
Trading Service

CRC for Distributed Systems Technology (DSTC)

DSTC has been active in the specification, implementation, and standardisation of the ODP Trading Function and is delighted to contribute this expertise to the adoption of a Trading Service for CORBA.

Implementations and configurations can satisfy particular quality-of-service requirements.

Introduction

This Trading Service specification:

- is based on the concepts of the ODP Trading Function
- is specified to conform with the OMG philosophy, model, and style

Introduction

This Trading Service Specification provides:

- benefits to the users
- interfaces are simple, elegant, and powerful
- easy to use and understand
- benefits to the OMG
- object-oriented
- re-uses the Object Property Service
- benefits to the implementers
- easy to implement
- permits a variety of implementations and configurations

Overview

- Description of the Trading Service
- Service Offers
- Service Types
- Import and export
- Federation
- Summary
- Discussion (Security etc.)

Distributed object systems span heterogeneous software platforms, hardware platforms, and heterogeneous network environments. In order to use services in such systems, service users must be aware of potential services and service providers. Furthermore, locations and versions of services change quite frequently in large distributed systems, which makes late binding between service users and service providers essential. To support late binding, mechanisms must be provided to locate and access services dynamically.

For example, a bank advertises its savings accounts by announcing the account's interest rate and customer services (e.g. cheque-writing facility). Meanwhile, a customer wants to find a bank that has a savings account with a high interest rate and a cheque-writing facility. The role of the Trading Service is to "introduce" the bank to the customer.

Notes

Trading Service

Why?

Service users must find service providers at run-time

What?

The Trading Service provides a matchmaking service for objects

- Service providers use the Trading Service to advertise their services
- Service users consult the Trading Service to obtain information about suitable services

Traders

a registry of advertised service offers (object references)

operations for exporters - advertising services:

registers a service offer with the trader
withdraw a service offer from a trader

But modify Service Offers directly using its inherited PropertySetDef operations - not via the trader!

operations for importers - finding services:

search the trader for a set of ServiceOffers that

- satisfy the importer's matching constraints
- select a single ServiceOffer from the trader that
 - satisfies the importer's matching constraints and
 - best satisfies the importer's selection criteria.

In some cases, a service is associated with only one interface. Other services involve multiple interfaces (e.g. an event service might be comprised of an event-supplier interface and an event-consumer interface).

At the time of preparation of this submission, there is discussion within OMG on whether an object can have more than one interface. Therefore, this submission uses terms like “interface reference” in preference to “object reference” to avoid any confusion while the interface-object relationship is resolved.

Thus, CORBA and the Object Services already provide the basic building blocks for describing a service offer. Since it is possible to store interface types, interface references, and interface attributes¹ as properties in the Property Service, our ServiceOffer interface inherits from the PropertySetDef interface of the Property Service.

¹ While an interface attribute cannot be directly used as a property value, it can be used indirectly.

Notes

Service Offers

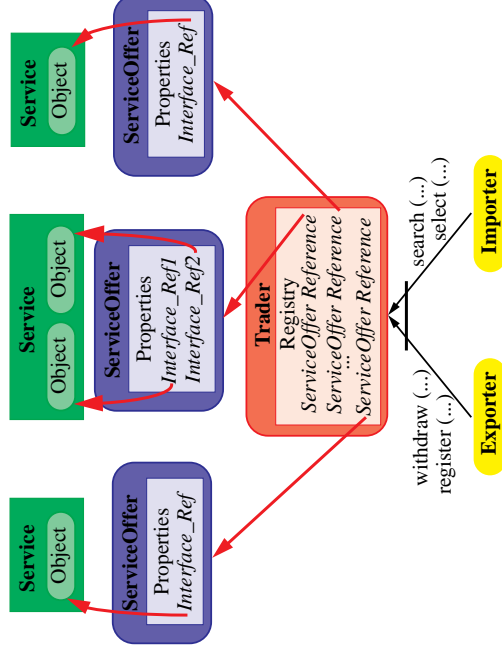
A service is provided by one or more objects.

A service offer is a description of a service, involving:

- details of each interface of the service:
 - interface reference
 - interface type
- other characteristics of the service, e.g. quality of service
 - attributes which are statically defined in the interface definition
 - properties which are dynamically defined by use of the Property Service

The ServiceOffer interface inherits from the PropertySetDef interface of the Property Service.

A Trader, its ServiceOffers and their Services



Property Service

The Property Service provides operations to store and retrieve properties.

Notes

Properties

- a name (of type string)
- a typed value (of type any)
- a mode (of enumeration type `PropertyModeType`):

PropertyModeType	Meaning
<code>normal</code>	Property can be updated or deleted.
<code>read_only</code>	Property cannot be updated, but can be deleted.
<code>fixed_normal</code>	Property can be updated, but not deleted.
<code>fixed_readonly</code>	Property cannot be updated and cannot be deleted.
<code>undefined</code>	Not relevant; only used in exceptions.

Service Offer

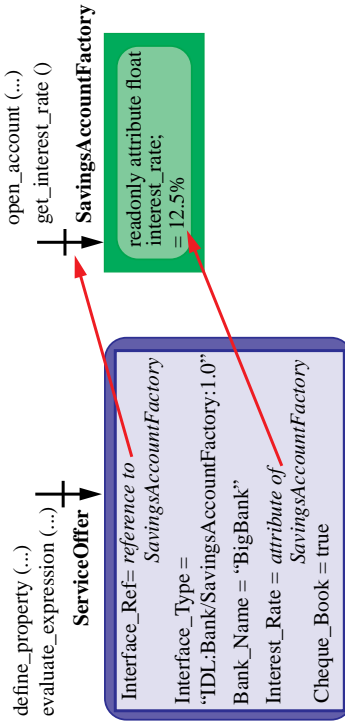
Inherited operations (from the `PropertySetDef`)

- define named properties and their associated strongly-typed values and modes
- retrieve the values and modes of properties
- delete properties

Extra operation

- evaluate expressions involving its property values.

A ServiceOffer for a SavingsAccountFactory



An example of a service offer for a bank is illustrated. The bank has created a service offer for its `SavingsAccountFactory` interface (which contains an operation to open a bank account). Most of the property values are stored within the service offer. However, the `SavingsAccountFactory` has an `interest_rate` attribute that can be accessed via the service offer; the bank does not need to update its service offer every time the interest rate fluctuates.

When a customer wants to open a savings account with a high interest rate and a cheque-writing facility, the customer can ask the service offer if it satisfies the requirement “`Interest_Rate > 10.00` and `Cheque_Book`”. In this example, the banking service would meet the customer’s requirements.

ServiceOffer Interface

```
typedef string ErrorMessage;
typedef string Expression;

exception InvalidExpression {ErrorMessage error_message;};

interface ServiceOffer: CosPropertyService::PropertySetDef
{
    any evaluate_expression (in Expression expr)
        raises (InvalidExpression);
};
```

Language for Expressions

At a minimum, the expression language should support:

- property names and constant values
 - comparison operators (e.g. =, !=, <)
 - logical connectives (e.g. and, or, not)
 - normal precedence and grouping (e.g. ())
- as described in Appendix 5B of the Life Cycle Service

Examples

- “Interest_Rate > 10.00 and Cheque_Book”
- “max(Interest_Rate)”

Properties of a ServiceOffer

- its service type
- service_Type of type Trading: :ServiceType
- the properties defined by the service type
 - with the same name, type, and mode
 - with the same value if the property is fixed_readonly in the ServiceType

which include a pair of properties for each interface:

Property Name	Property Type	Property Mode
Interface_Refsuffix	Object	fixed_normal
Interface_Typesuffix	RepositoryId	fixed_readonly

- other properties if permitted by implementation

Attributes as Properties

ServiceOffer interface permits properties to have values which are attributes of another interface

Such properties are if type Attribute

```
typedef struct Attribute_s {
    string attrib_name;
    TypeCode attrib_type;
    Object attrib_if;
} Attribute;
```

Attributes as Properties

example/diagramme whatever?

Service Types

A service type is used to define the basic set of properties needed to describe a particular kind of service.

Usage

- **exporters:**
create service offers from from service types
- **importers:**
to restrict the trader's search and select operations to only those ServiceOffers that conform to the nominated Service-Type
- **traders:**
to partition its ServiceOffer references for greater search efficiency

Service Types

ServiceType interface inherits from the PropertySetDef

Specific advantages of inheriting the PropertySetDef interface:

- default values for properties,
- pre-definition of certain property values
(by making them `fixed_readon1y`)

Service Type for Savings Account Factory

Property Name	Property Type	Initial/Default Value (if any)	Property Mode
Interface_Ref	SavingsAccountFactory (CORBA interface type)	nil	fixed_normal
Interface_Type	RepositoryId (string)	"IDL:Bank/SavingsAccountFactory:1.0"	fixed_readonly
Bank_Name	string		fixed_normal
Interest_Rate	float	0.0	fixed_normal
Cheque_Book	boolean	false	fixed_normal

ServiceType Interface

```
interface ServiceType: CosPropertyService::PropertySetDef
{
    boolean matches (in ServiceType st)
        raises (InvalidServiceType);
};
```

matches compares the ServiceType st with this ServiceType to determine if st is substitutable for this ServiceType

true if the ServiceType st is substitutable for this ServiceType.

false if the ServiceTypes are not substitutable.

InvalidServiceType exception

is raised if any problems arise while accessing st's interface.

Substitutability of Service Types

ServiceType_A is substitutable for ServiceType_B every property in ServiceType_B must have a corresponding property in ServiceType_A with:

- the same name
 - a same property type or for interface types, the type of the property in ServiceType_A to be substitutable for the type of the property in ServiceType_B
 - a substitutable property mode (fixed_readonly is substitutable for fixed_normal)
 - the same value if the property is fixed_readonly in ServiceType_B
- If the property value is an interface type (i.e. the property type is RepositoryId), then it is sufficient for the value of the property in ServiceType_A to be substitutable for the value of the property in ServiceType_B.

Notes

Properties of a ServiceType

ServiceTypes must contain a pair of Interface_Refsuffix and Interface_Typesuffix properties for each interface of the service

The ServiceType provides default values for properties; ServiceOffer may or may not be able to override the default values depending on the mode. Therefore, the modes of the properties of a ServiceType (which are copied to the ServiceOffers) must be either:

- fixed_normal
Allows a ServiceOffer to override the default value specified in the ServiceType, but does not allow the property to be deleted. Most properties of a ServiceType have this mode.
 - fixed_readonly
Prevents a ServiceOffer from overriding the value specified in the ServiceType and prevents the property from being deleted. The Interface_Typesuffix properties are predetermined in this manner in the ServiceType.
- Other modes (which permit deletion) are not used in ServiceTypes.

ServiceTypeFactory Interface

ServiceTypes can be created using the ServiceTypeFactory interface.

```
interface ServiceTypeFactory
{
    ServiceType create_servicetype ();
    servicetype create_constrained_servicetype (
        in CosPropertyService::PropertyTypes allowed_property_types,
        in CosPropertyService::PropertyDefs allowed_property_defs)
        raises (CosPropertyService::ConstraintNotSupported);
    ServiceType create_initial_servicetype (
        in CosPropertyService::PropertyDefs initial_property_defs)
        raises (CosPropertyService::MultipleExceptions);
};
```

ServiceTypeFactory Interface

create_servicetype

returns an empty ServiceType interface
it contains no properties and no property
constraintscreate_constrained_servicetype

create_constrained_servicetype

returns an empty ServiceType interface with constraints

- if allowed_property_types is not empty, then the returned ServiceType can only contain properties of the types included in allowed_property_types. If allowed_property_types is empty, then any property type is permitted.

- if allowed_property_defs is not empty, then the returned ServiceType can only contain properties (names, types, and modes) defined in allowed_property_defs.

The exception ConstraintNotSupported (from the Property Service) is raised if allowed_property_types or allowed_property_defs contain any invalid names, types, or modes.

CRC for Distributed Systems Technology

OMG Tokyo meeting

November 1995

Trading Service

25

ServiceTypeFactory Interface

create_initial_servicetype

creates an initialised ServiceType with the initial set of
properties described by initial_property_defs.

MultipleExceptions exception (from the Property Service) is
raised if any of the initial_property_defs have a property
name, type, or mode not supported by the implementation.

ServiceOfferFactory Interface

The ServiceOfferFactory interface creates ServiceOffers from a
nominated ServiceType.

```
exception InvalidServiceType {ErrorMessage error_message;};  
  
interface ServiceOfferFactory  
{  
    ServiceOffer create_serviceoffer (  
        in ServiceType st)  
        raises (InvalidServiceType);  
  
    ServiceOffer create_initial_serviceoffer (  
        in ServiceType st,  
        in CosPropertyService::PropertyDefs initial_property_defs)  
        raises (CosPropertyService::MultipleExceptions,  
            InvalidServiceType);  
};
```

CRC for Distributed Systems Technology

OMG Tokyo meeting

November 1995

Trading Service

27

CRC for Distributed Systems Technology

OMG Tokyo meeting

November 1995

Trading Service

26

CRC for Distributed Systems Technology

OMG Tokyo meeting

November 1995

Trading Service

26

ServiceOfferFactory Interface

`create_serviceoffer`

creates a default `ServiceOffer`. The new `ServiceOffer` contains:

- all of the properties of the `ServiceType st` with the default values provided by `st`
- a property named `service_type` of type `Trading::ServiceType` which references the `ServiceType` interface from which this `ServiceOffer` was created

The `InvalidServiceType` exception is raised if any problems arise while accessing `st`'s interface

Notes

CRC for Distributed Systems Technology

OMG Tokyo meeting

November 1995

Trading Service

30

ServiceOfferFactory Interface

The `MultipleExceptions` exception (from the `Property Service`) is raised if any of the `initial_property_defs`:

- attempts to override a read-only property (as defined by the `ServiceType st`)
- has a property value of the wrong type (as defined by the `ServiceType st`)
- has a property name, type, or mode not supported by the implementation

The `InvalidServiceType` exception is raised if any problems arise while accessing `st`'s interface.

CRC for Distributed Systems Technology

OMG Tokyo meeting

November 1995

Trading Service

31

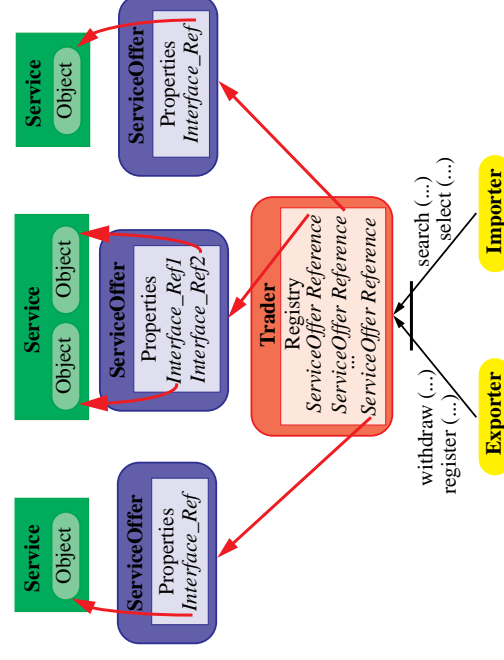
ServiceOfferFactory Interface

`create_initial_serviceoffer`

creates an initialised `ServiceOffer` containing

- all of the properties of the `ServiceType st` with the default values provided by `st`, augmented by `initial_property_defs` which either override property values in the `ServiceType` (where the mode permits), or define additional properties (if the implementation permits)
- a property named `service_type` of type `Trading::ServiceType` (a particular `Object` type) which references the `ServiceType` interface from which this `ServiceOffer` was created

A Trader, its ServiceOffers and their Services



CRC for Distributed Systems Technology

OMG Tokyo meeting

November 1995

Trading Service

32

Export Interface

The Export interface provides operations to allow a ServiceOffer to be registered and deregistered with a trader.

```
typedef sequence <octet> UniqueId;
typedef UniqueId ExportId;
exception ServiceOfferDoesNotExist {};
exception InvalidServiceOffer {ErrorMessage error_message;};
interface Export
{
    ExportId register (in ServiceOffer so)
        raises (InvalidServiceOffer);
    void withdraw (in ExportId export_id)
        raises (ServiceOfferDoesNotExist);
};
```

CRC for Distributed Systems Technology

OMG Tokyo meeting

November 1995

Trading Service

33

Export Interface

register

registers the ServiceOffer so with this trader
returns an ExportId, required to identify so within this trader
raises InvalidServiceOffer exception if the ServiceOffer so is invalid, e.g. if so is nil.

withdraw

deregisters a ServiceOffer identified by export_id from this trader.
raises ServiceOfferDoesNotExist exception if export_id does not identify a currently registered ServiceOffer.

The internal content of the ExportId is an implementation decision.

CRC for Distributed Systems Technology

OMG Tokyo meeting

November 1995

Trading Service

34

Import Interface

The Import interface enables an importer to find ServiceOffers registered with this trader (and possibly other traders) that match the importer's particular requirements.

```
typedef string MatchingConstraints;
typedef string SelectionCriteria;
typedef sequence <ServiceOffer> ServiceOffers;
exception InvalidServiceType {Error error_message;};
exception InvalidConstraintExpression {ErrorMessage error_message;};
exception InvalidSelectionExpression {ErrorMessage error_message;};
```

CRC for Distributed Systems Technology

OMG Tokyo meeting

November 1995

Trading Service

35

Import Interface

```
interface Import
{
    void search (
        in ServiceType st,
        in MatchingConstraints matching_constraints,
        in unsigned long how_many,
        out ServiceOffers so_list,
        out ServiceOfferIterator rest)
        raises (InvalidServiceType,
              InvalidConstraintExpression);
    void select (
        in ServiceType st,
        in MatchingConstraints matching_constraints,
        in SelectionCriteria selection_criteria,
        out ServiceOffer so)
        raises (InvalidServiceType,
              InvalidConstraintExpression,
              InvalidSelectionExpression);
};
```

CRC for Distributed Systems Technology

OMG Tokyo meeting

November 1995

Trading Service

36

Import Interface

search

returns all the ServiceOffers registered with this trader (or other traders linked by federation) that are of ServiceType *st* (or of substitutable ServiceType) and satisfy the *matching_constraints* expression.

A sequence *so_list* of at most *how_many* matching ServiceOffers is returned

the remainder of the matching ServiceOffers are returned via *rest*, a ServiceOfferIterator interface *rest*. If there are no ServiceOffers remaining to be returned via the ServiceOfferIterator interface, then *rest* is nil.

raises `InvalidConstraintExpression` exception is raised if *matching_constraints* is not a valid expression.
raises `InvalidServiceType` exception if the ServiceType *st* is invalid.

CRC for Distributed Systems Technology

OMG Tokyo meeting

November 1995

Trading Service

37

Import Interface

select

returns a single ServiceOffer of ServiceType *st* (or of a substitutable ServiceType) that satisfies the *matching_constraints* and best satisfies the *selection_criteria*.

is returns a nil reference only if no matching ServiceOffer can be found.

raises `InvalidConstraintExpression` exception if *matching_constraints* is not a valid expression.

raises `InvalidServiceType` exception if the ServiceType *st* is invalid.

raises `InvalidSelectionExpression` exception if *selection_criteria* is not a valid selection expression.

It is an implementation decision whether to use trader federation to import ServiceOffers

CRC for Distributed Systems Technology

OMG Tokyo meeting

November 1995

Trading Service

38

ServiceOfferIterator Interface

The ServiceOfferIterator interface:

- enables an importer to process a potentially large list of ServiceOffers (returned by the *search* operation)
- adopts the design of the iterators in other object services

```
typedef sequence <ServiceOffer> ServiceOffers;  
interface ServiceOfferIterator  
{  
    void reset ();  
    boolean next_one (out ServiceOffer so);  
    boolean next_n (  
        in unsigned long how_many,  
        out ServiceOffers so_list);  
    void destroy ();  
};
```

CRC for Distributed Systems Technology

OMG Tokyo meeting

November 1995

Trading Service

39

ServiceOfferIterator Interface

reset

resets this iterator to its first ServiceOffer, if one exists.

next_one

returns the next ServiceOffer in this iterator in the out parameter *so*.

returns true if there is a ServiceOffer to return.

false signifies the end of ServiceOffers in the iterator and that *so* does not hold a meaningful value.

next_n

returns the next *how_many* ServiceOffers in this iterator (or as many as remain) in the out parameter *so_list*.

returns true if *so_list* is not empty

returns false if *so_list* is empty

destroy

destroys the iterator.

CRC for Distributed Systems Technology

OMG Tokyo meeting

November 1995

Trading Service

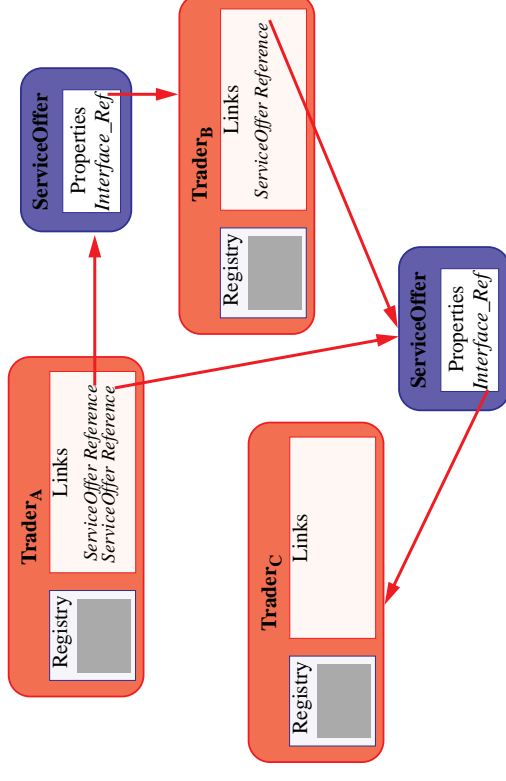
40

Trader Federation

In large distributed systems, there might be many trader objects. Although the importer could individually import ServiceOffers from many individual traders, it is more convenient for the importer to interact with a single trader that accesses other traders on its behalf. In a trader federation, traders act recursively as clients to other traders.

A trader federation is a set of traders with links between them. A link from Trader_A to Trader_B means that import operations performed at Trader_A can propagate to Trader_B, but not vice versa. A link is a description of a trading service provided by another trader; that is, a link can be realised as a ServiceOffer which describes a trader.

Trader Federation and Links



Interfaces for Trader Federation

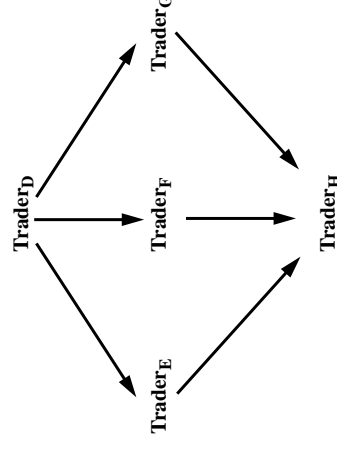
- LinkManagement interface provides operations to
 - add links to other traders and
 - remove those linksfor the use of trader administrators

Import operations can propagate through the trader federation

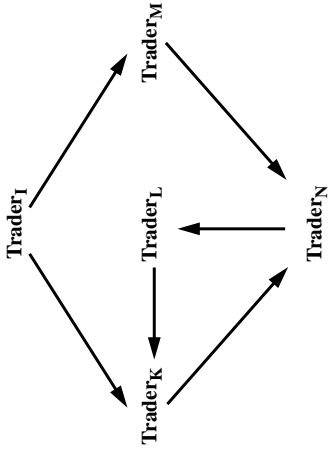
- Federation interface provides specialised import operations which detect an import operation that has already been processed at this interface

The trader inherits the LinkManagement and Federation interfaces.

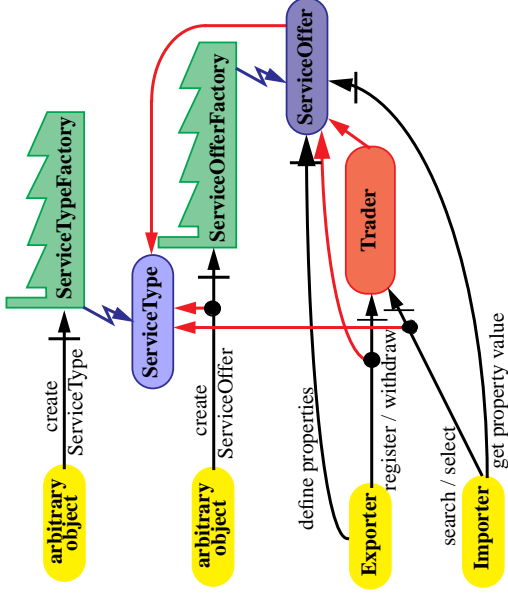
Multiple Paths in the Trader Link Graph



Loops in the Trader Link Graph



A Typical Trading Scenario



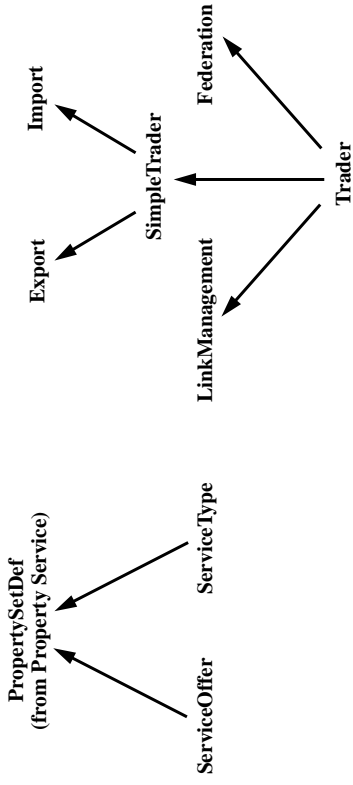
Interfaces of the Trading Service

Interface	Purpose	Primary Client(s)
ServiceOffer	Describes a service using strongly-typed properties and evaluates expressions involving those properties	Exporters define properties; importers query properties; traders evaluate expressions
ServiceType	Defines the basic set of properties for a particular kind of service	ServiceOfferFactories create ServiceOffers from ServiceTypes; exporters and importers use ServiceTypes
ServiceOfferFactory	Creates ServiceOffer objects	Exporters
ServiceTypeFactory	Creates ServiceType objects	Any object

Interfaces of the Trading Service

Interface	Purpose	Primary Client(s)
Export	Manipulate the trader's registry of ServiceOffers	Exporters
Import	Choose suitable ServiceOffers	Importers
ServiceOfferIterator	Returns suitable ServiceOffers singly or in small groups	Importers
LinkManagement	Manipulate the links between federated traders	Trader administrators
Federation	Choose suitable ServiceOffers through propagating import operations along the links of a trader federation	Trader objects
SimpleTrader	Inherits Export and Import	as for base interfaces
Trader	SimpleTrader, LinkManagement, and Federation combined	as for base interfaces

Interface Inheritance Hierarchy



Bootstrapping: Finding the Trader

The ORB Interface specifies an operation available in the CORBA : :ORB interface called `resolve_initial_references` which is used to resolve a set of well-known objects (service name strings) into references for fundamental interfaces that ORB users require, such as the Interface Repository and the Naming Service.

For this Trading Service specification, the following table provides identifiers and interface types to be supported by the `resolve_initial_references` operation.

Identifier	Interface Type
"Trader"	Trading::Trader
"ServiceTypeFactory"	Trading::ServiceTypeFactory
"ServiceOfferFactory"	Trading::ServiceOfferFactory

As the Trader interface is derived from the base interfaces of Simple Trader (comprising Exporter and Importer),

LinkManagement, and Federation, the initial reference to the Trader interface also gives access to all of these base interfaces.

Why not use the ODP Trading Function IDL ?

The ODP Trading Function IDL is inappropriate for adoption by OMG because:

- the goals of the ODP Trading Function are different to the goals of the Object Services
- the ODP Trading Function uses CORBA IDL as an architecture-neutral notation and does not imply
 - that implementations must use CORBA
 - that the standard is appropriate for adoption by OMG

Why not use the ODP Trading Function IDL ?

This Trading Service specification was developed:

- by adopting the concepts and essential functions of the ODP Trading Function
- by devising a set of interfaces that are consistent in approach and style with CORBA and existing Object Services
- to enable mappings between this Trading Service specification to the ODP Trading Function specification, enabling interworking between CORBA and non-CORBA systems

Why not use the ODP Trading Function IDL?

This Trading Service specification is based on the concepts of the ODP Trading Function and provides the essential trading services specified by the ODP Trading Function.

- Object Service goals
- be object-oriented
 - re-use existing OMG object services
 - be conformant with the OMA
 - be consistent with OMG style and conventions

Specification Use of other OMG Services

This Trading Service specification uses the Property Service in two ways:

- the ServiceOffer interface and ServiceType interface inherit the PropertySetDef interface of the Property Service
- the ServiceTypeFactory interface adopts the design of the PropertySetDefFactory interface of the Property Service

This Trading Service specification uses the Interface Repository in two ways:

- the RepositoryId type is used for describing interface types in service offers and service types
- the Interface Repository to determine interface substitutability in ServiceType matching

No extensions to the OMA, CORBA, or any object service are required.

Implementation use of other object services

Implementations of this Trading Service specification might use:

- the Event Service to trigger the register/withdrawal of service offers when the corresponding service becomes available/unavailable
- the Externalisation and/or Persistence Services to make service offers, service types, and traders persistent
- the Life Cycle Service for creating, copying, and deleting objects
- the Naming Service to assign names to service types and traders
- the Query Service to evaluate matching constraints and selection criteria to find suitable service offers for importers
- the Relationship Service to represent the graph of links between traders
- the Security Service to provide access control, authentication, and auditing
- the Time Service to limit the extent of federated operation

Security Issues

Assumptions:

- that objects are encapsulated
- that the client identity (and other security information) can be implicitly obtained by a server object during an invocation
- that security-failure exceptions are implicitly raise-able by all operations

The Trading Service cannot judge the accuracy of the service description contained in a service offer; this is the responsibility of the exporter. Audit trails could be used to identify the exporter of a service offer and any importers of that service offer to support investigations or corrective actions involving erroneous service offers. To achieve this, exporters and importers must be principals (authenticated entities).

Object encapsulation is fundamental to security; no object can be secure if there are mechanisms by which the state or actions of an object are manipulated other than those defined by the interface behaviour.

It is not desirable to have security parameters visible in the operations at the IDL level (except for interfaces that form part of the Security Service). However, they might be visible as additional parameters at the language-binding level.

Again, it is not desirable to have security exceptions visible in operations at the IDL level (except for interfaces that form part of the Security Service). However, they should be available in the same manner as CORBA standard exceptions.

The following table indicates the likely access rights that would be assigned to different kinds of principal with respect to different interfaces and operations. Here, "principal" means an authenticated entity or any delegate of it (perhaps via a capability). Particular implementations and configurations might deliberately assign greater or lesser rights to reflect their particular enterprise goals.

Security Issues

Interface	Operation	Principal
ServiceTypeFactory	create a service type	Any object
ServiceType	"write" operations, e.g. define_property, delete_property	Its creator
	"read" operations, e.g. get_property_value, is_property_defined	Service offer factories need to "read" the ServiceType; typically, any object can "read" a ServiceType
	matches operation	A trader needs to match service types; typically, any object can do so

Security Issues

Interface	Operation	Principal
ServiceOfferFactory	create a service offer	Any object
ServiceOffer	"write" operations, e.g. define_property, delete_property	Its creator
	"read" operations, e.g. get_property_value, is_property_defined	An importer needs to "read" the service offer; typically, any object can do so
	evaluate_expression	A trader needs to evaluate expressions on service offers exported to that trader; typically, any object can do so

Security Issues

Interface	Operation	Principal
Exporter	export operation	A creator of a service offer needs to export it; typically, any object can export a service offer
	withdraw operation	The exporter of that service offer
Importer	all operations	Any object
ServiceOfferIterator	all operations	Importer which invoked the search operation that created the ServiceOfferIterator

Security Issues

Interface	Operation	Principal
LinkManagement	all operations	Trader administrators
Federation	all operations	Traders which have links to this Federation interface
SimpleTrader		as for the base interfaces
Trader		as for the base interfaces

Summary

[Something needed here]

Submission Contact Point

Enquiries and comments about this submission are most welcome, and should be directed to:

Kerry Raymond

CRC for Distributed Systems Technology

University of Queensland

Brisbane 4072

Australia

Phone: +61 7 3365 4310

Fax: +61 7 3365 4311

Email: kerry@dstc.edu.au, omg-trader@dstc.edu.au

WWW: <http://www.dstc.edu.au/CORBATradingService>

What we left out

no naming - use the Naming Service

no contexts - use federation plus naming

trader works with service offers and not properties