

Development Services of Open Source Software Platforms

Technical Investment for Developing Common
Desktop Base

Technical Specifications

December 2006

Information-technology Promotion Agency, Japan

Trademarks

- Microsoft, MS, Windows, and others are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
- Java, JDK, and others are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.
- Linux is a registered trademark or trademark of Linus Torvalds, owner of the trademark on a world-wide basis.
- OpenOffice.org is a registered trademark of Team OpenOffice.org e.V.
- Mozilla and the Mozilla logo are trademarks of the Mozilla Foundation.
- UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.
- Qt is a registered trademark of TrollTech in Norway.
- All other company names and product names are the property of their respective owners.
- TM and other trademarks are not appended to the registered trademarks and trademarks in this document.

Technical Specifications
- Table of Contents -

1 OVERVIEW	5
1.1 TITLE	5
1.2 BACKGROUND.....	5
1.3 PURPOSES	5
1.4 POSITIONING OF THESE TECHNICAL SPECIFICATIONS.....	6
1.5 SCOPE OF THESE TECHNICAL SPECIFICATIONS	7
2 SYSTEM REQUIREMENTS.....	8
3 REQUIREMENTS.....	9
4 HANDLING OF EXISTING TECHNOLOGIES AND THE COMMON DESKTOP PLATFORM, AND MESSAGE COMMUNICATION SPECIFICATIONS	10
4.1 DCOP AND QTDBUS	10
4.1.1 <i>Object Management</i>	10
4.1.2 <i>Interface (IDL)</i>	10
4.1.3 <i>Common Components</i>	10
4.1.4 <i>Communication</i>	11
4.2 ORBit2	11
4.2.1 <i>Object Management</i>	11
4.2.2 <i>Interface</i>	11
4.2.3 <i>Common Components</i>	11
4.2.4 <i>Communication</i>	12
4.3 XPCOM	12
4.3.1 <i>Object Management</i>	12
4.3.2 <i>Interface</i>	12
4.3.3 <i>Common Components</i>	12
4.3.4 <i>Communication</i>	12
4.4 UNO	13
4.4.1 <i>Object Management</i>	13
4.4.2 <i>Interface</i>	13
4.4.3 <i>Common Components</i>	13
4.4.4 <i>Communication</i>	13
4.5 D-BUS.....	14
4.5.1 <i>Naming Service</i>	14
4.5.2 <i>Interface</i>	14
4.5.3 <i>Communication</i>	14
4.6 NET FRAMEWORK	14
4.6.1 <i>Life Cycle Management</i>	15
4.6.2 <i>Activation</i>	15
4.6.3 <i>Handling of Threads</i>	15
4.7 WSDL	15
4.7.1 <i>Interface</i>	15
4.8 SOAP	16
4.8.1 <i>Interface</i>	16
4.9 JAX-WS	18
4.9.1 <i>Mapping to SOAP</i>	18
4.9.2 <i>Synchronous and Asynchronous Communications</i>	18
4.10 AXIS2.....	19
4.10.1 <i>Code Generation from WSDL</i>	19

Technical Specifications

4.10.2 Mapping to SOAP.....	19
4.11 XFIRE	19
4.11.1 Mapping to SOAP	19
4.12 JBI.....	20
4.12.1 Communication.....	20
4.12.2 Component Management and Component Life Cycle.....	20
4.13 SERVICEMIX	20
4.13.1 Mapping between Multiple Message Communication Protocols.....	20
4.14 SCA.....	21
4.14.1 Service Assembly Model	21
4.14.2 Mapping Between Multiple Message Protocols.....	21
4.15 TUSCANY	21
4.15.1 Mapping Between Multiple Message Communication Protocols.....	21
5 SYSTEM ARCHITECTURE.....	22
5.1 CHARACTERISTICS OF THIS ARCHITECTURE.....	22
5.2 LOGICAL ARCHITECTURE.....	22
6 SPECIFICATIONS FOR IDL.....	25
6.1 RULES FOR INTERFACE NAMES	25
6.2 RULES FOR METHOD NAMES	26
6.3 RULES FOR ARGUMENT NAMES	26
6.4 RULES FOR ANNOTATIONS.....	26
6.5 TYPE NAMES THAT CAN BE SPECIFIED IN IDL.....	27
7 FUNCTIONAL DESIGN SPECIFICATIONS	28
7.1 COMMON COMPONENT DEFINITION	28
7.2 COMMON COMPONENT CREATION FUNCTION	29
7.2.1 Use-case of Creating a Common Component.....	35
7.2.2 Sequence in the Case of QtDBUS	40
7.2.3 Sequence in the Case of ORBit2	41
7.2.4 Sequence in the Case of XPCOM.....	42
7.2.5 Sequence in the Case of UNO.....	43
7.3 COMMON COMPONENT REGISTRATION FUNCTION	44
7.3.1 Sequence	46
7.4 WEB SERVICE REGISTRATION FUNCTION	47
7.5 COMMON COMPONENT SEARCH FUNCTION	49
7.5.1 Sequence	52
7.6 COMMON COMPONENT REFERENCE FUNCTION.....	53
7.6.1 Synchronous Communication Processing.....	53
7.6.2 Asynchronous Communication Processing.....	56
7.6.3 Use-case of Referring to a Common Component.....	59
7.7 COMMON COMPONENT DELETION FUNCTION	61
7.8 WEB SERVICE CALL FUNCTION.....	63
8 ISSUES TO BE ADDRESSED IN FUTURE	66
8.1 TECHNICAL ISSUES.....	66
8.2 APPROACH TO COMMUNITIES.....	67

1 Overview

1.1 Title

The title of this document is "Technical Specifications."

1.2 Background

Many ISV, System Integrators (SIs), and software houses do not select an open source desktop environment, such as Linux, as a platform for developing business applications and systems because it is more difficult to develop business applications in an open source environment compared to the Windows environment. There are some barriers caused by insufficient realization of the following features:

- (1) Collaboration features among desktop applications
 - ① Data transfer between applications
 - ② Linking and embedding of objects
 - ③ Binding of objects
- (2) Collaboration features between network services and desktop applications

The features in (1) can be realized to some extent in a closed environment of one of the desktop component technologies for open source software ("OSS" hereinafter); DCOP, QtDBus (KDE); ORBit2 (GNOME); XPCOM (Mozilla); or UNO (OpenOffice.org). However, it is impossible to interoperate across the above component technologies because standards and systems for open source desktop environments have not been established sufficiently. Therefore, desktop applications can satisfy needs by complementing each other but cannot collaborate with each other. The feature in (2) is not realized in OSS.

1.3 Purposes

This framework aims realization of a common component platform and service linkage platform as a common desktop base ("common desktop platform" hereinafter) that satisfy the following use cases to solve the above problems' demands.

- (1) To realize easy development of rich UI applications by combining common (abstract) components, such as word processors, spreadsheets, mailers, and schedulers.
 - The deployments of this system architecture become popular solution for project managers, developers, and CIO at ISV, System Integrators, and software houses, etc. in order to reduce costs of system and application developments.
- (2) Service collaboration technology
 - ① Realizes applications that use combined components to display the results of a web service by calling components' service.
 - ② Realizes creation of a system that operates in combination with multiple web services (e.g. stock transaction system service, passport-issuing system) as a desktop application.
- (3) To enable easy creation of a business application by combining the features in (1) and (2) using script languages such as Ruby or Python.

1.4 Positioning of These Technical Specifications

This project acts as the first step to progressively realize a system that can interoperate across DCOP, QtDBus (KDE), ORBit2 (GNOME), XPCOM (Mozilla), and UNO (OpenOffice.org) in order to achieve the purposes in 1.3.

In this project, to achieve the purposes in 1.3, D-BUS and SOAP technologies are used as a platform that links the desktop component technologies and a protocol for web services, respectively. D-BUS was selected because it was developed as a desktop common bus by freedesktop.org project and there is a tendency that increasing number of open source softwares for desktop, e.g., HAL (Hardware Abstraction Layer) and SkypeAPI, are adopting it. SOAP was selected because it was a most extensively used protocol for the web service.

The findings in this project are summarized in two documents: "Research Report" summarizes the examination results of existing technologies, and "Technical Specifications" (this document) describes the technical specifications derived from those examination results.

These technical specifications describe the design in external specifications of "the system to be realized." Chapter 4 describes the common desktop platform specifications that are based on the examination results, especially message definitions and message communication mechanism specifications, as well as the specifications for converting messages into SOAP messages. Chapter 5 describes the logical configuration of the system. Chapter 6 describes the IDL specifications of the system. Chapter 7 describes the functional specifications of the system. Chapter 4 is based on considerations in the Research Report, and see the detail for it.

The examination period of this project was between July 2006 and December 2006. Version or time of each technology when examined is shown below.

- KDE: ver. 3.3.5
- Qt: ver. 4.2
- ORBit2: ver. 2.14.3
- Bonobo: ver. 1.0.22
- XPCOM: As of September 2006
- UNO: As of September 2006
- D-Bus: ver. 0.9.1 (Specification 0.11)
- .NET Framework 3.0: As of September 2006
- 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 Scope of These Technical Specifications

These technical specifications describe the basic features to satisfy the purposes mentioned above.

- (1) Specifications for IDL
 - Specifications for IDL used to define primitive data types and behaviors of common (abstract) components
- (2) Common component definition feature
 - A feature that enables inheritance relationships between common (abstract) components and specific components in each component technology to be described.
- (3) Common component creation feature
 - A feature that creates a specific component that has inherited a common (abstract) component.
- (4) Common component registration feature
 - A feature that enables search for references to common (abstract) components and specific components that have inherited those common (abstract) components.
- (5) Web service registration feature
 - A feature that enables search for references to web services
- (6) Common component search feature
 - A feature that searches for a specific component that has inherited a common (abstract) component.
- (7) Common component reference feature
 - A feature that enables reference to a common (abstract) component as data across component technologies.
- (8) Common component deletion feature
 - A feature that deletes a specific component that has inherited a common (abstract) component.
- (9) Web service call feature
 - A feature that enables a service on a network to be called and a component on a network to be referred to.

2 System Requirements

The requirements of this system (framework) are as follows.

- (1) The tool kits etc. that use this system (framework) can receive services by using the functions provided in the D-BUS library and newly developed libraries that have implemented the specifications described in this document.
 - When the functions in D-BUS are insufficient, function addition should be proposed to the D-BUS Community.
- (2) The target shall be enterprise desktops and workstations.
 - However, they must have D-BUS operative.

3 Requirements

The design requirements to be satisfied by this system (framework) are as follows.

- (1) Cross-reference is possible between multiple component technologies
 - Both caller and callee should be considered
 - For security reasons, the situation that the callee **cannot be called** must be also possible
- (2) Objects (data) can be handled abstractly
 - The objects mentioned here shall be spreadsheets or editors
- (3) Calls in each technology should not be changed whenever possible
- (4) The mechanisms should be lightweight
- (5) Components can be easily combined
 - Ease of combination shall be realized by using script languages
- (6) There should be no memory leaks
- (7) Local processes and network services can be handled transparently
- (8) Network security should be considered

4 Handling of Existing Technologies and the Common Desktop Platform, and Message Communication Specifications

This project aimed to design a fundamental technology that enables collaboration among desktop component technologies and also between such technologies and web services. For this purpose, DCOP and QtDBus (KDE), ORBit2 (GNOME), XPCOM (Mozilla), and UNO (OpenOffice.org) technologies were examined. Based on the results of the examination, this chapter describes the following aspects of the handling of existing technologies and the common desktop platform, as well as the message communication specifications:

- (1) Object management
- (2) Interface (IDL)
- (3) Common components
- (4) Message definitions and message communication mechanism specifications
- (5) Specifications for converting messages of the common components into SOAP messages

KParts and Bonobo, high components above QtDBus and ORBit2, respectively, were also examined.

4.1 DCOP and QtDBus

To analyze and study object management and communication methods, message definitions and mechanisms to extract for the common component platform, and IDLs, DCOP (Desktop COmmunication Protocol), a component technology used in KDE ver. 3.x, current stable, and earlier, the QtDBus module (wrapper for D-BUS) in KDE ver. 4.0, next release, and subsequents, and KParts, an object technology, were examined. Specifically, the message and IDL specifications of DCOP, and the implementation and development trends of KDE4 or Qt related to QtDBus were studied. Specifications to be realized by the "common component platform and the service linkage platform," derived from the results, are described below:

4.1.1 Object Management

Because the garbage collector (GC) function of Qt is used for the life cycle management of objects, only the creation of objects has to be considered. The GC function should be incorporated in the common component platform for object management. To ensure the correct life cycle management of objects, the GC function of the common component platform must be synchronized with that of Qt. For more details, refer to Chapter 7.

DCOP and QtDBus are not designed to enable objects to be accessed or referred to from the outside. They do not support a mechanism for activation of objects. The functions to support activation must be provided as common components in the common component platform. For example, UNO objects must be accessible from KDE objects for reference and vice versa. To achieve this, QtDBus must be expanded to enable objects to be searched, referred to, and activated from the outside. This requires collaboration with the development communities of KDE and Qt, and needs to be addressed in the future.

4.1.2 Interface (IDL)

The IDL of DCOP is almost completely compatible with C++ and realizes highly flexible expression. An IDL created in the common component platform can be easily converted into the IDL of DCOP. The IDL of QtDBus is based on the same DTD as the introspection described in 4.1 of the "D-BUS Research Report." Thus, the IDL of QtDBus shall be used as an IDL created in the common component platform.

4.1.3 Common Components

Component types provided by KParts shall be used as common components provided by the common component platform without changes.

Technical Specifications

Original KParts is independent of DCOP or QtDBus. In KOffice, on the other hand, it is expanded to work in concert with DCOP or QtDBus. In implementing common components in the common component platform, the communication, search, reference, and activation functions of the component technologies must work in concert with KParts. This requires collaboration with the development communities of KDE and Qt, and needs to be addressed in the future.

4.1.4 Communication

Basically, QtDBus supports MethodCallMessage, which means a synchronous call, and SignalMessage, which means asynchronous communication (*Signal*), as the necessary types of messages. DCOP and QtDBus are designed to use all data types of C++ and extensible data types of D-BUS (ARRAY and STRUCT), respectively, to send any type of data.

The message types of DCOP and QtDBus compare favorably with those of ORBit2, XPCOM, or UNO. Thus, there is no need to add message types for implementing the common component platform.

The data types of DCOP and QtDBus compare favorably with those of ORBit2, XPCOM, or UNO. Thus, there is no need to add data types for implementing common components. Considering possible collaboration with new descriptions, an extensible mechanism should be provided for the common component platform.

4.2 ORBit2

To analyze and study object management and communication methods, message definitions and mechanisms to extract for the common component platform, and IDLs, Bonobo, a component object model used for GNOME, and ORBit2, a communication platform, were examined. Specifically, the message and IDL specifications, and the implementation were studied. Specifications to be realized by the "common component platform and the service linkage platform," derived from the results, are described below:

4.2.1 Object Management

Bonobo supports life cycle management using a reference counter in addition to life cycle management at the CORBA level. The GC function should be incorporated in the common component platform for object management. For that reason, Bonobo must implement the GC function too. In addition, there should be no discrepancies in references to the objects between the common component platform and Bonobo. To avoid those discrepancies, the GC function of the common component platform must be synchronized with that of Bonobo. For details, refer to Chapter 7.

The search, reference, and activation functions of objects are implemented in Bonobo as described in 3.2 "Search and Reference" and 3.3 "Activation" of "ORBit2 Research Report." Therefore, the functions required for implementing the basic functions to be provided in the common component platform are already implemented. However, to actually work with a common component platform, it is required to coordinate the name specification method used for the search and reference functions with the search keys used in the common component platform. DCOP, QtDBus, ORBit2, XPCOM, and UNO have different name and object coordination methods, which requires new definition of that coordination method. For details, refer to 7.2 (6).

4.2.2 Interface

As described in 4.1.1 "Data Types" of "ORBit2 Research Report," the data types that can be described as IDL also correspond to the data types of D-BUS, which does not require addition of data types. However, ORBit2 cannot differentiate synchronous and asynchronous communications, and thus, needs to be expanded. Oneway operation should be supported by the D-BUS *Signal*.

4.2.3 Common Components

A list of common components provided by Bonobo shall be used as a list of common components provided by the common component platform without changes.

4.2.4 Communication

As described in 6.1 and 6.3 of "ORBit2 Research Report," ORBit2 can use normal synchronous call and Oneway communications. These are equivalent to D-BUS method call and *Signal*. In addition, asynchronous call using callback has been implemented in CORBA3.0 or later and is also required in the common desktop platform: thus, the function needs to be added to ORBit2 too. This requires collaboration with the development community of GNOME, and needs to be addressed in future.

4.3 XPCOM

To analyze and study object management and communication methods, message definitions and mechanisms to extract for a common component platform, and IDLs, XPCOM (Cross-Platform Component Object Model), a component object model used for Mozilla, was examined. Specifically, the message and IDL specifications, and the implementation were studied. Specifications to be realized by the "common component platform and the service linkage platform," derived from the results, are described below:

4.3.1 Object Management

As described in 3.1 "Life Cycle Management" of "XPCOM Research Report," object management in XPCOM will be simplified when the GC function, which is now under development, has been implemented. The GC function should be incorporated in a common component platform for object management. However, there should be no discrepancies in references to the objects between the common component platform and XPCOM. To avoid those discrepancies, the GC function of the common component platform must be synchronized with that of XPCOM. For details, refer to Chapter 7.

The search, reference, and activation functions of objects are implemented in XPCOM as described in 3.2 "Search and Reference" and 3.3 "Activation" of "XPCOM Research Report." Therefore, the functions required for implementing the basic functions to be provided in a common component platform are already implemented. However, to actually work with the common component platform, it is required to coordinate Class ID and Contract ID used for the search and reference functions, as described in 3.2 and 6.4 of "XPCOM Research Report," with the search keys used in the common component platform. DCOP, QtDBUS, ORBit2, XPCOM, and UNO have different name and object coordination methods, which requires new definition of that coordination method. For details, refer to 7.2 (6).

4.3.2 Interface

As described in 4.1.1 "Data Types" of "XPCOM Research Report," the data types that can be described as IDL cannot enable any or extensible data types to be described, which requires addition of data types. XPCOM cannot differentiate synchronous and asynchronous communications, and thus, needs to be expanded. This requires collaboration with the development community of Mozilla, and needs to be addressed in future.

4.3.3 Common Components

A list of common components provided by XPCOM shall be used as a list of common components provided by the common component platform without changes.

4.3.4 Communication

XPCOM supports the message passing function only, as described in 6.1 "Message Definition" of "XPCOM Research Report." This is equivalent to the *Signal* described in "DCOP and QtDBUS Research Report" and "D-BUS Research Report." In addition, DCOP, QtDBUS, D-BUS, and UNO support the method call communication, as described in each research report. Therefore, for consistency of a common component platform, the method call function needs to be added. This requires collaboration with the development community of Mozilla, and needs to be addressed in future.

Technical Specifications

Likewise, since only the PRUnichar* data type can be used for communications, a function that enables using the same data types as those of IDL needs to be added for consistency. This also requires collaboration with the development community of Mozilla, and needs to be addressed in future.

4.4 UNO

To analyze and study object management and communication methods, message definitions and mechanisms to extract for a common component platform, and IDLs, UNO (Universal Network Objects), a component object model used for OpenOffice, was examined. Specifically, the message and IDL specifications, and the implementation were studied. Specifications to be realized by the "common component platform and the service linkage platform," derived from the results, are described below:

4.4.1 Object Management

Object management in UNO differs depending on what language is used for implementing applications, C++ or Java. As described in 8.1 "Distributed Components" of "UNO Research Report," objects are managed using a reference counter when applications are implemented with C++, and by the GC function when Java is used. The GC function should be incorporated in the common component platform for object management. For that reason, the GC function needs to be implemented even if C++ is used. In addition, there should be no discrepancies in references to the objects between the common component platform and UNO. To ensure the correct life cycle management of objects, the GC function of the common component platform must be synchronized with that of UNO. For details, refer to Chapter 7.

The search, reference, and activation functions of objects are implemented in UNO as described in 3.2 "Search and Reference" and 3.3 "Activation" of "UNO Research Report." Therefore, the functions required for implementing the basic functions to be provided in a common component platform are already implemented. However, to actually work with the common component platform, it is required to coordinate the name specification method used for the search and reference functions with the search keys used in the common component platform. DCOP, QtDBus, ORBit2, XPCOM, and UNO have different name and object coordination methods, which requires a new definition of that coordination method. For details, refer to 7.2 (6).

4.4.2 Interface

As described in 4.1.1 (1) "Data Types" of "UNO Research Report," UNOIDL can describe the most data. Therefore, the specifications for the data that can be described as IDL in the common component platform should be defined based on the specifications for the data that can be described in UNOIDL. However, it does not have a method of describing synchronous and asynchronous communications, so it needs to be expanded. This requires collaboration with the development community of OpenOffice.org, and needs to be addressed in the future.

The specifications should be defined based on the fact that the IDL of common components is described by inheriting multiple interfaces, as described in 4.1.1 (4) "Service" of "UNO Research Report." For details, refer to Chapter 6.

4.4.3 Common Components

As described in 5.1 "Common Component Definition" of "UNO Research Report," common components are not listed in the examined documents. In order to popularize the desktop platform in the future, it requires a list of the common components for developers.

4.4.4 Communication

UNO supports synchronous and asynchronous communications. It has sufficient functionality to realize a common component platform.

Technical Specifications

It can also use the same data types as IDL for communications and therefore, is sufficient as a platform for common components.

4.5 D-BUS

To analyze and study message definitions and mechanisms to extract for the common component platform, and IDLs, D-BUS, which has been developed as a common bus for desktops by freedesktop.org, was examined. Specifically, the message and IDL specifications, and the implementation were studied. Specifications to be realized by the "common component platform and the service linkage platform," derived from the results, are described below:

4.5.1 Naming Service

As described in Chapter 3 of "D-BUS Research Report," D-BUS has a naming service mechanism, which can assign a name to each application for management. This name can be specified for the DESTINATION header field of the message described in 5.1 of "D-BUS Research Report" to enable applications to communicate with each other. In addition, as described in 3.2 of "D-BUS Research Report," this mechanism enables acquisition of a list of the names possessed by a Bus in order to search for or refer to an application name as a sending destination of D-BUS. It also enables activation of an application, as described in 3.3 of "D-BUS Research Report."

As described above, this naming service mechanism has necessary functions for object management in a common component platform. However, it targets applications only and cannot enable objects to be searched for, referred to, and activated. If these functions become available for object paths, the basic functions for the common component platform are realized: therefore, function addition is required. This requires collaboration with the development community of D-BUS, and needs to be addressed in the future.

4.5.2 Interface

The interface information presently provided as the mechanism for D-BUS introspection is sufficient to be used as IDL, as described in Chapter 4.1 of "D-BUS Research Report."

This interface information is in the XML format, which is easy to expand by adding a DTD. Therefore, a DTD should be expanded in order to use the information as the IDL for this system (framework), because the information that can be expressed as the results of DCOP, QtDBus, ORBit2, XPCOM, and UNO examinations is insufficient. For details, refer to Chapter 6.

4.5.3 Communication

D-BUS supports two communication types: the method call communication and the *Signal* (no return values), as described in 5.1 of "D-BUS Research Report." Also, from the viewpoint of the functions used for sending, the method call has two types of communications: synchronous communication and asynchronous communication (callback). The above-mentioned 3 types of communications in total are supported by DCOP, QtDBus, ORBit2, XPCOM, and UNO, which does not require function addition at present.

Considering that the data types available for communication include the ARRAY, STRUCT, and DICT_ENTRY types, the required data types for the common component platform are supported.

4.6 NET Framework

To analyze and study object management and communication methods, message definitions and mechanisms to extract for the common component platform, and IDLs, Microsoft Windows .NET Framework 3.0 was examined.

Technical Specifications

Specifically, COM/COM+, a component technology in .NET was studied. Specifications to be realized by the "common component platform and the service linkage platform," derived from the results, are described below:

4.6.1 Life Cycle Management

Since implementation of the GC function is already performed or being considered in .NET Framework3.0 CLR and other component technologies, it should be implemented in the common component platform too. In addition, since each component technology performs life cycle management independently, a technology that integrates and manages the life cycle management of other component technologies is also needed in the common component platform. For details, refer to Chapter 7.

4.6.2 Activation

Although COM provides a service of object activation that the common component platform will implement using various component technologies, there are no useful references for defining specifications.

As a future issue, if the common component platform has to control the start and end operations of each component technology directly, the activation function is required in the common component platform. The activation method in COM should be used as a reference when a prototype is created based on this Technical Specifications to determine internal specifications.

4.6.3 Handling of Threads

The common component platform does not handle thread management. Therefore, there are no useful references for defining specifications.

As a future issue, if the common component platform has to provide a function that hides availability of the thread handling function of each component technology, this thread handling function is required in the common component platform. The thread handling method in COM should be used as a reference when a prototype is created based on this Technical Specifications to determine internal specifications.

4.7 WSDL

To analyze and study communication methods and interface description methods to design the service linkage platform, WSDL was examined. Specifications to be realized by the "common component platform and the service linkage platform," derived from the results, are described below:

4.7.1 Interface

WSDL provides 2-phase definition, i.e. "abstract service model" and "specific service model," in order to define an interface without assumption of transport protocols and computer languages to develop, etc. The interfaces for D-BUS and the desktop platform are categorized in the specific service model.

D-BUS has a mechanism to describe an interface, which is called introspection (for more details, refer to Chapter 4 of "D-BUS Research Report"). The interface expression described by the introspection is shown in Figure 4-1.

Technical Specifications

```

<!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">
    ...(omitted)...
    <method name="Bazify">
      <arg name="bar" type="(iiu)" direction="in"/>
      <arg name="bar" type="v" direction="out"/>
    </method>
    ...(omitted)...
  </interface>
</node>

```

Figure 4-1 Example of Interface Expression Described by Introspection

As shown in Table 4-1, the interface information that can be described by this introspection has a structure that can be associated with the binding elements and service elements described in 3.1.6 and 3.1.7 of “WSDL Research Report,” respectively. Therefore, the interface definition by using the D-BUS introspection should be associated with the web interface definition by using WSDL as shown in Table 4-1.

Table 4-1 Association Relationship between WSDL Elements and D-BUS Introspection

WSDL	D-BUS introspection
endpoint elements among the service elements	node elements
binding elements	interface elements
operation elements	method elements
input/output/infault/outfault elements	arg elements (in/out should be specified with direction attribute)

4.8 SOAP

To analyze and study communication methods and interface description methods to design the service linkage platform, SOAP technology was examined. Specifically, the message definition methods and message conversion were studied. Specifications to be realized by the "common component platform and the service linkage platform," derived from the results, are described below:

4.8.1 Interface

This research mainly studied the message definition methods and message conversion using the SOAP technology.

Assuming that D-BUS and the desktop platform should be associated with web services by using the specifications and implementation architectures of JAX-WS, Axis2, and XFire as references, the basic concept would be that D-BUS messages should be associated with the SOAP body if SOAP is selected as a protocol. D-BUS ErrorMessage should be associated with SOAP faults.

In this case, the point for mapping of D-BUS messages to the SOAP body is how their types are associated with each other. Among the D-BUS types described in 5.2 of "D-BUS Research Report," OBJECT_PATH, ARRAY, and DICT_ENTRY do not have the corresponding types to be associated with. On the other hand, some SOAP types, such as basic types and enumerated types shown in Table 4-3, cannot be associated with D-BUS types. Association of these types should be realized by mapping rules. For details, refer to Chapter 7.8 (3).

Technical Specifications

Table 4-2 Association Relationship between D-BUS Types and SOAP Types

No.	D-BUS type name	Association with SOAP type
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	No types that can be directly associated with
12	ARRAY	No types that can be directly associated with
13	STRUCT	Structure
14	DICT_ENTRY	No types that can be directly associated with

Table 4-3 SOAP Simple Types That Cannot Be Associated with D-BUS Types

No.	SOAP simple type
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

Technical Specifications

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

4.9 JAX-WS

To analyze and study communication methods and interface description methods to design the service linkage platform, JAX-WS was examined. Specifically, Java and SOAP mapping specifications were studied. Specifications to be realized by the "common component platform and the service linkage platform," derived from the results, are described below:

4.9.1 Mapping to SOAP

The common component platform and service linkage platform are planned to employ a D-BUS-based communication method. In this case, to enable collaboration between component services, D-BUS messages need to be mapped to SOAP messages. Mapping of Java and SOAP in JAX-WS is a good reference for associating D-BUS messages with SOAP messages. As an example, association between the message header fields described in 5.1 of "D-BUS Research Report" and Java is shown in Table 4-4.

Table 4-4 D-BUS Header Fields

No.	D-BUS field type	Type	Corresponding Java element
1	PATH	OBJECT_PATH	Package
2	INTERFACE	STRING	Interface
3	MEMBER	STRING	Method
4	ERROR_NAME	STRING	Exception
5	REPLY_SERIAL	UINT32	None in particular
6	DESTINATION	STRING	Package
7	SENDER	STRING	None in particular

This structure is almost the same as the D-BUS introspection described in 4.1 of "D-BUS Research Report."

As for the function to assign mapping-related metadata by using Annotations, which is implemented in JAX-WS, D-BUS introspection described in 4.1 of "D-BUS Research Report" also has Annotations that can be used to describe similar additional information, and thus, those Annotations can be associated to realize the same architecture.

4.9.2 Synchronous and Asynchronous Communications

To enable collaboration between component services, messaging synchronization also needs to be considered, in addition to the mapping described in 8.1 of "JAX-WS Research Report." As described in 5.1 of "D-BUS Research Report," D-BUS only supports synchronous calls and asynchronous calls (callback). Since polling for asynchronous calls is supported as a web service mechanism in general, implementation of a polling mechanism in D-BUS needs to be considered.

However, polling can be wrapped by callback. In addition, assuming that the polling mechanism is used by a desktop application, regular result check may affect a real-time property. For these reasons, whether polling should be supported or wrapped by callback will be considered according to user demands in future.

4.10 Axis2

To analyze and study communication methods and interface description methods to design the service linkage platform, Axis2 was examined. Specifically, Java and SOAP mapping specifications, and implementation methods to realize it were studied. Specifications to be realized by the "common component platform and the service linkage platform," derived from the results, are described below:

4.10.1 Code Generation from WSDL

The common component platform and service linkage platform are planned to employ a D-BUS-based communication method. In this case, to enable collaboration between component services, D-BUS messages need to be mapped to SOAP messages. This section describes the implementation method to realize that mapping.

The interface definition for a web service (WSDL) differs in each web service. When the mapping processing is performed, it needs to be performed according to the interface. Axis2 enables this processing by generating stub or skeleton codes from WSDL. This architecture of Axis2 can be used as a reference for defining specifications. However, while Axis2 generates Java or C++ codes (which means Java/C++ is directly converted to SOAP), the components in the common component platform use D-BUS for communication. Therefore, corresponding D-BUS messages and rules for converting them to SOAP should be defined. In practice, a program realizes a web service call by sending those D-BUS messages. For details, refer to 7.8.

The problem is what D-BUS message should be sent, because the service that uses the D-BUS cannot see the conversion method to SOAP. To solve this problem, C or C++ codes that create corresponding D-BUS messages should be generated.

4.10.2 Mapping to SOAP

Axis2 uses the codes generated by WSDL for the mapping processing that depends on an interface, as described in 5.1 of "Axis2 Research Report." However, it uses the Handler mechanism for the processing that does not depend on an interface, e.g. SOAP header analysis, identification of a corresponding service object (application implementation), and dispatch of a request, as described in 3.7 of "Axis2 Research Report." Considering creating and implementing a module for service collaboration in the common component platform as a technology similar to DCOP, ORBit2, XPCOM, and UNO, the module should use an architecture that configures a SOAP message by the header, like Axis2, to realize conversion from D-BUS messages to SOAP messages. For details, refer to 7.8.

4.11 XFire

To analyze and study communication methods and interface description methods to design the service linkage platform, XFire was examined. Specifically, Java and SOAP mapping specifications, and implementation methods were studied. Specifications to be realized by the "common component platform and the service linkage platform," derived from the results, are described below:

4.11.1 Mapping to SOAP

XFire uses the codes generated by WSDL for the mapping processing that depends on an interface (refer to 7.2 of "XFire Research Report"). However, it uses the Handler mechanism for the processing that does not depend on an interface, e.g. SOAP header analysis, identification of a corresponding service object (application implementation), and dispatch of a request, as described in Chapter 3 of "XFire Research Report." Creation of a module for service collaboration implementation as a technology similar to DCOP, ORBit2, XPCOM, and UNO in a common component platform should be considered. In this case, conversion from D-BUS messages to SOAP messages should be realized by creating a module that uses an architecture that configures SOAP messages by the Handler. For details, refer to 7.8.

4.12 JBI

To analyze and study object management and communication methods, message definitions and mechanisms to extract for the common component platform, JBI (Java Business Integration), a common component platform using Java, was examined. Specifically, the service integration method was studied. Specifications to be realized by the "common component platform and the service linkage platform," derived from the results, are described below:

4.12.1 Communication

The relationship between Normalized Message (NM) and Message Exchange (ME) of JBI can be considered to resemble the relationship between the body and header of a D-BUS message, which is described in Chapter 5 of "D-BUS Research Report." Actually, EndPoint, which indicates the sending destination of ME, corresponds to Path of the D-BUS header field.

Considering that, it is found that context information, which is included in NM, is missing in D-BUS messages. Also, there are no mechanisms that can be used for name resolution of a sending destination during routing, such as EndPoint Reference (EPR). Although there is room to consider where such information and mechanism should be added, the header field or body, they are needed in order to realize web service functions. Expansion of the D-BUS messages requires collaboration with the development community of D-BUS, and needs to be addressed in future.

4.12.2 Component Management and Component Life Cycle

JBI has a dedicated interface for component management. The method call of this interface is used to control component life cycles. The idea that an interface should be determined to manage components can be used as a good reference for defining specifications when the component life cycle management needs to be synchronized between the common component platform and each component technology. Specifically, to enable collaboration of the GC functions between the common component platform and each component technology, a specification should be defined so that an interface should be registered and controlled in the common component platform. However, because it is difficult to realize the GC function on the common component platform, a prototype should be created and reviewed before determination, which needs to be addressed in future. For details, refer to Chapter 7.

4.13 ServiceMix

To analyze and study object management and communication methods, message definitions and mechanisms to extract for the common component platform, ServiceMix, an open source implementation of JBI (Java Business Integration), a common component platform using Java, was examined. Specifically, the JBI communication methods on ServiceMix and component addition/deletion methods were studied. Specifications to be realized by the "common component platform and the service linkage platform," derived from the results, are described below:

4.13.1 Mapping between Multiple Message Communication Protocols

ServiceMix is an implementation of JBI and is actually implemented as a Normalized Message (NM) and a Message Exchange (ME), which defines an interface and configurations only. For details of NM and ME, refer to 3.1 of "JBI Research Report."

A noteworthy point of those implementations is mutual conversion between their messages and external protocols (HTTP, SOAP, JMS, etc). This point is not specifically described in JBI. It was found that in actual processing, each data is assigned to NM according to the contents of each protocol, or vice versa. Even when an external conversion module such as XFire is used, the operation is just delegated to it and the processing is the same in essence.

Likewise, for realization of web service collaboration, a common interface should be determined and actual conversion processing should be implemented in each protocol. This requires consideration on internal specifications, and needs to be addressed in future.

4.14 SCA

To analyze and study object management and communication methods, message definitions and mechanisms to extract for the common component platform, SCA (Service Component Architecture), open specifications for the implementation based on a service-oriented architecture, was examined. Specifically, the service integration method of SCA was studied. Specifications to be realized by the "common component platform and the service linkage platform," derived from the results, are described below:

4.14.1 Service Assembly Model

In the SCA service assembly model, service binding relationship can be described independently of a specific technology or language. Although granularities are different, common components in the common component platform also need to be described as the bound components independent of a specific technology or language. And then, if an application can be described as a combination of those components, reproducibility of components will be increased and thus, development efficiency will also be improved.

Specifically, when an application is created as a combination of the components across multiple component descriptions (e.g. spreadsheet in Calc of OpenOffice, address book in KOrganizer of KDE), application functions and interface can be described independently of a specific component, and therefore, the binding level of each component is decreased, which enables efficient application building.

To realize this, the SCA service assembly model is a good reference to define specifications. To be more precise, an application should be described by making the interface elements, Component elements, and the Composite element correspond to component functions, components, and an application, respectively. This requires creation of a prototype for review before determination, and needs to be addressed in future.

4.14.2 Mapping Between Multiple Message Protocols

SCA requires mapping between multiple message protocols to be described as Binding elements. These Binding elements are extensible by using protocols and interface descriptions. However, how to process protocols and interface descriptions in the Binding elements is not defined in SCA, so depends on how the SCA specifications are implemented.

Likewise, when communication between multiple protocols including web services is to be established in a common component platform, only communication-related definitions should be determined as the specifications of the common component platform and mapping between multiple protocols should be implemented depending on the protocols.

4.15 Tuscany

To analyze and study object management and communication methods, message definitions and mechanisms to extract for the common component platform, Tuscany, a project approved as Apache incubation, was examined. Specifically, the service integration method and specifications for mapping between multiple message communication protocols were studied. Specifications to be realized by the "common component platform and the service linkage platform," derived from the results, are described below:

4.15.1 Mapping Between Multiple Message Communication Protocols

The study showed that Tuscany enables the execution environment to automatically generate a Java interface for communication between multiple protocols by referring to the Binding elements of SCA that comply with the SCA specifications. In the case of a Binding element that supports a web service, for example, Tuscany refers to the corresponding interface description, such as WSDL, from that description to automatically generate a Java interface and enable communications with other technologies.

Likewise, when communication between multiple protocols including web services is to be established in a common component platform, the problem is the descriptions indicating how the interface is published and how an external interface is accessed. To solve this problem, the above-mentioned SCA Binding elements and extended models can be used as a reference for defining specifications. This requires creation of a prototype for review before determination, and needs to be addressed in future.

5 System Architecture

5.1 Characteristics of This Architecture

This logical configuration is designed to have the following characteristics in order to achieve the purposes described in 1.3, based on the background shown in 1.2.

- (1) Utilizes the functions of a D-BUS primitive message communication bus to realize collaboration between component technologies.
 - ① Does not design a new component technology on the D-BUS.
 - ② Introduces a concept of common components.
- (2) Abstracts the components used to create a custom application in order to make application development easier.
- (3) This is a gradual design considering the consistency with existing component technologies.
 - ① It is difficult to immediately begin activities to integrate the component technologies in the current open source desktop environment for historical reasons.
 - ② Minimizes the interface technology of each technology in order to have communities accept the design.

5.2 Logical Architecture

The system architecture to be realized in this project is as follows:

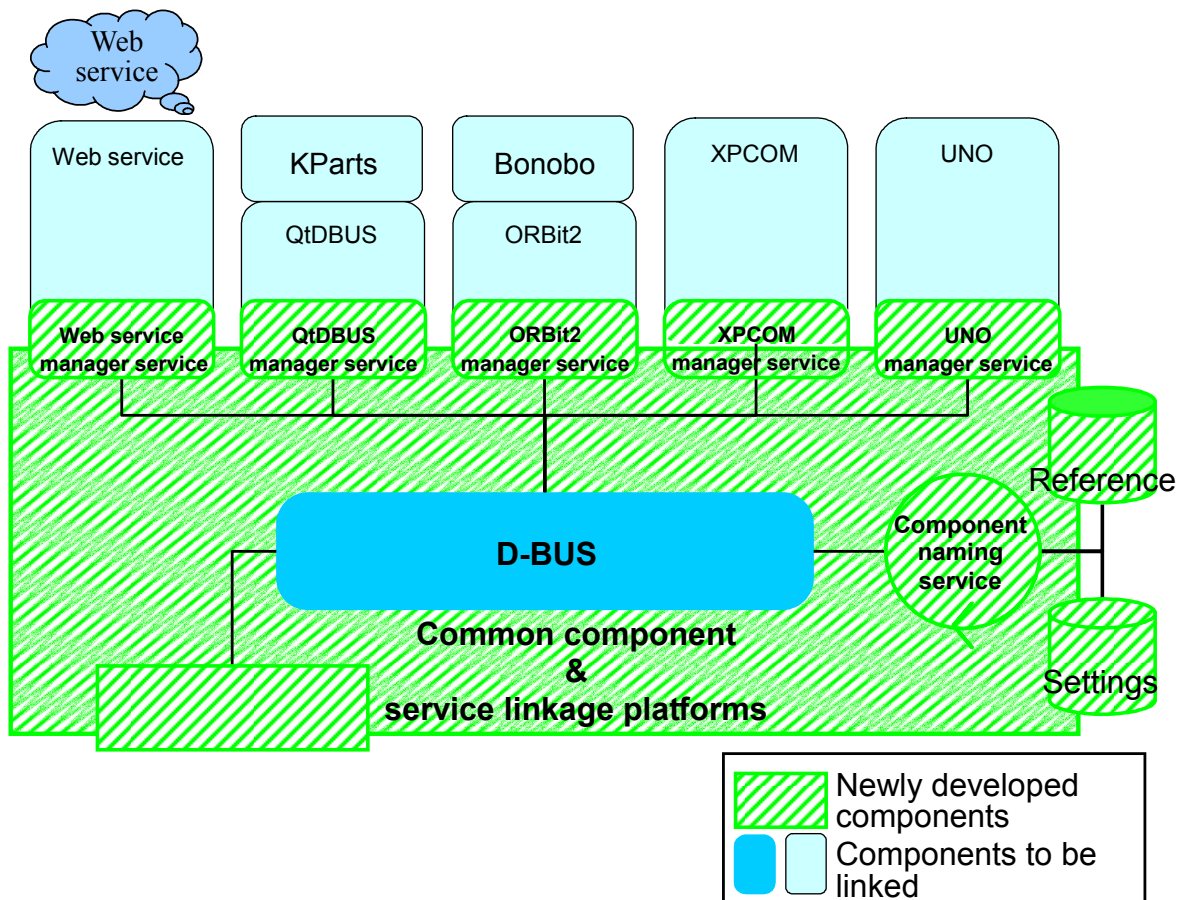


Figure 5-1 System Architecture

Technical Specifications

The technologies to be implemented are filled with green diagonal strokes. Specifically, the following are included:

Note that hereinafter we use generic communication technology names (i.e., ORBit2, QtDBus) to refer to desktop technologies that have different technologies for communication and components (such as Bonobo, ORBit2, KParts, and QtDBus).

- (1) Component naming service
 - Activates a component service.
 - ① Activates D-BUS.
 - ② Activates Session Bus along with the activation of D-BUS.
 - ③ Activates a component naming service with the application name `":component_namingservice"` immediately after Session Bus activation.
 - Creates, deletes, or searches for common components.
 - Creates a common component that is requested for creation from a custom application by referring to the settings. Registers the object path, common component name, and common component inheritance relationship of the created object in the reference. For details, refer to 7.2.
 - Deletes the common component that has been deleted in the custom application. For details, refer to 7.7.
 - Searches for the specific component that corresponds to the common component that is requested for searching from the custom application. For details, refer to 7.5.
- (2) QtDBus manager service
 - It runs as an application on D-BUS immediately after the component naming service is activated (application name: `":qtdbus_mgr."`)
 - Creates a specific component and corresponding adapter and keeps the adapter.
 - Transfers the method call or signal that is called via the D-BUS to an object on QtDBus.
- (3) ORBit2 manager service
 - It runs as an application on D-BUS immediately after the component naming service is activated (application name: `":orbit2_mgr."`)
 - Creates a specific component and corresponding adapter and keeps the adapter.
 - Converts the method call or signal that is called via the D-BUS for Bonobo/ORBit2 communication method.
- (4) XPCOM manager service
 - It runs as an application on D-BUS immediately after the component naming service is activated (application name: `":xpcom_mgr."`)
 - Acts as a proxy of XPCOM and D-BUS.
 - Creates a specific component and corresponding adapter and keeps the adapter.
 - Activates corresponding applications, such as Firefox and Thunderbird when creating a specific component.
 - Activation is not performed when the applications are already activated.
 - Converts the method call or signal that is called via the D-BUS for communication method on XPCOM.
- (5) UNO manager service
 - It runs as an application on D-BUS immediately after the component naming service is activated (application name: `":uno_mgr."`)
 - Creates a specific component and corresponding adapter and keeps the adapter.
 - Activates OpenOffice.org when creating specific components.
 - Activation is not performed when it is already activated.
 - Converts the method call or signal that is called via the D-BUS for communication method on UNO.

Technical Specifications

- (6) Web services manager service
 - It runs as an application on D-BUS immediately after the component naming service is activated (application name: ":webservice_mgr.")
 - Loads WSDL for web services and keeps the created interface.
 - Creates an adapter that corresponds to the interface and registers it to the component naming service on activation.
 - Transfers the method call or signal that is called via the D-BUS as a message for the web service.

- (7) Application Helper Library
 - This is a library used to develop a custom application by using the "common component platform and service linkage platform."
 - Sends a request for creating a common component to the component naming service.
 - Creates a proxy for the created common components.

6 Specifications for IDL

The IDL for the "common component platform and service linkage platform" has the D-BUS introspection (refer to Chapter 4 of "D-BUS Research Report"), which has the following changes, and is written in XML format.

- The interface elements are used as attributes to express common components.
- The extends attribute can be added to the interface elements to represent an inheritance relationship.
 - When multiple interface elements are inherited, they are delimited with a comma.
- The org.freedesktop.DBus.Method.Async annotation can be appended to a method element to represent asynchronous communication.
- The signal and property elements cannot be used.

The DTD shall be as follows:

```

<!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>
    
```

Figure 6-1 DTD for IDL

The following summarizes specifications other than the XML tags used to write the IDL.

6.1 Rules for Interface Names

An interface (tag name: interface) shall have a STRING-type name (attribute name: name) in a valid UTF8 format. The following restrictions are for an interface name:

- An interface name consists of the elements delimited with one or more periods. All the elements are composed of one or more characters.
- All the elements use ASCII characters ("[A-Z][a-z][0-9]_") only and must not start with a number.
- An interface name must include one or more periods. In other words, it must be composed of two or more elements.
- An interface name must not start with a period.
- An interface name must not exceed the maximum name length (255 characters).

6.2 Rules for Method Names

A method (tag name: method) shall have a STRING-type name (attribute name: name) in a valid UTF8 format. The following restrictions are for an interface name:

- It uses ASCII characters ("[A-Z][a-z][0-9]_") only and must not start with a number.
- It must not include periods.
- It must not exceed the maximum name length (255 characters).
- It must have the length of one character at minimum.

6.3 Rules for Argument Names

The rules for method names are the same as those for argument names.

6.4 Rules for Annotations

An annotation (tag name: annotation) shall have a name (attribute name: name) and value (attribute name: value). The following combinations of names and values can be specified:

Table 6-2 Combination of Names and Values That Can be Specified By Annotation

No.	Name	Value	Default	Value
1	org.freedesktop.DBus.Method.NoReply	true false	false	When true, response to a method call should not be expected.
2	org.freedesktop.DBus.Method.Async	true false	false	When true, a function call is an asynchronous call.

6.5 Type Names That Can Be Specified in IDL

The type names that can be specified in IDL are as follows.

Table 6-3 Type Names That Can Be Specified in IDL

No.	Type name	Description
1	INVALID	A type that does not comply with the specifications.
2	BYTE	An 8-bit unsigned integer.
3	BOOLEAN	A boolean type in which 0 represents FALSE and 1 represents TRUE.
4	INT16	A 16-bit signed integer.
5	UINT16	A 16-bit unsigned integer.
6	INT32	A 32-bit signed integer.
7	UINT32	A 32-bit unsigned integer.
8	INT64	A 64-bit signed integer.
9	UINT64	A 64-bit unsigned integer.
10	DOUBLE	A double type that is defined in IEEE754.
11	STRING	A string type in the UTF8 format. nul must be included as a termination character.
12	OBJECT_PATH	A name of an object instance. No limitations of the length. Example: /org/freedesktop/sample_object
13	SIGNATURE	A type signature. Up to 255 bytes. There are depth limitations in the recursive structure: 32 for both ARRAY and STRUCT.
14	ARRAY	An array. For example, an array including two INT32 is "aai."
15	STRUCT	A structure. For example, a structure including one INT32 and one UINT32 is "r(iu)."
16	VARIANT	A variant type.
17	DICT_ENTRY	A dictionary type expressed by an array containing pairs of keys and values.

7 Functional Design Specifications

7.1 Common Component Definition

To define a common component, the IDL described in Specifications for IDL of Chapter 6 is used. To enable each specific component to use common components, the IDL for common component is converted to the IDL for specific components. The converted IDL for the specific component will be QtDBus IDL, CORBA IDL, XPIDL, and UNOIDL.

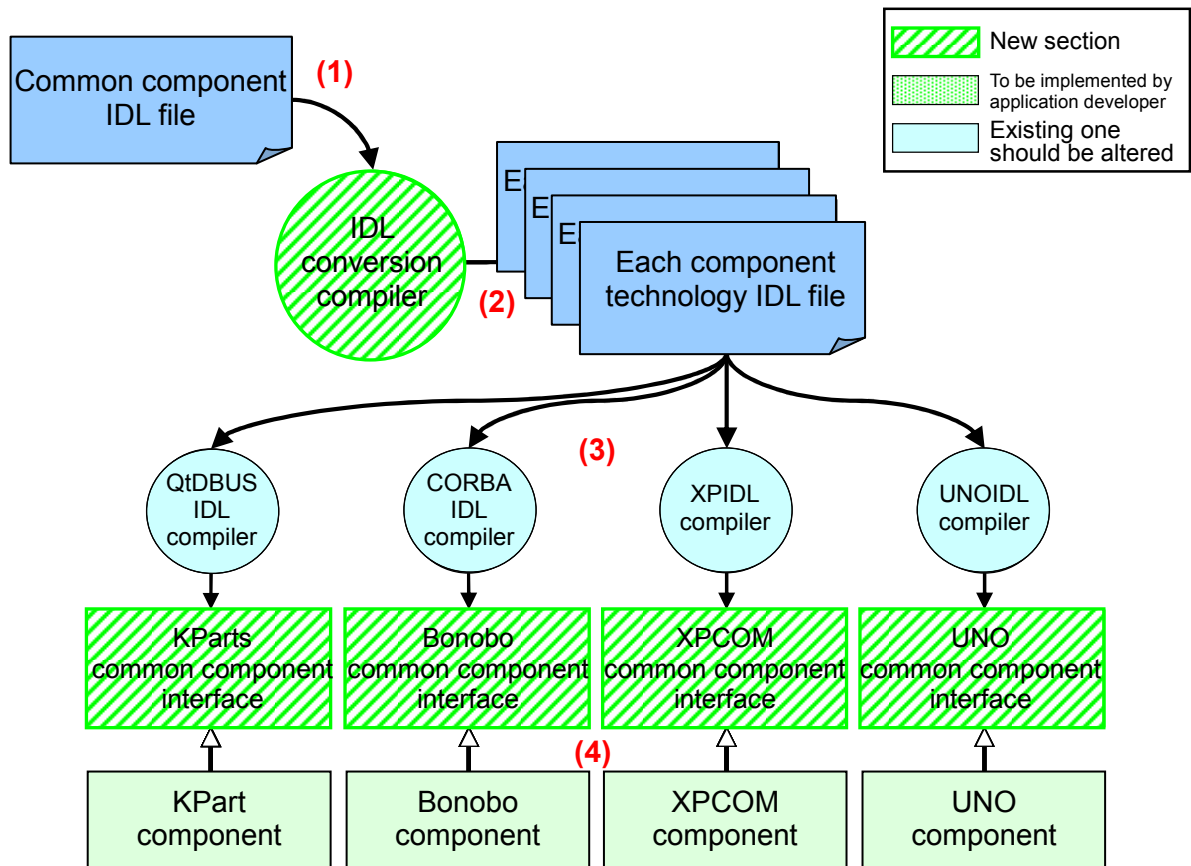


Figure 7-1 Definition of Common Component

- (1) Create an IDL for common components (Figure 7-1)
 - Create an IDL for common components by using the IDL described in Chapter 6.
- (2) Convert the IDL (Figure 7-1)
 - Convert the IDL for common components to the IDL for each component technology by using the IDL conversion compiler of this system (framework).
 - As described in Handling of Existing Technologies and the Common Desktop Platform, and Message Communication Specifications of Chapter 4, each existing component technology has a section that cannot be mapped by an abstracted IDL. Therefore, extension is required for that section.
- (3) Compile the IDL (Figure 7-1)
 - Create an interface for common components by using the IDL compiler of each component technology.
- (4) Implement specific components (Figure 7-1)
 - Implement specific components by Inheriting the created interface for common components.

7.2 Common Component Creation Function

The common component creation process flow in a sample case when UNO specific component is the corresponding component of the specified common component is shown in Figure 7-2 to Figure 7-8. According to this, the data types that are used and stored for communication, setting, and reference should be written. Note that higher-level technologies of each component technology will not be described hereinafter. Also note that the numbers (1) to (10) and ① and ② in the figures correspond to the numbers in the description.

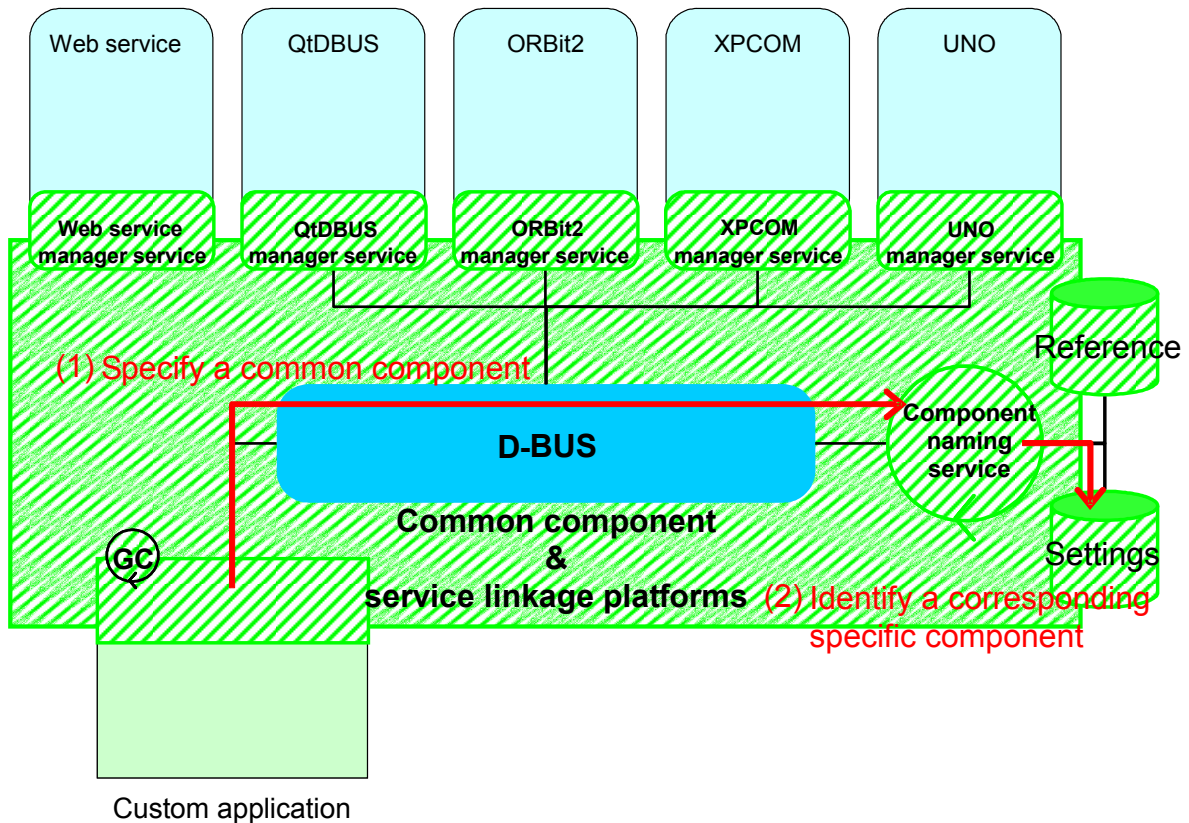


Figure 7-2 Common Component Creation 1

- (1) Specify a common component (Figure 7-2)
 - A custom application requests the component naming service (:component_namingservice) to create a common component (e.g.: Spreadsheet) by using the application helper library.
- (2) Identify a corresponding specific component (Figure 7-2)
 - The component naming service (:component_namingservice) loads the settings for the specific component that corresponds to the common component that has been requested for creation (e.g. spreadsheet) to identify the specific component to be created.
 - The settings for the specific component that corresponds to the common component shall be in XML format. The format shall be as follows:
 - common component element: Setting entry for common components
 - name attribute: The name of the common component
 - mgrservice attribute: The name of the manager service that requests creation of a specific component (manages the life cycle).
 - component attribute: The name of the specific component

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

Figure 7-3 Setting Example of Common Component

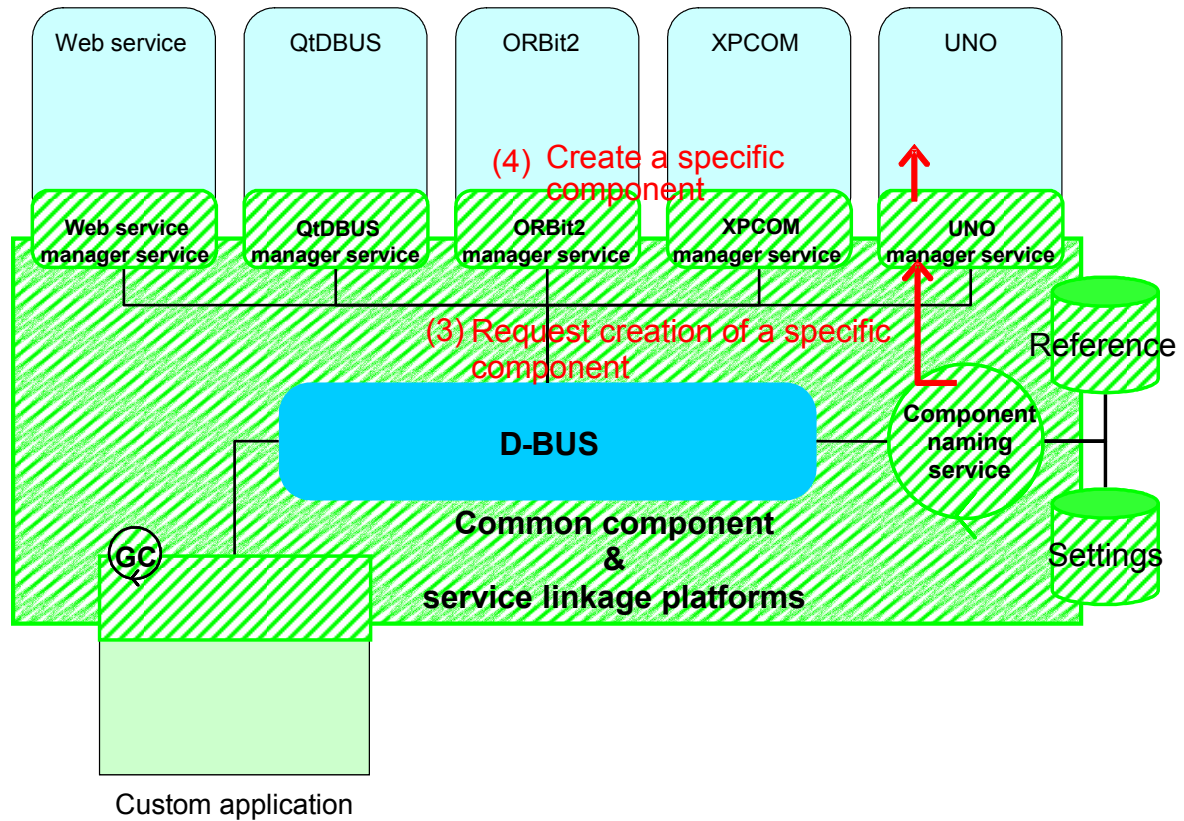


Figure 7-4 Common Component Creation 2

- (3) Request creation of a specific component (Figure 7-4)
 - The component naming service requests each manager service that has been loaded from the settings (e.g. for UNO, uno_mgr) to create a specific component.
 - A specific component name should be specified when creation is requested.
- (4) Create a specific component (Figure 7-4)
 - Create a specific component according to the mechanism of the component technology for which each manager service is responsible.
 - For example, when UNO is the component technology, a specific component is created according to the following procedures:
 - ① Activate OpenOffice.org in the listening mode that enables external access (Figure 7-5).
 - ② Request OpenOffice.org to create a requested specific component (Figure 7-5).

Technical Specifications

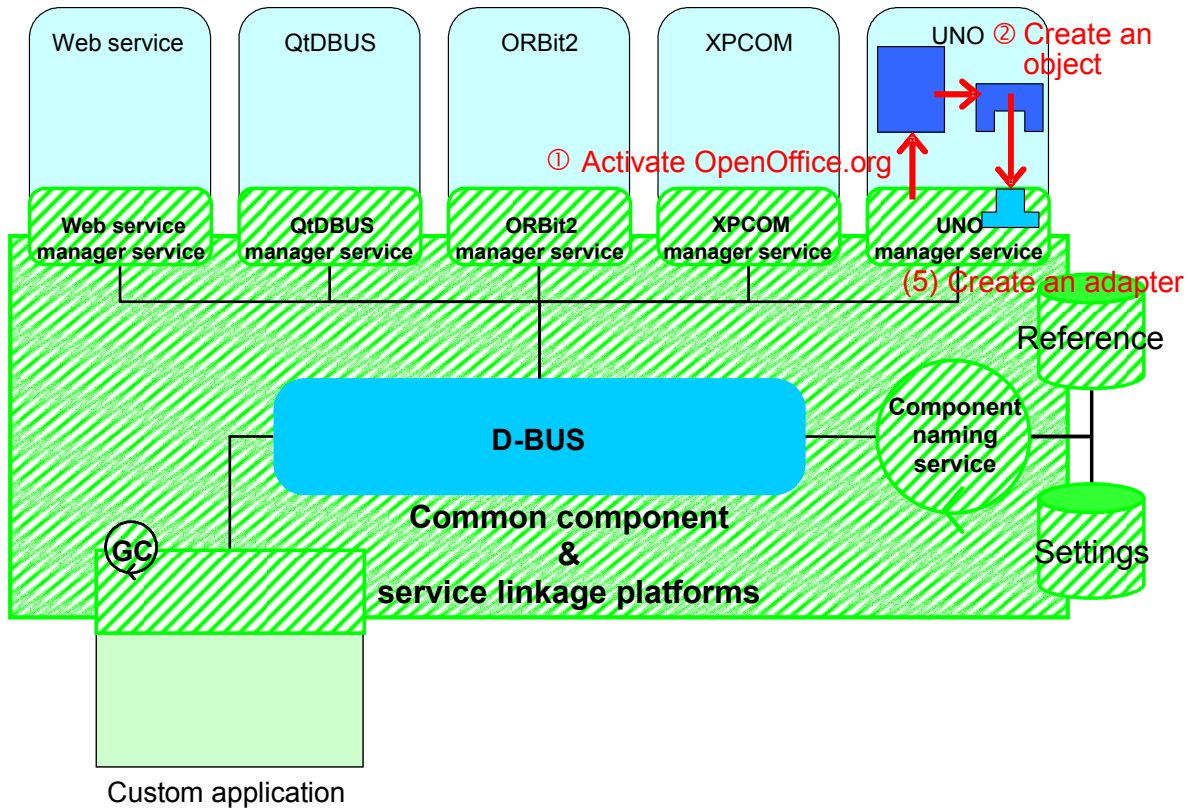


Figure 7-5 Common Component Creation 3

(5) Create an adapter (Figure 7-5)

- Create an adapter that converts a D-BUS message for the created specific component to the communication method of each component technology.

Technical Specifications

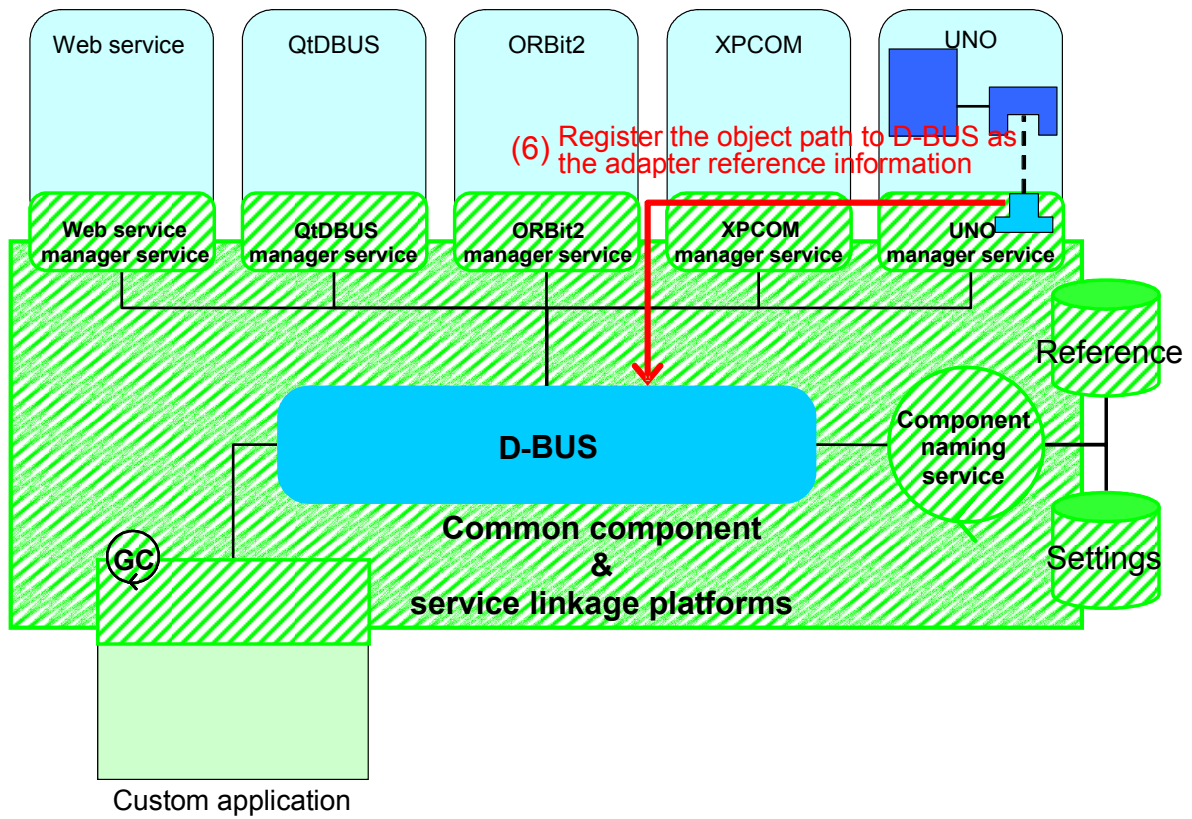


Figure 7-6 Common Component Creation 4

(6) Register the object path of the adapter to D-BUS (Figure 7-6)

- Register the object path of the adapter as an object on each manager service.
 - `/[Application Name]/[Class ID]/[Object ID]`
 - Example:
`/OpenOffice.org/uno:socket,host=localhost,port=2002;urp;StarOffice.ServiceManager/1`
 - Example: `/Mozilla/@mozilla.org/file/directory_service;1/2`

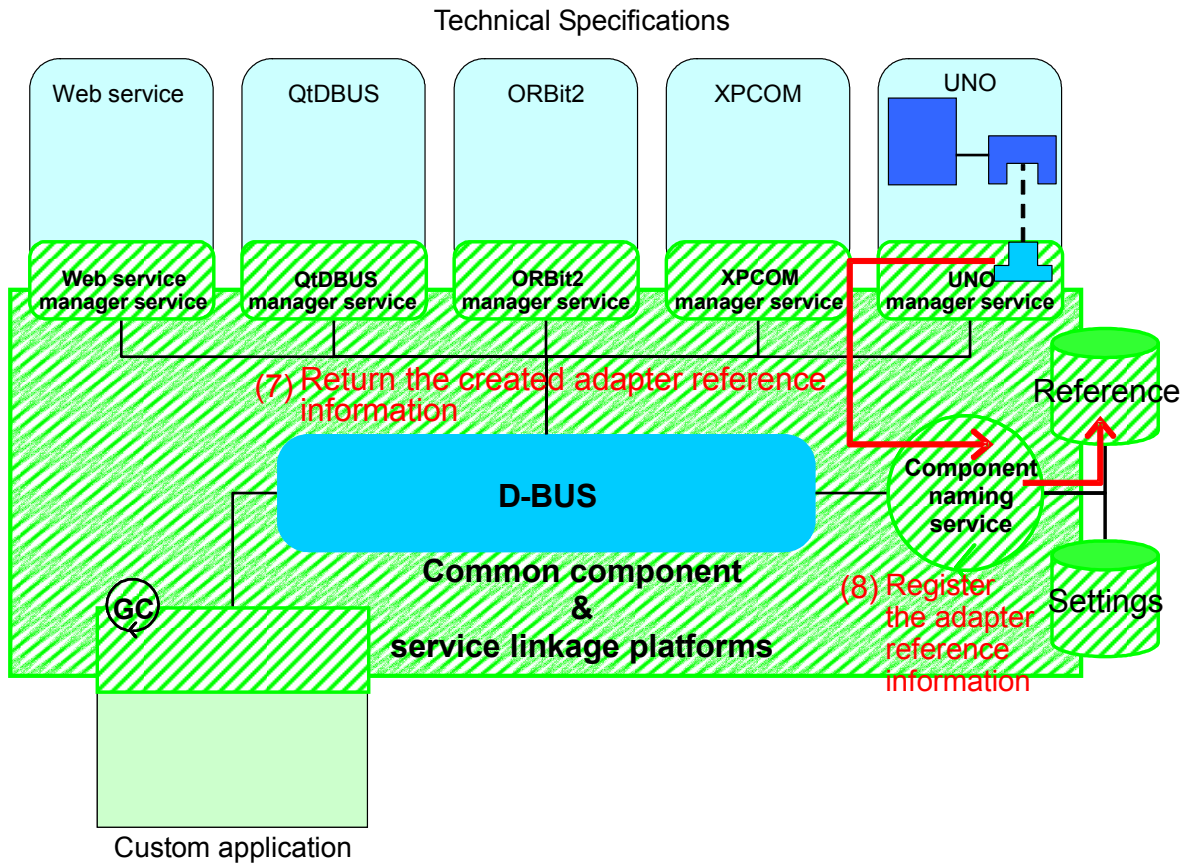


Figure 7-7 Common Component Creation 5

- (7) Return the created adapter reference information (Figure 7-7)
- Return the reference information of the adapter that has been created by each manager service to the component naming service.
 - The reference information is returned by the METHOD_RETURN message of D-BUS.
- (8) Register the adapter reference information (Figure 7-7)
- Register the obtained adapter reference information to the reference information possessed by the component naming service.
 - The items of the reference information to be registered shall be as follows: The format shall be determined after a prototype is verified for usability and retrieval performance because the retrieval performance and operation performance of web services etc. must be considered.
 - Component technology name (example: UNO)
 - Created manager service name (example: uno_mgr)
 - Object path (example: /Calc/com.sun.star.sheet.Spreadsheet/sheet1)
 - Common component name (example: spreadsheet)
 - Inheritance relationship of common component
 - Signature of the method (including synchronous and asynchronous information)

Technical Specifications

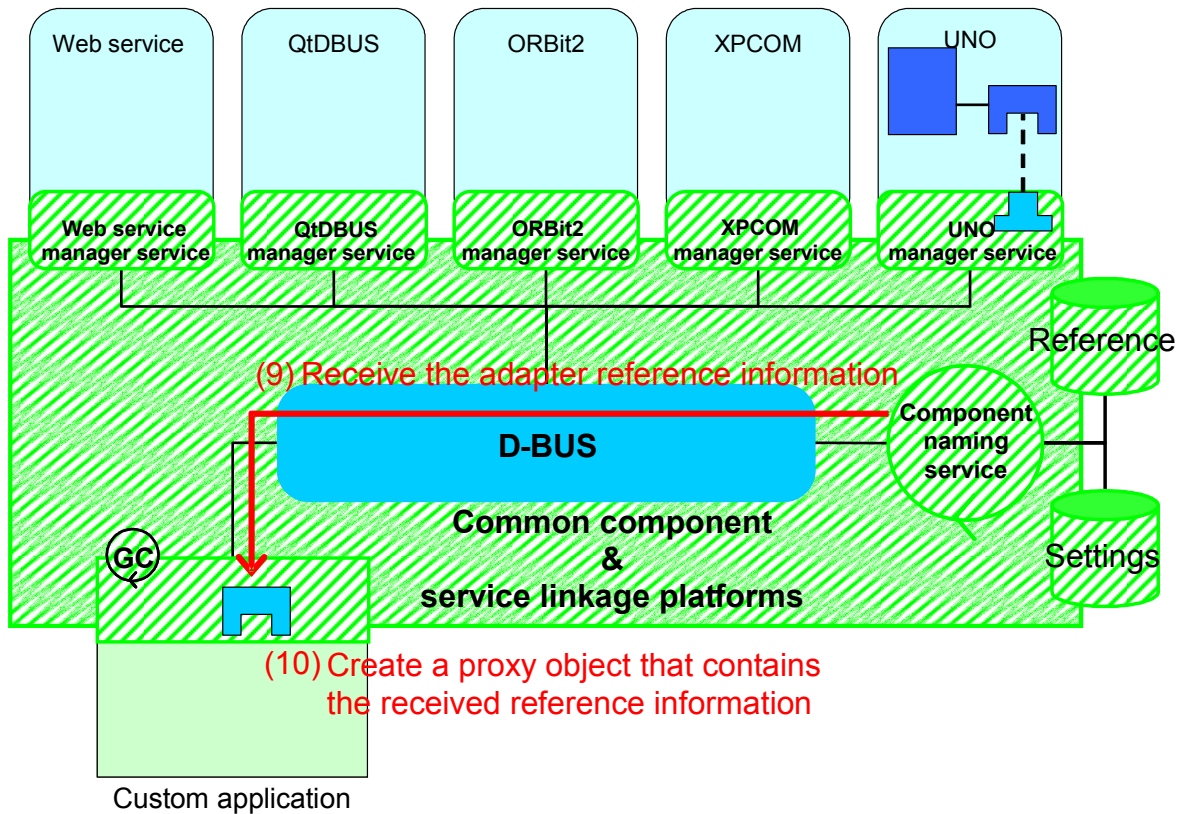


Figure 7-8 Common Component Creation 6

- (9) Receive the reference information of the created adapter (Figure 7-8)
- Transfer the reference information of the adapter that has been registered from the component naming service to the application helper library.
 - The reference information is transferred with the METHOD_RETURN message of D-BUS.
 - The reference information to be transferred shall be as follows:
 - Created manager service name (example: uno_mgr)
 - Object path (example: /Calc/com.sun.star.sheet.Spreadsheet/sheet1)
 - Common component name (example: spreadsheet)
 - Inheritance relationship of common component
 - Signature of the method (including synchronous and asynchronous information)
- (10) Based on the received reference information, create a proxy object (Figure 7-8)
- The application helper library creates a proxy object that contains the received reference information.

7.2.1 Use-case of Creating a Common Component

[1] Mail box

The following shows the use-case in which a custom application creates a common component for a mail client to open a mail box. In this description, Thunderbird is used as a sample specific component for the mail client.

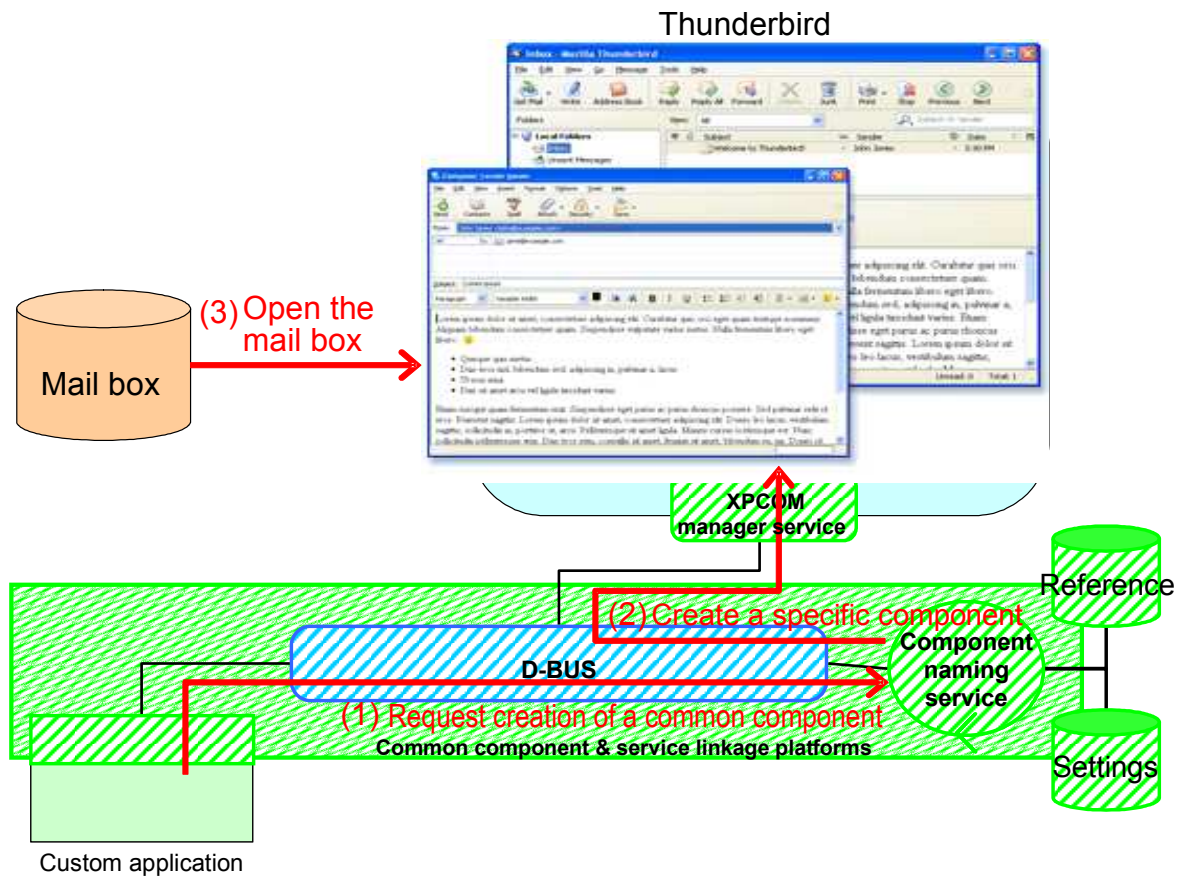


Figure 7-9 Use-case of Creating a Common Component (Mail Box)

- (1) A custom application requests the component naming service to create a common component for a mail client.
- (2) The component naming service creates a specific component, Thunderbird, via XPCOM manager service.
- (3) Thunderbird opens the mail box of a user according to settings that have been made in advance.

Technical Specifications

[2] Rendering (HTML/XML+CSS)

The following shows a use-case in which a custom application creates a common component that renders HTML/XML+CSS files to perform rendering of HTML/XML+CSS files. In this description, FireFox is used as a sample specific component.

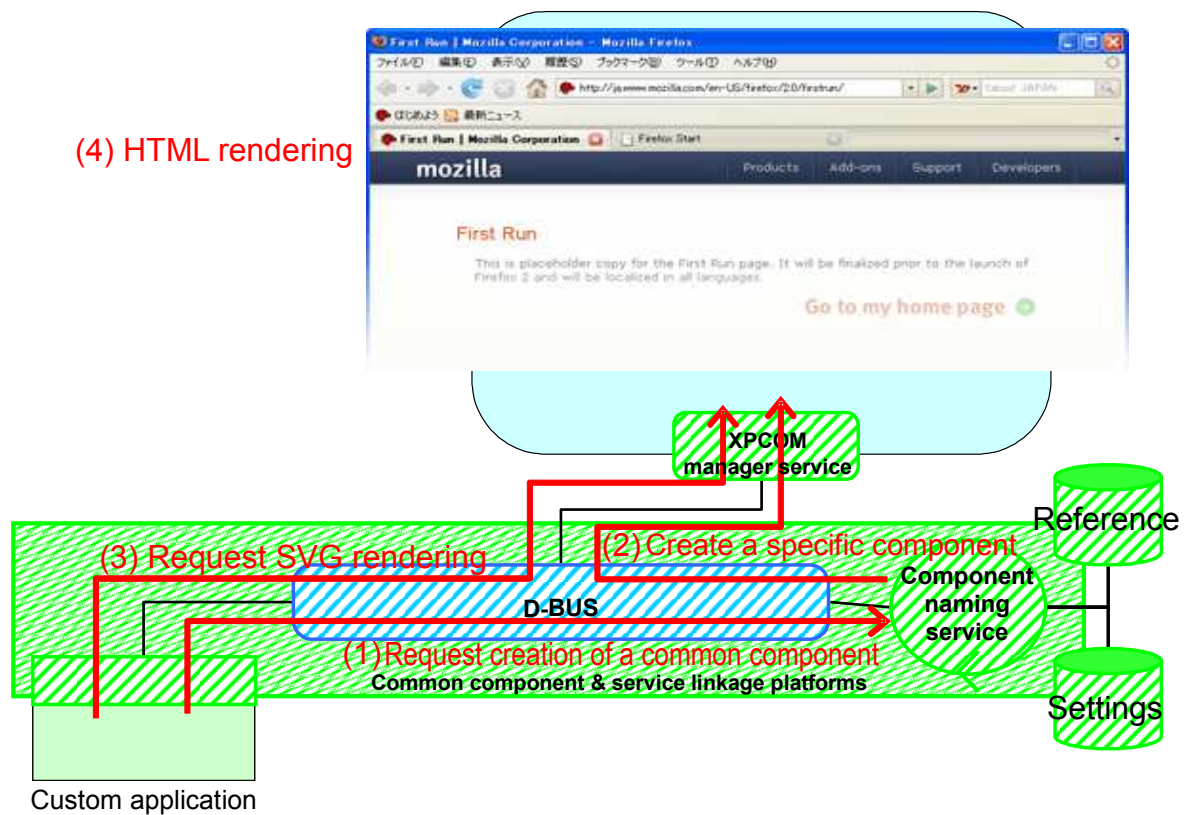


Figure 7-10 Use-case of Creating a Common Component (Rendering (HTML/XML+CSS))

- (1) A custom application requests the component naming service to create a common component that renders HTML and XML+CSS files.
- (2) The component naming service creates a specific component, FireFox, via XPCOM manager service.
- (3) A custom application transfers HTML data to the common component and requests HTML rendering.
- (4) FireFox renders the HTML data.

Technical Specifications

[3] Rendering (SVG)

The following shows a use-case in which a custom application creates a common component for SVG Renderer to render a SVG file. In this description, KSVG of KDE is used as a sample specific component for the SVG Renderer.

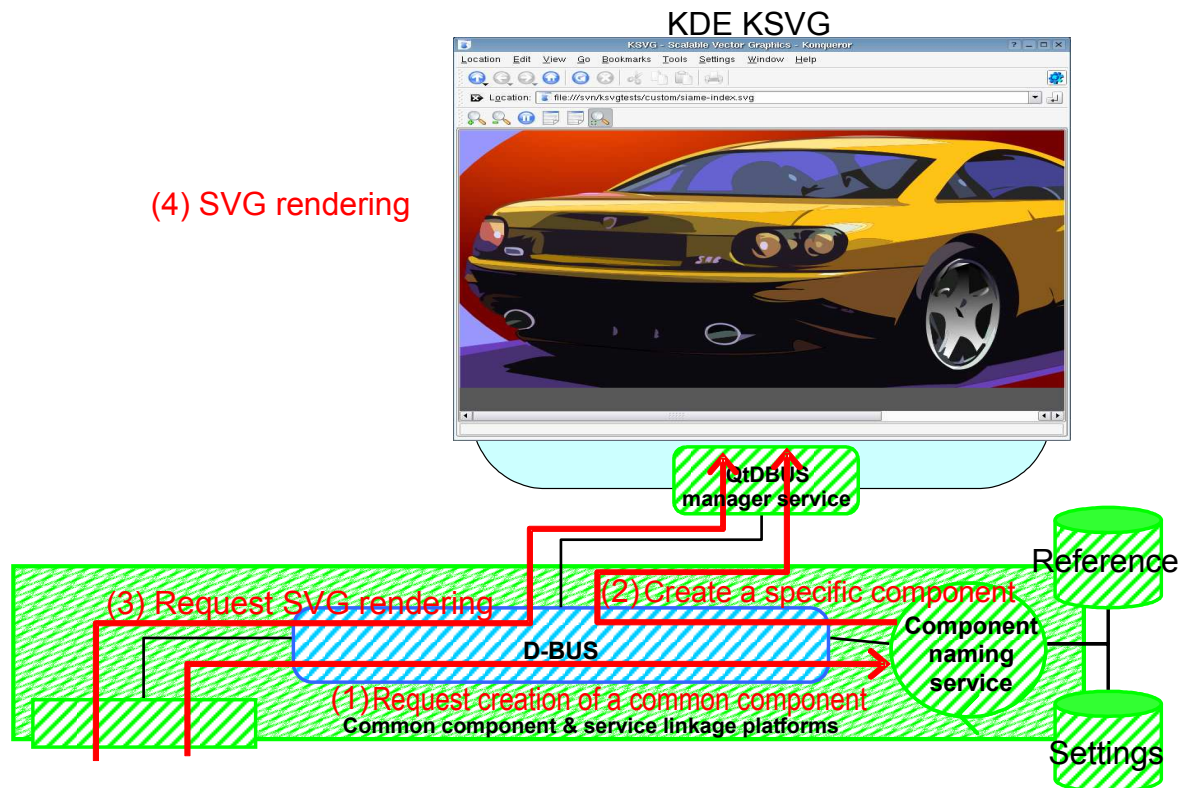


Figure 7-11 Use-case of Creating a Common Component (Rendering (SVG))

- (1) A custom application requests the component naming service to create a common component that renders SVG files.
- (2) The component naming service creates a specific component, KSVG of KDE, via QtDBus manager service.
- (3) A custom application transfers SVG data to the common component and requests SVG rendering.
- (4) KSVG renders the SVG data.

Technical Specifications

[4] Rendering (PDF)

The following shows a use-case in which a custom application creates a common component for PDF Renderer to render a PDF file. In this description, KPDF of KDE is used as a sample specific component for the PDF Renderer.

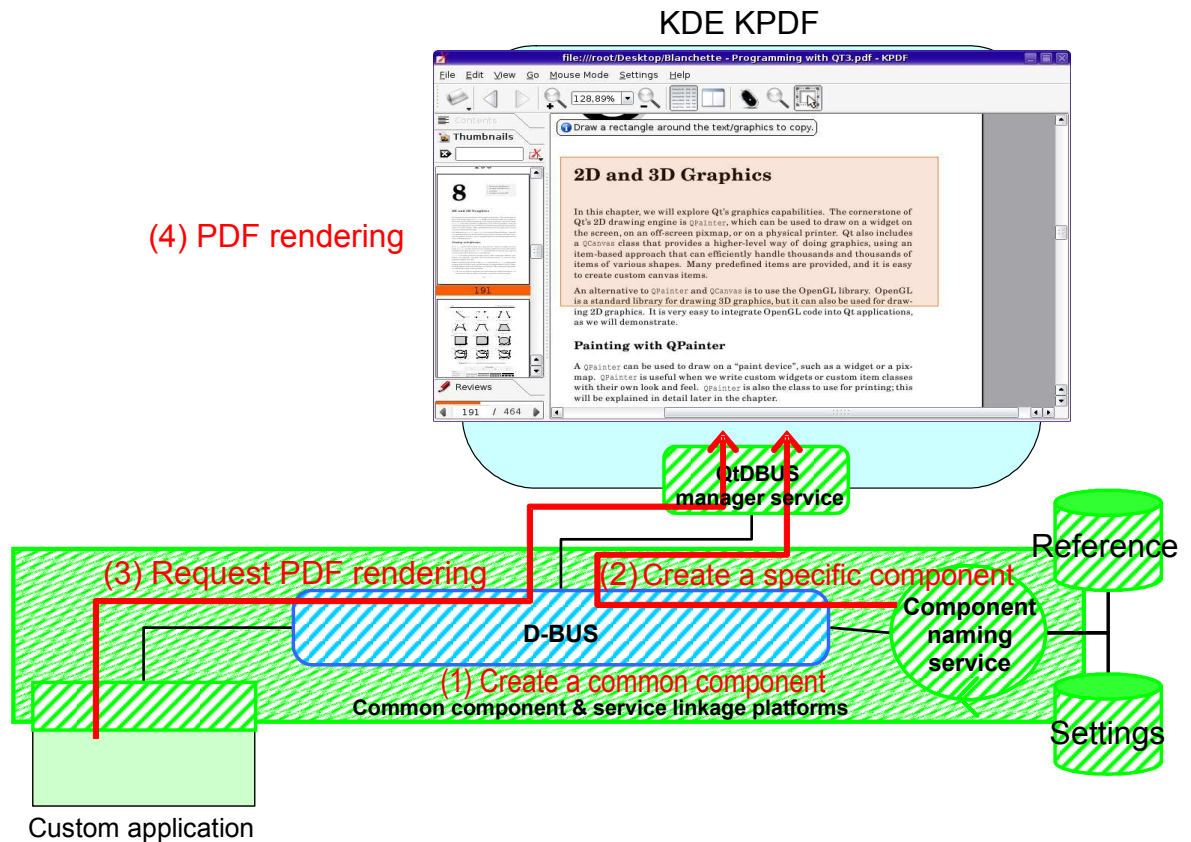


Figure 7-12 Use-case of Creating a Common Component (Rendering (PDF))

- (1) A custom application requests the component naming service to create a common component that renders PDF files.
- (2) The component naming service creates a specific component, KPDF of KDE, via QtDBus manager service.
- (3) A custom application transfers PDF data to the common component and requests PDF rendering.
- (4) KPDF renders the PDF data.

Technical Specifications

[5] Database

The following shows a use-case in which a custom application creates a common component for a database to render the database. In this description, Base of OpenOffice.org is used as a sample specific component for the database.

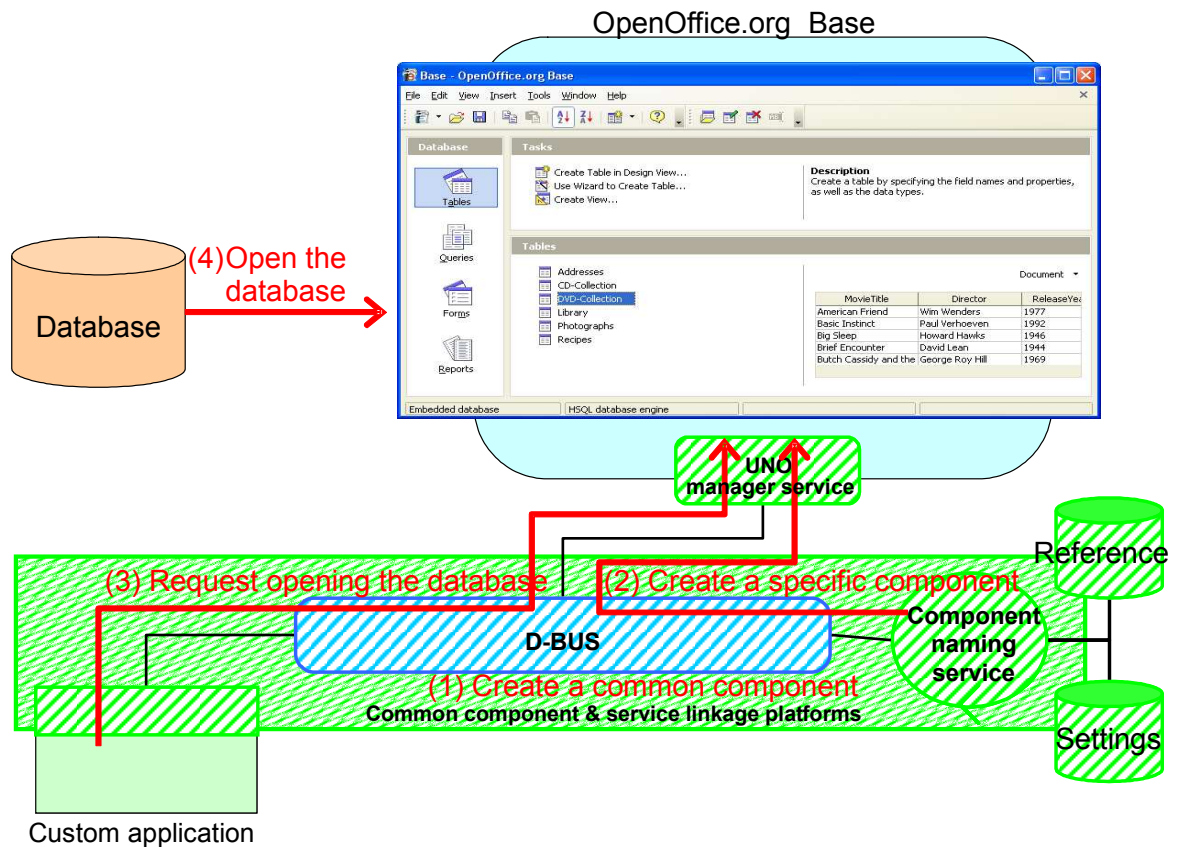


Figure 7-13 Use-case of Creating a Common Component (Database)

- (1) A custom application requests the component naming service to create a common component for a database.
- (2) The component naming service creates a specific component, OpenOffice.org Base, via UNO manager service.
- (3) A custom application requests the common component to open the specified database.
- (4) OpenOffice.org Base opens the specified database.

7.2.2 Sequence in the Case of QtDBus

The following shows the sequence when a specific component on QtDBus is created.

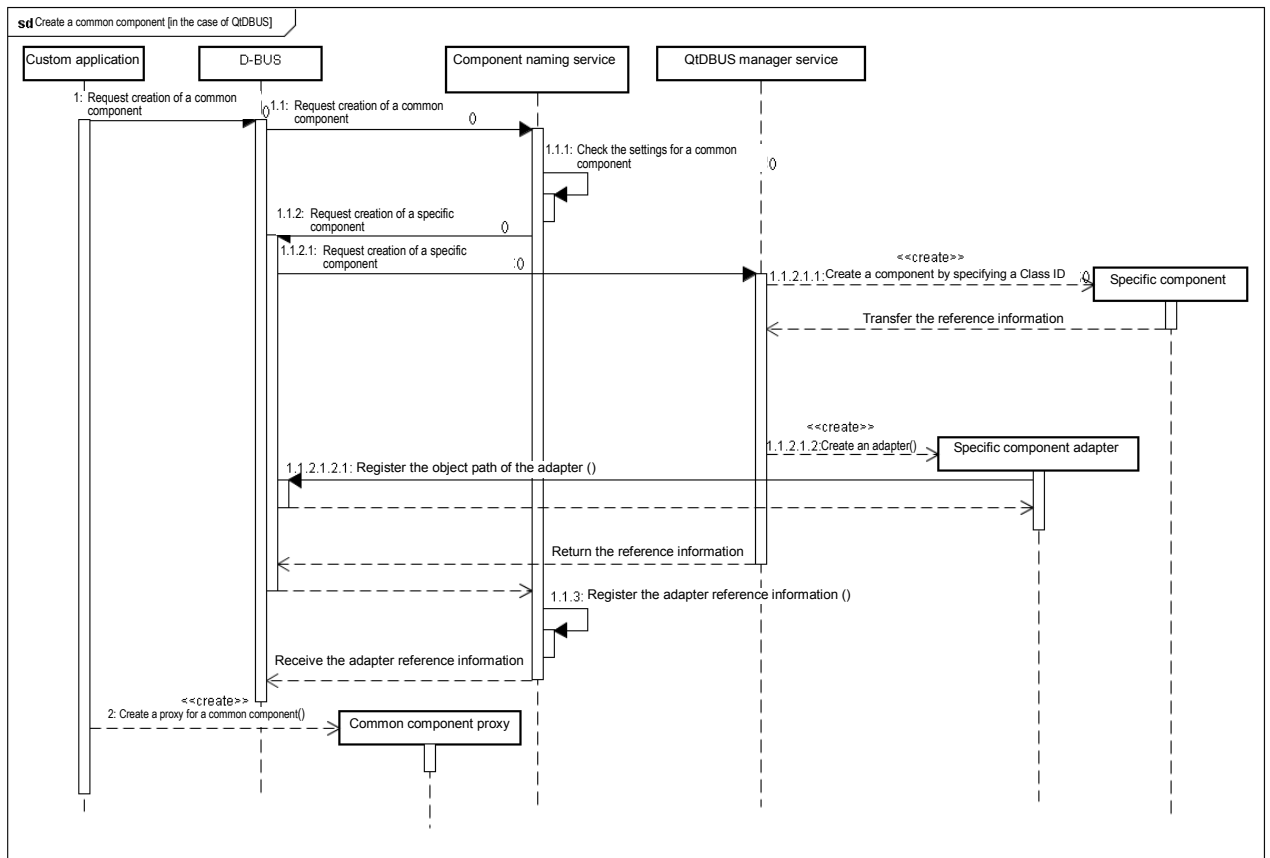


Figure 7-14 Creation Sequence of Common Component on QtDBus

7.2.3 Sequence in the Case of ORBit2

The following shows the sequence when a specific component on ORBit2 is created.

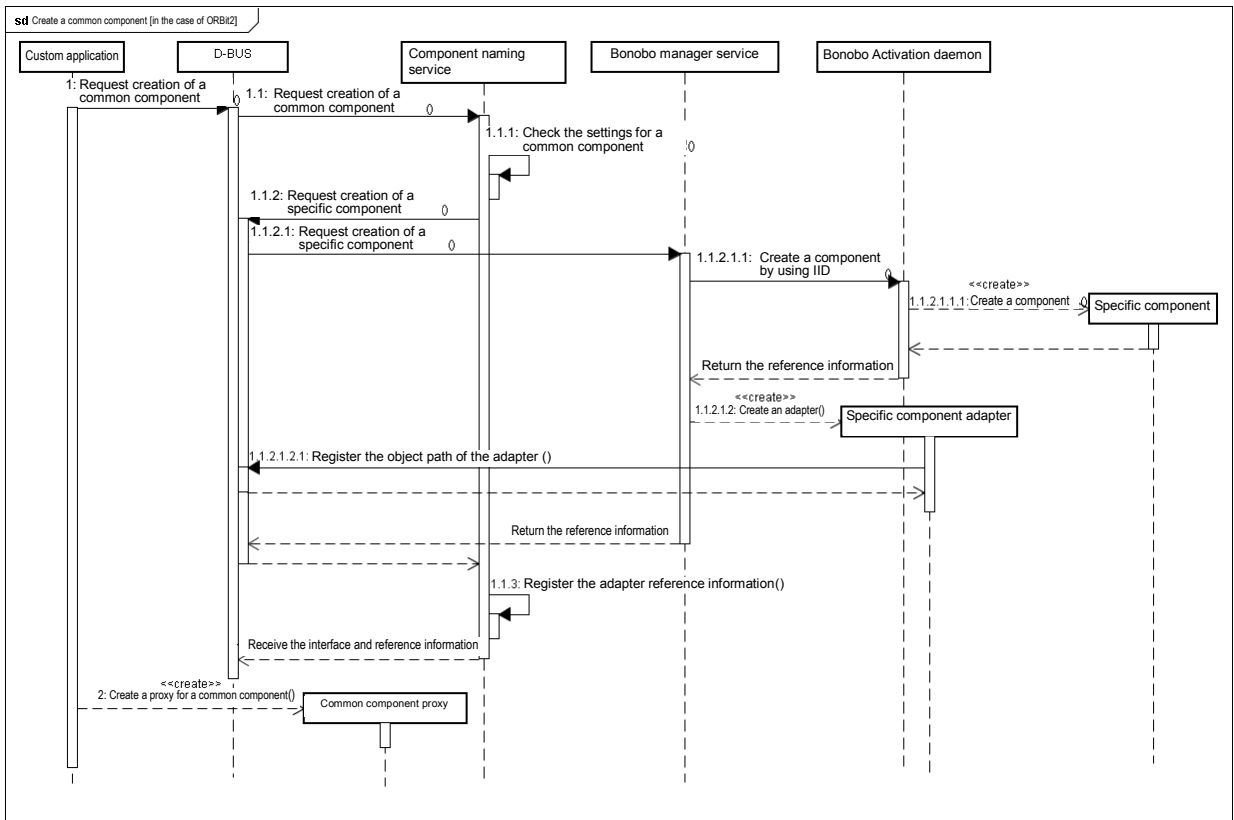


Figure 7-15 Creation Sequence of Common Component on ORBit2

7.2.4 Sequence in the Case of XPCOM

The following shows the sequence when a specific component on XPCOM is created.

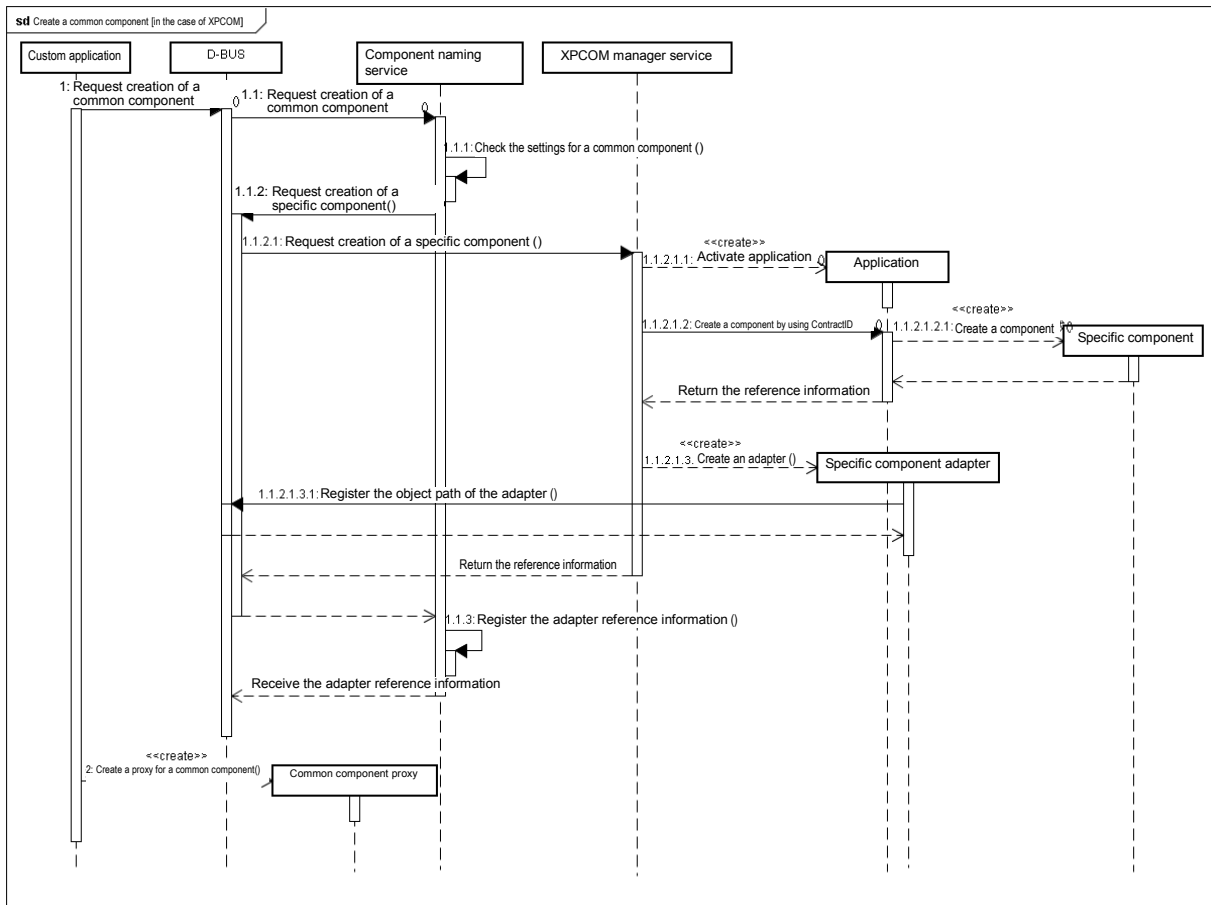


Figure 7-16 Creation Sequence of Common Component on XPCOM

7.2.5 Sequence in the Case of UNO

The following shows the sequence when a specific component on UNO is created.

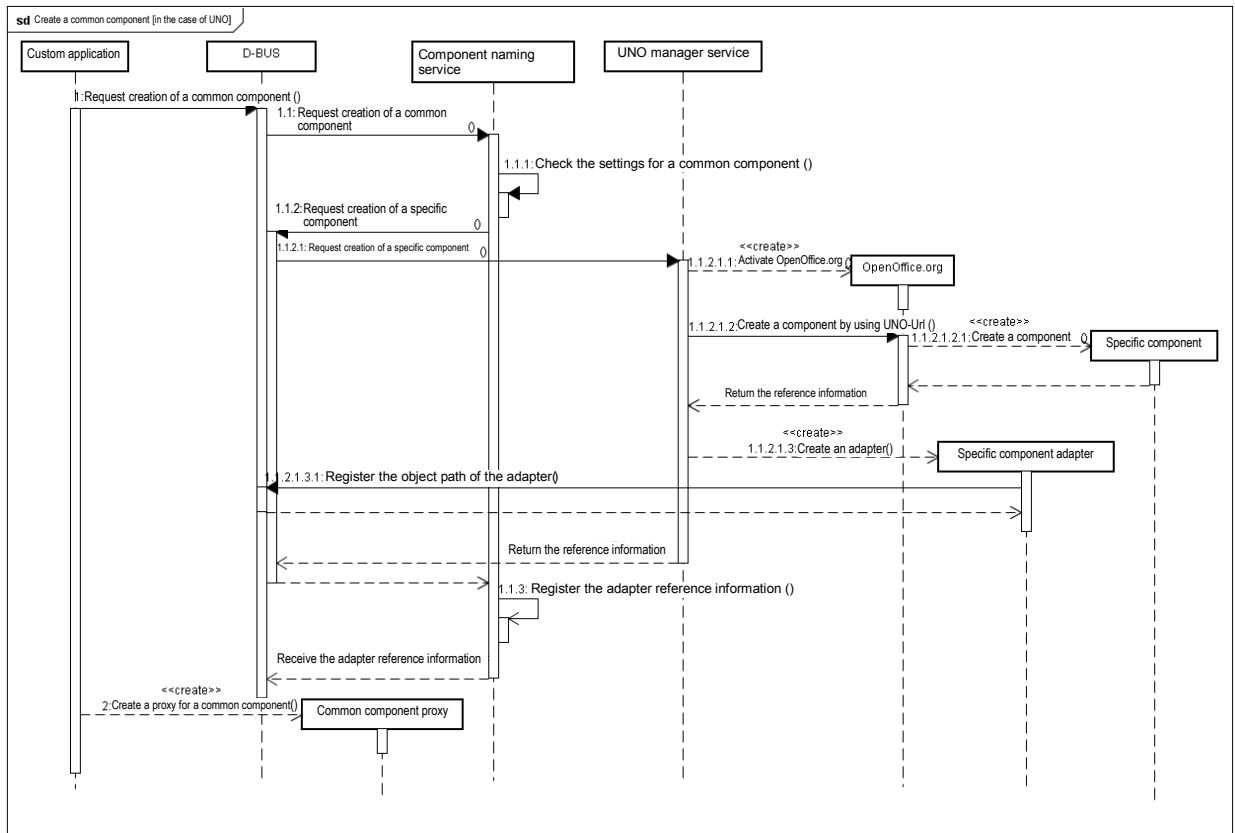


Figure 7-17 Creation Sequence of Common Component on UNO

7.3 Common Component Registration Function

The registration flow of common component reference information is shown in Figure 7-18 to Figure 7-19. This process is performed when multiple specific components are created during system activation, such as KDE and GNOME.

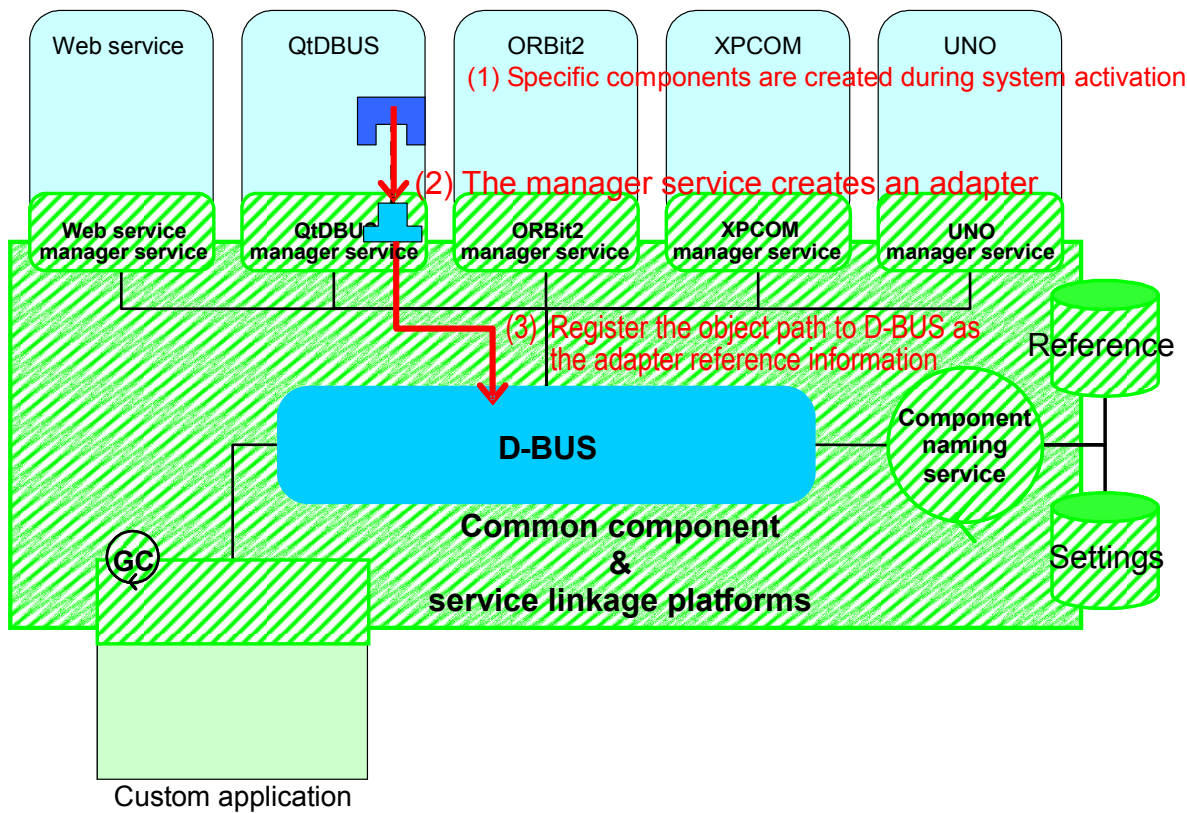


Figure 7-18 Common Component Registration 1

- (1) Specific components are created during system activation (Figure 7-1-18)
 - Specific components are created when a system (such as KDE or GNOME) is activated.
- (2) The manager service creates an adapter (Figure 7-1-18)
 - The created specific components request each manager service to create an adapter.
 - The function that requests manager services to create adapters when specific components are created must be added to KParts and Bonobo.
- (3) Register the object path of the adapter to D-BUS (Figure 7-1-18)
 - Register the object path of the adapter as an object on each manager service.

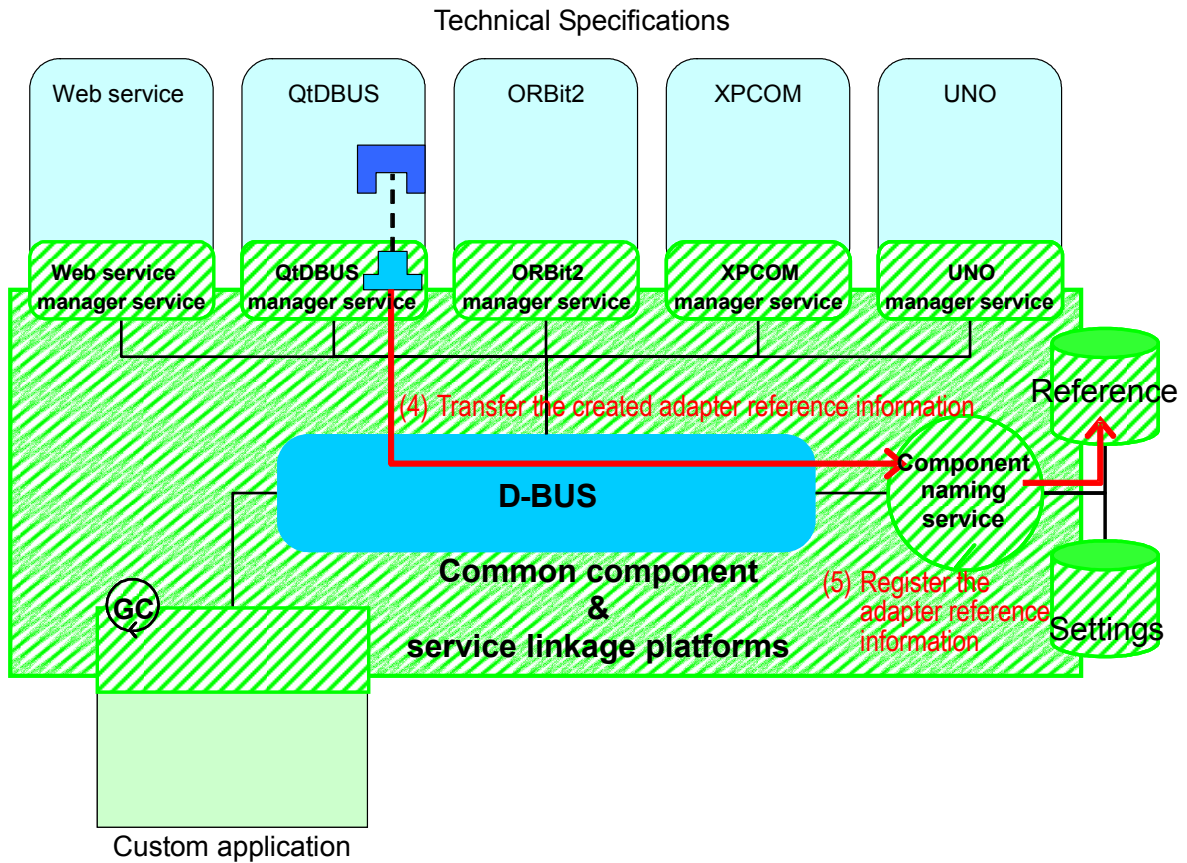


Figure 7-19 Common Component Registration 2

- (4) Transfer the created adapter reference information (Figure 7-1-19)
 - Transfer the reference information of the adapter that has been created by each manager service to the component naming service.
 - The reference information is returned by the METHOD_CALL message of D-BUS.
- (5) Register the adapter reference information (Figure 7-1-19)
 - Register the obtained adapter reference information to the reference information possessed by the component naming service.

7.3.1 Sequence

The sequence of common component reference information registration process is shown below:

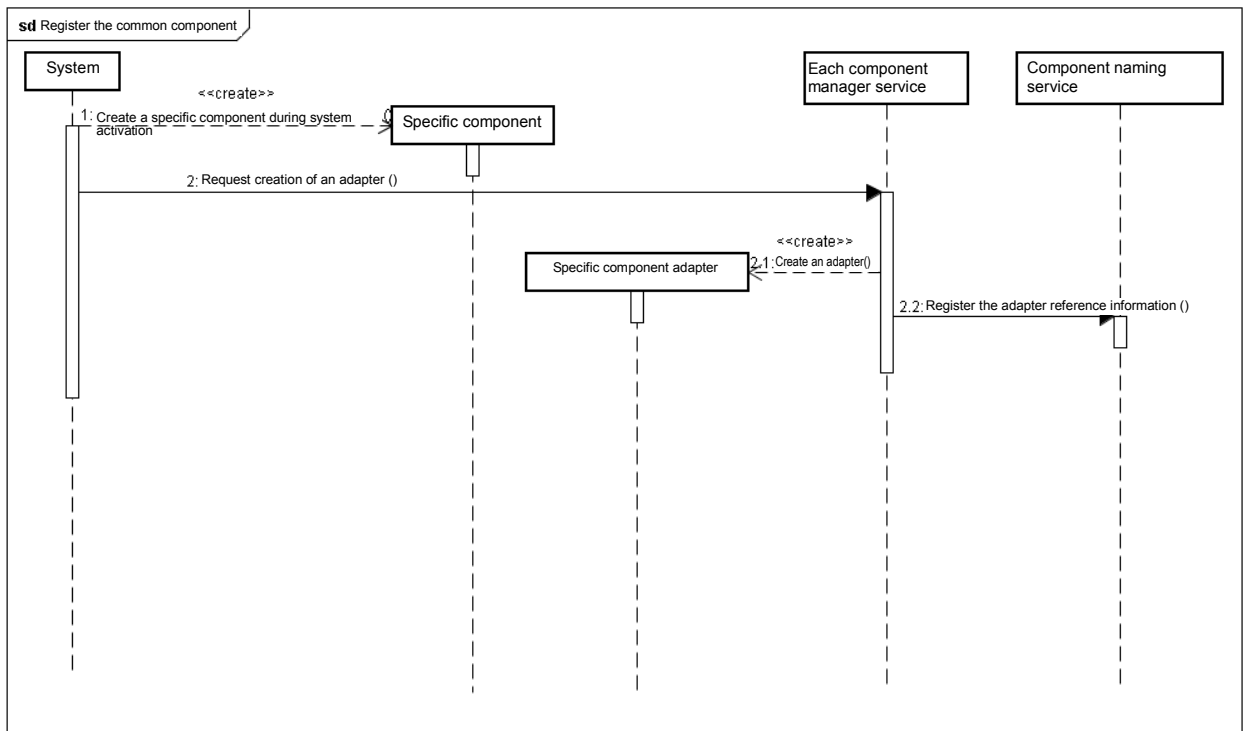


Figure 7-20 Common Component Reference Information Registration Sequence

7.4 Web Service Registration Function

The registration flow of web service interface information is shown in Figure 7-22 to Figure 7-23. The WSDL conversion processing shown in Figure 7-21 must be performed in advance. Also, the web service interface shall be registered in the component naming service when a web service manager service is activated.

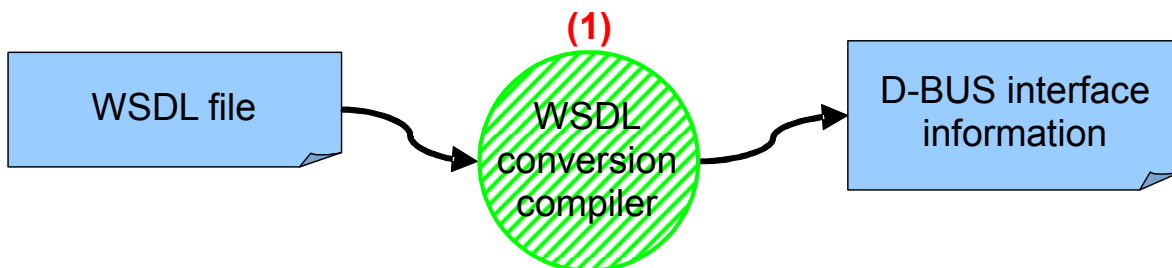


Figure 7-21 Creating D-BUS Interface from WSDL

- (1) Convert the WSDL (Figure 7-1-21)
 - Convert the WSDL file to be used for a web service to D-BUS interface information (introspection) that corresponds to the web service by the WSDL conversion compiler of this system (framework).
 - **As described in Chapter 4, make the WSDL file correspond to the introspection, the interface information of D-BUS.**

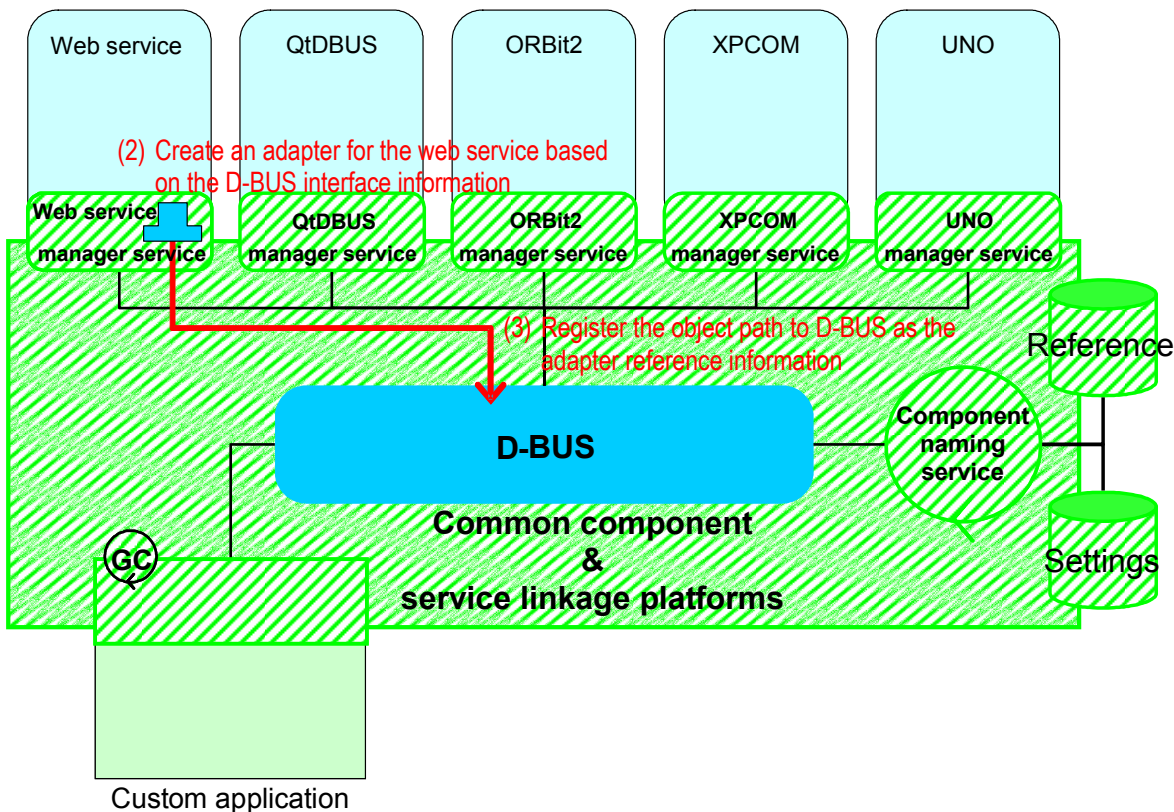


Figure 7-22 Web Service Interface Information Registration 1

- (2) Creates an adapter for a web service (Figure 7-1-22)
 - Create an adapter for the web service based on the D-BUS interface information that has been created in (1).

Technical Specifications

- (3) Register the object path of the adapter to D-BUS (Figure 7-1-22)
 - Register the object path of the adapter as an object on the web service manager service.
 - The object path shall be the URL of the web service.

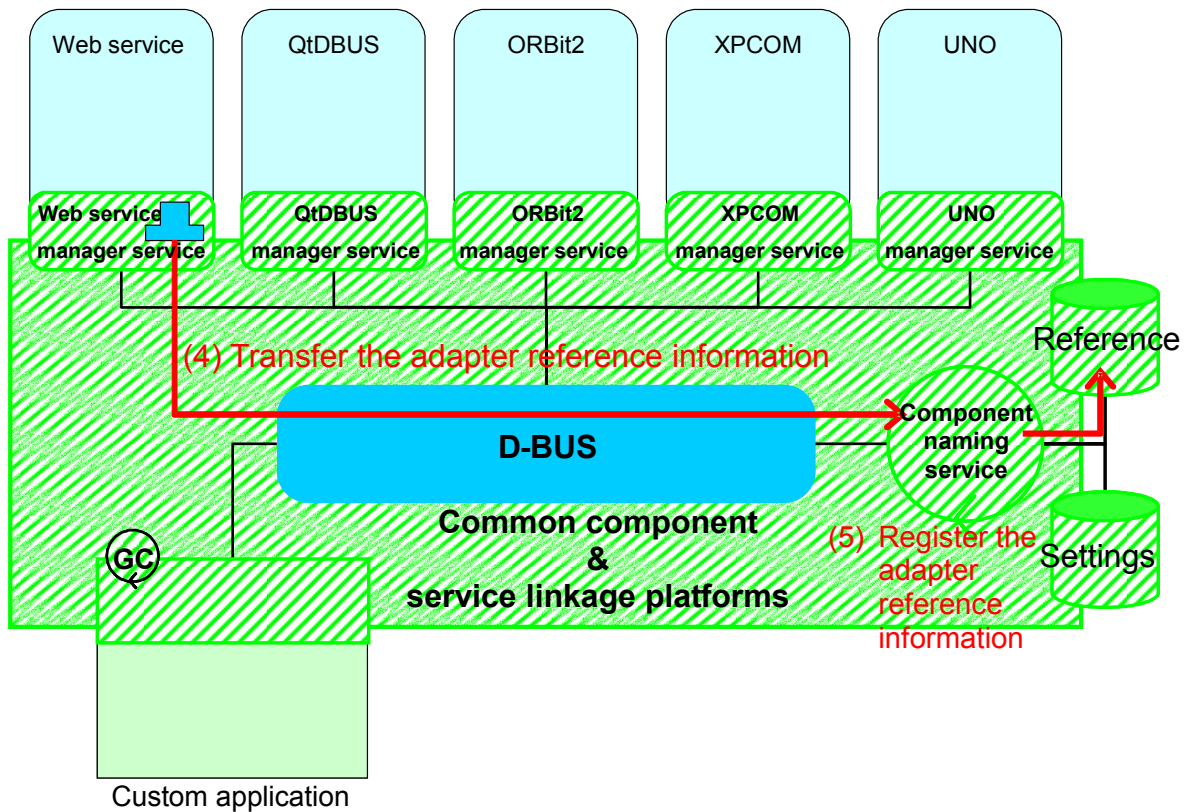


Figure 7-23 Web Service Interface Information Registration 2

- (4) Transfer the adapter reference information (Figure 7-1-23)
 - Register the reference information of each adapter to the component naming service.
- (5) Register the adapter reference information (Figure 7-1-23)
 - Register the reference information of each adapter to the reference information possessed by the component naming service.

7.5 Common Component Search Function

The common component search process flow in a sample case when UNO specific component is the corresponding component of the specified common component is shown in Figure 7-24 to Figure 7-26.

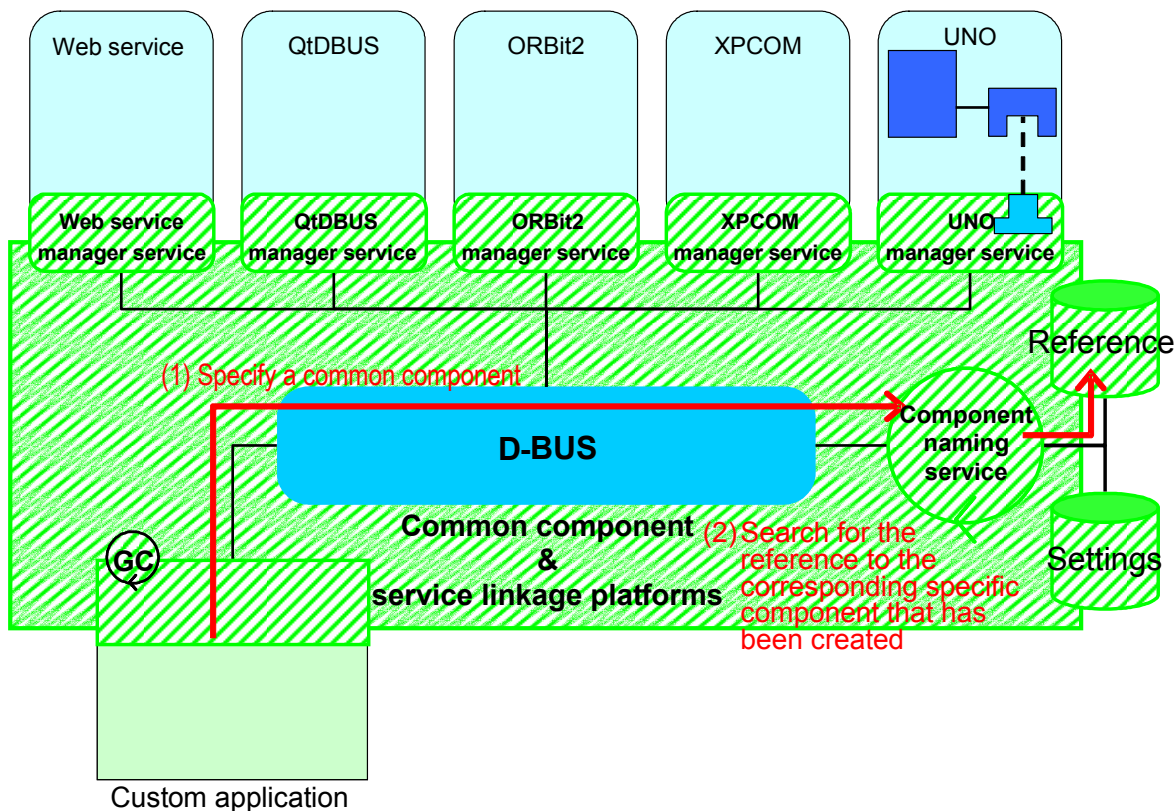


Figure 7-24 Common Component Search 1

- (1) Specify a common component (Figure 7-1-24)
 - A custom application requests the component naming service (:component_mgr) to search for a common component (e.g.: Spreadsheet) to be obtained by using the application helper library.
- (2) Search for the reference to the corresponding specific component that has been created (Figure 7-1-24)
 - The component naming service (:component_mgr) searches the reference information possessed by the service for the reference to the specific component that corresponds to the common component to be obtained (e.g. spreadsheet).

Technical Specifications

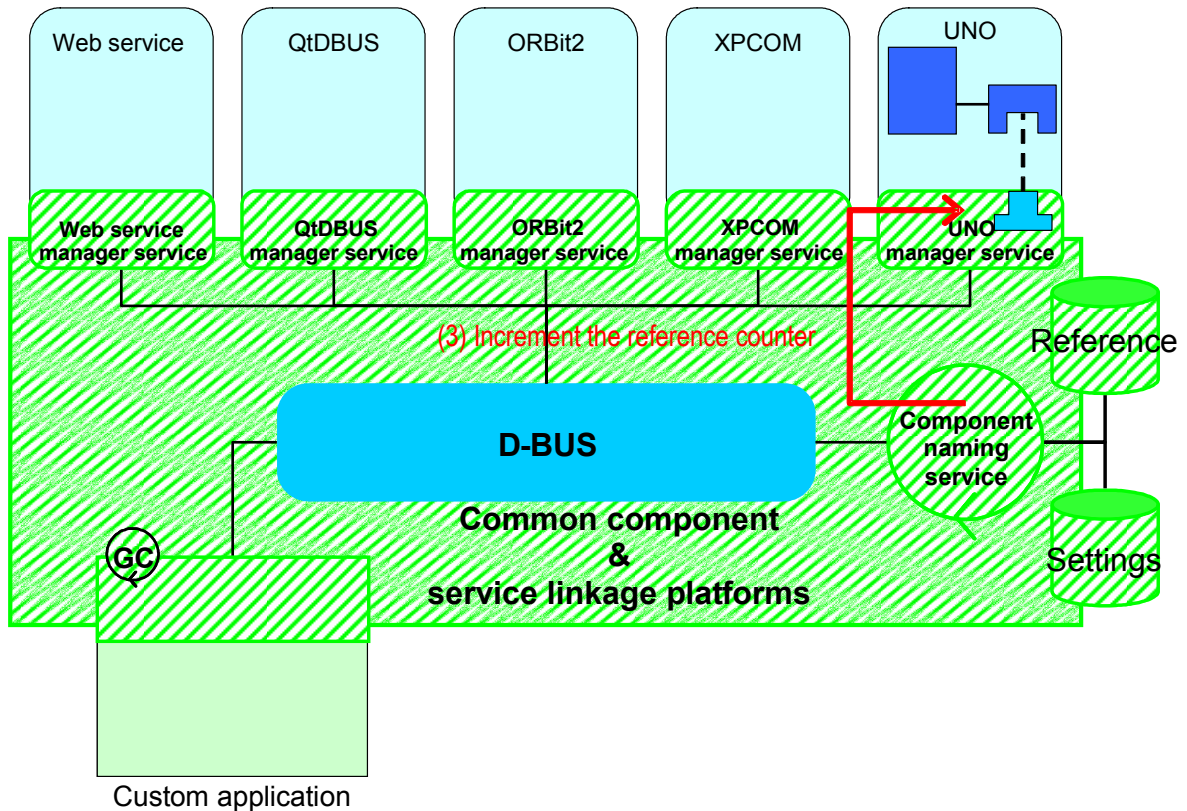


Figure 7-25 Common Component Search 2

- (3) Increment the reference counter (Figure 7-1-25)
- The component naming service requests each manager service to increment the reference counter of the specific component to be referred to or create a reference pointer.
 - **When the GC function is used in each component technology, increment of the reference counter that is used as a reference for executing the GC function or creation of the reference pointer is required.**
 - **It is preferable but difficult to use the GC function (and synchronize the GC function) in the common component platform, which should be determined after a prototype is created and verified.**

Technical Specifications

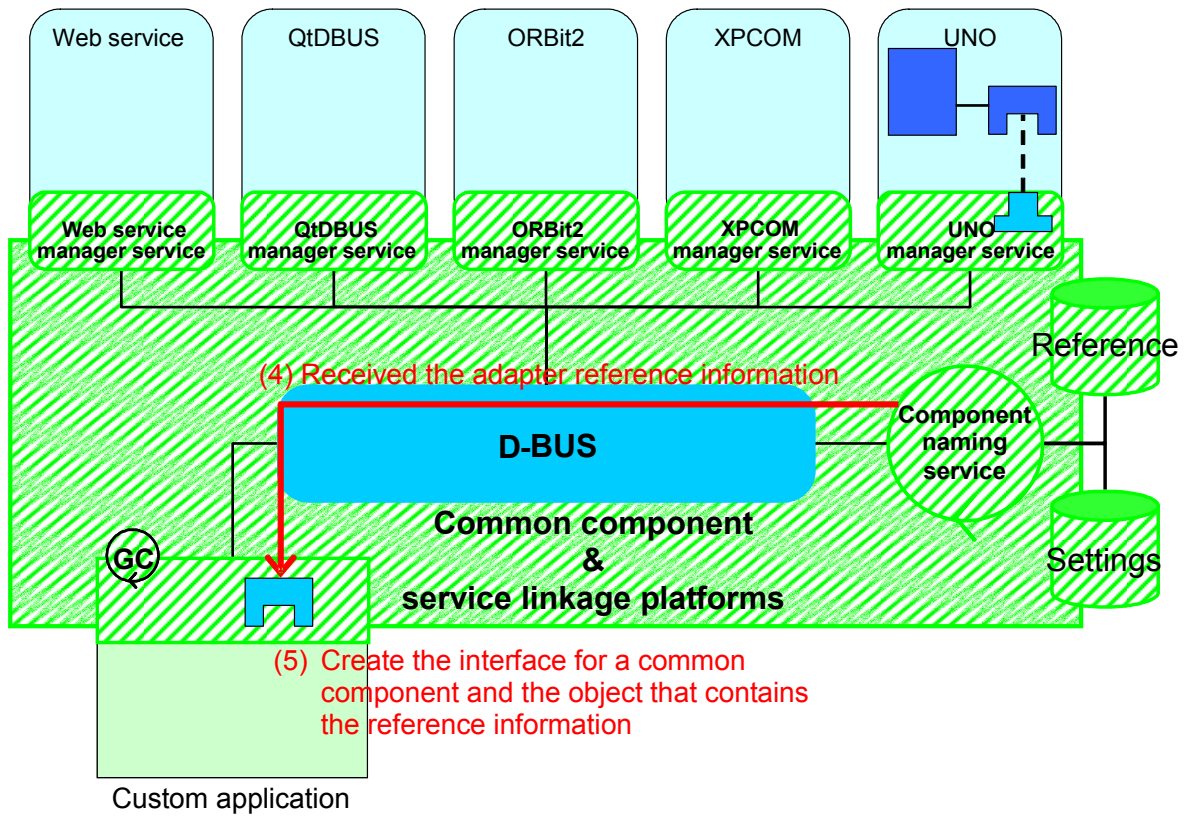


Figure 7-26 Common Component Search 3

- (4) Receive the adapter reference information (Figure 7-1-26)
 - Transfer the reference information of the adapter that has been found by the search to the application helper library.
 - The reference information is transferred with the METHOD_RETURN message of D-BUS.
 - The reference information to be transferred shall be the same as the created one.
- (5) Based on the received reference information, create a proxy object (Figure 7-1-26)
 - The application helper library creates a proxy object that contains the received reference information.

7.5.1 Sequence

The sequence of the common component search process is shown below:

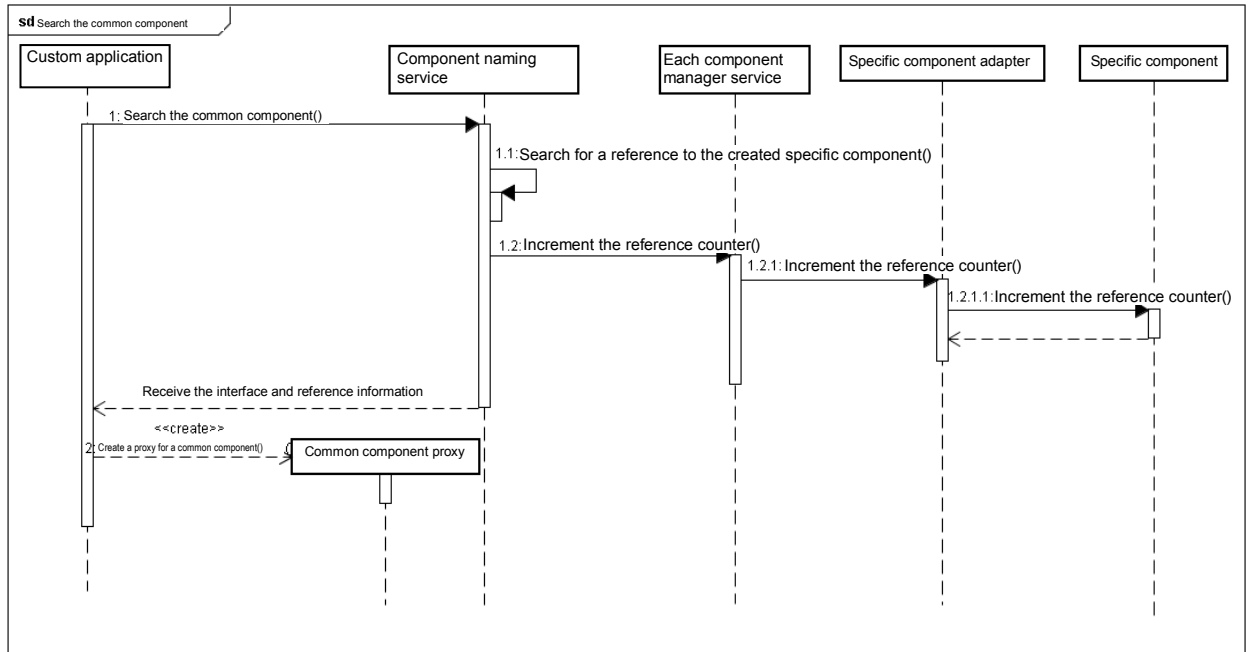


Figure 7-27 Common Component Search Sequence

7.6 Common Component Reference Function

The common component reference process (method call) flow in a sample case when UNO specific component is the corresponding component of the specified common component as shown in Figure 7-28 to Figure 7-33. According to it, the data types and settings to be communicated should be written.

7.6.1 Synchronous Communication Processing

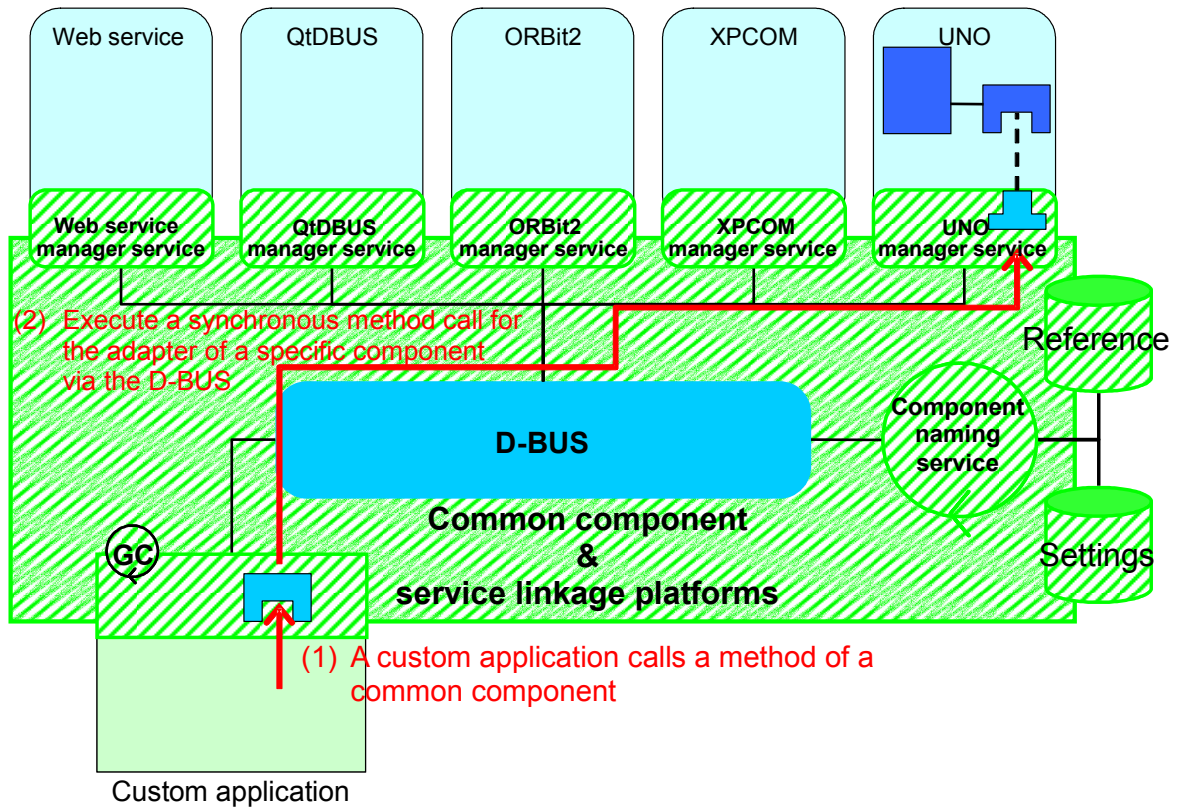


Figure 7-28 Synchronous Communication Processing 1

- (1) A custom application calls a method of a common component (Figure 7-1-28)
 - A custom application executes a synchronous method call for a proxy of a common component.
- (2) Communicate it to the adapter of a specific component via D-BUS (Figure 7-1-28)
 - The proxy of the specific component uses the application helper library to call a method of the specific component adapter via the D-BUS.

Technical Specifications

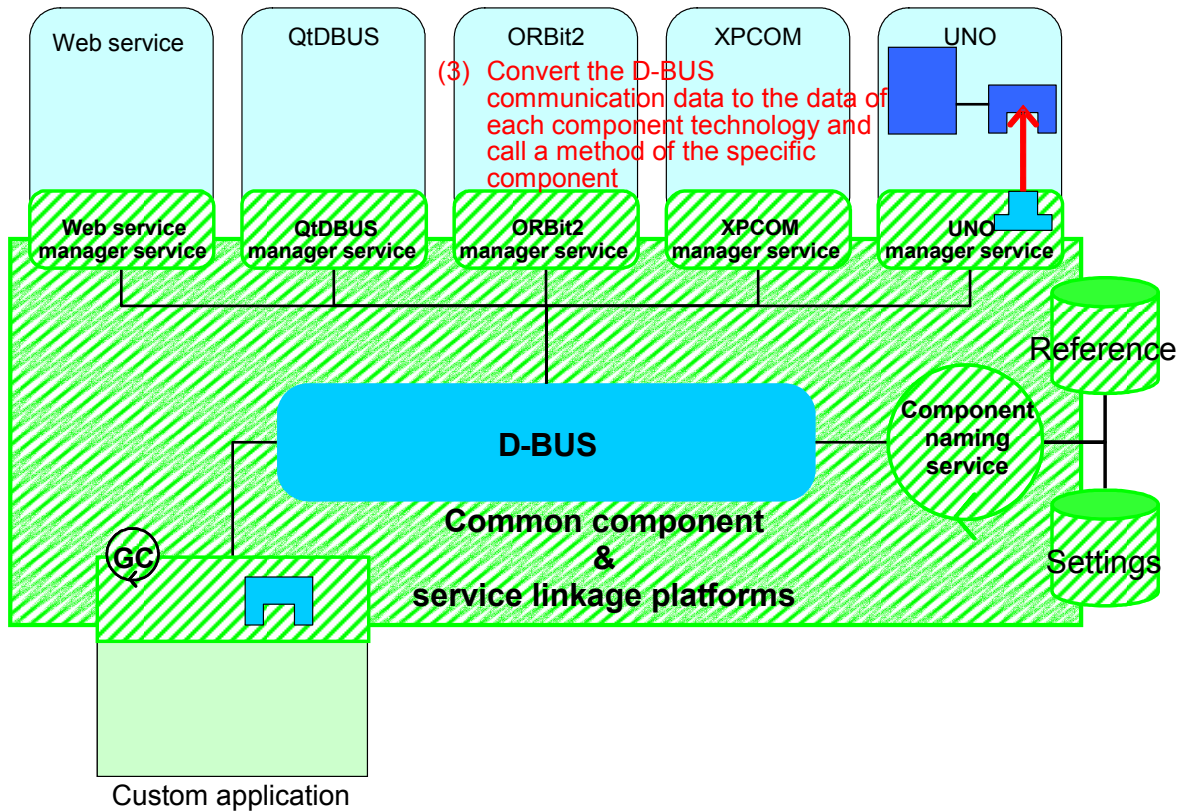


Figure 7-29 Synchronous Communication Processing 2

- (3) Call a method of a specific component (Figure 7-1-29)
- A manager service of each component technology converts the D-BUS communication data to the data of each component technology and calls a method of the specific component.
 - **As described in Chapter 4, conversion is possible because all D-BUS communication data types correspond to the data types of each component technology.**

Technical Specifications

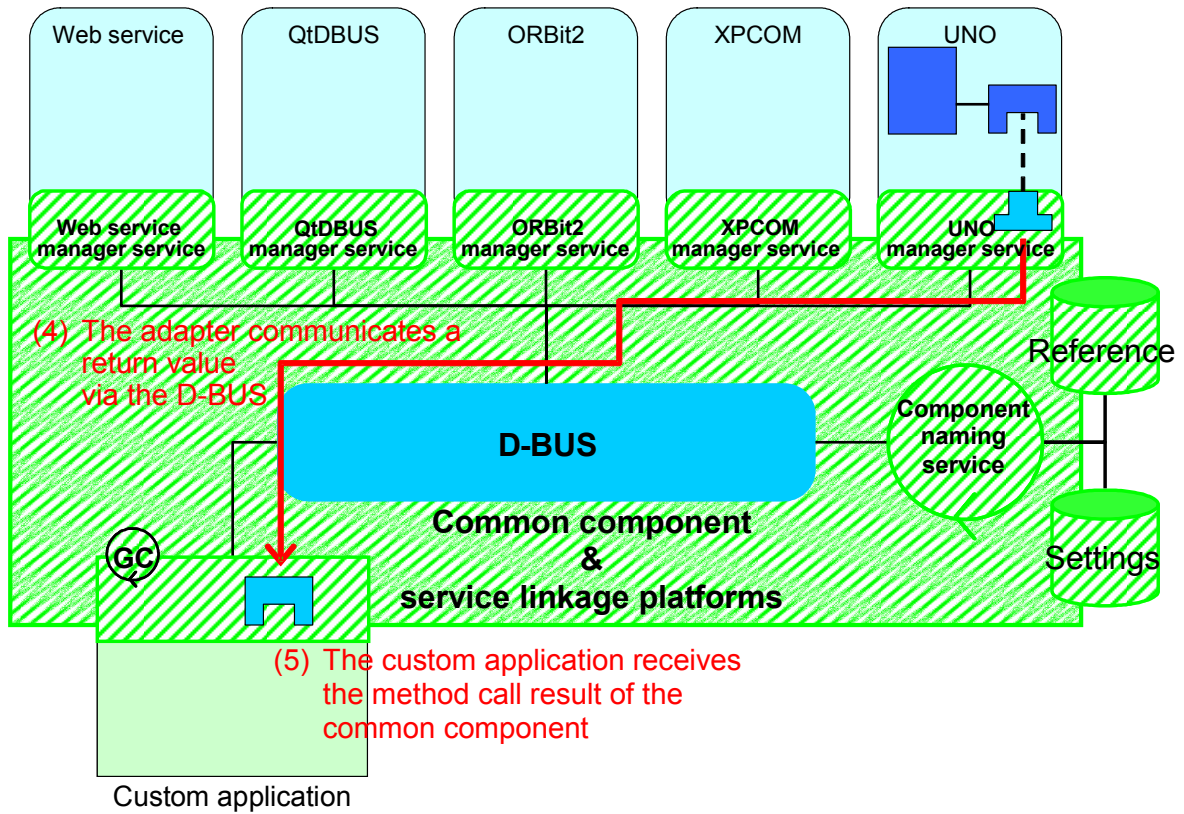


Figure 7-30 Synchronous Communication Processing 3

- (4) The adapter communicates a return value via the D-BUS (Figure 7-1-30)
 - A manager service of each component technology converts the data of each component technology to the D-BUS communication data and communicates the result with the D-BUS METHOD_RETURN message.
- (5) The custom application receives the result (Figure 7-1-30)
 - The custom application common component, proxy, receives the result via the D-BUS.

7.6.2 Asynchronous Communication Processing

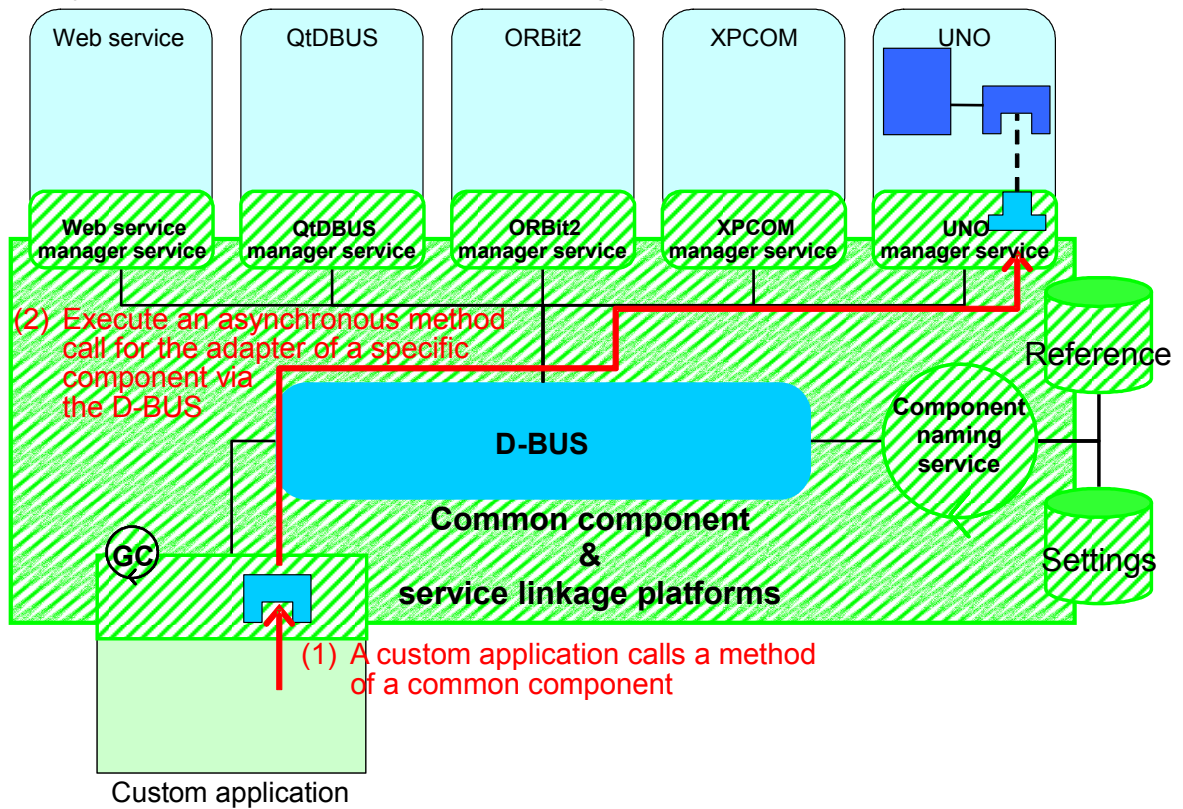


Figure 7-31 Asynchronous Communication Processing 1

- (1) A custom application calls a method of a common component (Figure 7-1-31)
 - A custom application executes an asynchronous method call for a proxy of a common component.
- (2) Communicate it to the adapter of a specific component via D-BUS (Figure 7-1-31)
 - The proxy of the specific component uses the application helper library to call a method of the specific component adapter via the D-BUS.

Technical Specifications

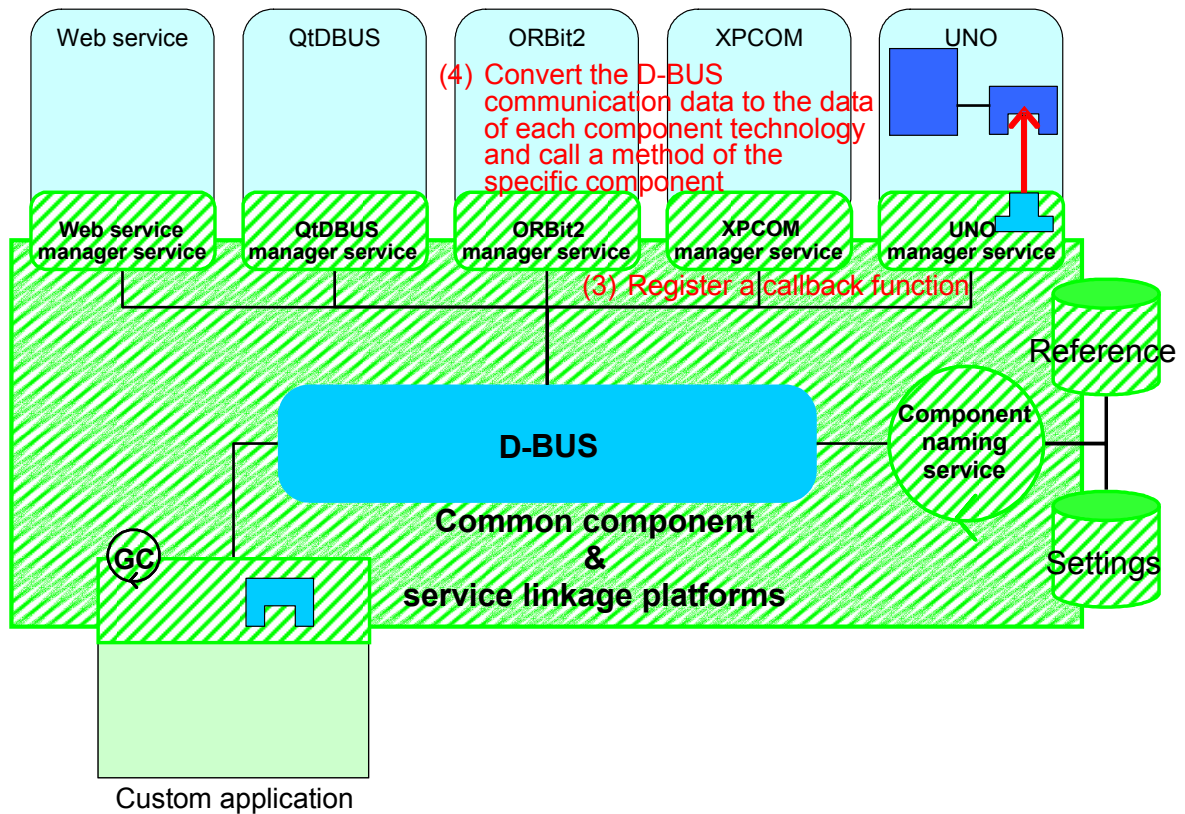


Figure 7-32 Asynchronous Communication Processing 2

- (3) Register a callback function (Figure 7-1-32)
 - Register the callback function for a D-BUS asynchronous call.

- (4) Call a method of a specific component (Figure 7-1-32)
 - A manager service of each component technology converts the D-BUS communication data to the data of each component technology and calls a method of the specific component.
 - **As described in Chapter 4, conversion is possible because all D-BUS communication data types correspond to the data types of each component technology.**

Technical Specifications

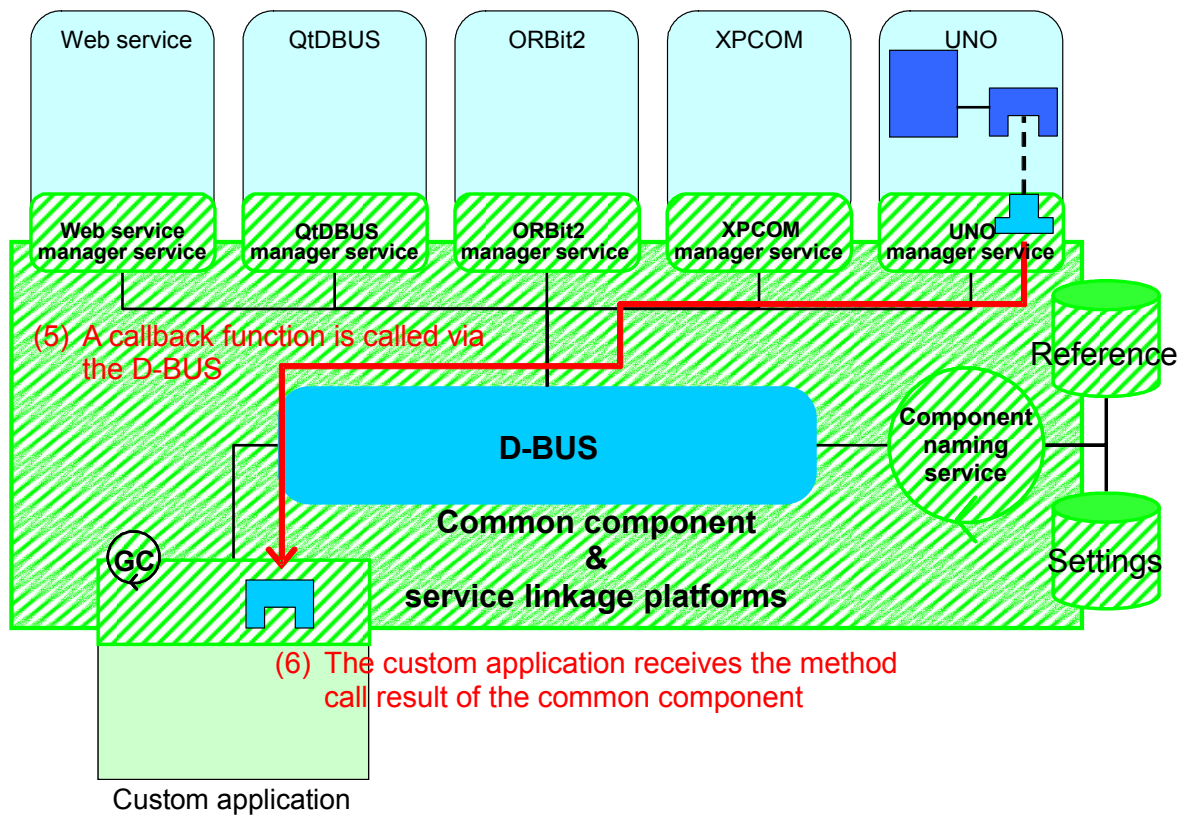


Figure 7-33 Asynchronous Communication Processing 3

- (5) A callback function is called via the D-BUS (Figure 7-1-33)
 - A callback function is called via the D-BUS.
- (6) The custom application receives the result (Figure 7-1-33)
 - The custom application common component, proxy, receives the result via the D-BUS.

7.6.3 Use-case of Referring to a Common Component

[1] Office Application

The following shows the use-case in which a custom application creates common components of a spreadsheet and word processor and obtains data from the spreadsheet (which is referred to from the custom application), and inserts the data to the word processor (which is referred to from the custom application) after processing (This assumes the situation in which, for example, a custom application adds up and performs statistical processing of numeric data in a spreadsheet, and then inserts the result in the report being edited in a word processor). In this description, OpenOffice Calc and KOffice KWord are used as sample specific components of Office application.

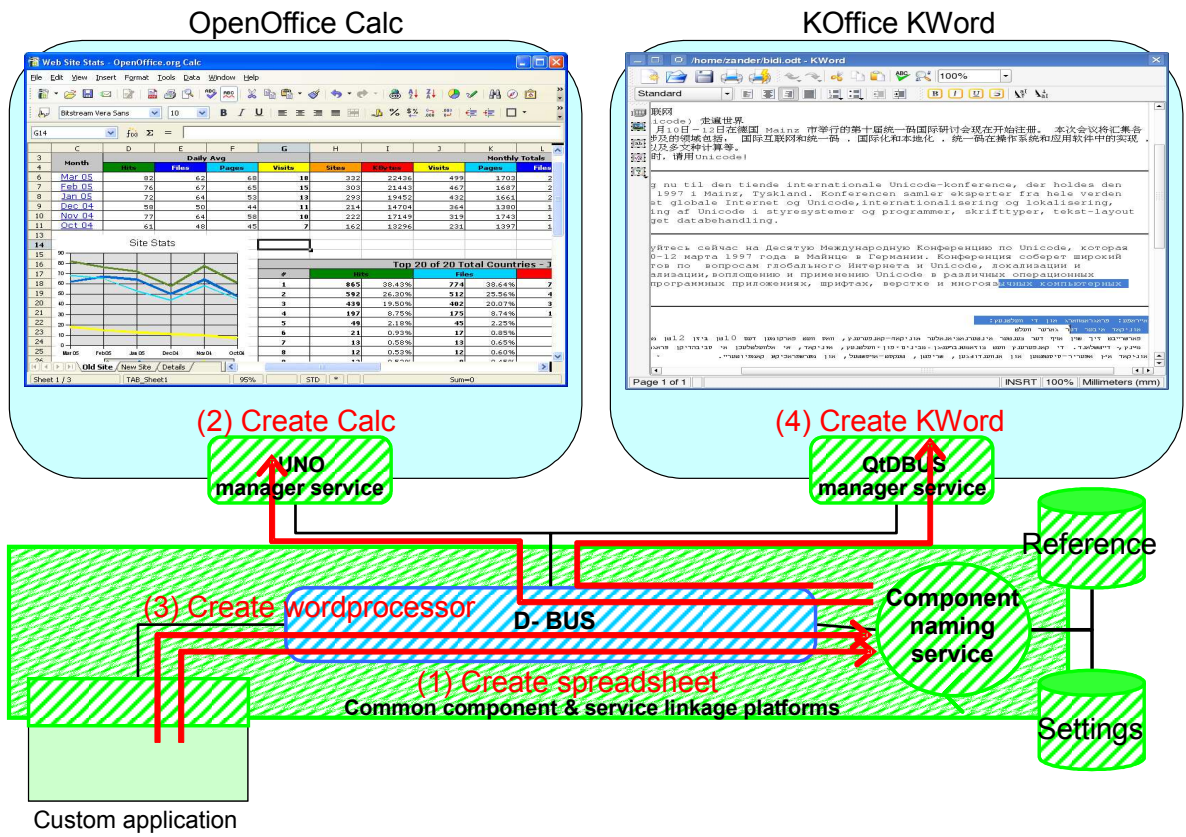


Figure 7-34 Use-case of Referring to Common Component 1

- (1) A custom application requests the component naming service to create a common component (spreadsheet) for Office application.
- (2) The component naming service creates a specific component, OpenOffice Calc, via UNO manager service.
- (3) A custom application requests the component naming service to create a common component (wordprocessor) for Office application.
- (4) The component naming service creates a specific component, KDE KWord, via QtDBus manager service.

Technical Specifications

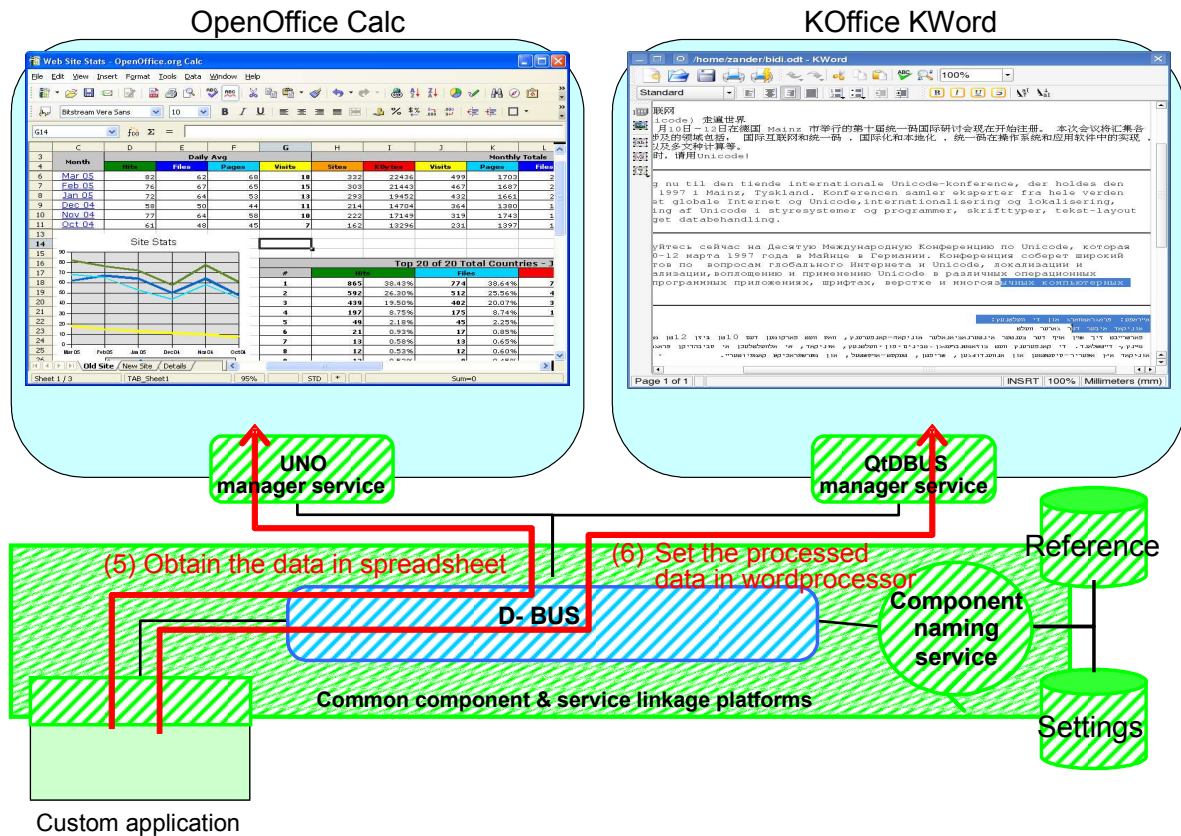


Figure 7-35 Use-case of Referring to Common Component 2

- (5) The custom application refers to the common component of a spreadsheet to obtain a series of numeric data in the spreadsheet.
- (6) The custom application processes the data obtained from a spreadsheet and inserts the results to the wordprocessor by referring to its common component.

7.7 Common Component Deletion Function

The common component deletion process flow in a sample case when UNO specific component is the corresponding component of the specified common component as shown in Figure 7-36 to Figure 7-37.

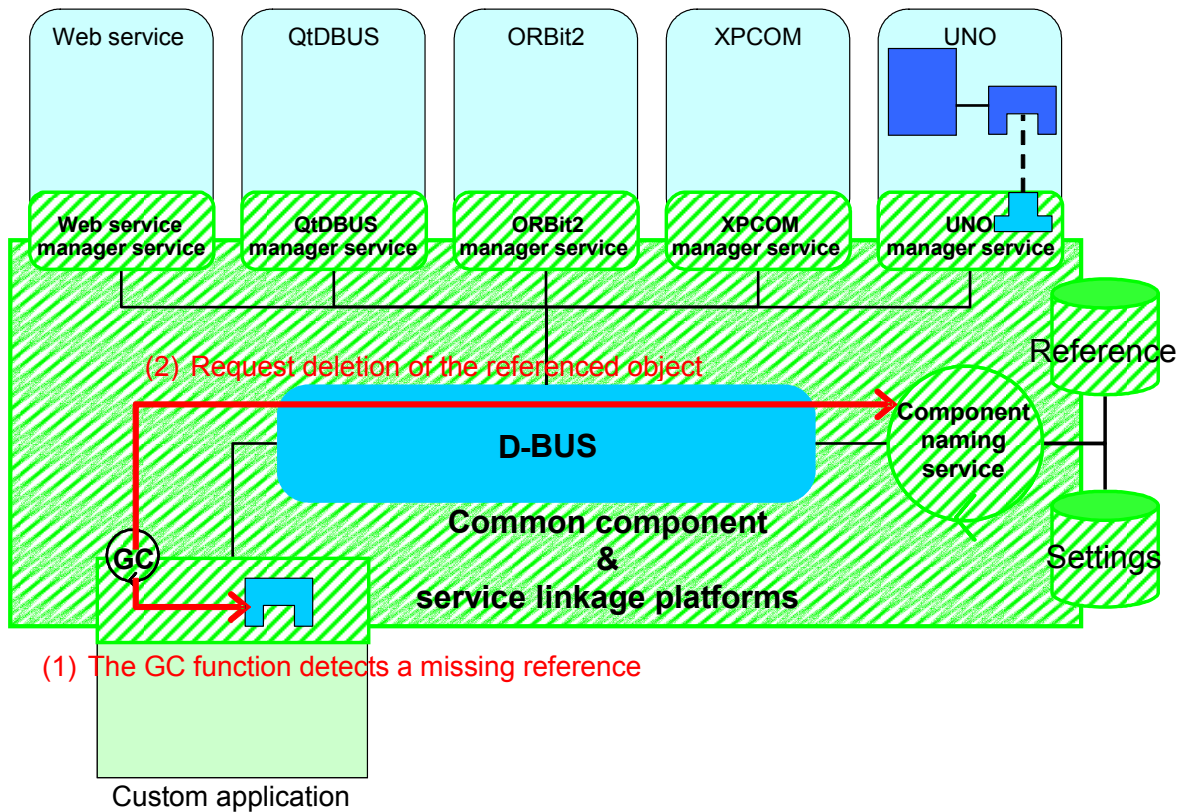


Figure 7-36 Common Component Deletion 1

- (1) The GC function detects a missing reference (Figure 7-1-36)
 - The GC function in the application helper library detects a missing reference to a proxy.
- (2) Request deletion of the referenced object (Figure 7-1-36)
 - The GC function requests the component naming service to delete the referenced object of a proxy.

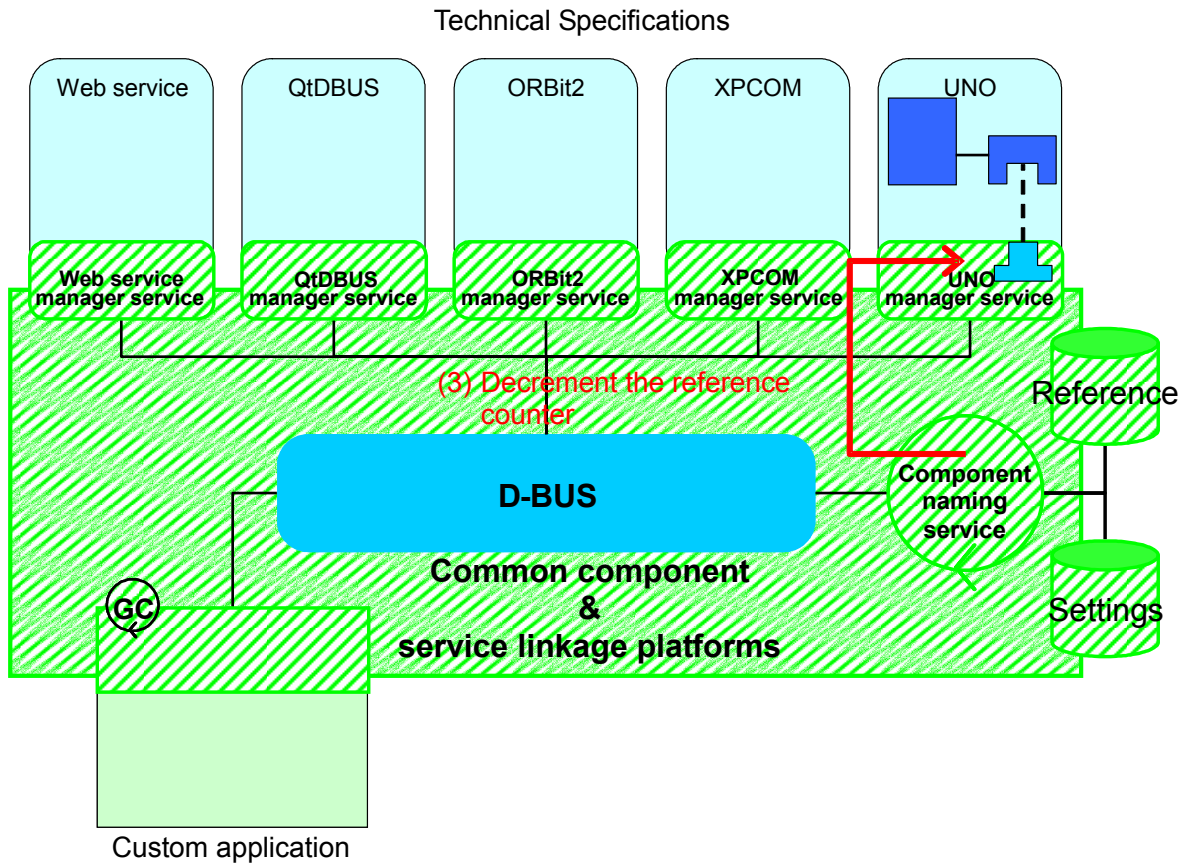


Figure 7-37 Common Component Deletion 2

- (3) Decrement the reference counter (Figure 7-1-37)
- The component naming service requests each manager service to decrement the reference counter of the specific component to be referred to or delete a reference pointer.
 - **When the GC function is used in each component technology, decrement of the reference counter or deletion of the reference pointer is required.**
 - **It is preferable but difficult to use the GC function (and synchronize the GC function) in the common component platform, which should be determined after a prototype is created and verified.**

7.8 Web Service Call Function

The web service call process flow is shown in Figure 7-38 to Figure 7-40.

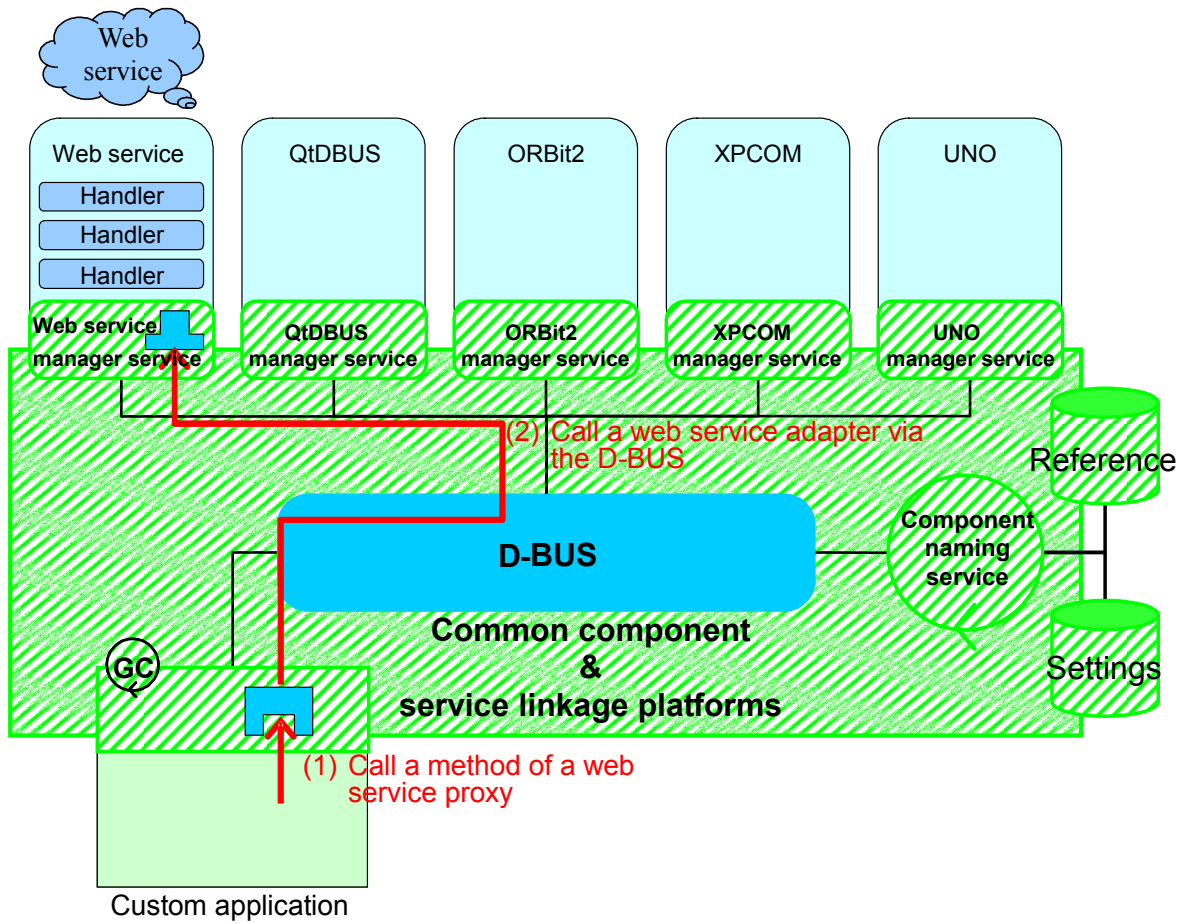


Figure 7-38 Web Service Call 1

- (1) Call a method of a web service proxy (Figure 7-1-38)
 - The application helper library calls a method of a web service proxy.
- (2) Call a web service adapter via the D-BUS (Figure 7-1-38)
 - Send a D-BUS METHOD_CALL message to the web service adapter.

Technical Specifications

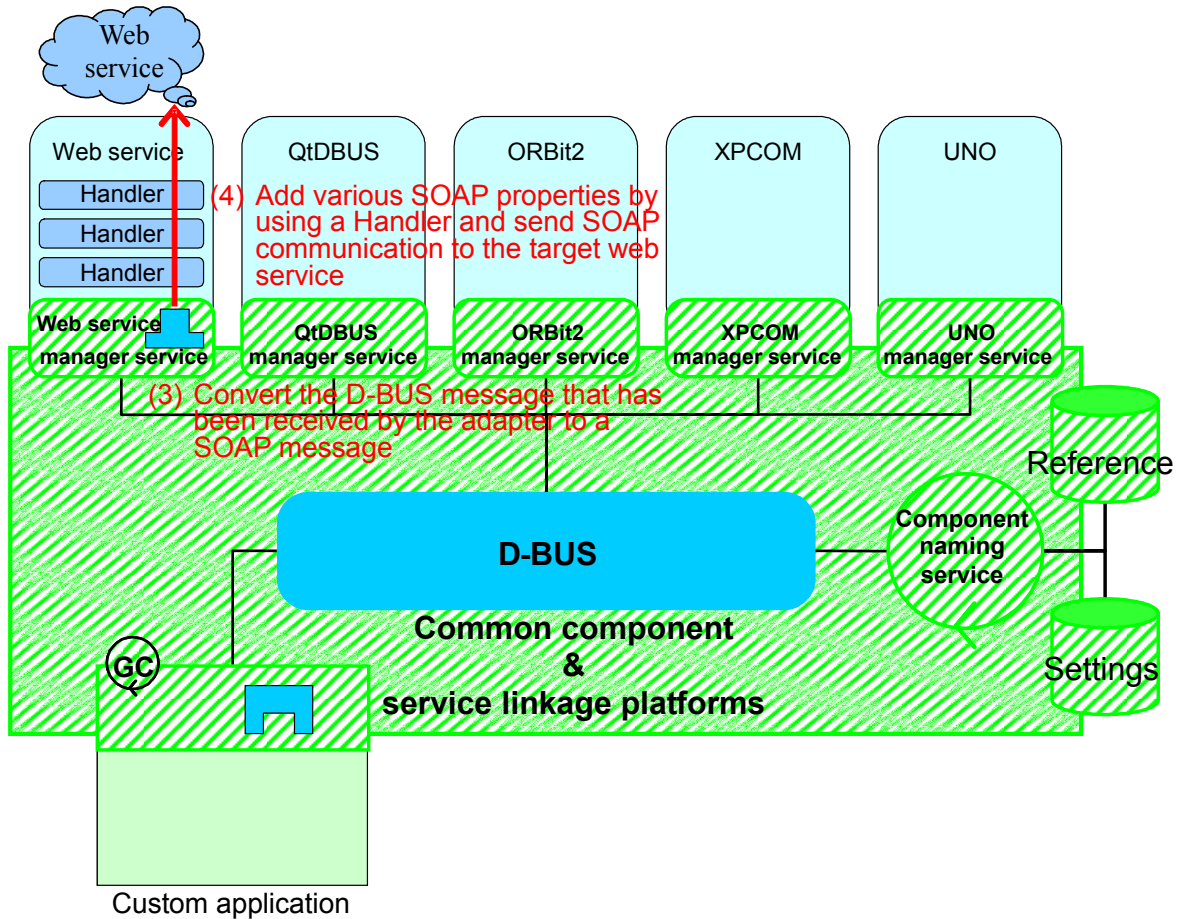


Figure 7-39 Web Service Call 2

- (3) An adapter converts the D-BUS message to a SOAP message (Figure 7-1-39)
 - An adapter converts the received D-BUS message to a SOAP message.
 - **The mapping specifications of D-BUS messages and SOAP messages are described in Chapter 4.**
 - **Mapping rules can be applied to D-BUS and SOAP messages that have no corresponding messages. The setting method requires verification by creating a prototype, which will be determined after the verification.**
- (4) Add various SOAP properties by using a Handler (Figure 7-1-39)
 - The SOAP properties that cannot be added directly from a D-BUS message are added by using a Handler while a SOAP message is sent.

Technical Specifications

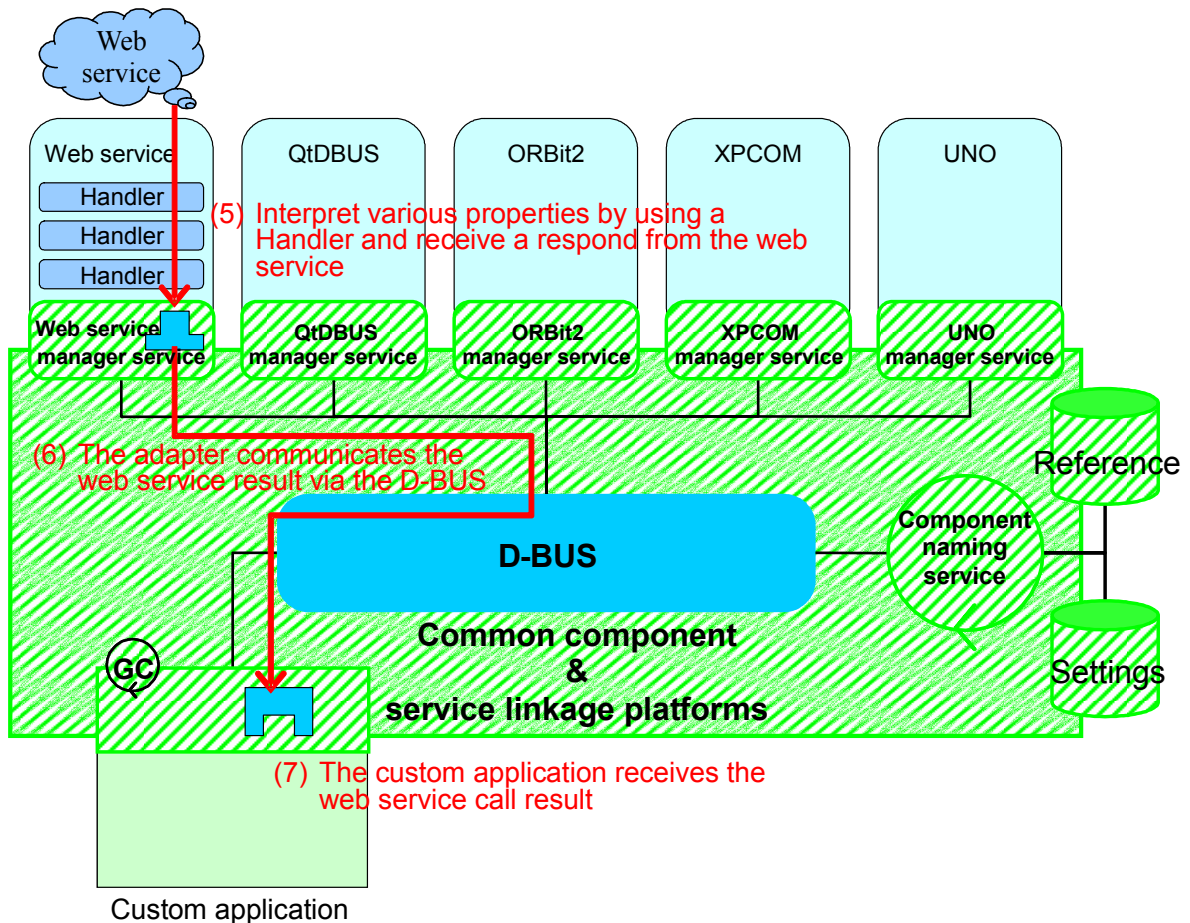


Figure 7-40 Web Service Call 3

- (5) Interpret various SOAP properties by using a Handler (Figure 7-1-40)
 - Interpret the properties of the SOAP message that has been received as a result of a web service by using a Handler.
 - An adapter converts the resulting SOAP message after property interpretation to a D-BUS message.
- (6) The adapter communicates the web service result via the D-BUS (Figure 7-1-40)
 - Communicate the D-BUS message that has been converted by the adapter to the custom application.
- (7) The custom application receives the web service call result (Figure 7-1-40)
 - The custom application receives the web service call result as a D-BUS message.

8 Issues to Be Addressed in Future

There are the following issues left in this design.

8.1 Technical Issues

(1) Implementing the GC Function in the Common Component Platform

It is preferable that modern component technologies should have the GC function like .NET Framework. To realize such function, the specifications have been defined assuming the GC function will be installed in the common desktop platform too.

However, actual object management will be performed in each component technology. Therefore, the GC function in the common desktop platform needs to be synchronized with the GC function in each component technology.

Although it is difficult to realize the GC function, it is further difficult to synchronize multiple GC functions. This will be left as a technical issue, which should be detailed after verification by creating a prototype in future.

(2) Assembly Model

As a result of the study, one possible specification is that binding of multiple components should be written as an assembly model in XML, as described in 4.14 and 4.15.

However, whether introduction of an assembly model in XML is appropriate for the common desktop platform cannot be determined without verification by creating a prototype. This will be left as a technical issue to be addressed in future.

(3) Support for Distributed Components

In general, desktop environments have a distributed component technology, like .NET Framework, COM+, UNO, and Bonobo/ORBit2.

In addition, realizing the distributed component technology will increase the solutions for open source desktop environments, such as thin client (for example, a component that controls application display operates on a client while a component that handles document files operation on a network).

However, to implement the distributed environment in the common desktop platform, expansion needs to be performed by working with existing development teams so that compatibility is maintained among component technologies. This will be left as a technical issue to be addressed in future.

(4) Standardization of Component Technologies

This design realizes collaboration among component technologies by connecting the component technologies based on the D-BUS. As a result, some processes such as a naming service are required. In a normal situation, those technologies should be integrated as a component technology.

However, to standardize component technologies including the naming service technology etc., the component technologies examined in this study need to be built from scratch. To do this, an architecture etc. need to be designed by discussing with communities including the developer of each component technology.

Therefore, as a first step, a gradual design considering the consistency with existing component technologies was made this time. Discussion with communities is an issue to be addressed in future.

Technical Specifications

(5) Functional Expansion to Support Web Services

As a result of the study, one possible specification for implementing web service functions in the common desktop platform is that the polling function should be added to the following D-BUS, as described in Chapter 4 (for details, refer to 4.9.2).

However, demand for those functions will vary depending on how web service functions are used in the common desktop platform. Specifically, the polling function may affect the real-time property for a desktop.

Therefore, this is a technical issue that will be considered according to user demand in the course of the common desktop platform implementation in future.

8.2 Approach to Communities

To actually develop the common component platform, which is to be realized in this design, function addition to existing components and D-BUS is required, which means that cooperation of communities is essential. Specifically, the function addition includes the following:

- (1) Provide QtDBus with the functions to execute search, reference, and activation of objects from outside.
- (2) Provide KParts with the functions to execute communication, search, reference, and activation.
- (3) Provide ORBit2 with an asynchronous communication function.
- (4) Provide IDL for XPCOM with the function to specify a synchronous or asynchronous communication.
- (5) Provide XPCOM communication with a method call function.
- (6) Provide XPCOM communication with the function to use the same data types as IDL.
- (7) Provide IDL for OpenOffice.org with the function to specify a synchronous or asynchronous communication.
- (8) Provide D-BUS with the functions to execute search, reference, and activation for an object path.
- (9) Expand D-BUS messages to add ERP and other information to a message header.

Approaching communities in order to add the above functions is an issue to be addressed in future.

End