DEVELOPMENT OF A FORMAL SPECIFICATION OF THE ODP-TRADER

Joachim Fischer, Eckhardt Holz
Humboldt-Universität zu Berlin
Lindenstraße 54 a
D-10099 Berlin - Germany

{fischer|holz}@informatik.hu-berlin.de

This article describes the development of a formal specification of the ODP-Trader using the ITU-T specification language SDL´92 in combination with ASN.1. Due to the limited space (the complete specification totals more then 70 pages) only small parts of the specification can be given here. Therefore the emphasis of this article lies more on the description of the design decisions. Problems in the informal specification are pointed out and solutions are proposed. The most markable achievement is the introduction of a multilevel specification, each level being depend on the more abstract preceding level. generic placeholder components are used to encapsulate non-normative parts of the Trader specification as well as parts which will become subject of future ODP component standards. The complete SDL specification of the Trader is given in [8].

1 Trading - Overview

Trading is a required support function of ODP. It enables clients to choose appropriate servers (service providers). Trading is provided by a third-party object, the Trader, which links clients and servers in a distributed system.

- A Trader accepts service offers from exporters of services (servers), when a server wishes to advertise a service offer. A service offer contains the characteristics of a service that an exporter is willing to provide. Service offers are stored by the Trader in a centralized or distributed database.
- A Trader accepts a service request from importers of services (clients), when a client requests a service. A service request is an expression of service requirements made by an importer when a service is needed.
- A Trader searches its service offer database to match the importer's service request. If required, a Trader can select the most appropriate service offer (if one exists) that satisfies the importers service request. The matched list of service offers or the selected service offer is returned to the importer.

The matching and selection of an appropriate service at run time by a Trader allows to configure client objects into an ODP system without prior knowledge of server objects that provide the required services. After a successful match, the client can directly interact with the server object. An object can act as an exporter of one service and as an importer of another service, importer or an exporter can even be another Trader.

2 SDL as an ODP Description Technique

The ODP Reference Model Part 4 [4] identifies the four languages Lotos, SDL, Estelle and Z for the application as Computational and Information Language. Moreover, rules for their use are given and pros and cons are pointed out. Languages with object-oriented and generic concepts like SDL have a good chance to cover the most important ODP concepts in a structurally equivalent manner. The general guidelines to use SDL for modelling ODP concepts are defined by the ODP-BRM Part4.

The use of SDL for the as computational language and its application in the specification of the Trader showed the need for some enhancements:

- dynamic creation of composite objects (blocks),
- remote procedure call as a basic concept comprising as well synchronous as asynchronous forms,
- interrupt concepts,
- graphical construct to visualize clearly the import and export of remote procedures,
- object oriented data types.

These enhancements should be considered in the process of further development of SDL. For the data description ASN.1 was used. The combination of SDL´92 and ASN.1 is standardized in Z.105 [7].

Additionally to SDL a description technique similar to the Message Sequence Charts (Z.120) was used for the Trader specification. This was done to bridge the gap between the plain english text specification of the informal Trader's description on one side and the formal SDL92/ASN.1 specification on the other. In Figure 1 a summary of the concepts of this technique is given.

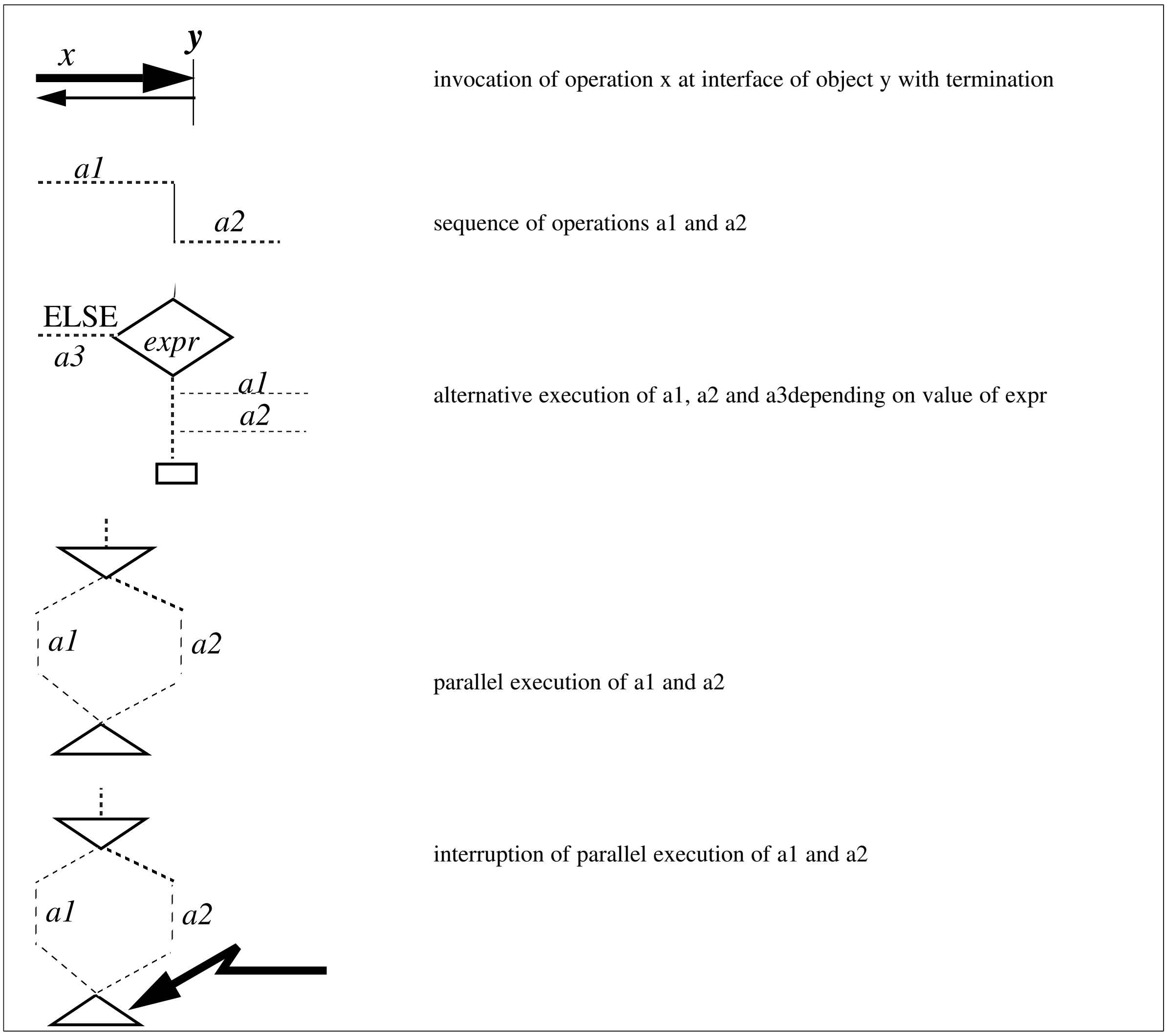

Figure 1

3 An Informal Behavior Description of the Trader

3.1 Introd1uction

This paragraph describes the behavior of the trading function using the import operation as demonstration example. The charts in Figure 2, Figure 3 and Figure 4 show the communication between a client (importer) and a Trader, i.e an invocation by the client and various terminations (successful or not) by the Trader. Additionally the behavior of the import operation is given by an acyclic graph of actions according to ODP-BRM Part 2. For those parts of the behavior where the Trader requires other ODP-functions (e.g. the Storage Function) placeholder objects whose behavior is not yet specified are introduced.

The specification of the behavior and the interfaces of the placeholder objects will be part of further standardization activities. However, these objects are necessary to define the Trader's behaviour.The following generic placeholder objects have been introduced:

a) Security Policy Maker Object
- provides the access control

b) Search Policy Maker Object
- unifies the traders and the importers policies
- determines the search scope (including check whether a linked Trader shall be included in the search)
- modifies the search policy before invoking the import operation at a linked Trader
- evaluates the matching constraint and the preference criteria
- is able to interrupt the operation

c) Type Management Object
- determines a set of compatible types for an given service type
- checks whether the required properties are supported by a service type

d) Link Space Object
- allows the access to the Link Space

e) Offer Space Object
- allows the access to the Offer Space Actions Graph Description

The following figures define the interactions between the client object, the Trader and the additional placeholder objects.

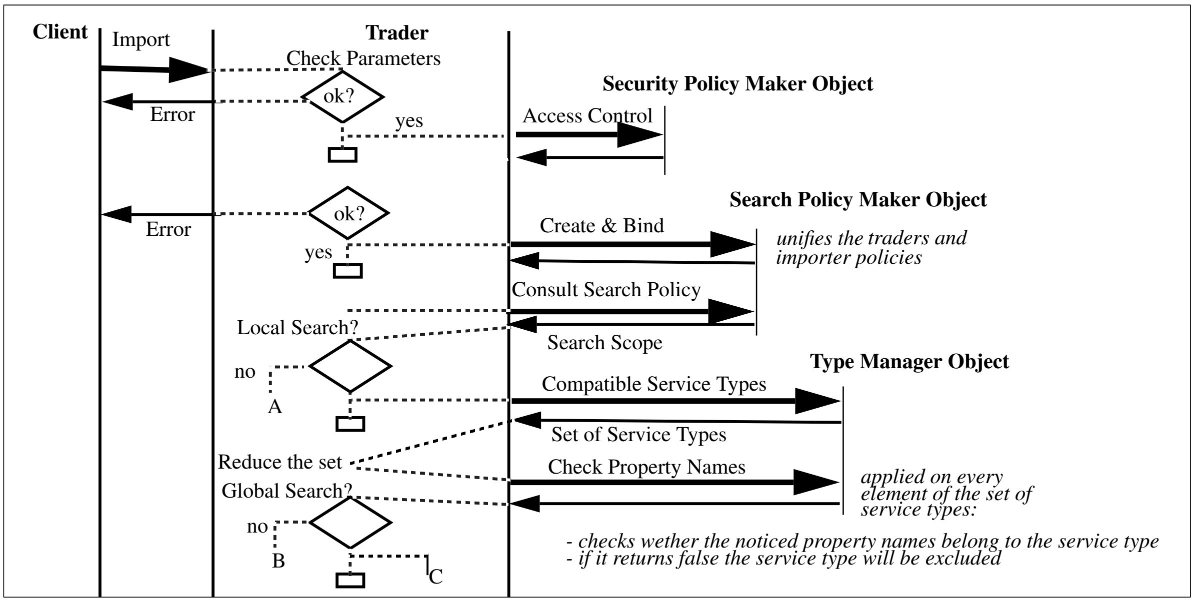

Figure 2

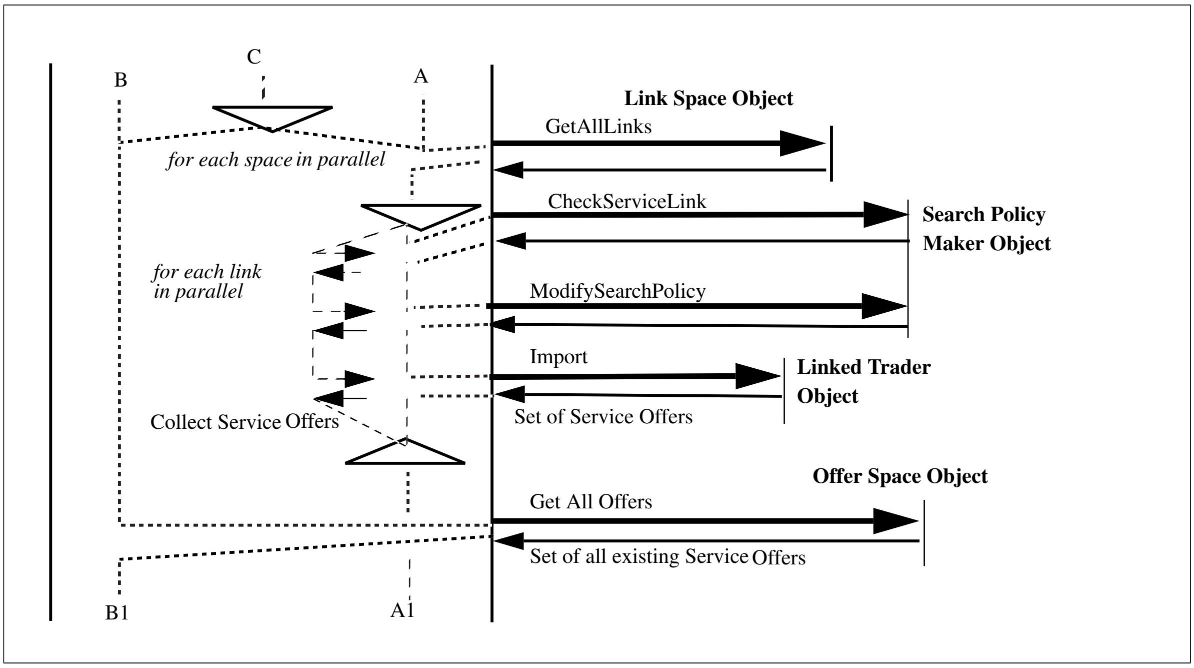

Figure 3

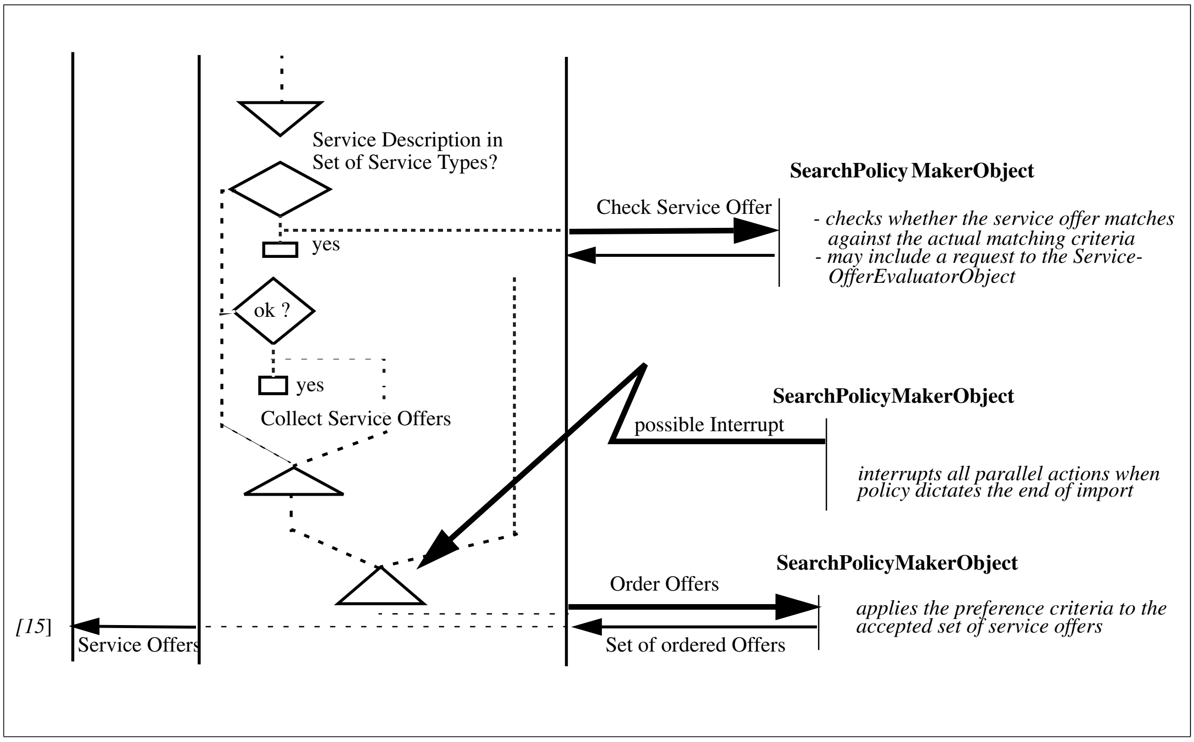

Figure 4

4 Interface Signature of the Computational Trading Function Specification in SDL/ ASN.1

The data types and operation signatures of the formal computational model of the ODP trading function are specified using SDL'92 (Z.100) in combination with ASN.1 (Z.105). To avoid confusion the term ***type*** is used here in the sense of data types in ASN.1 and not in the sense of ODP, where types are predicates. However, there is a correspondence between the ASN.1 ***type*** and the ODP ***template***.

The basic types *IdentifierType, InterfaceIdentifierType, NameType* and the types *RuleType, QualificationCodeType, PropertyType* are introduced as context parameters. That is, they can appear in the definition of other types (for example *PropertyType* is used to define *PropertyValueType*) as well as in the definition of operation signatures without a concrete definition. A context parameter is a generic parameter which has the function of a placeholder and can be replaced by a concrete type in further specifications by inheritance.

The specification is structured in the following SDL packages:

- Common Definitions
  - as defined in the informal Trader's description
- Invocation and termination types
  - Types for Trading Service Interface
  - Types for Trading Management Interface
    For each operation two types are defined: one for the invocation (a structure for all parameters) another for the termination (a structure for all return values) except those operations with only a single parameter and/or a single return value (*Withdraw*).
- SDL 92 definition of the Trader template
  - The template consists of the two interface definitions (TSI, TMI) and the declaration of all normative operations. The signatures are given with the operation definitions using the data types, invocation types, termination types.

4.1 Relation to OMG's CORBA IDL

Many of the definitions using SDL/ASN.1 are equivalent or similar to definitions in IDL. In Table 1 the relation between the SDL representation and the specification in OMG's CORBA IDL is given.

OMG's CORBA IDL	SDL/ ASN.1 interface description
Basic types are not defined; these types can be defined using `any`	Basic types are defined as context parameters
PropertyValueType defined with `any`	PropertyValueType defined with the context parameter PropertyType
ConstraintSpecificationType, PolicySpecificationType, PreferenceSpecificationType are defined as `any`	these types are defined as RuleType which is a context parameter
ServiceOfferDetailType is defined	this type is not provided in the informal Trader's description it is used as a termination type of the import operation (ImportRType)
ServiceDescriptionType is defined in different ways	only defined as IdentifierType
Signature definition	invocation parameters are combined in a structure as well as termination values

Table 1 Divergencies between different IDL's

5 Behavior Specification of the Computational Trading Function in SDL/ ASN.1

5.1 Introduction

The behavior of the normative operations of the trading service interface template and the trading management interface template in the formal computational model of the ODP trading function are specified using SDL'92

(Z.100) in combination with ASN.1(Z.105). The specification sketched in Chapter 4 is specialized here by the inheritance mechanism provided by SDL. That implies in particular

a) all general data type definitions introduced in the Trader template are available,

b) all identified normative operations with their signatures at the trading service interface template and management interface template defined as abstract virtual remote procedures can now be extended with behavior descriptions.

According to the basic modelling concepts of ODP-BRM (Part2, Part3) the behavior description of an interface operation is given by an acyclic graph of actions and interactions which perform the Trader's state. So the behavior description given by one interface template can not be defined without an identification of interactions with interfaces defined by other interface templates of the Trader. The basic problem is that these interfaces are not standardized yet. This is a task of a further ODP component standard (security function, storage function, type repository,...). The proposed solution for this dilemma is the introduction of generic place-holder components in the behavior description, which may be specialized/modified in reaction to the further ODP component standardization. The description of such generic components is done in the SDL description using object-oriented techniques like virtual functions and virtual structural components in combination with generic types.

The introduced additional generic components for the definition of the traders behavior have to specify the interactions to such objects like *ServiceOfferSpace, LinkSpace* and to the *PolicyMakerObjects*. The property ‚generic' of these components means, that the access to the interfaces of the objects is not described. It is open, because these objects with their interfaces do not exist.

5.2 Trader Template

Figure 5 shows the Trader template according to its interface specification (Chapter 4) The signatures of the operations (*export, import*,...) are given, but the behavior description of these interfaces is still open. So the Trader template is an abstract or generic one. A Trader instantiation of that template as it is shown makes no sense, only if all generic components of the Trader template are substituted by concrete ones, the instantiation is possible.

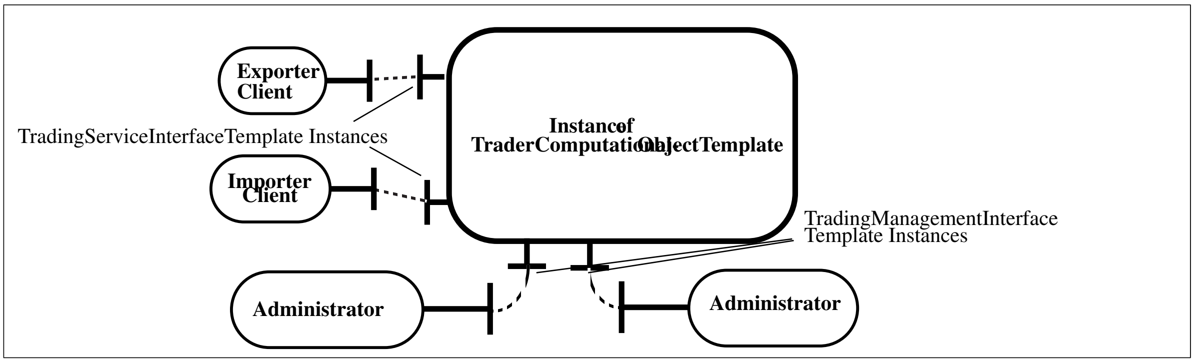

Figure 5 Application of a Trader template with normative interface templates

The task now is to extend the Trader template including its two operational interface templates *TradingService-Interface* and *TradingManagementInterface* templates with behavior descriptions. This is done in SDL by a

Traderspecialization of the abstract Trader template to a new Trader template, where the empty virtual procedures are redefined.

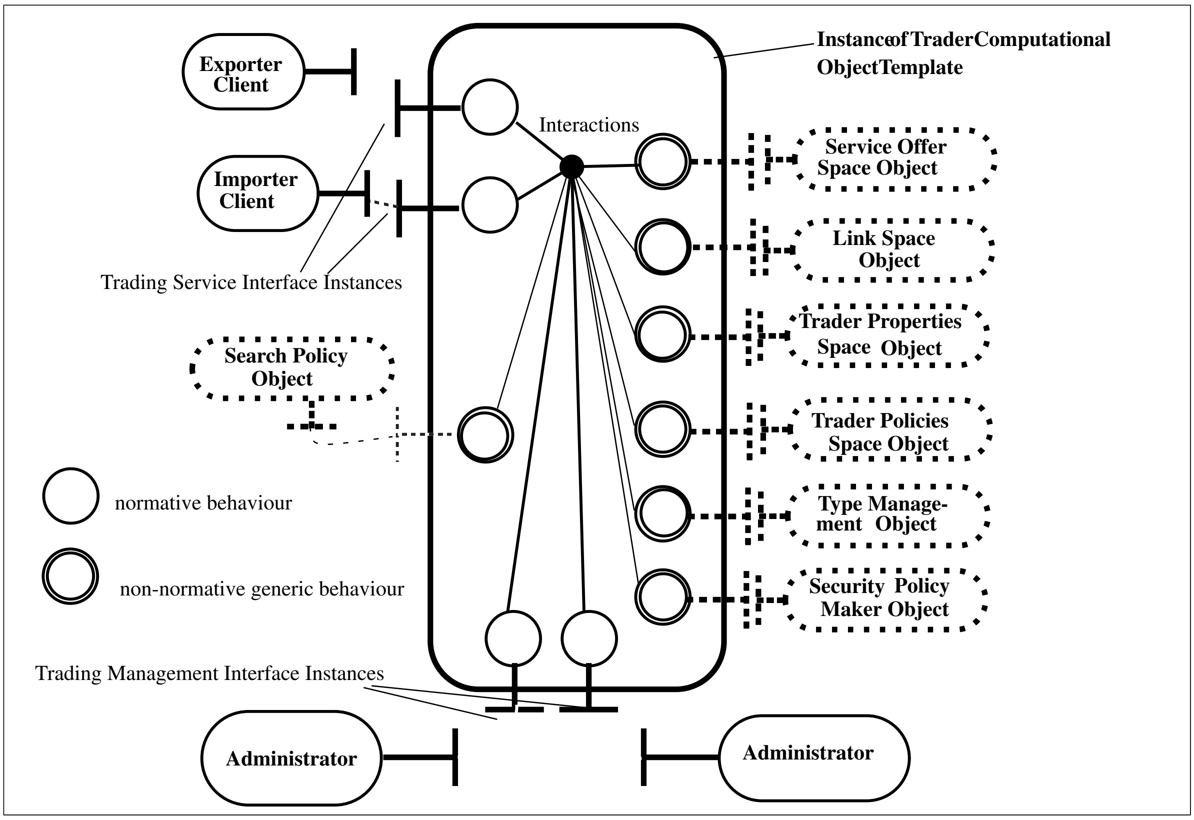

Figure 6 A specialized Trader template with generic place-holder behavior components

Figure 6 represents a possible application of the specialized Trader template. The description of the Trader's behavior is realized by interactions of various behavior parts (represented as circles) into which the behavior description of the Trader can be decomposed. Of course the structure of the decomposition is not normative. The problem is to specify the computational behavior description of the Trader as a graph of interactions with further objects which offer special services to the Trader's behavior. This is independent of the decision whether these objects are defined inside or outside of the Trader. In each case interfaces to these subobjects/objects have to be identified. Because the description of these objects depends on the further ODP component standardization, this problem can not be solved sufficiently by the Trader standardization.

The following objects and functionality can be identified (cf. Chapter 3.1)

a) SecurityPolicyMakerObject

b) SearchPolicyMakerObject

c) Type Management Object

d) Link Space Object

e) Offer Space Object

f) Trader Properties Space Object
 • allows the access to the Trader Properties Space

g) Trader Policies Space Object
 • allows the access to the Trader's Policies Space
 • computes the Placement Policy

The solution given here is again generic. That means, the behavior description of the Trader by those components which have to represent the interactions to the computational objects are specified as virtual interactions with an empty behavior part (calls of virtual remote procedures in SDL), which can be redefined by specializations in different ways.

5.3 Modelling of Service Offers

A major problem of mapping identified information entities of the informal description onto computational entities concerns the representation of the anonymous Trader's state in the computational model of the Trader. Seen from the computational viewpoint it is necessary to describe this phenomena, because the behavior description of operations at the normative interfaces are based on actions, which manipulate this state. The difficulty to model the Trader's state is of course to find out an abstract and implementation independent representation. A first step in this direction is the definition of a template type for service offers, i.e. service offer

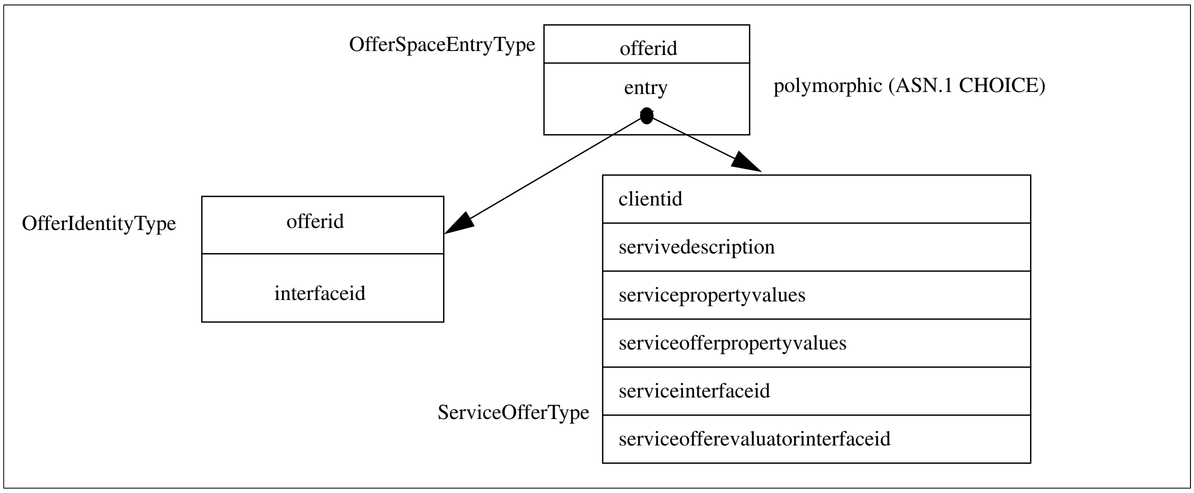

Figure 7 Template types to handle service offers

template type (represented in SDL by the data type *ServiceOfferType*, see Figure 7). The template type defines only the common structure of all service offers, what is just the collection of all parameters given by an *export*. As a consequence the Trader is now able to generate instances of the service offer template.

The next step is to derive a model for the service offer space (as part of the Trader's state) guarantying the implementation independency. In Figure 6 a place-holder component was introduced which can be substituted later by interactions with an interface to another computational object, namely the *ServiceOfferSpace* object. Note that the decision, whether this object will be a subobject of the Trader or an object outside of the Trader, is open. For the computational model only one (simple and abstract) solution has to be given which can be substituted by different approaches.The access operations to the offer space are realized as virtual remote procedure calls.

To model the mechanisms of distributed service offer spaces by different traders, the SDL type *OfferIdentityType* in correspondence to the informal Trader's description was introduced. So, modifications of the offer space of two different traders by an *export* can be distinguished. For that purpose the placement policy is consulted to decide, where the offer has to be stored (Figure 8).The following strategy is used for the storage of service offers by the Trader:

a) If the service offer is stored by the invoked Trader (*Trader1*) in its offer space, an offer identifier is returned (*y11*) and a structure is instantiated including all necessary information about the service offer (*ServiceOfferType*).

b) Otherwise, if the placement policy decides to store the offer at another Trader (*Trader2*), the invoked Trader (*Trader1*) stores an offer identifier *y12* unique to its service offer space. Furthermore the invoked Trader stores a structure including the offer identifier *y2* unique to the other Trader's (*Trader2*) service offer space and the interface identifier *x2* of the trading service inter-

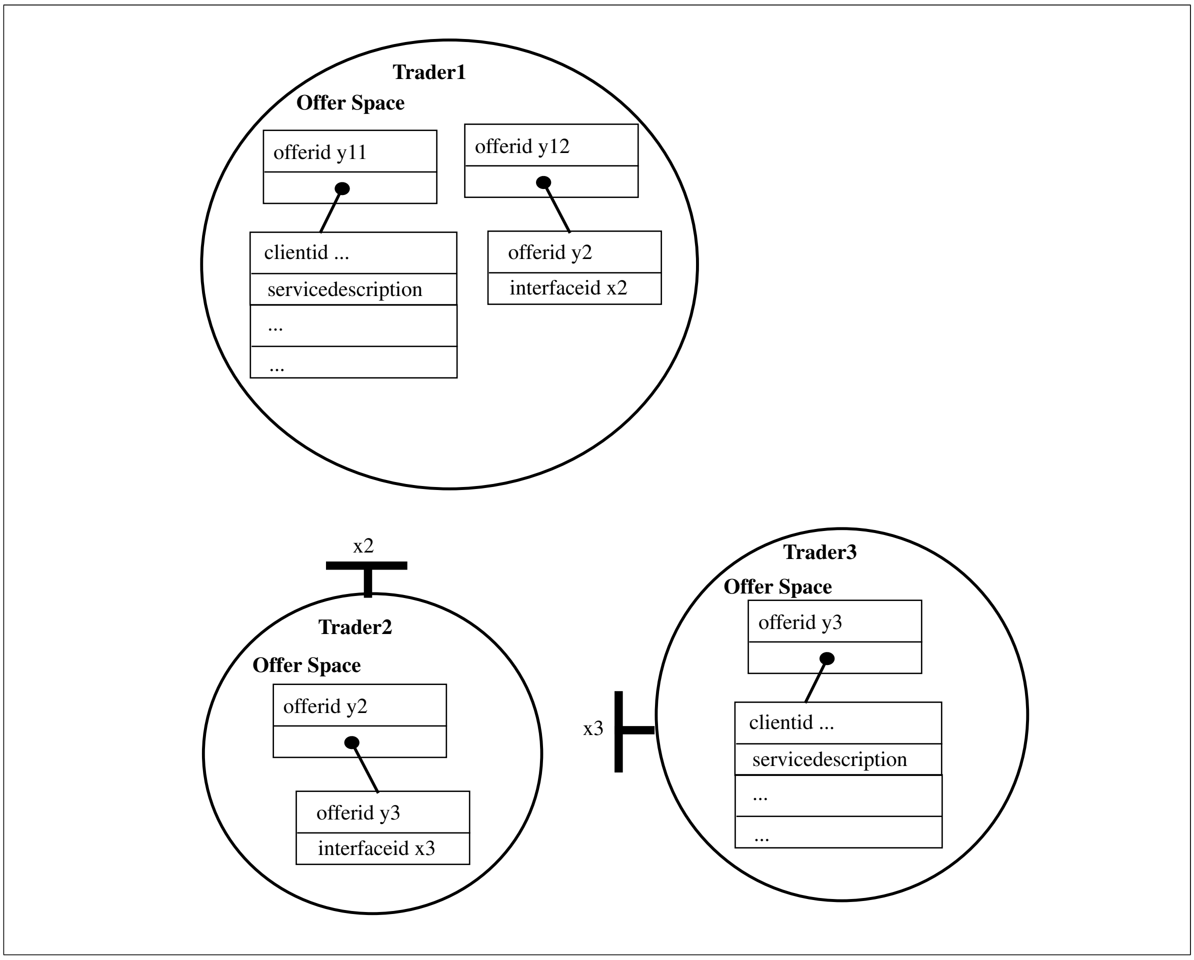

Figure 8 Model of a distributed service offer space

face of that Trader (Trader2) where the offer is stored (*OfferIdentityType*).

5.4 Modelling of Trader Links

The mapping of a Trader link seen from the information viewpoint to the computational viewpoint results in a binding of operational interfaces of two traders with fixed client/server roles. A graph of actions, which manipulate the Trader's state, has to be given in order to describe the behavior of the link management operations. Special computational entities (template, instances) represent that state.

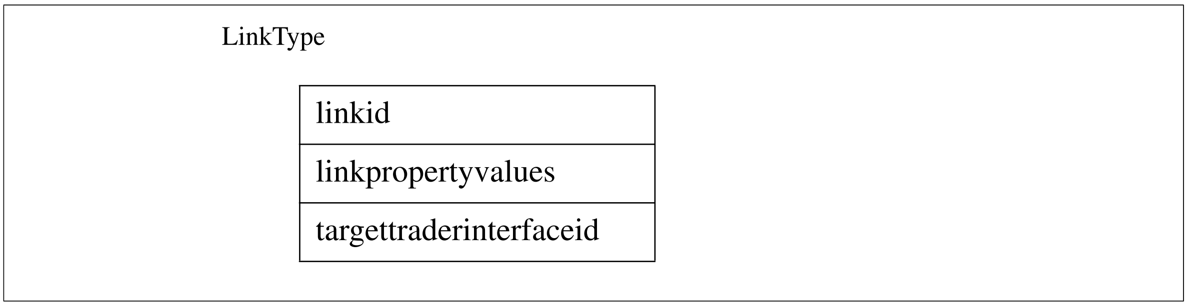

Figure 9 Template to handle Trader links

This template is *LinkType* (Figure 9), given in the formal description by an ASN.1 data type. The idea of L*ink-Type* is similar to *ServiceOfferType*. It is just a collection of the parameters of the *AddLin*k operation. The

Trader can instantiate this template during runtime and is able to change the state by using the place-holder object for communication with the *LinkSpaceObject*.

5.5 SDL Specification Overview

The given SDL specification of the generic Trader template for the description of the behavior of the normative interface templates is realized as a generic SDL system type. This system type is a specialization of the much more abstract ones, given by paragraph 3. Now the system type includes the definitions of

- global data structures, and
- a generic block type, which represents the Trader template.

The block type consists of

- process types which represents the behavior component structure of the Trader template which were identified by Figure 6,
- remote procedure definitions, which describe the behavior of the Trader (their signatures are defined by inheritance).

The system type *TraderCommunity* is generic because only two of the six inherited context parameters are actu-

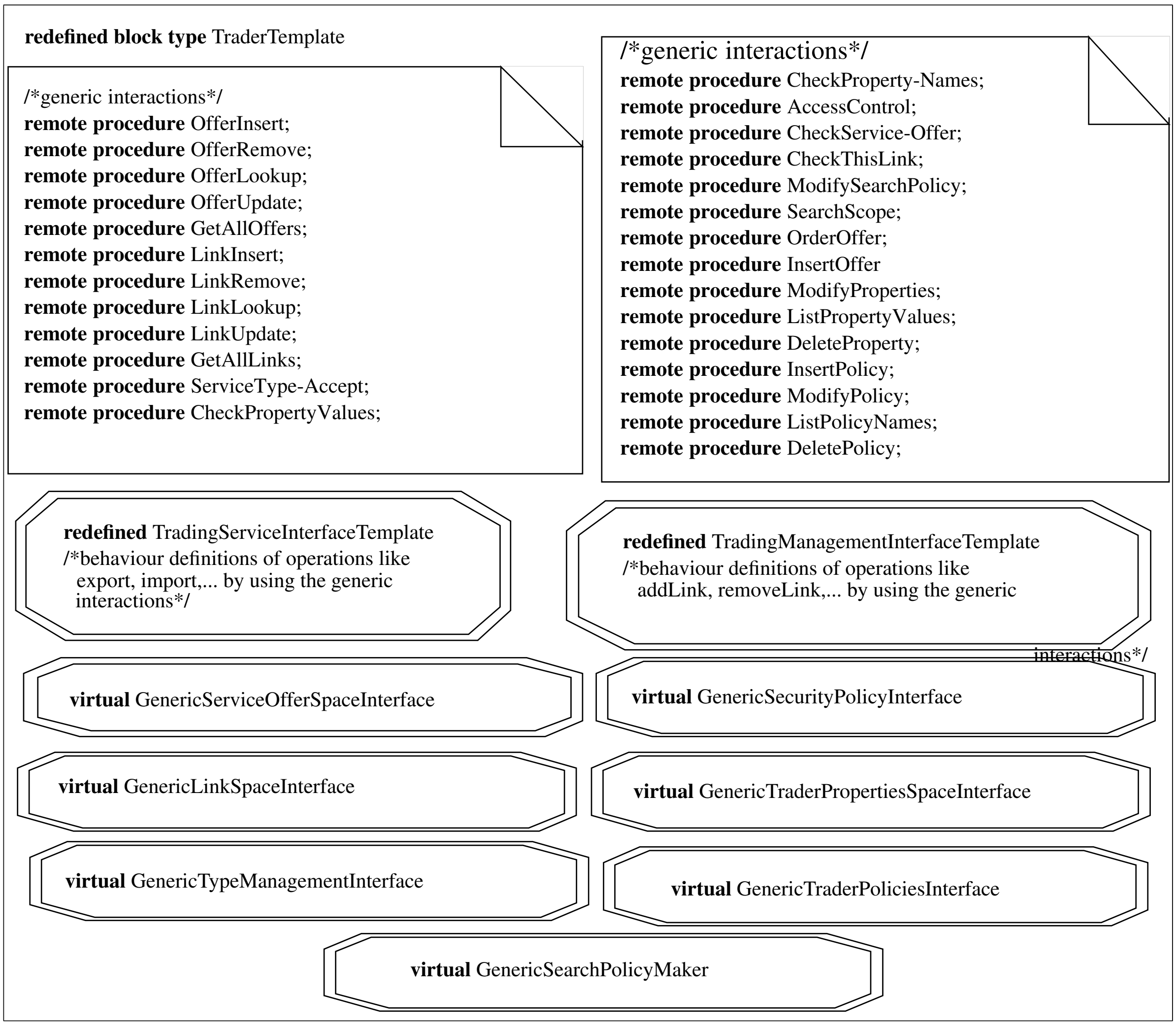

Figure 10 Graphical SDL representation of the Trader‘s computational object template

alized. The first *QualificationCodetype*, which is used for the representation of messages as results of certain operations, is substituted by *IA5STRING*. The second parameter *InterfaceIdentifier-Type* is defined as *PId* (SDL type for object identification) Furthermore the system type is an abstract type because the included process types use some virtual procedures, whose semantics are still undefined. Only their signatures are defined here. These procedures model the generic place-holder components.

The SDL representation of the Trader‘s template structure given by Figure 6 is shown by Figure 10. It includes the specifications of all generic interactions with the place-holder objects. Only the signatures of the corresponding remote procedure are defined (per reference to the further SDL text) but not the behavior. The *TradingServiceInterfaceTemplate* and the *TradingManagementTemplate* are redefined here because their already defined signatures will now extend by behavior definitions (see to the SDL text). Furthermore the types for the place-holder objects are defined in form of process type definitions. The instances of all defined process types of the block type *TraderTemplate* define by their interworking the behavior of the Trader.

5.6 SDL Specification

The complete SDL/ASN.1 specification of the Trader template is given in the SDL/PR form in [8] The syntactical correctness as well as the correctness of the static semantics have been checked using the SDLEnvironment [10].

6 References

[1] ITU-T/ISO: ITU-T X.901 | ISO/IEC 10746-1 ODP Reference Model Part 1. Committee Draft (CD) output from the meeting in Southampton (U.K.), 19-29 July 1994

[2] ITU-T/ISO: Revised Text of CD 10746-2.3, Information Technology - Open Distributed Processing - Reference Model - Open Distributed Processing - Part 2.: Descriptive Model, forwarded to the ITTF for DIS registration and letter ballot

[3] ITU-T/ISO: Revised Text of CD 10746-3.2, Information Technology - Open Distributed Processing - Reference Model - Open Distributed Processing - Part 3: Prescriptive Model,. forwarded to the ITTF for DIS registration and letter ballot

[4] ITU-T/ISO: ITU-T X.901 | ISO/IEC 10746-4 ODP Reference Model Part 4: Architectural Semantics,. Committee Draft (CD) output from the meeting in Southampton (U.K.), 19-29 July 1994

[5] ITU-T/ISO: Report on the Joint Meeting of ISO/IEC JTC1/SC21/WG7 and SG7.Q16 (ODP) on ODP Trader, Southampton, UK, 20-29 July 1994

[6] ITU-T: Z.100 Specification and Description Language SDL, Geneva 1993

[7] ITU-T: Z.105 SDL in Combination with ASN.1, Geneva 1994

[8] Fischer, J.: Contributions for the Formal Specification of the ODP Trader using SDL’92 and ASN.1, Informatik-Preprint Nr.37, Humboldt Universität zu Berlin, to appear 1/1995

[9] Fischer, J.; Prinz, A.; Vogel, A.: Different FDT’s Confronted with Different ODP-Viewpoints of the Trader, Proceedings of the FME’93, Springerverlag, 1993

[10] Fischer, J.; Holz, E.: Towards an Object-Oriented Technology for Specification and Implementation of Distributed Systems, Proceedings of the TOOLS-USA’93, Prentice-Hall 1994