An object-oriented description of services in a distributed system

C. Popien, A. Kuepper
Aachen University of Technology, Department of Computer Science IV
Ahornstr. 55, D-52056 Aachen, Germany, Phone: +49(241)8021415,
Fax: +49(241)8888 220, popien@informatik.rwth-aachen.de

Abstract

This paper studies the requirements of services in a distributed system. It examines management possibilities for describing the service trading scenario according to the computational viewpoint of the Open Distributed Processing (ODP) Reference Model. Because of similar architectures and properties ODP services, service offers, types, exporters and traders are mapped onto management components and modelled as managed objects. Therefore, the Guidelines for the definition of Managed Objects (GDMO) are used. The concept allows a precise study of the service trading scenario and provides means for exporting and importing of service offers in a distributed environment.

1 INTRODUCTION

For global networks of interconnected computers, new services to manage resources of distributed systems are needed. A growing number of components offering services leads to an open service market. The client's problem to select a service arises. Therefore, the client/server model is extended to a three-party model: the importer/exporter/trader model.

The tasks of the ODP trader are divided into domain and type management, respectively. Domain management supports finding a service, type management supports interface matching. While existing papers like [Sl 90] and [Dr 92] do not consider these topics, realization of type management is independend from management concepts, but there are suitable relations between the object-oriented modelling techniques of ODP and the managed object based description methods. It is the aim of this paper to study this relationship between management and ODP in order to get a formal basis for describing a service trading scenario.

The paper is structured as follows. The second chapter gives an overview about the requirements of service trading described in the ODP Reference Model. The third chapter discusses management concepts defined in the management standards. The ODP properties and requirements are compared with available techniques in GDMO. Subsequently, the fourth section formally specifies some ODP components. In order to operate on service offers ODP enables export and import of services. Finally, the fifth section derives the conclusions and mentions some open questions.

2 SERVICES IN DISTRIBUTED SYSTEMS

An ODP trader is an object that performs trading, primarily satisfying identification requirements [ODP Tr], [MaBl 92]. A trader can be seen as an object from which another object can buy (import) its needs and sell (export) its services in a distributed environment.

In order to study the process of service trading in more detail some notions are introduced. A *service* is a function provided by an object at a computational interface. It is a set of capabilities available at an interface of an object. Every service is an instance of a *service type*. Associated with each service type is an *interface type*, which determines the computational behaviour. Interfaces of the same type provide the same functionality. However, instances of the same service type may differ in some noncomputational aspects. These additional aspects are called *service properties.*

In the trading context, service properties can be classified as either static or dynamic. A *static service property* is a property that changes infrequently. The values are asserted by an exporter in a service offer and are stored in the *service directory* (SD) of the trader, see Fig. 1. A *dynamic service property* is a property whose values change much more frequently than the rate at which the trader is asked to perform a match based on that property value. Values of dynamic properties are obtained on demand by the trader. A *service offer* describes a service that is being traded. It is an assertion made by an exporter about a service that is offered for use by other objects at a computational interface. In addition to service type and service property values, a service offer can also have *service offer property values.* Considering the trader in more detail, it performs two major functions: the *domain management function* and *type management function.*

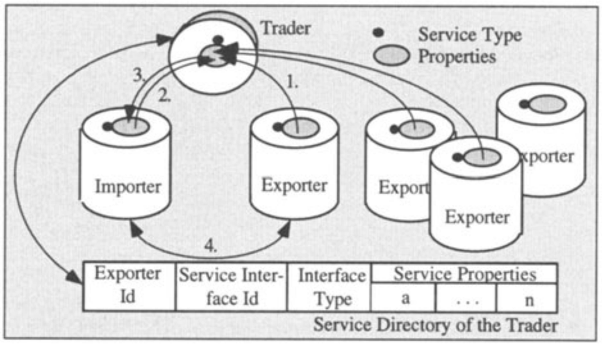

Figure 1 Service trading in distributed systems

The type management function realizes the management of the subtype relationship between types. In more detail, the set of all service types known to the trader is organized into a service type hierarchy. The service type hierarchy is represented as a directed acyclic graph, called *interface offer graph,* in which each node is a service type and each directed edge represents supertype to subtype relationships.

The domain management function manages the *service offer space.* The service offer space within a trader may be structured into sets called contexts. A context structure is defined by the containment relationship between contexts. A trader context structure can be represented as a directed acyclic graph, called *trading context graph,* with the nodes representing the trading contexts, and arcs representing the containment relationship.

Considering the description of a service offer, it is possible to derive a structure for service offers of an importer. A service offer consists of *exporter Id* (Eid), *service interface Id* (SII), *service type description* and *service offer properties,* where a service type description can be further split into an *interface type description* (IT) and a*service property type description.* Service properties are *static service properties* (SSP) or *dynamic service properties* (DSP), respectively, described as ordered pairs

<service property name, service property value>.

If a local trader fails to return a service offer to its clients it can establish an *import contract* with a remote trader. This trader acts as exporting trader and has to establish an *export contract.* The new system is called *trading federation* [BeRa 91], [PoMe 93].

In large distributed systems, changes of configurations are likely to appear quite often. Therefore, the behaviour of management and administration services must be flexible to cope with these dynamically changing configurations and requirements. In order to achieve a flexible object behaviour, it has to be determined by a policy.

3 GDMO AND SERVICES IN DISTRIBUTED SYSTEMS

Managing the services in a distributed system requires powerful concepts for efficient service management. First approaches of service management have been presented in [PoKM 94] and [PoKu 94].

To establish a service management, some functionality can be yield from the Open Systems Interconnection (OSI) respective standards [OSI MF, MI, MO, SM]. This holds for instance for the model of *managed objects* (MO) and its specification language Guidelines for the Definition of Managed Objects (GDMO) which is applied to ODP architectures in the following. Subsequently, a management scenario is created that demonstrates the abstraction of a whole ODP environment by MOs [RPB 93].

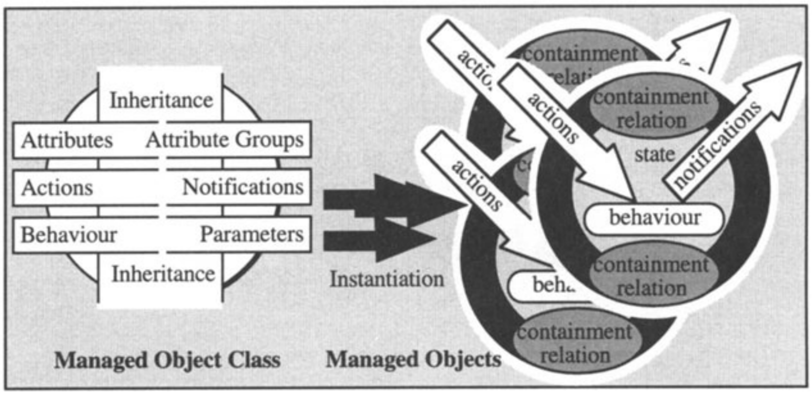

Figure 2 Deriving MOs from its common class

The use of management functions presumes information about the resources that are to be managed. This holds particulary for the configuration management [Pa 91]. This information is contained in MOs which are characterized by their respective MO class as can be seen in Fig. 2. A MO is an abstract view of a resource which can be a hardware or a software component. According to the OSI management framework, a MO consists of several elements like attributes to represent the resource's state and properties, actions to handle the resource, notifications for the generation of reports recording the occurence of particular events or confirming operations, and its relationship to other MOs. These properties of a MO are given by several templates that establish the MO class.

In the following, a management scenario for an ODP environment will be established, subdivided into an exporter and a trader. The exporter in turn is subdivided into several functional units, called *service type database*, to store the exporter's offered service types, the *service entity* which contains the services and performs them, the *service assignement* which assigns the incoming importers to their required services, and a *service supplier* which is responsible for establishing service offers in the service directory. The service directory is the most important entity of the trader and contains all service offers of the related ODP environment.

The MOs, as well as their classes, are arranged hierarchically. MOs are arranged in a containment relation, whereas the hierarchy between classes is called inheritance relation. The containment relation is useful for abstracting real world hierarchies like the ODP environment.

A MO can contain several MOs, but it is contained in exactly one MO, except for the MO establishing the top of the containment tree. Moreover, the containment relation is responsible for the naming of MOs. Each MO has an identifier called *relative distinguished name* (RDN). The RDN of a MO must be distinct from the RDNs of all other MOs below the same, superior MO. The *distinguished name* (DN) of a MO formed by combinig its RDN with the DN of the superior MO. The containment relation is realized by a name binding template.

The inheritance relation is a class hierarchy and enables a class to take over all properties from the superior classes. It is possible to manipulate these properties in a restrictive way or to add further elements, but not to delete the derived properties.

This architecture, including service types, services and service offers, should be mapped onto MOs and reproduced by the containment relation which is shown in Fig. 3. This is done by defining a particular MO class for each of these components. The inheritance relation between these MO classes is illustrated at the top. In our example, the MO class `ServiceOffer` takes over properties from `Service`. `Service` is derived from `ServiceType`, and all MO classes are subordinated to `ODPTop`, which is a generic class, that is a class which is not instantiated. It is useful for specifying elements that have to be included in all MOs of its subordinated classes. For example, each MO has to possess an attribute which contains its RDN. By specifying this attribute in the MO class `ODPTop` and arranging it at the top of the inheritance tree, all subordinated classes automatically contain this attribute.

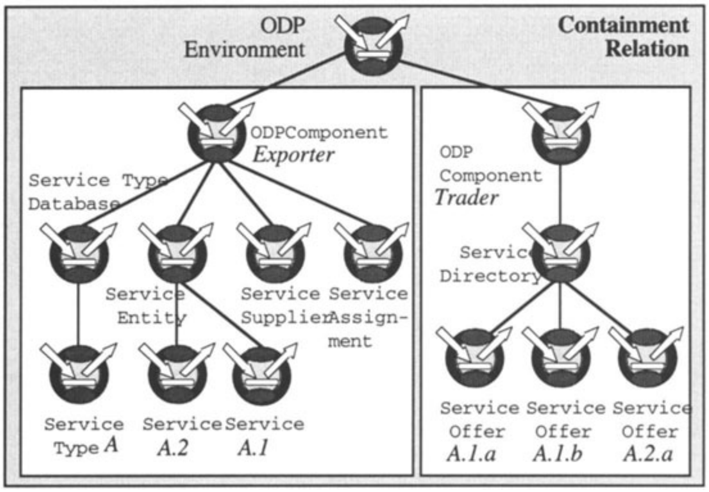

Figure 3 Containment relation

By defining special operations on MOs it is possible to realize the functions of an exporter. MOs which represent a service type have to possess a function for instantiating it, i.e. for creating a service. The MO of a service must contain a function `destroy` for eliminating a service and `export` for creating a service offer and establishing it in the service directory of the trader. Furthermore, the MO of a service offer realizes its manipulation by the function `replace` and its elimination by the function `withdraw`. The functions `export`, `withdraw`, and `replace` are compatible with the given functions of the ODP-RM.

The attributes of the MOs representing a service offer are subdivided into attributes describing static properties and those representing dynamic properties. For example, static properties of a service offer are the offer identifier and the associated service type; dynamic properties contain the state of the associated service. The actual state of a service is realized by three attributes called operational state, usage state and administrative state. The first one shows, if the concerning resource is installed and ready for working. The second indicates the frequence of using the service, whereas the third enables the locking of the service, for example to install a new version.

```
<template-label> TEMPLATE-NAME
CONSTRUCT-NAME [<construct-argument>];
[CONSTRUCT-NAME [<construct-argument>];]*
[REGISTERED AS <object-identifier>];
[supporting productions
[<definition-label>  -> [<syntatic definition>]*]
```

Figure 4 Structure of a template

Notifications realized in the MO of the service can report the change of one of these state attributes. Changes of the operational state or usage state are automatically sent to the responsible management process, whereas the change of the administrative state executed by a management process is transmitted as a confirmation.

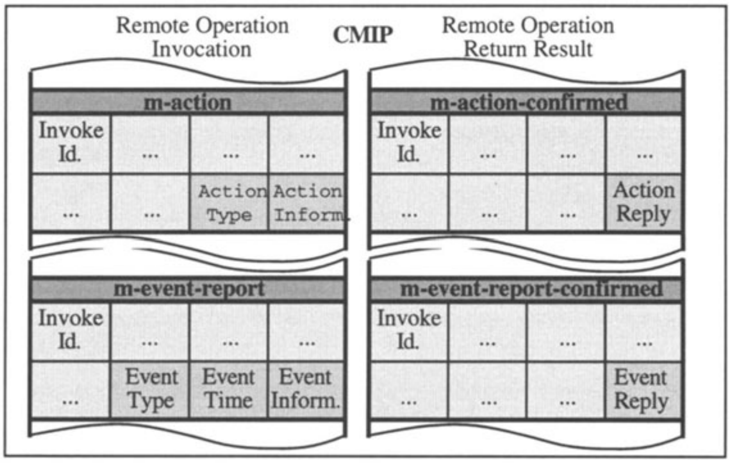

Figure 5 Parameter fields in a management PDU

The Guidelines for the Definition of Managed Objects is a specification language for the definition of MOs. The properties of a MO are given by its MO class that is composed from several templates. Each template is a specification of a template structure whose typical construction is shown in Fig. 4. It consists of a template label combined with the name of a template structure, at least one construct, identified by `CONSTRUCT-NAME`, a non-ambigious object name for registration and supporting productions. `template-label` is used as a local reference in other templates, whereas `object-identifier` allows the identification in a global open system. Therefore, it has a hierarchical structure which contains the name of the global system as the root of the naming tree and the identifiers of the subordinated entities as the nodes. The constructs are used for the defintion of the classes' properties. The supporting productions, if available, allow a more detailled specification of one of the constructs. Furthermore, a construct permits referencing to other templates.

Data types are required for the transmission of management information by a management protocol. Therefore, several template structures enable the reference to a module of another specification language, Abstract Syntax Notation One (ASN.1).

The Common Management Information Service (CMIS) offers several functions for handling MOs. For example, these functions allow changing of attributes' values, creation or deletion of MOs, etc. The CMIS functions and their parameters are mapped to the Common Management Information Protocol (CMIP) which enables the transmission of management information between two management processes. Fig. 5 shows some parameter fields of the action and event-report operation in CMIP. Action enables the execution of a resource-specific function described on a MO by the action template. The function is identified by the field action type, their parameters are contained in action information and action reply. Event-report is used for the transmission of notifications specified by a notification template of a MO. The handling of the function in the management protocol is analogous to action.

The ODP-RM defines several requirements that specification languages have to fulfill. Most of them are taken from object-oriented languages. Fig. 6 shows a comparison between the most important concepts of ODP and their equivalents in GDMO.

Requirements of ODP	Available technique in GDMO
Objects	Managed Objects
Templates	Managed Object Class Template
Types and Classes	Managed Object Class
Subtyping and Subclassing	Indirectly available
Incremental Inheritance	DERIVED FROM
Composition	Containment Relation

Figure 6 Comparison of ODP requirements concerning specification languages with GDMO

In ODP all entities are modelled as objects. An object is characterized by its behaviour, its state and its relationship to other objects. The concepts of behaviour, state and relationship to other objects are interrelated. For example, the actual state of an object is determined by its behaviour and the influences of other objects. An object is described by its template which specifies features like operations or data types. Similar objects can be described by the same template. An object is of a certain type, if it posseses the properties described by this type. All objects of a common type build a set, called class.

The concept of a 'type' and a 'class' are equivalent. Moreover, objects need not be similar to be of the same type. This fact implies a hierarchy called subtyping or subclassing. One type is a subtype of another one exactly if, for all objects, satisfaction of the first type implies satisfaction of the second type. Thus, the predicates of the first type build a subset of those of the second one. Another hierarchy is the incremental inheritance. Inheritance means that one template can derive elements from another template. Incremental inheritance enables the modification of these elements which can be of any kind, for example including, deleting or manipulating properties. In some cases the incremental inheritance may be equivalent to the concept of subtyping but in general these two hierarchies are not the same thing. With the concept of composition and decompostion it is possible to put together several objects into a bigger unit or to divide them into smaller ones.

4 SPECIFYING SERVICES OFFERS

In the following, a GDMO specification of the MO classes `ServiceOffer` is given. Fig. 7 shows the corresponding MO class templates. Generally, a class can inherit its properties from several classes. `ServiceOffer` has a package providing attributes, notifications, and actions concerning the mentioned state. `CHARACTERIZED BY` enables the establishment of one or several packages. In this case, the corresponding package shall be instantiated only in the specified class, not in its subclasses.

An exporter which offers several services has to advertise them to make them available in an ODP-environment. For this purpose, the exporter has to place an associated service offer in the service directory of the trader. This is done by the operation `export` which has to be performed on a MO of the MO class `Service`. This operation has to be extended with parameters which put together the service offer.If the execution of the service was successful, the exporter gets an offer identifier as a confirmation of `export`, otherwise an error code is returned. Furthermore, the exporter has the ability to remove the service offer by the operation `withdraw` or to manipulate it by `replace`.

```
ServiceOffer MANAGED OBJECT CLASS
DERIVED FROM    ODPTop;
CHARACTERIZED BY

    staticProperties     PACKAGE
          ATTRIBUTES     InterfaceId                   GET-REPLACE,
                         Interface                     GET-REPLACE,
                         ServiceInterfaceId            GET-REPLACE,
                         ServiceOfferId                GET-REPLACE,
                         ContextId                     GET-REPLACE,
                         ServiceTypeDescription        ADD-REMOVE,
                         ServiceProperties             ADD-REMOVE,
                         ServiceOfferProperties        ADD-REMOVE,
    REGISTERED AS {...}

    dynamicProperties    PACKAGE
          ATTRIBUTES     operationalState              GET,
                         usageState                    GET,
                         administrativeState           GET;
    REGISTERED AS {...}

    ServiceOfferAction   PACKAGE
          ACTIONS        replace, withdraw;
    REGISTERED AS {...}

REGISTERED AS {...}
```

Figure 7 Specification of the MO class `service offer`

Import operations are realized by a formal language called Service Request Description Language (SRDL) [PoMK 94]. Interpreting the service request is equivalent to reading the attributes of a Managed Object. This is done by the operation `get`. `get` requires an invoke identifier, the object class, and a corresponding object instance. It returns the invoke identifier. This identifier is used to evaluate the formal semantics of the `search` and `select` operations.

6 CONCLUSIONS

It has been shown that Open Distributed Processing has a greater functionality than the management theory requires it. However, it is possible to explain all properties of ODP with constructs of the management service. The result of this paper is the possibility of a formal description of ODP service trading architectures. Further work should consider the structure of a service in more detail.

Another possibility to enhance this formal model arises from the value-added service of service combining [PoHe 94]. By sequential performance of several services it could be possible to get a new service which has the requested interface type or properties. This possiblity should be included into the SRDL and studied from the management point of view.

Currently, the ODP Working Group of our Computer Science Department is busy with implementing this language in order to support the ANSAware trading realization. Based on tools supporting BNFs we have implemented a syntax checker for that language and are now working on the interface between the service directory and the processing of that language. Thus, one or more services can be searched or selected by specifying the conditions which are important for the user of distributed environments. So far, the ANSAware trader considers only the static service properties. Further work has to be done to include the dynamic service properties into the formal GDMO description presented here, in order to manage service trading.

REFERENCES

[BeRa 91] Bearman, M.; Raymond, K.: *Federating Traders: An ODP Adventure.* In: IFIP Workshop on Open Distributed Processing, Berlin 1991, North Holland

[Dr 92] Dreo, G. et al: *Using the OSI management information model for ODP.* In: Open Distributed Processing, Elsevier Science Publishers B.V., North Holland, 1992

[MaBl 92] Macartney, A. J.; Blair, G.S.: *Flexible Trading in distributed multimedia systems.* In: Computer Networks and ISDN Systems 25 (1992) 145-157, Elsevier Science Publishers B.V., North Holland, 1992

[ODP P1-4] ISO/IEC JTC1/SC21/WG 7 N885(preliminary): *Basic Reference Model of Open Distributed Processing - Part 1:4*, 1995

[ODP Tr] ISO/IEC JTC1/SC21 N8409: *Working Document - ODP Trading Function* Jul. 1994

[OSI MF] ISO/IEC 7498-4: *Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 4: Management framework*, 1989

[OSI MI] ISO/IEC JTC1/SC21 DIS 10165-1: *Information Technology - Open Systems Interconnection - Structure of Management Information - Part 1: Management Information Model*, Nov. 1991

[OSI MO] ISO/IEC JTC1/SC21 DIS 10165-4: *Information Technology - Open Systems Interconnection - Structure of Management Information - Part 4: Guidelines for the Definitons of Managed Objects*, Nov. 1991

[OSI SM] ISO/IEC JTC 1/SC 21 DIS 10164-2: *Information Technology - Open Systems Interconnection - Systems Management - Part 2: State Management Function*, Oct. 1991

[Pa 91] Park, H. J. et al: *Configuration management of object groups.* In: The Australian Computer Journal, Vol. 23, No. 4, Dezember 1991

[PMK 94] Popien, C.; Kuepper, A.; Meyer, B.: *A Formal Description of Open Distributed Processing (ODP) Trading* Vol.2, No. 4, 1994, pp. 383-400

[PoHe 94] Popien, C.; Heineken, M.: *Trading Enhancement by Service Combination in ODP.* In: Proceedings of the IFIP International Conference on Open Distributed Processing. North Holland Amsterdam London New York 1994 pp. 384 - 387

[PoKM 94] Popien, C.; Kuepper, A.; Meyer, B.: *Service Mangement - The new answer of ODP Requirements.* In: Proceedings of the IFIP International Conference on Open Distributed Processing. North Holland Amsterdam London New York 1994, pp. 408 - 411

[PoKu 94] Popien, C.; Kuepper, A.: *A Concept for an ODP Service Management.* In: IEEE/IFIP 1994 Network Operations and Management Symposium, Hyatt Orlando, Kissimmee, Florida, Febr. 14-17, 1994

[PoMe 93] Popien, C.; Meyer, B.: *Federating ODP Traders: An X.500 Approach.* In: Proceedings of the IEEE International Conference on Communications ICC'93, May 1993, Geneva, Switzerland, pp. 313 - 318

[RPB 93] Roos, J.; Putter, P.; Bekker, C.: *Modelling Management Policy using Enriched Managed Objects.* Proceedings of IFIP International Symposium on Integrated Network Management. North Holland 1993, pp. 207 - 215

[Sl 90] Sloman, M.: *Management for Open Distributed Processing.* Second IEEE Workshop on Future Trends of Distributed Computing Systems, IEEE Comp. Soc. Press, New York 1990