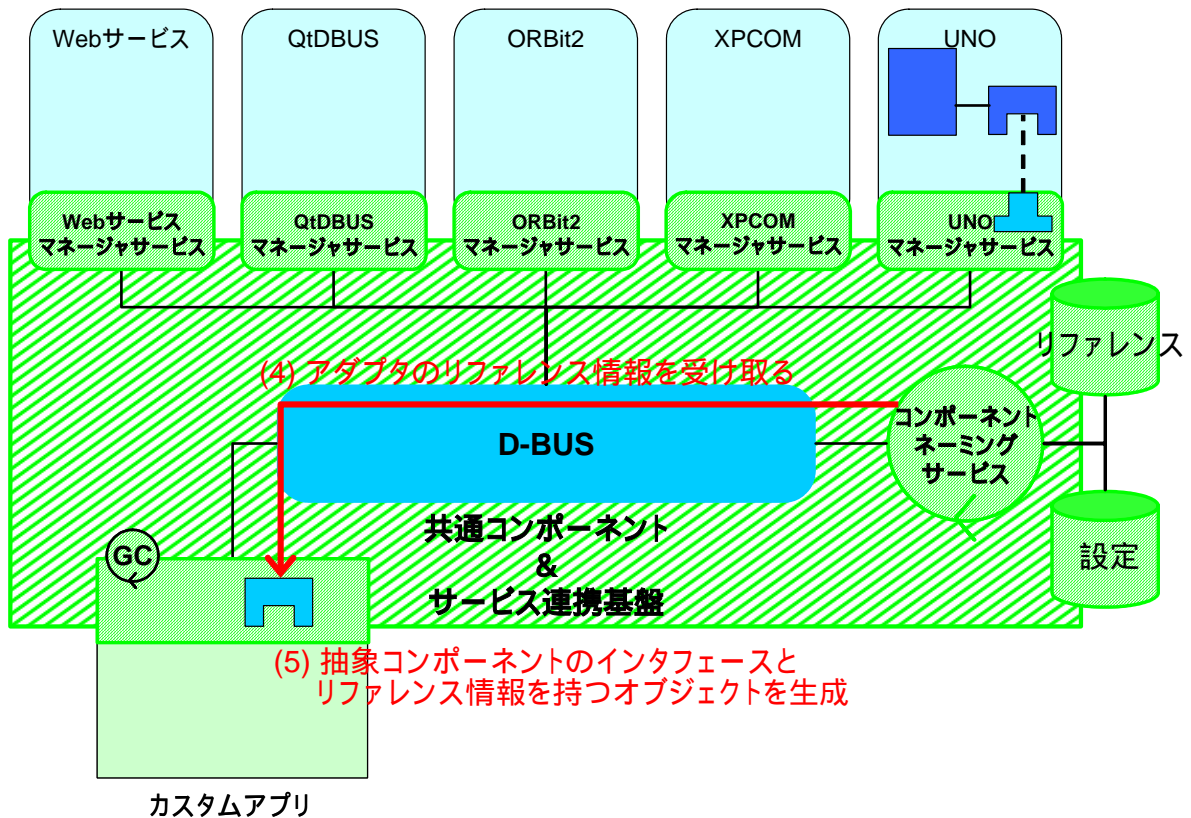


図 7-26 抽象コンポーネントの検索 3

- (4) アダプタのリファレンス情報を受け取る (図 7-1-26)
- 検索した結果見つかったアダプタのリファレンス情報をカスタムアプリ作成ライブラリに渡す。
 - リファレンス情報の受け渡しは、D-BUS の METHOD_RETURN メッセージで行う。
 - 受け渡すリファレンス情報は生成時と同じとする。
- (5) 受け取ったリファレンス情報を元に proxy オブジェクトを生成 (図 7-1-26)
- カスタムアプリ作成ライブラリは受け取ったリファレンス情報を持つ proxy オブジェクトを生成する。

7.5.1 シーケンス

抽象コンポーネントの検索処理のシーケンスを以下に示す。

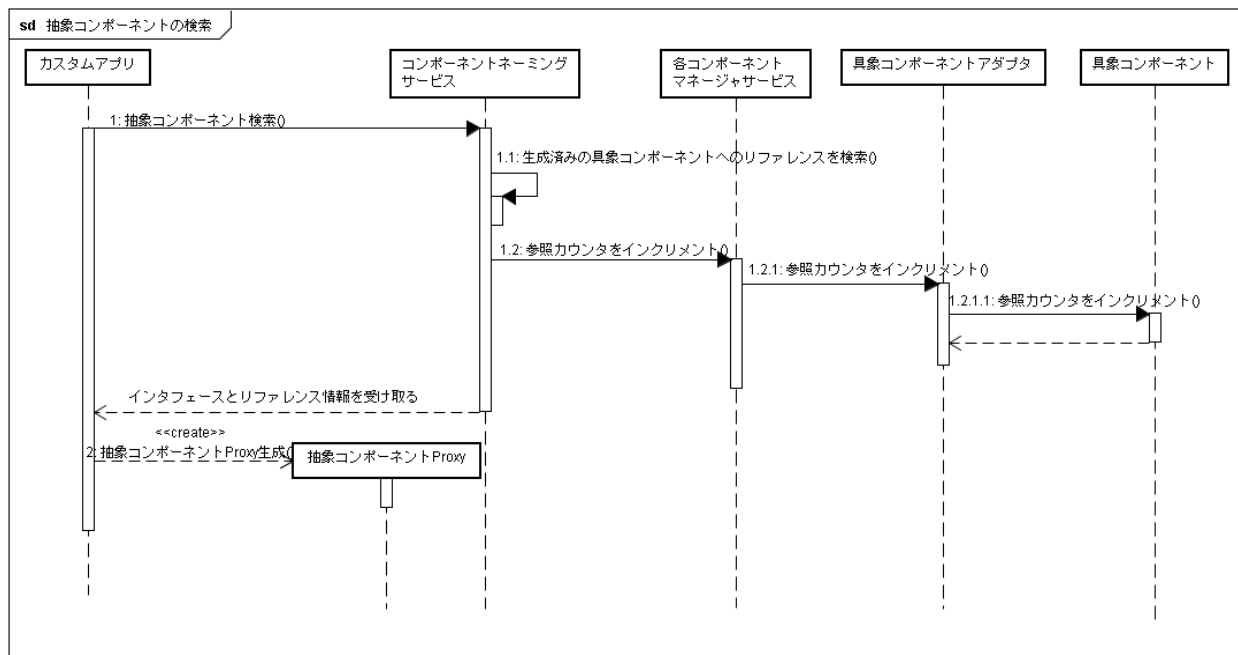


図 7-27 抽象コンポーネントの検索シーケンス

7.6 抽象コンポーネントの参照機能

抽象コンポーネントの参照処理（メソッド呼び出し）の流れを、指定する抽象コンポーネントに対応する具象コンポーネントが UNO のコンポーネントである場合を例に図 7-28～図 7-33 で示す。それに合わせて、通信するデータ形式、設定を記述する。

7.6.1 同期通信処理

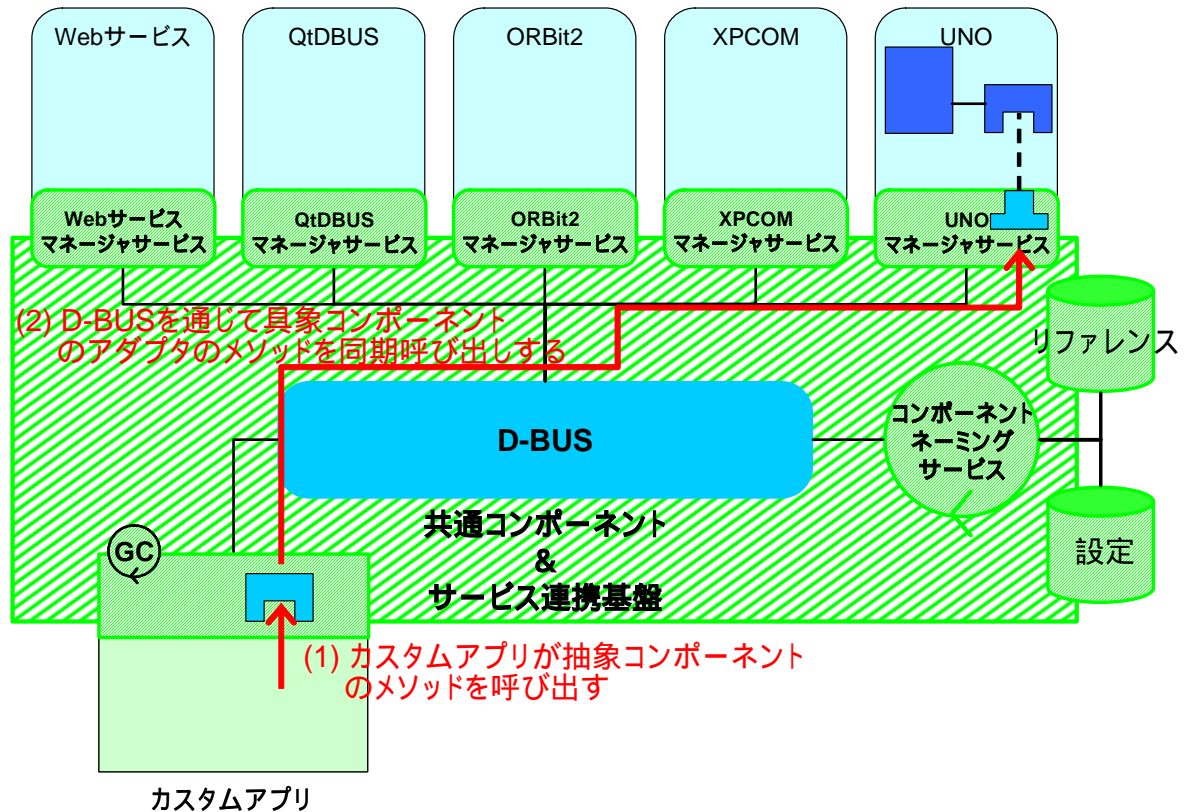


図 7-28 同期通信処理 1

- (1) カスタムアプリが抽象コンポーネントのメソッドを呼び出す（図 7-1-28）
 - カスタムアプリが抽象コンポーネントの proxy に対してメソッドの同期呼び出しを実行する。
- (2) D-BUS を通じて具象コンポーネントのアダプタに通信する（図 7-1-28）
 - 抽象コンポーネントの proxy は、カスタムアプリ作成ライブラリを利用して、D-BUS を経由して具象コンポーネントのアダプタのメソッドを呼び出す。

技術仕様書

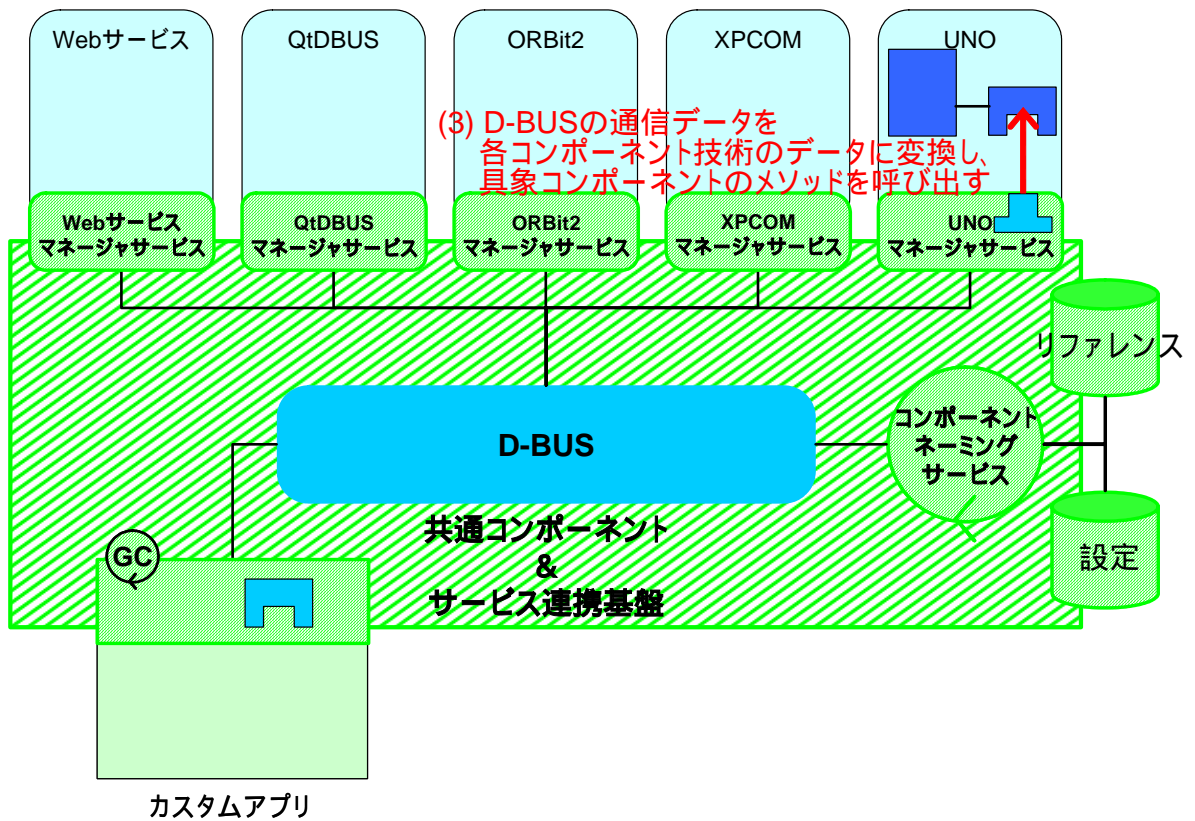


図 7-29 同期通信処理 2

(3) 具象コンポーネントのメソッド呼び出し (図 7-1-29)

- 各コンポーネント技術のマネージャサービスにて、D-BUS の通信データを各コンポーネント技術のデータに変換し、具象コンポーネントのメソッドを呼び出す。
- 4章に記述したように、D-BUS の通信データ型から各コンポーネント技術のデータ型への対応に不足が無いため、変換可能である。

技術仕様書

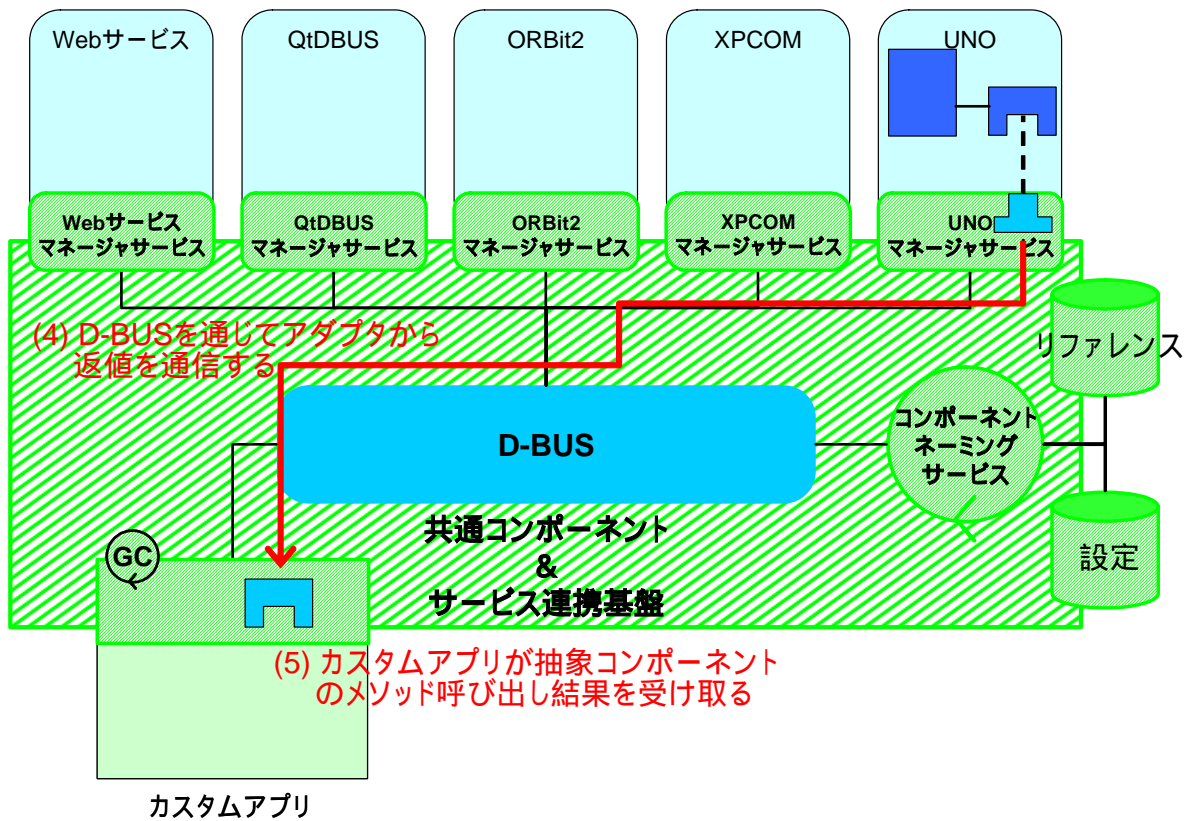


図 7-30 同期通信処理 3

- (4) D-BUS を通じてアダプタから返値を通信する (図 7-1-30)
- 各コンポーネント技術のマネージャサービスにて、各コンポーネント技術のデータを D-BUS の通信データに変換し、D-BUS の METHOD_RETURN メッセージで結果を通信する。
- (5) カスタムアプリが結果を受け取る (図 7-1-30)
- D-BUS を経由してカスタムアプリの抽象コンポーネント proxy が結果を受け取る。

7.6.2 非同期通信処理

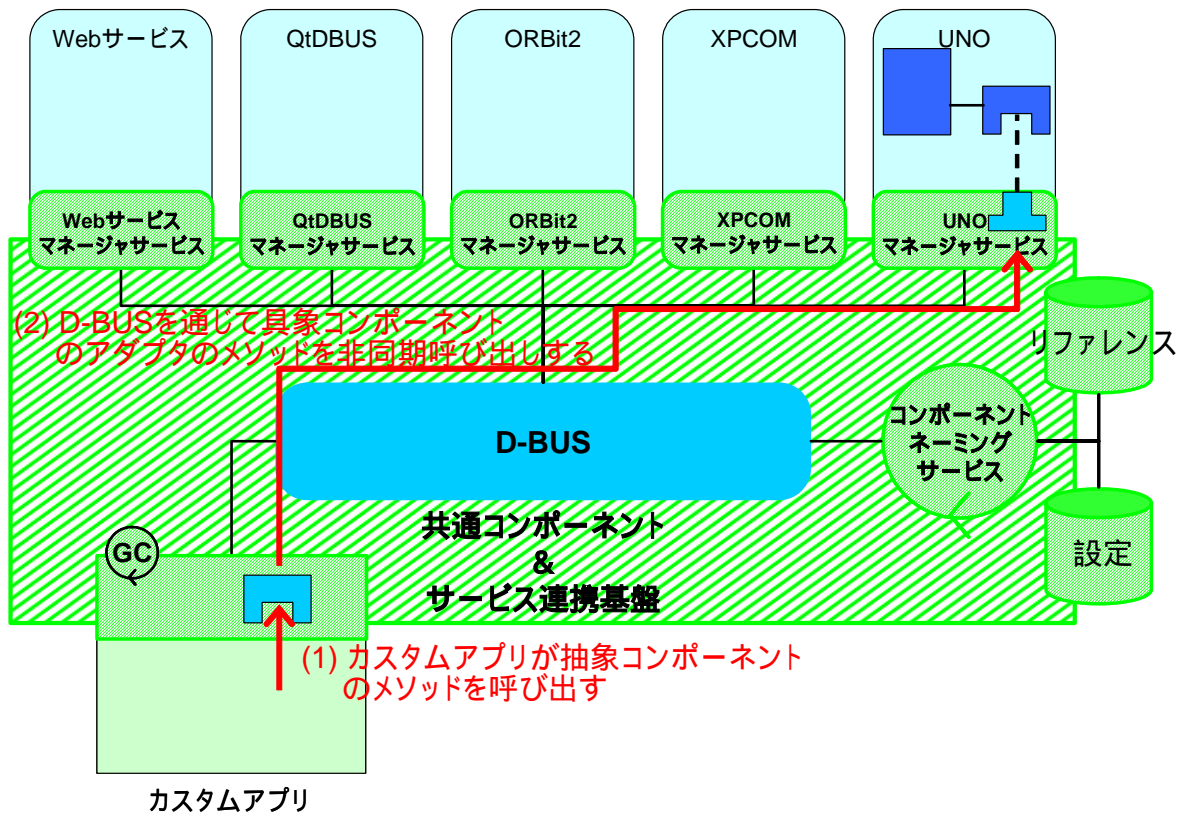


図 7-31 非同期通信処理 1

- (1) カスタムアプリが抽象コンポーネントのメソッドを呼び出す (図 7-1-31)
 - カスタムアプリが抽象コンポーネントの proxy に対してメソッドの非同期呼び出しを実行する。
- (2) D-BUS を通じて具象コンポーネントのアダプタに通信する (図 7-1-31)
 - 抽象コンポーネントの proxy は、カスタムアプリ作成ライブラリを利用して、D-BUS を経由して具象コンポーネントのアダプタのメソッドを呼び出す。

技術仕様書

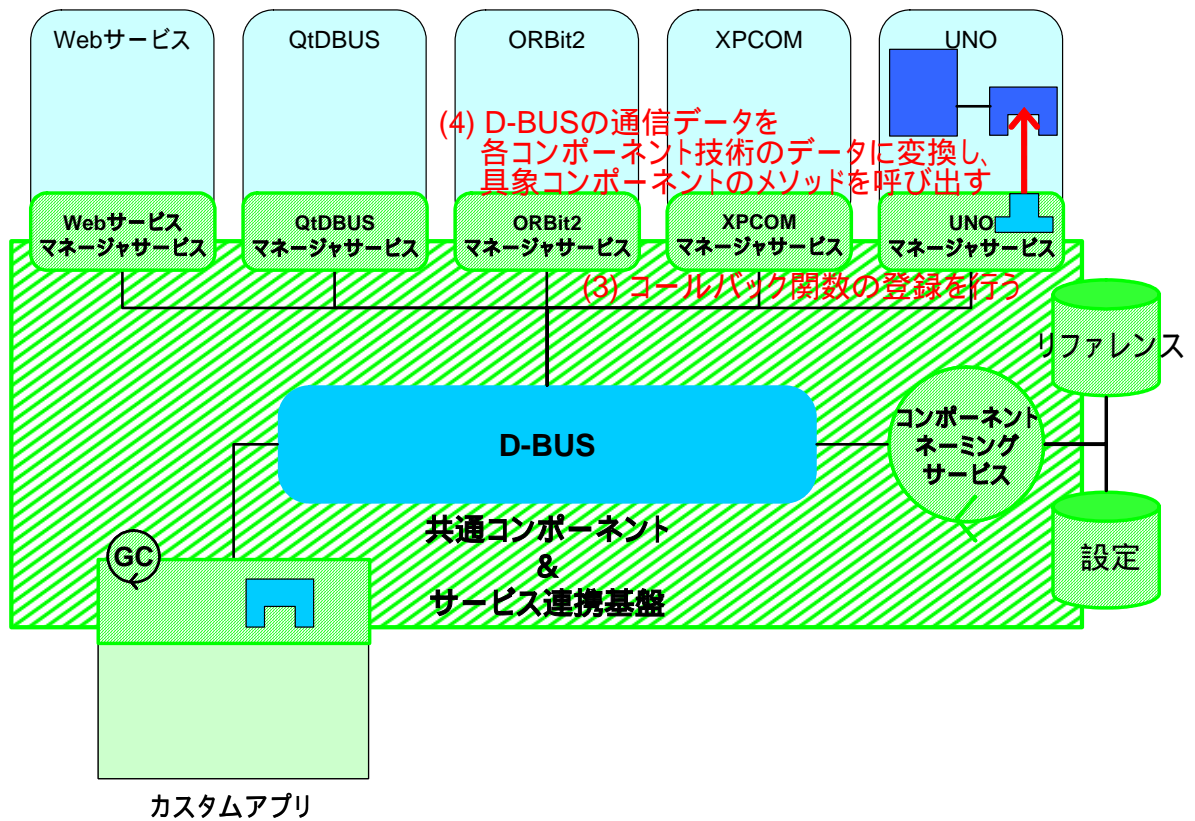


図 7-32 非同期通信処理 2

- (3) コールバック関数の登録を行う (図 7-1-32)
 - D-BUS の非同期呼び出しのコールバック関数を登録する。
- (4) 具象コンポーネントのメソッド呼び出し (図 7-1-32)
 - 各コンポーネント技術のマネージャサービスにて、D-BUS の通信データを各コンポーネント技術のデータに変換し、具象コンポーネントのメソッドを呼び出す。
 - 4章に記述したように、D-BUS の通信データ型から各コンポーネント技術のデータ型への対応に不足が無いため、変換可能である。

技術仕様書

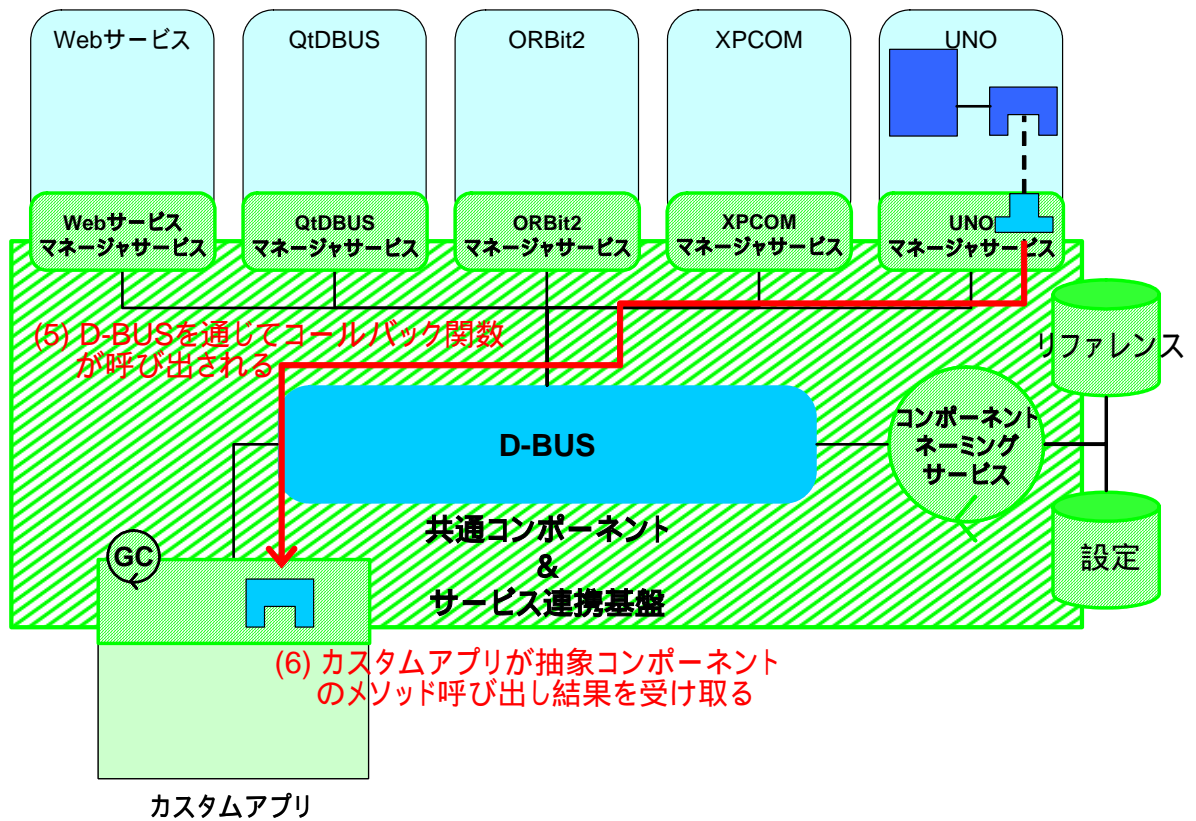


図 7-33 非同期通信処理 3

- (5) D-BUS を通じてコールバック関数が呼び出される (図 7-1-33)
- D-BUS を経由してコールバック関数が呼び出される。
- (6) カスタムアプリが結果を受け取る (図 7-1-33)
- D-BUS を経由してカスタムアプリの抽象コンポーネント proxy が結果を受け取る。

7.6.3 抽象コンポーネント参照のユースケース

【1】 Office アプリケーション

カスタムアプリケーションからスプレッドシートおよびワードプロセッサの抽象コンポーネントを生成し、スプレッドシートからデータを取得（カスタムアプリケーションから参照）し、カスタムアプリケーションにて加工後、ワードプロセッサに挿入（カスタムアプリケーションから参照）するユースケースを以下に示す（例えば、スプレッドシート上の数値データをカスタムアプリで集計・統計処理し、ワードプロセッサで編集集中の報告書に挿入する、といったシチュエーションを想定）。Office アプリケーションの具象コンポーネントとして OpenOffice Calc および KOffice KWord を例に説明する。

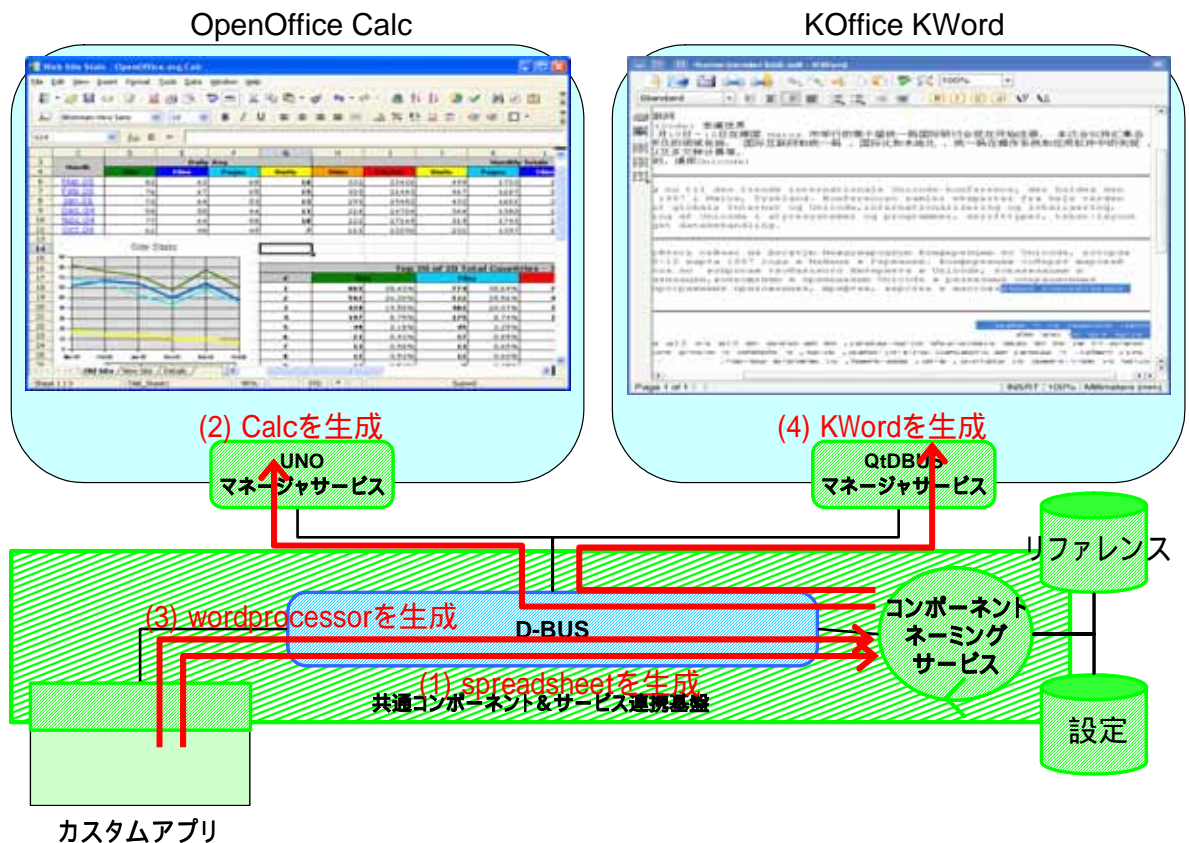


図 7-34 抽象コンポーネント参照のユースケース 1

- (1) カスタムアプリケーションはコンポーネントネーミングサービスに、Office アプリケーションの抽象コンポーネント（spreadsheet）の生成を要求する。
- (2) コンポーネントネーミングサービスは、UNO マネージャサービスを通じて具象コンポーネント OpenOffice Calc を生成する。
- (3) カスタムアプリケーションはコンポーネントネーミングサービスに、Office アプリケーションの抽象コンポーネント（wordprocessor）の生成を要求する。
- (4) コンポーネントネーミングサービスは、QtDBus マネージャサービスを通じて具象コンポーネント KDE KWord を生成する。

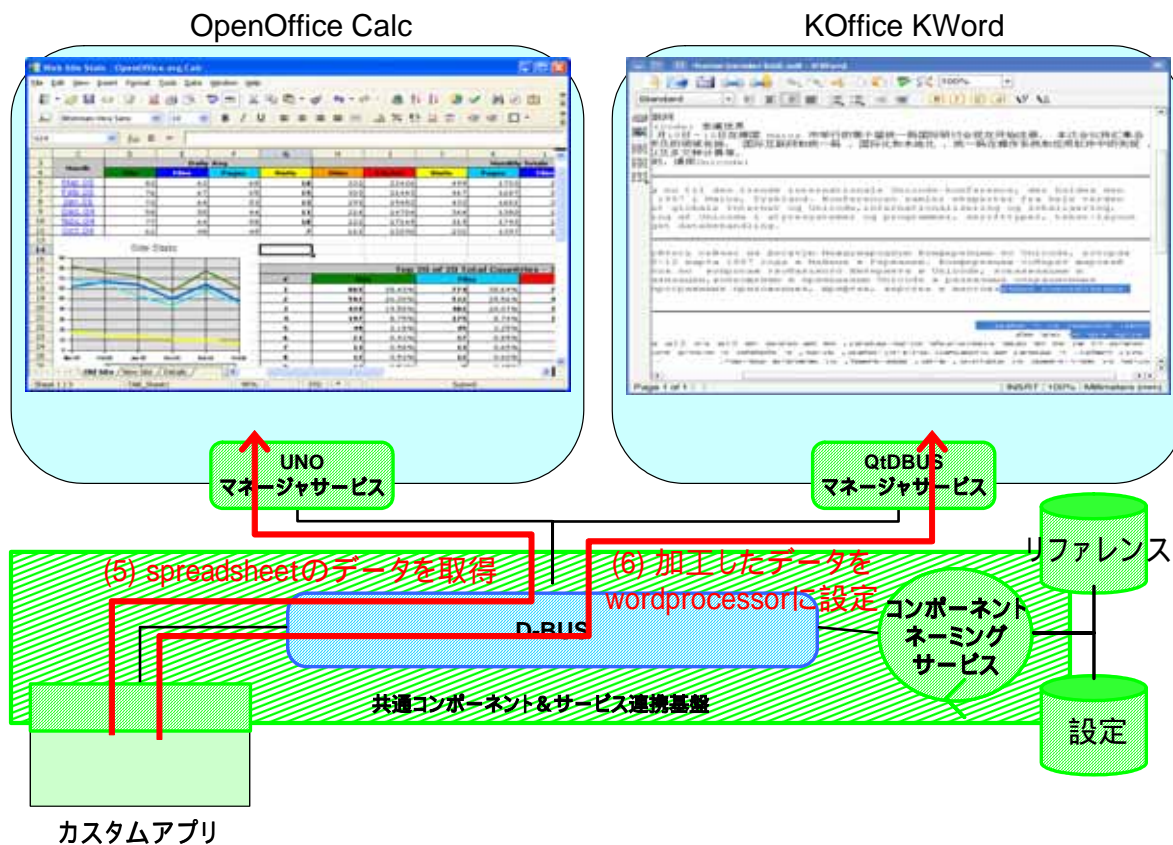


図 7-35 抽象コンポーネント参照のユースケース 2

- (5) カスタムアプリケーションは、spreadsheet の抽象コンポーネントを参照し、spreadsheet 上の一連の数値データを取得する。
- (6) カスタムアプリケーションは、spreadsheet から取得したデータを処理し、その結果を wordprocessor の抽象コンポーネントを参照し、データを挿入する。

7.7 抽象コンポーネントの削除機能

抽象コンポーネントの削除処理の流れを、指定する抽象コンポーネントに対応する具象コンポーネントが UNO のコンポーネントである場合を例に図 7-36 ~ 図 7-37 で示す。

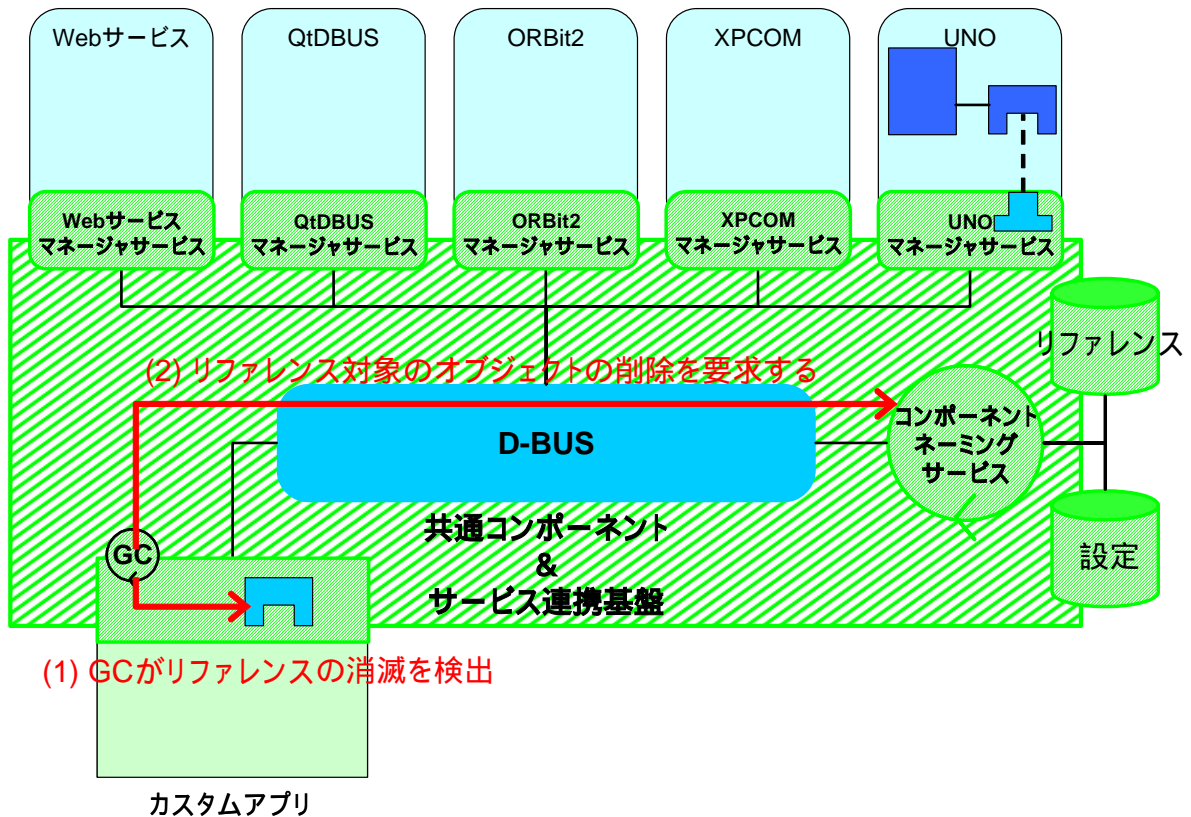


図 7-36 抽象コンポーネントの削除 1

- (1) GC がリファレンスの消滅を検出 (図 7-1-36)
 - カスタムアプリ作成ライブラリが持つ GC が proxy へのリファレンスの消滅を検出する。
- (2) リファレンス対象のオブジェクトの削除を要求する (図 7-1-36)
 - GC が proxy のリファレンス対象のオブジェクトの削除をコンポーネントネーミングサービスに要求する。

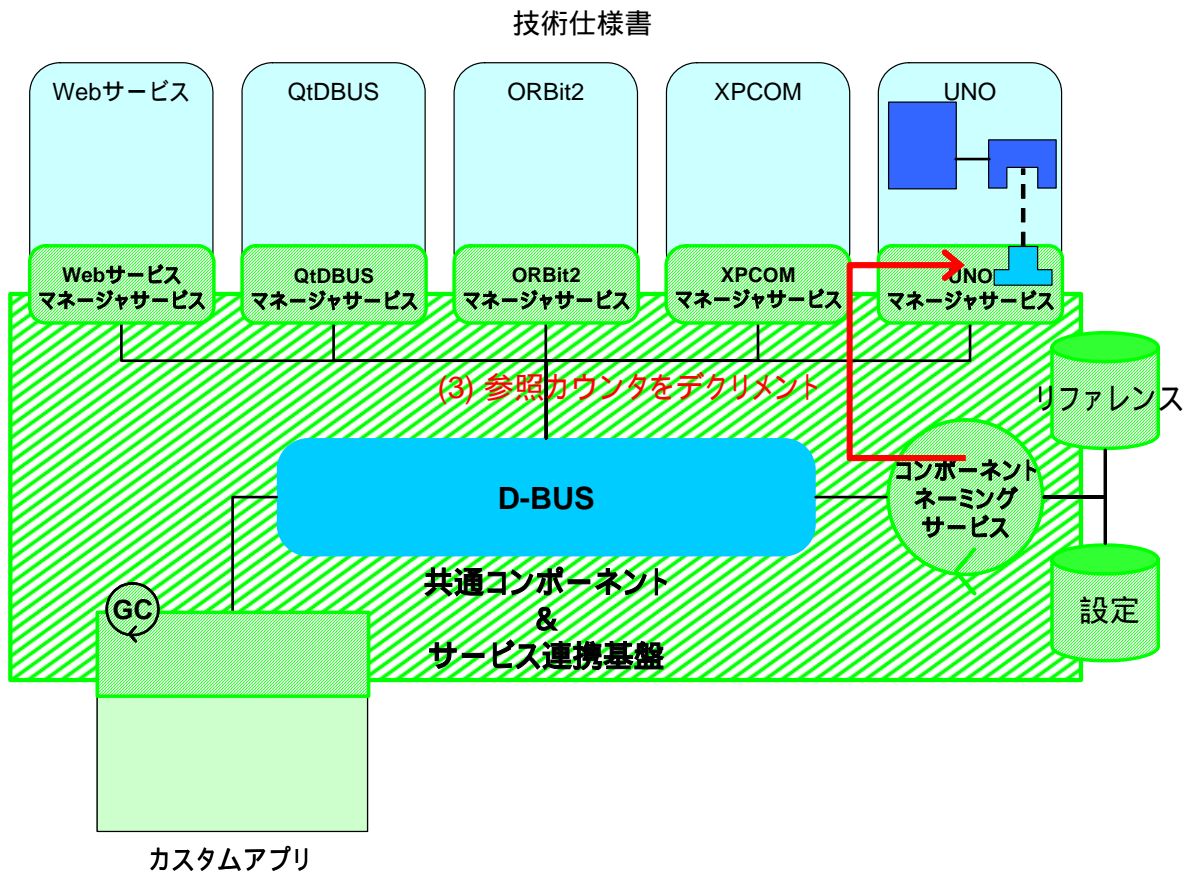


図 7-37 抽象コンポーネントの削除 2

(3) 参照カウンタをデクリメント (図 7-1-37)

- コンポーネントネーミングサービスから各マネージャサービスに対して、参照する具象コンポーネントの参照カウンタのデクリメント、もしくは参照ポイントの削除を要求する。
- **各コンポーネント技術が GC を用いている場合には、具象コンポーネントの参照カウンタのデクリメント、もしくは参照ポイントの削除が必要になる。**
 - **共通コンポーネント基盤における GC の利用 (および GC の同期) は望ましいが実現が難しいため、プロトタイプ等を作成し検証した上で決定する課題とする。**

7.8 Web サービス呼び出し機能

Web サービス呼び出し処理の流れを、図 7-38 ~ 図 7-40で示す。

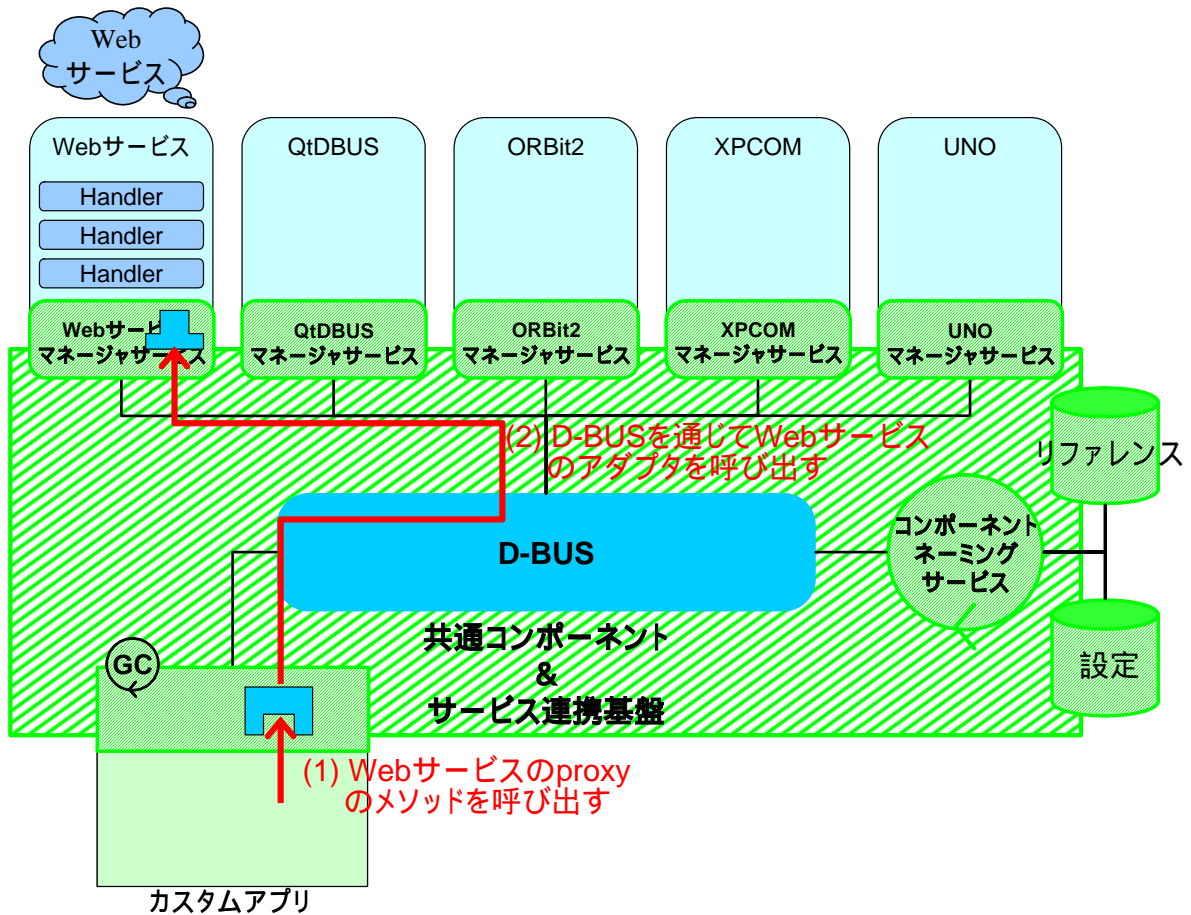


図 7-38 Web サービス呼び出し 1

- (1) Web サービスの proxy のメソッドを呼び出す (図 7-1-38)
 - カスタムアプリ作成ライブラリは Web サービスの proxy のメソッドを呼び出す。
- (2) D-BUS を通じて Web サービスのアダプタを呼び出す (図 7-1-38)
 - Web サービスのアダプタに対して D-BUS の METHOD_CALL メッセージを送信する。

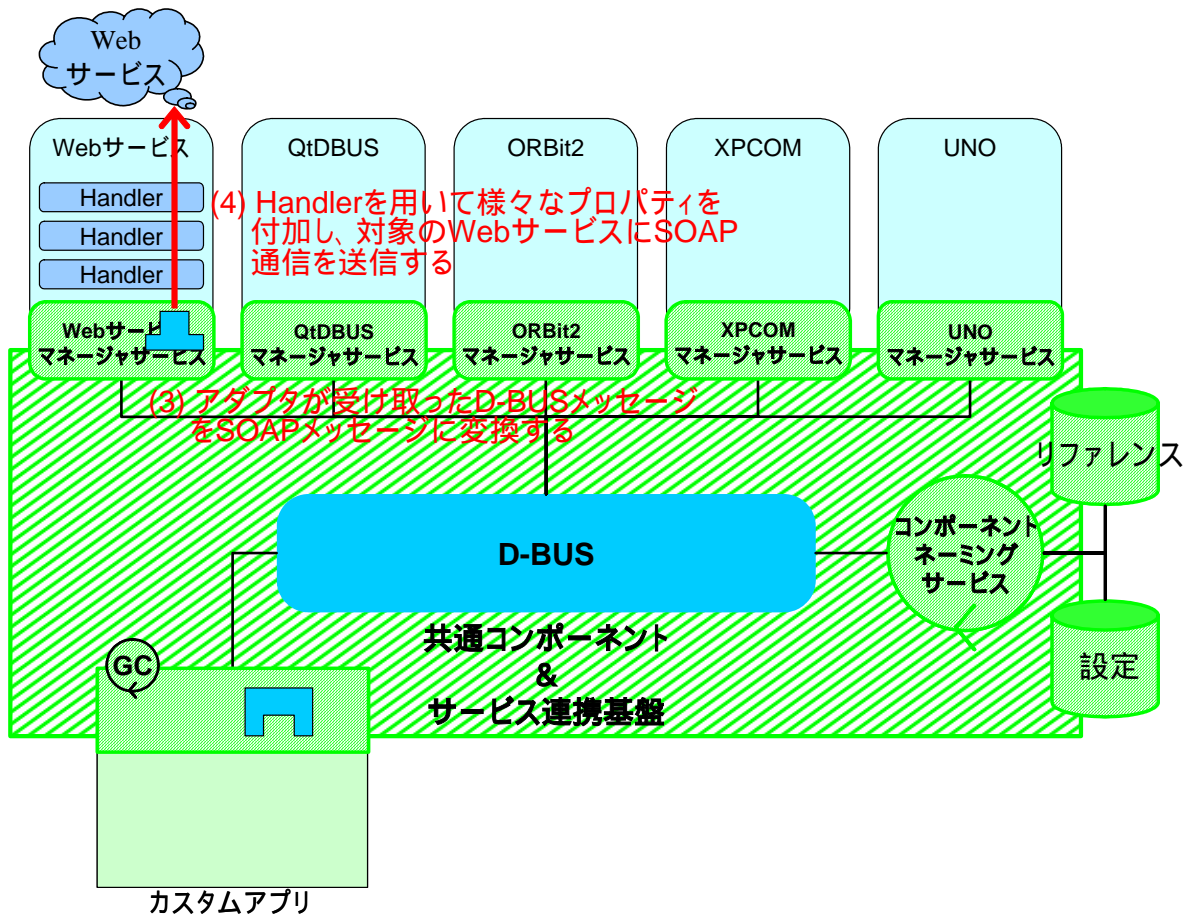


図 7-39 Web サービス呼び出し 2

- (3) アダプタが D-BUS メッセージを SOAP メッセージに変換 (図 7-1-39)
- アダプタが受け取った D-BUS メッセージを SOAP メッセージに変換する。
 - **D-BUS メッセージと SOAP メッセージのマッピング仕様については 4 章に記述している。**
 - **D-BUS メッセージと SOAP メッセージが直接対応付けられない物については、マッピングルールを設定可能とする。設定方法については、プロトタイプ作成等による検証が必要となるため、検証の後に決定する。**
- (4) Handler を用いて様々な SOAP プロパティを付加する (図 7-1-39)
- SOAP メッセージの送信時に、直接 D-BUS メッセージからは付加できない SOAP のプロパティを Handler により付加する。

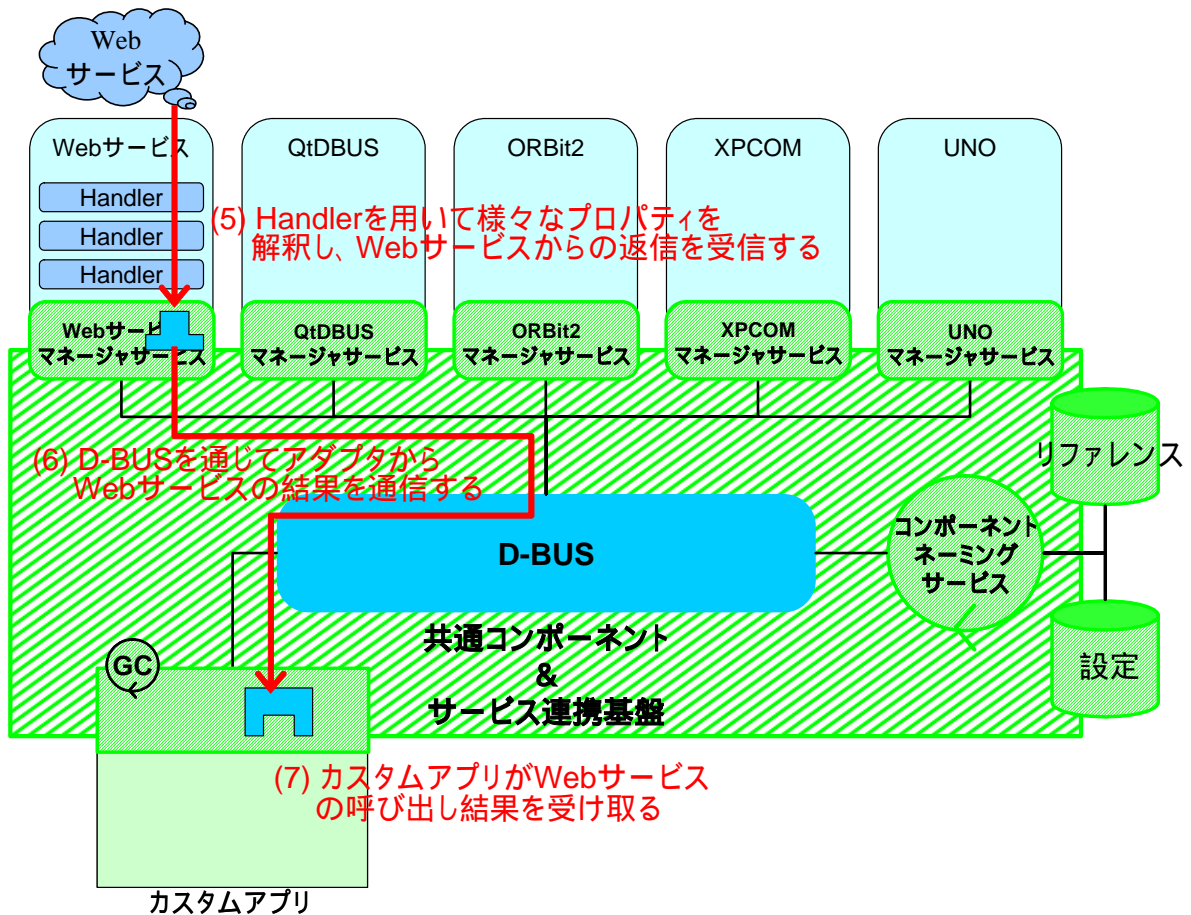


図 7-40 Web サービス呼び出し 3

- (5) Handler を用いて様々な SOAP プロパティを解釈する (図 7-1-40)
- Web サービスの結果として受信した SOAP メッセージのプロパティを Handler により解釈する。
 - プロパティを解釈した結果の SOAP メッセージをアダプタで D-BUS メッセージに変換する。
- (6) D-BUS を通じてアダプタから Web サービスの結果を通信する (図 7-1-40)
- アダプタが変換した D-BUS メッセージを D-BUS を経由してカスタムアプリに通信する。
- (7) カスタムアプリが Web サービスの呼び出し結果を受け取る (図 7-1-40)
- カスタムアプリが Web サービスの呼び出し結果を D-BUS メッセージとして受信する。

8 今後の課題

本設計には、以下のような課題が残っている。

8.1 技術的課題

(1) 共通コンポーネント基盤における GC の実現

近代的なコンポーネント技術としては、.NET Framework のように GC が搭載されていることが望ましい。そのため、共通デスクトップ基盤にも GC を搭載するものとして仕様の策定を行った。

しかし、実際のオブジェクトの管理は各コンポーネント技術が行うことになる。そのために、共通デスクトップ基盤の GC と各コンポーネント技術との GC は同期をとる必要がある。

GC は実現するのも難しいが、複数の GC の同期をとることはさらに難しい。そのため、今後プロトタイプ作成による検証を行いながら詳細を決定する技術的課題として残す。

(2) アセンブリモデル

調査の結果、4.14 および 4.15 に述べた通り、複数のコンポーネントの結合を XML でアセンブリモデルとして記述するという仕様が考えられる。

しかし、共通デスクトップ基盤として XML によるアセンブリモデルの導入が適切かどうかは、プロトタイプ作成による検証がなければ判断できない。そのため、今後の技術的課題として残す。

(3) 分散コンポーネント対応

.NET Framework、COM+、UNO、Bonobo/ORBit2 などのようにデスクトップ環境では分散コンポーネント技術が存在することが一般的である。

また、分散コンポーネント技術を実現することによって、シンクライアント（アプリケーションの表示を行うコンポーネントはクライアント側で動作し、ドキュメントファイルを扱うコンポーネントはネットワーク上で動作する等）など、オープンソースのデスクトップ環境のソリューションが増える。

しかし、共通デスクトップ基盤として分散環境を実現するためには、既存の開発チームと連携をしながら拡張を行わなければ、各コンポーネント技術の互換性を維持したまま拡張をすることができない。そのため、今後の技術的課題として残す。

(4) コンポーネント技術の共通化

本設計では、D-BUS をベースとし、各コンポーネント技術をつなぐことでコンポーネント技術の連携を実現している。その結果、ネーミングサービス等の処理が必要となっている。本来であれば、これらの技術はコンポーネント技術として統合されているべきである。

しかし、ネーミングサービス等の技術を含んだコンポーネント技術の共通化を実現するためには、本調査の対象とした各コンポーネント技術を一から作り直す必要がある。そのためには、各コンポーネント技術の開発者を含むコミュニティとの議論をしながらアーキテクチャ等の設計を行う必要がある。

今回は、そのための第一歩として、既存コンポーネント技術との整合性を考慮した漸進的な設計を行った。コミュニティでの議論は今後の課題である。

(5) Web サービス対応のための機能拡張

調査の結果、4章に述べた通り共通デスクトップ基盤の一部として Web サービスの機能を実現するためには、以下の D-BUS へのポーリング機能追加が考えられる（詳細は 4.9.2 参照）。

しかし、これらの機能は共通デスクトップ基盤の Web サービス機能の利用方法によって要望が変化する。具体的には、ポーリングはデスクトップへのリアルタイム性への影響が考えられる。

そのため、今後共通デスクトップ基盤の実装過程で、ユーザの要望によって実現を検討する技術的課題として残す。

8.2 コミュニティへの働きかけ

本設計で実現しようとしている共通コンポーネント基盤を実際に開発するには、既存コンポーネントへの機能追加、D-BUS への機能追加が必要となるため、コミュニティの協力が必要不可欠である。具体的に必要な機能追加は以下の通りである。

- (1) QtDBus に外部からオブジェクトの検索、参照、アクティベーションを実行可能にすること。
- (2) KParts に通信、検索、参照、アクティベーションなどの機能を追加すること。
- (3) ORBit2 における非同期通信機能の追加すること。
- (4) XPCOM の IDL に同期・非同期の指定機能の追加すること。
- (5) XPCOM の通信にメソッド呼び出しの追加すること。
- (6) XPCOM の通信に IDL と同様のデータ型を使用可能にすること。
- (7) OpenOffice.org の IDL に同期・非同期の指定機能の追加すること。
- (8) D-BUS の検索、参照、アクティベーションをオブジェクトパスに対して実行可能にすること。
- (9) D-BUS メッセージを拡張し、ERP などの情報をメッセージヘッダに追加すること。

上記の機能追加を各コミュニティへ働きかけることは、今後の課題とする。

以上