OTIF Protocol Description

Andras Lang
andras.lang@vanderbilt.edu

Institute for Software-Integrated Systems
Vanderbilt University
March 2004

Table of Contents

1. Introduction

This document gives you a detailed overview about the communication protocol used by the OTIF framework.

You should be familiar with the *OTIF.pdf* and *OTIF_Usage_Guide.pdf* documents to use this documentation effectively.

The OTIF framework uses the CORBA mechanism to communicate between the client and the server application.

The **Backplane** is the server application. The **Manager** and any kind of **ToolAdaptor** and **Translator** applications are the client applications. The **ToolAdaptor** and **Translator** applications can be implemented *with* or *without* the libraries of the OTIF framework.

This document is very useful for those users who do not want to use the OTIF libraries to implement any ToolAdaptor or Translator applications. At this case the user should be familiar with the communication protocol of the OTIF.

The communication protocol is defined by the CORBA IDL files. The OTIF framework is using two IDL files.

The *otifdata.idl* file defines the CORBA structure of

- the document and
- the paradigm.

The document is an entity which is shipped between the client and the server applications. The OTIF document and an UDM document (datanetwork) describe the same information but the data of these documents are represented in a different way.

Note: Of course any kind of data modeling technology - which uses the UML defined metamodeling technique - can be used instead of the UDM. OTIF only gives you an opportunity to use UDM but you are free to use your own data modeling technology.

The paradigm is a metamodel conforms to the UML.

The *otif.idl* file defines the CORBA communication protocol. It will be introduced to you in the next section.

2. CORBA communication protocol

The protocol defined in the *otif.idl* file defines 7 interfaces. These interfaces are instantiated by the Backplane server and the clients application.

Backplane interface

This interface may be implemented only in the Backplane application. The user can reach all the Backplane functionality through this interface.

- readonly attribute NameVersion_S Paradigms

  It provides a list of known metamodels (paradigms) registered in the Backplane server.

- readonly attribute TranslatorVersion_S Translators

  It provides a list of known translators registered in the Backplane server.

- readonly attribute DocumentInfo_S Documents

  It provides a list of available datanetworks (documents) shipped to the Backplane server by the ToolAdaptor or Translator applications.

- readonly attribute Name_S ToolAdaptorTypes

  It provides a list of available types of ToolAdaptors registered in the Backplane server.

- void lockParadigm(in NameVersion paradigm) raises(InvalidParadigm, ParadigmLocked)

  It locks a metamodel. You have to specify the name and the version parameter of the metamodel (paradigm) in the *NameVersion* CORBA structure.

- Diagram fetchParadigm(in NameVersion paradigm) raises(InvalidParadigm)

It fetches a metamodel. The return value is the CORBA structure of the document.

Note: The paradigm has exactly the same data structure as a document.

- void unlockParadigm(in NameVersion paradigm) raises(InvalidParadigm, ParadigmNotLocked)

It unlocks a metamodel. You have to specify the name and the version parameter of the metamodel (paradigm) in the *NameVersion* CORBA structure.

- ManagerSession loginManager(in NameString key, out SessionId session_id) raises(InvalidKey, OnlyOneManager)

All Manager application - which implement the ManagerSession interface- should login first to the Backplane server with this function. Upon login the client (Manager application) must supply a key for authentication.

The input parameter is the password of the Backplane server.

The output parameter is a session_id for the Manager application which can be used by the client to identify itself to the Backplane application.

- void logoutManager(in SessionId session_id) raises(InvalidSession)

Before the Manager application quits it should logout from the Backplane server. *It is necessary!*

The input parameter is the ID of the session of the Manager application.

- ToolAdaptorSession loginToolAdaptor(in string ToolAdaptorType, out SessionId session_id) raises(InvalidToolAdaptorType)

All ToolAdaptor application should login first to the Backplane server with this function. The ToolAdaptor has to identify its type to the Backplane server.

The input parameter is the type of the ToolAdaptor application.

The output parameter is a session_id for the ToolAdaptor application which can be used by the client to identify itself to the Backplane application.

- ToolAdaptorInfo getToolAdaptorInfoByType(in NameString taType) raises(InvalidToolAdaptorType);

It provides detailed information (supported publishing and fetching paradigms) for a tool adaptor type.

- ToolAdaptorInfo getToolAdaptorInfo(in SessionId session_id) raises(InvalidSession);

It provides detailed information (supported publishing and fetching paradigms) about the ToolAdaptor for an active session.

- NameString getToolAdaptorType(in NameVersion paradigm, in taMode mode) raises(InvalidParadigm);

It provides the name of the type of the ToolAdaptor which fetches/publishes in a given paradigm.

- void logoutToolAdaptor(in SessionId session_id) raises(InvalidSession)

Before the ToolAdaptor application quits it should logout from the Backplane server. *It is necessary!*
The input parameter is the ID of the session of the ToolAdaptor application.

- void fetchRootTypeName(in DocumentID docId, out NameString rootTypeName) raises(InvalidDocument);

It provides the name of the type of a document's root object. This information can be necessary if you want to search for the root object of a paradigm. It is necessary when you allow loading the paradigm dynamically in your ToolAdaptor or Translator application. (For example, it is how the generic ToolAdaptor works)

- void now(out timeStamp stamp)

It returns the current clock of the Backplane server.

- void uploadStartup(in string xml) raises(InvalidStartupFile)

It sends an xml configuration file to the Backplane server. The Backplane resets itself and loads the new configuration.

ManagerSession interface

This interface may be implemented by the Manager application.

- enum RegistrationAction { add, replace }

The registration procedure supports two kinds of registration mode. The add mode is used to register a new one. The replace mode is used to replace a registered one.

- void destroyDocument(in DocumentID docId) raises(InvalidDocument)

It destroys a document in the Backplane server.
The input parameter is the ID of the document.

- void modifyDocumentName(in DocumentID docId, in string name) raises(InvalidDocument)

It modifies the name parameter of the document.
The input parameters are the ID of the document and the new name parameter of the document.

- void modifyDocumentVersion(in DocumentID docId, in string version) raises(InvalidDocument)

It modifies the version parameter of the document.
The input parameters are the ID of the document and the new version parameter of the document.

- void modifyDocumentRemarks(in DocumentID docId, in string remarks) raises(InvalidDocument);

It modifies the remark parameter of the document.
The input parameters are the ID of the document and the new remarks parameter of the document.

- void modifyDocumentKeepAlive(in DocumentID docId, in long keepAlive) raises(InvalidDocument)

It modifies the keepAlive parameter of the document.
The input parameters are the ID of the document and the new keepAlive parameter of the document.

- Diagram registerMetaModel(in NameVersion paradigm, in UdmDataNetwork metaData, in RegistrationAction action) raises(ParadigmExists, ParadigmLocked, ParadigmInUse)

It registers a metamodel (paradigm) into the Backplane server. The paradigm should be converted into an UdmDataNetwork structure (paradigm represented in CORBA structure).
The input paramteres are the name and version parameter of a paradigm in the *NameVersion* CORBA structure, the UdmDataNetwork CORBA structure and a RegistrationAction.

- Diagram registerMetaModelURL(in NameVersion paradigm, in NameString fileName, in RegistrationAction action) raises(ParadigmExists, ParadigmLocked, ParadigmInUse,InvalidFile)

It registers a metamodel (paradigm) local file into the Backplane server. This file should be in the "paradigm" directory of the Backplane server.
The input paramteres are the name and version parameter of a paradigm in the *NameVersion* CORBA structure, the name of the file and a *RegistrationAction*.

- void unregisterMetaModel(in NameVersion paradigm) raises(ParadigmInUse, ParadigmLocked, InvalidParadigm)

It unregisters a metamodel (paradigm). A paradigm only can be unregistered if it is not locked or not in use.
The input paramteres are the name and version parameter of a paradigm in the *NameVersion* CORBA structure.

- void registerTranslator(in TranslatorVersion translator, in string activation, in RegistrationAction action) raises(TranslatorExists, TranslatorInUse, InvalidTranslator, InvalidParadigm)

It registers a translator into the Backplane server. The translator is activated by the activation command. The activation command identifies the Translator application which can be an *.exe* file or an *.xsl* document.

The input parameters are the *TranslatorVersion* CORBA structure, the filename of the Translator application and a *RegistrationAction*.

- void unregisterTranslator(in TranslatorVersion translator) raises(TranslatorInUse, InvalidTranslator)

It unregisters a translator.
The input parameter is the *TranslatorVersion* CORBA structure.

- void change_password(in string old_password, in string new_password) raises(InvalidKey)

It changes the Backplane server password.
The input parameters are the old and the new password.

- oneway void shutdown()

It terminates the Backplane server and all object references become invalid.

ToolAdaptorNotifyCallback interface

This interface is implemented only in the ToolAdaptor applications.

A **ToolAdaptorNotifyCallback** object can be used to be callbacked by the Backplane server when a document is available for the ToolAdaptor application.

The notification carries a unique document id, which can be used in a subsequent fetch operation by the ToolAdaptor application.

- oneway void notify(in NameVersion documentName, in string remarks, in DocumentID docId, in long keepAlive, in long numberOfDocuments)

It notifies the ToolAdaptor application when a document is available in the Backplane server.

The input parameters:

- the *NameVersion* CORBA structure stores the name and the version parameter of the available document
- the *remarks* parameter stores the remarks of the document
- the *docId* parameter identifies the available document in the Backplane server. It is used to fetch the document
- the *keepAlive* parameter tells how long (in seconds) the Backplane server will keep the document alive
- the *numberOfDocuments* parameter shows the number of the structured elements of the document.

- oneway void changedExistingDocument(in NameVersion documentName, in string remarks, in DocumentID docId, in long keepAlive, in long numberOfDocuments)

It notifies the ToolAdaptor application if a parameter of a document is changed. This function is useful when the ToolAdaptors displays the available documents on a window.

The input parameters are the same as the *notify* function has.

ToolAdaptorSession interface

This interface is implemented only in the ToolAdaptor applications.

- DocumentID publishDocument(in SessionId session_id, in NameVersion paradigm, in NameVersion document, in string remarks, in long keepAlive, in Document docData, in DocumentID previous_docId) raises(InvalidParadigm, InvalidSession, InvalidDocument)

It publishes a document. The document should be converted into a Document structure (document represented in CORBA structure).

The publisher can specify a time limit for how long the document should be kept alive in the document cache in the Backplane server.

The input parameters:

- the *session_id* parameter identifies the ToolAdaptor application
- the paradigm *NameVersion* CORBA structure stores the name and the version of the paradigm of the document
- the document *NameVersion* CORBA structure stores the name and the version parameter of the document
- the *remarks* parameter stores the remarks of the document
- the *keepAlive* parameter tells how long (in seconds) the Backplane server will keep the document alive
- the *Document* CORBA structure stores the document
- the *previous_docId* should be NO_PREVIOUS_DOCUMENT whenever a completely new document is published, otherwise it is the backplane-generated ID of an existing, previously published document whose new version is in *docData*. If that previous document does not exist anymore, an *InvalidDocument* exception will be raised.

It returns the backplane-generated ID of the published document.

- void subscribe(in SessionId session_id, in ToolAdaptorNotifyCallback callback)

It subscribes to documents of matching type. After that, the ToolAdaptor application will receive notifications.

The input parameters are the *session_id* - which identifies the ToolAdaptor application - and the *callback* – which will be callbacked by the Backplane application - parameter.

- Document fetchDocument(in DocumentID docId) raises(InvalidDocument)

  It fetches an available document.
  The input parameter is the ID of the available document.
  The output parameter is the *Document* CORBA structure which stores the document.

- DocumentInfo_S getDocuments(in SessionId session_id) raises(InvalidSession)

  It gets the information of the corresponding documents of the ToolAdaptor for an active session.

- NameVersion getDocumentParadigmName(in DocumentID docId) raises(InvalidDocument)

  It gets the paradigm of a document.
  The input parameter is the ID of the available document.
  The output parameter is the the *NameVersion* CORBA structure which stores the name and the version parameter of the document.

- ToolAdaptorInfo getToolAdaptorInfo(in SessionId session_id) raises(InvalidSession);

  It provides detailed information (supported publishing and fetching paradigms) about the ToolAdaptor for an active session.

TranslatorNotifyCallback interface

This interface is implemented only in the Translators applications. This is used to callback the Translator if a document is available to fetch.

A **TranslatorNotifyCallback** object can be used to be callbacked by the Backplane server when a document is available for the Translator application.

The notification carries a unique document id, which can be used in a subsequent fetch operation by the Translator application.

- oneway void notify(in NameVersion documentName, in DocumentID docId, in long keepAlive, in long numberOfDocuments);

  It notifies the Translator application when a document is available in the Backplane server.

  The input parameters:
  - the *NameVersion* CORBA structure stores the name and the version parameter of the available document
  - the *remarks* parameter stores the remarks of the document
  - the *docId* parameter identifies the available document in the Backplane server. It is used to fetch the document
  - the *keepAlive* parameter tells how long (in seconds) the Backplane server will keep the document alive.
  - the *numberOfDocuments* parameter shows the number of the structured elements of the document.

TranslatorFeed interface

This interface is implemented only in the Translators applications.

A **TranslatorFeed** object can be used by the Translator application to subscribe to documents of matching type and receive documents from the Backplane server.

- void subscribe(in TranslatorNotifyCallback callback)

It subscribes to documents of matching type. After that, the Translator application will receive notifications.

The input parameter is the *callback* – which will be callbacked by the Backplane application - parameter.

- Document fetchDocument(in DocumentID docId) raises(InvalidDocument)

It fetches an available document.

The input parameter is the ID of the available document.

The output parameter is the *Document* CORBA structure which stores the document.

TranslatorSink interface

This interface is implemented only in the Translators applications.

A **TranslatorSink** object can be used by the Translator application to send documents to the Backplane server.

- void publishDocument(in NameVersion documentName, in string remarks, in long keepAlive, in Document docData, in DocumentID source_docId) raises(InvalidSession);

  It publishes a document. The document should be converted into an UdmDataNetwork structure (document represented in CORBA structure).

  The publisher can specify a time limit for how long the document should be kept alive in the document cache in the Backplane server.

  The input parameters:

  - the *NameVersion* CORBA structure stores the name and the version parameter of the document
  - the *remarks* parameter stores the remarks of the document
  - the *keepAlive* parameter tells how long (in seconds) the Backplane server will keep the document alive
  - the *Document* CORBA structure stores the document
  - the *source_docId* parameter stores the ID of the input document which was translated.

The following exceptions are used in the protocol

- exception OnlyOneManager

It is thrown when a Manager application tries to connect to the Backplane when another Manager application is active and the Backplane server runs in the "only one active manager is allowed" mode.

- exception InvalidKey

It is thrown when the authentication key supplied by the Manager application is invalid.

- exception InvalidParadigm

It is thrown when a ToolAdaptor or a Translator application tries to use a paradigm which is not registered in the Backplane server. (Paradigm is unsupported/unknown)

- exception ParadigmLocked

It is thrown when an application tries to lock a locked paradigm. The paradigm is already locked.

- exception ParadigmNotLocked

It is thrown when an application tries to unlock a not locked paradigm. The paradigm is not locked.

- exception InvalidSession

It is thrown when an application refers to an ID of the session which does not exist. The session is invalid.

- exception ParadigmExists

It is thrown when a Manager application tries to register a paradigm which is already exists.

- exception ParadigmInUse

It is thrown when a Manager application tries to unregister a paradigm which is being used.

- exception TranslatorExists

It is thrown when a Manager application tries to register a Translator which is already exists.

- exception TranslatorInUse

It is thrown when a Manager application tries to unregister a Translator which is being used.

- exception InvalidDocument

It is thrown when an application refers to an ID of a document which does not exist. The supplied document ID is invalid.

- exception InvalidFile

It is thrown when a Manager application refers to a file which does not exist. The filename is invalid.

- exception InvalidTranslator

It is thrown when a Manager application refers to a Translator which does not exist. The Translator is invalid.

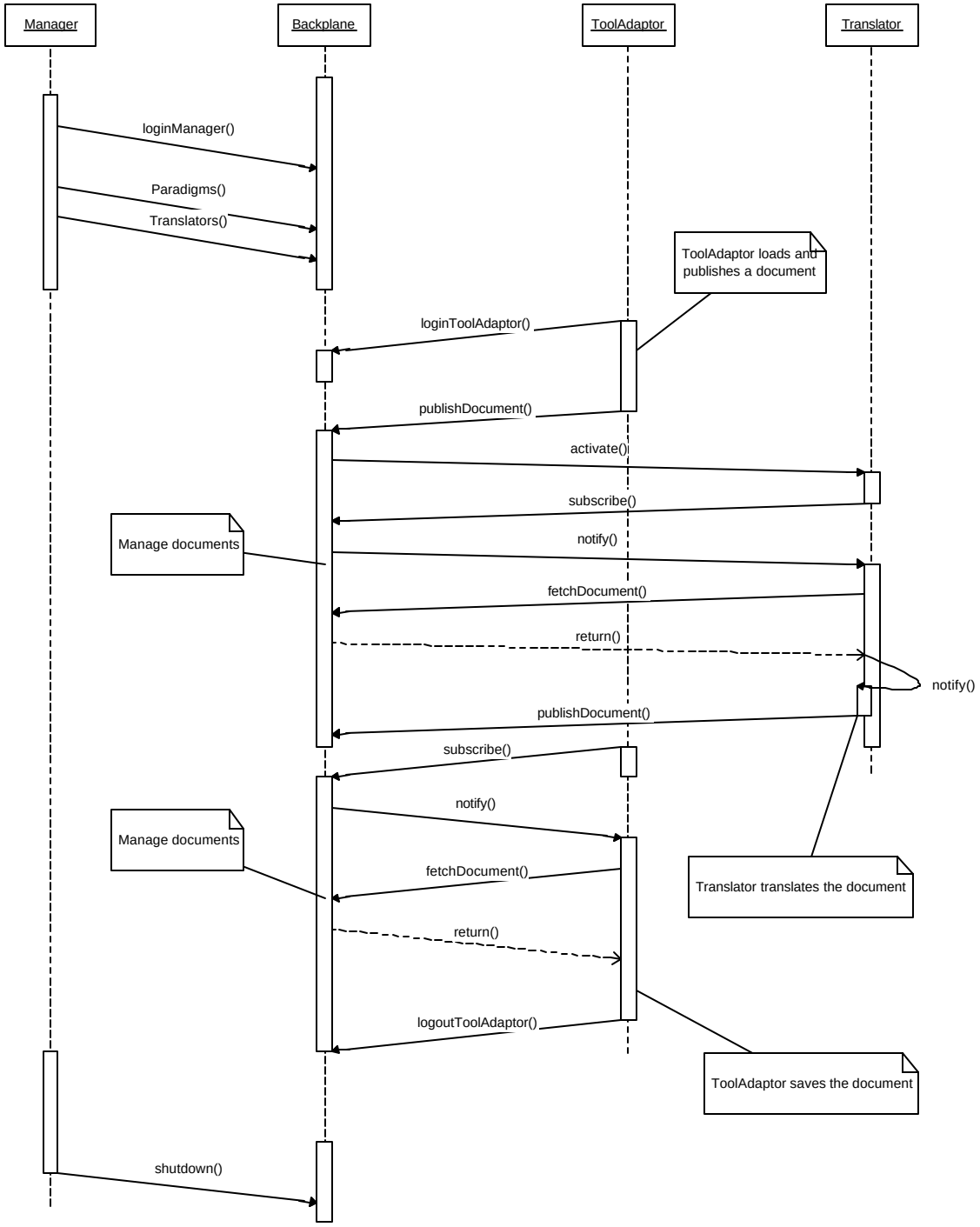
- exception InvalidStartupFile

It is thrown by the Backplane server when an invalid configuration xml file is uploaded to the Backplane server and it cannot be processed.

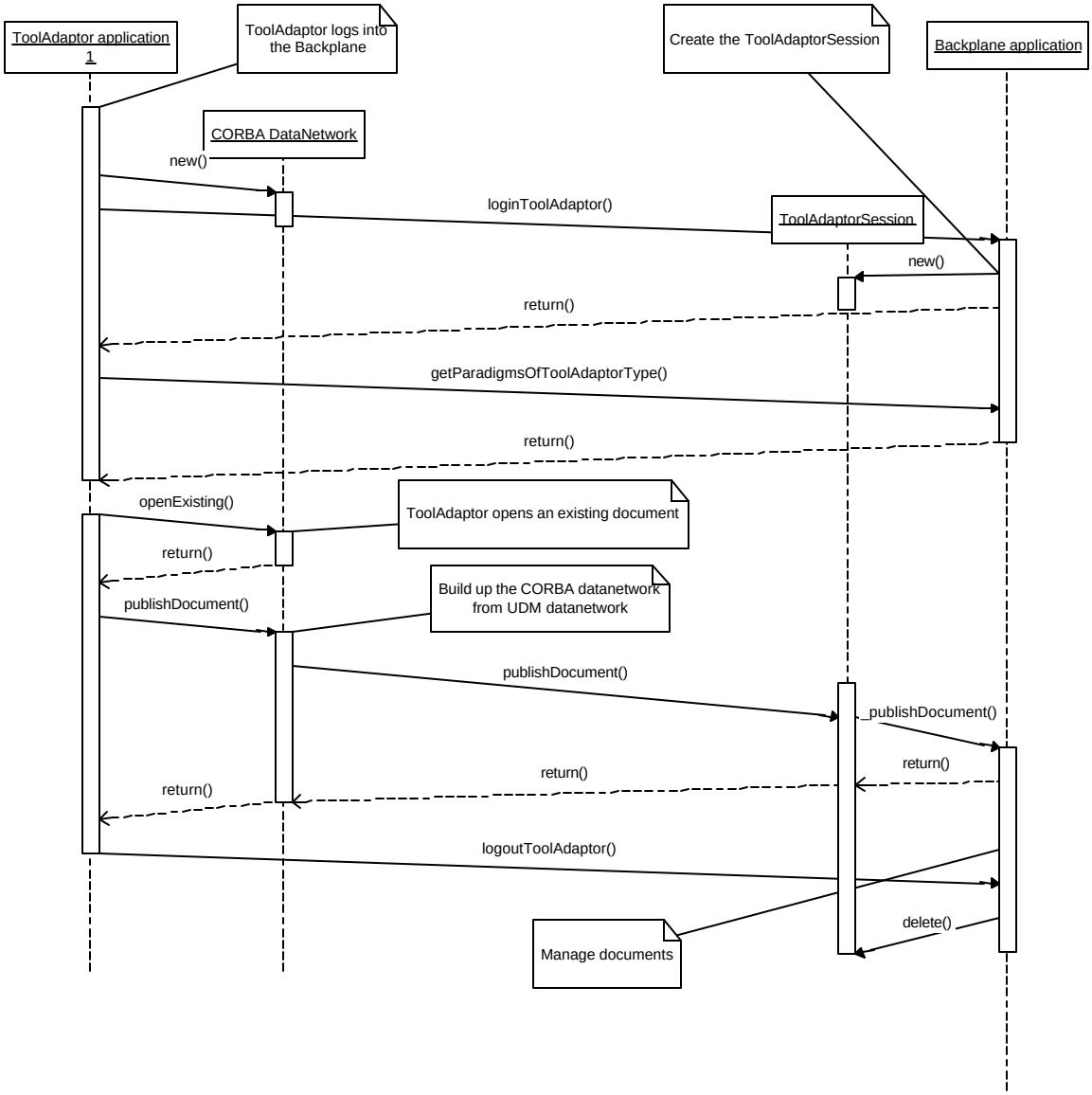
- exception InvalidToolAdaptorType

It is thrown when a ToolAdaptor application tries to log into the Backplane with an unsupported (unregistered) ToolAdaptor type.

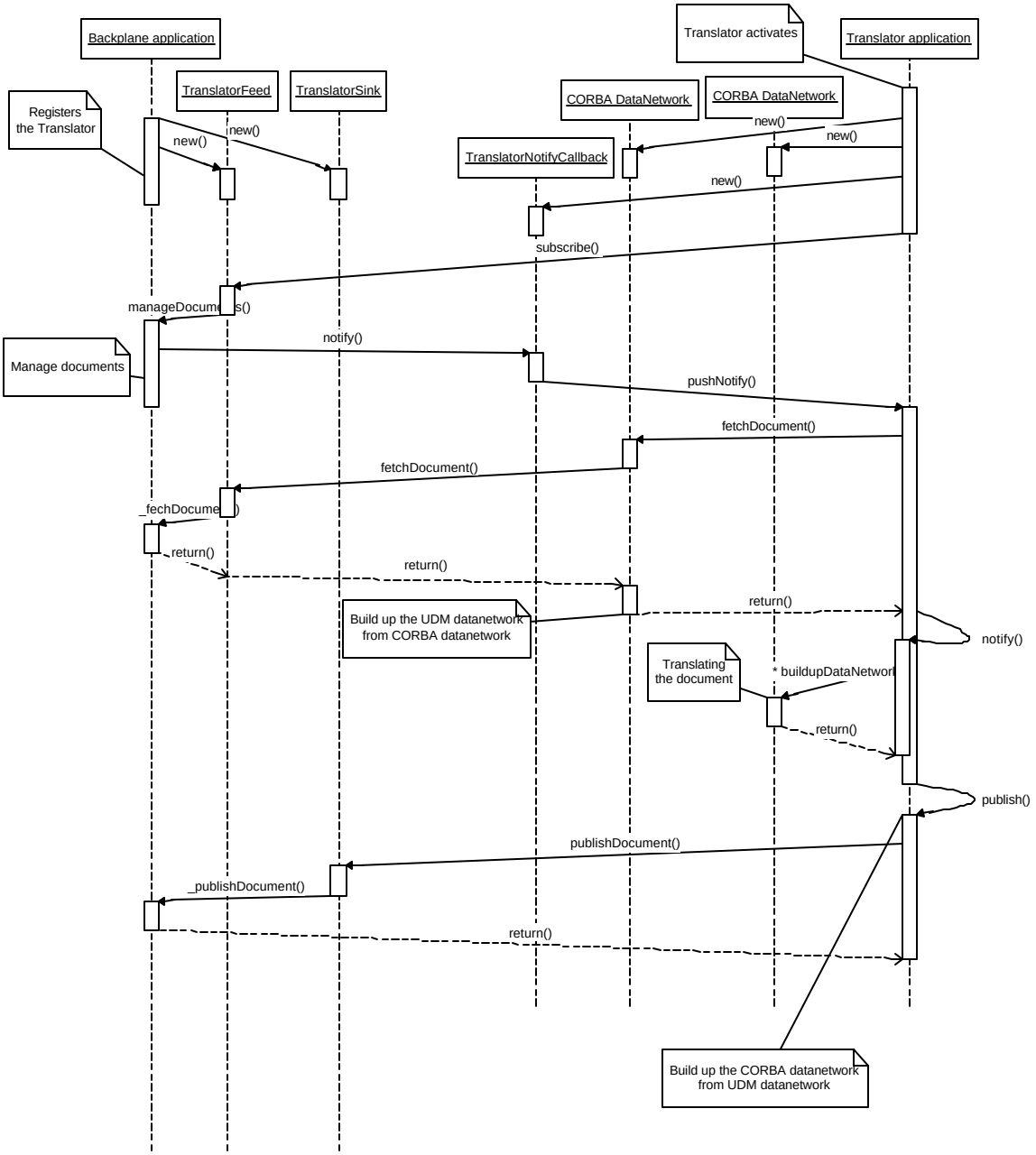
3. Protocol UML sequence diagrams



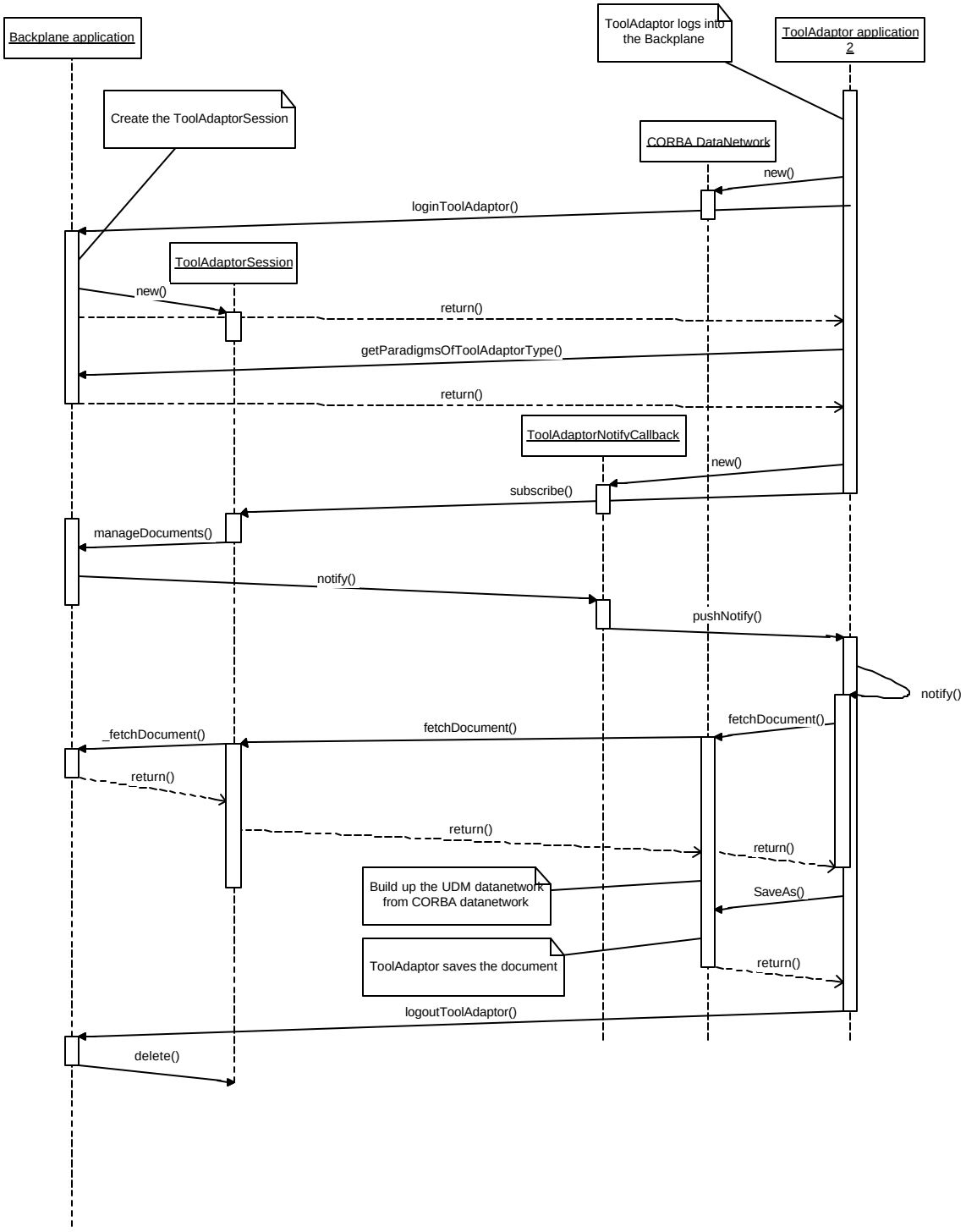
This diagram shows an overview of a simple lifetime of the OTIF framework.



This diagram shows a detailed **ToolAdaptor – Backplane** communication. The ToolAdaptor publishes a document to the Backplane, logs out and quits.



This diagram shows a detailed **Backplane – Translator** communication. The Backplane starts the Translator application when a document arrives. The Translator subscribes to the Backplane and waits for the notification. When it gets the notification it fetches, translates and publishes the document to the Backplane and quits.



This diagram shows a detailed **Backplane – ToolAdaptor** communication. The ToolAdaptor subscribes to the Backplane and waits for a notification. When it gets the notification it fetches the corresponding document, logs out and quits.