A Service Request Description Language

Claudia Popien and Bernd Meyer

Aachen University of Technology, Department of Computer Science IV, Ahornstr. 55
D-52056 Aachen, GERMANY. E-mail: popien@informatik.rwth-aachen.de

Abstract

To meet the new requirements of growing computer networks the client-server model has to be improved by new concepts with encreased functionality. Therefore, a service management in distributed systems is realized by a service trading concept.

This paper presents a language given in Backus-Naur Form which enables specification of service requests. First, a matching criteria is specified as conditioned Boolean expression in order to evaluate appropriate service offers. Additionally, a selection criteria can be used to yield the most suitable service. To limit the scope of a search it is possible to specify policies and search constraints on service properties which restrict the search on particular trading objects. The Service Request Description Language presented here is related to the ANSAware trader to increase its functionality by giving means for service searching and selecting.

Keyword Codes: F.4.3, D.2.1, D.3.1
Keywords: ODP, Service Trading, SRDL

1. INTRODUCTION

The current development of information processing systems leads to global network of interconnected computers. Therefore, new services for managing resources of distributed systems and networks are needed. A growing number of components which offer services leads to a new situation which enforces an open service market. The problem of the client's need to select a server arises. Therefore, the client-server model is extended to a three-party model. A so-called trader enables binding between client and server at run time. To realize trading a service directory is used, which stores service types and their associated service property values. Different trading realizations are compared in [Ke 93].

In this paper, trading is considered according to the Reference Model of Open Distributed Processing (ODP) [ODP P1-P4], [Ste 90]. The tasks of the ODP trader are devided into domain and type management, respectively. Domain management supports finding a service, type management supports interface matching. It is the aim of this paper to improve trading concepts given in the ODP Reference Model by presenting matching criteria and search constraints in a language with formal syntax and semantics. In addition, the concept of a policy will be elaborated formally and applied to trading.

The following section introduces elementary ODP trading concepts. The notions of a service and its properties are explained. After defining the structure of a service offer, a query language for service import is defined in section three. Both, matching and selection criteria are given in Backus-Naur Form, accompanied by example queries. In order to restrict the search space or scope of an importing query, search constraints are defined in section four. In addition, the scope for imports of the requested trader has to be considered, which is defined by a trader search policy. A notation for specifying policies is given, too. Whereas the previous sections deal with syntactical constructs, their semantics is given in section five. It consists of four steps for import operations on a given service database. Section six is concerned with service management, especially with the evaluation of dynamic service properties. Finally, this paper concludes with a brief summary of contributions made and a view on future work.

2. SERVICE TRADING IN DISTRIBUTED SYSTEMS

A trader is an object that performs trading, primarily satisfying identification requirements [Gei 92]. In the following traders based on ODP are considered [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, see Fig. 1.

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*.

For trading, service properties can be classified as static or dynamic, see Fig. 2. 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. A *dynamic service property* is a property whose values change much faster 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.

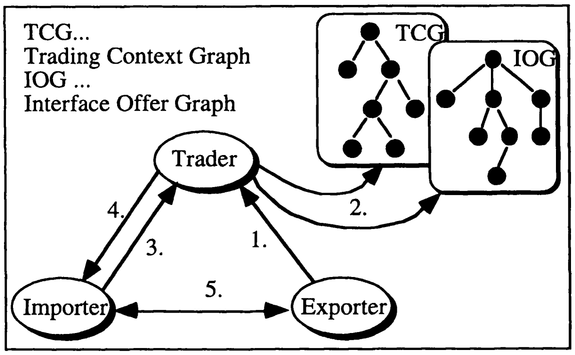

Fig. 1: The process of trading in ODP

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*.

The type management function realizes the management of the subtype relationship between types. The service type hierarchy is represented as a directed acyclic graph, called *Interface Offer Graph* (IOG), 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 which are defined by a containment relationship between contexts. A trader context structure can be represented as a directed acyclic graph, called *Trading Context Graph* (TCG), where nodes represent trading contexts and arcs represent a containment relationship. A service offer is a member of a trader context, and of any super context of that trader context. A trader service offer space can reflect administrative structures and organizations, or service types.

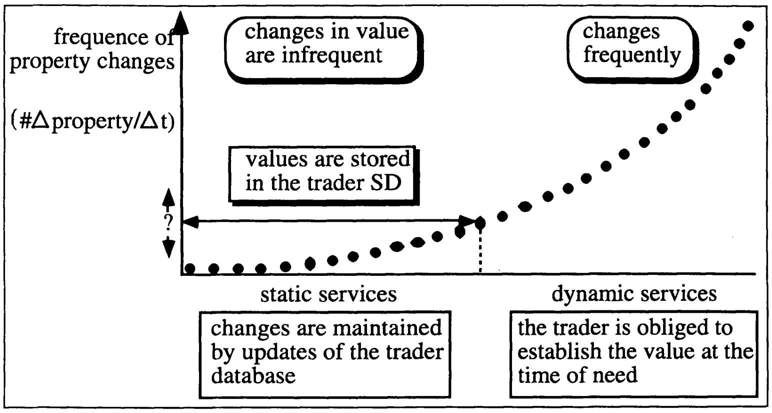
frequence of property changes
(#Δproperty/Δt)
?
changes in value are infrequent
values are stored in the trader SD
changes frequently
static services
dynamic services
changes are maintained by updates of the trader database
the trader is obliged to establish the value at the time of need

Fig. 2: Static vs. dynamic service properties

3. DESCRIBING MATCHING CRITERIA AND SELECTION CRITERIA IN SERVICE REQUESTS

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", "Service Interface Id", "Service Type Description" and "Service Offer Properties", where a "Service Type Description" can be further refined into an "Interface Type Description" and a "Service Property Type Description". Service properties are "Static Service Properties" or "Dynamic Service Properties", respectively described as ordered pairs

<service property name, service property value>.

The structure of a service offer without service offer properties is described in Fig. 3.

The importer has two different functions which are used to find suitable services. These functions are `search` and `select`. Search is the behaviour which searches the service offers of a trader and returns a set of service offers which satisfy some matching criteria. Its inputs are the service type of the desired service and a *matching criteria.* The requirement specification comprises two parts:

- the required computation service behaviour, given by a service type description and
- the required properties, expressed by a matching criteria.

To perform a `search` operation, the matching criteria must be studied in more detail. A matching criteria is a set of rules which are applied to the total set of service offers in order to yield a smaller set of acceptable service offers. The result of a `search` operation is the set of service offers with the correct service type. These service offers also satisfy the matching constraint. In this sense, a matching criteria is a criteria on service properties against which the service offers are evaluated. The successful execution of this operation will return a list of interface identifiers, service properties and service offer properties.

Until now, no further recommendations have been given, in which way these functions should be performed. ANSAware [ANSA] only offers a reduced syntax for matching constraints. Therefore, a concept is presented that enables specifying parameters of these functions. In this concept, a matching criterion is specified as a Boolean expression. The criterion is specified in a Backus-Naur Form (BNF) as followes.

Exporter Identifier	Service Interface Identifier	Service Type Id	Static Service Properties	Dynamic Service Properties

Fig. 3: The detailed structure of a service offer

The language presented here, is called Service Request Description Language (SRDL). SRDL specifications given according to the BNF are called SRDL-texts.

service-request ::= <service_request_operation> <search-constraint> "**END SERVREQ**"

service_request_operation ::= <search_operation>
| <select_operation>

search-operation ::= "**SEARCH**" <service-type-identifier> "**WITH**" <conditional-matching-criteria>
select-operation ::= "**SELECT**" <service-type-identifier> "**WITH**" <conditional-selection-criteria>

Therefore, a conditional_matching_criteria is defined based on a basic_matching_criteria and a simple_matching_criteria:

basic-matching-criteria ::= <service-property-identifier> <general-relation> <real-number>
| <service-property-identifier> <equal-or-not-symbol> <service-property-value-identifier>
| <service-property-value-identifier> <element-or-not-symbol> <service-property-identifier>.
simple-matching-criteria ::= <basic-matching-criteria>
| "**NOT**" <simple-matching-criteria>
| <simple-matching-criteria> <and-or-or-symbol> <simple-matching-criteria>.
conditional-matching-criteria ::= <simple-matching-criteria>
| "**IF SUCCESS**" <conditional-matching-criteria> "**THEN**" <conditional-matching-criteria> "**ELSE**" <conditional-matching-criteria> "**END**"
| "**IF SUCCESS**" <conditional-matching-criteria> "**THEN**" "**TAKE THAT**" "**ELSE**" <conditional-matching-criteria> "**END**".

The used notions and identifiers are explained in the following:

letter ::= "a" | "b" | ... | "z" | "A" | "B" | ... | "Z".
digit ::= "0" | "1" | ... | "9".
normal-character ::= <letter> | <digit>.
identifier ::= <letter> [{<normal-character> | "_"} normal-character].
service-property-identifier ::= <identifier>
| <identifier> "/d".

service-property-value-identifier ::= <identifier>.

service-type-identifier ::= <identifier>.

real-number ::= {<digit>} <digit> ["." <digit> {<digit>}].
relation-symbol ::= "<" | "≤" | ">" | "≥".
equal-or-not-symbol ::= "=" | "≠".
general-relation ::= <relation-symbol> | <equal-or-not-symbol>.
element-or-not-symbol ::= "∈" | "∉".
and-or-or-symbol ::= **"AND"** | **"OR"**.

As an example a SRDL-text is given so that all services of service type PRINTER with special static service properties "location" and "cost_per_page" and a special value for the dynamic service property "queue_length" should be searched which fulfill the following matching constraint:

```
SEARCH PRINTER WITH
  IF SUCCESS   location = CompCent AND
               cost_per_page ≤ 0.10
  THEN   TAKE THAT
  ELSE   IF SUCCESS cost_per_page < 0.10 AND queue_length/d ≤ 3
         THEN cost_per_page < 0.10 AND queue_length/d ≤ 5
         ELSE location = CompCent END END
```

A matching criterion needs no specific values for all properties of an exported service offer. The non-specified properties are essentially "don't care" values and are "true" when matched with any values for those properties in an exported service offer.

The `select` function searches a set of service offers and returns a single service offer which satisfies some selection criteria. It chooses the most suitable service offer that satisfies the importer's matching and selection criteria. This operation can be regarded as a simple case of a search behaviour.

To study the selection function it is necessary to consider *selection criteria*. A selection criterion is a set of rules which are applied to the total set of service offers to yield a single service offer. It is a criterion which will be applied to select one service offer from this set of service offers resulting from the matching criteria. Based on the Backus-Naur Form the selection criteria can be specified as follows.

select-operation ::= **"SELECT"** <service-type-identifier> **"WITH"** <conditional-selection-criteria>

Therefore, a conditional_selecion_criteria is defined based on a basic_selection_criteria and a simple_selecion_criteria:

basic-selection-criteria ::= <min-or-max-symbol> "(" <service-property-identifier> ")"
| <choice-symbol> "(" <conditional-matching-criteria> ")".
simple-selection-criteria ::= <basic-selection-criteria> {"**AND**" <simple-matching-criteria>}.
conditional-selection-criteria ::= <simple-selection-criteria>
| "**IF SUCCESS**" <conditional-selection-criteria>
"**THEN**"<conditional-selection-criteria> "**ELSE**"
<conditional-selection-criteria> "**END**"
| "**IF SUCCESS**" <conditional-selection-criteria>
"**THEN**" "**TAKE THAT**" "**ELSE**"
<conditional-selection-criteria> "**END**".

The additionally used notions and identifiers are explained in the following:

min-or-max-symbol ::= "**MINIMUM**" | "**MAXIMUM**".
choice-symbol ::= "**FIRST**" | "**LAST**" | "**RANDOM**".

An example for selecting a service of type PRINTER is given in the following:

```
SELECT PRINTER WITH
IF SUCCESS
    RANDOM (type = Laser AND cost_per_page < 0.10 AND A3 ∈ paper_size)
    THEN FIRST (MINIMUM (cost_per_page) AND type = Laser AND A3 ∈
    paper_size)
    ELSE MINIMUM (cost_per_page) AND A3 ∈ paper_size END
```

This SRDL language to describe matching and selection criteria presented here, has a formal semantics. By mapping Boolean expressions onto the values "true" and "false" it is possible to get corresponding services.

4. CONSIDERING POLICIES AND SEARCH CONSTRAINTS

A search constraint specifies a restriction for matching import requests. This object is a constraint on service and link properties that result in the set of traders in which the search is performed. A search constraint includes criteria for limiting the scope of a search and focusing a search to particular trader objects. When evaluated at a particular starting trader object, a search constraint may be resolved to a set of trader objects. 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 exporting contract, see Fig. 4.

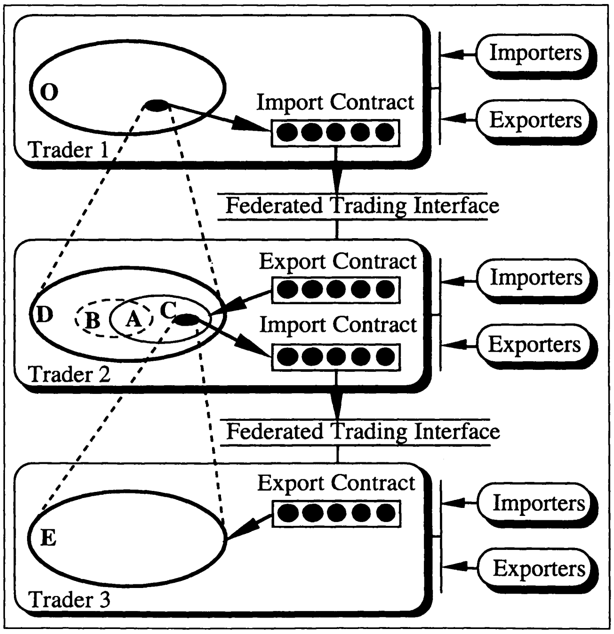

Fig. 4: A search path within a trading federation

The new system is called *trading federation* [BeRa 91], [PoMe 93]. The *trading offer domain* (TOD) is extended. It contains the set of all service offers that describe services available to objects in a federated trading offer domain, including those service offers that have been made available through federation.

Links to federated operations' service offers are set up in the appropriate search spaces of the importing trader when an import contract is established. When a trader receives a search request, it looks up its database to find suitable exports by accessing its relevant search spaces. If one of the entries in the search space is a link to an import contract, this contract is used to generate an equivalent federated operation, i.e. a federated-search, using the information in this import contract. On the other hand, when an exporting trader receives a federated search-request, it associates the search with an export contract identified by the requesting trader. The search is restricted to the search spaces of the exporting trader specified in the export contract. To describe the restricted search space, a context relative naming model is used in the following. Each name is a path which describes the route to a particular context.

4.1 Defining Search Constraints

Each context can have multiple names provided that the names are unique. In a context relative naming model, unique names are guaranteed if each edge in the graph has a unique label relative to the source of the edge.

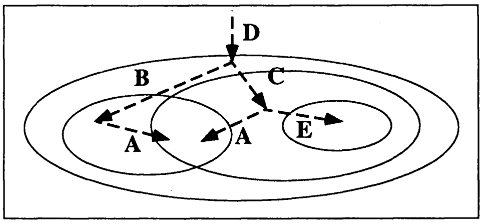

Fig. 5: An example for a context-relative naming of a context structure

In our example in Fig. 5, trading context A is named both "/D/B/A" and "/D/C/A". It is not necessary to assign the label "A" to both of the edges (TC-B, TC-A) and (TC-C, TC-A). The edge (TC-B, TC-A) could be given any label while the edge (TC-C, TC-A) could be given any label that is different from the label of the edge (TC-C, TC-E). The subset relationships can be derived from the names. The context named "/D/B/A" is a subset of the context named "/D/B", which is a subset of the context named "/D".

The search constraint can be described as a search path in the context structure. The BNF given before is extended to:

```
search-constraint ::=   "IN" <search-expression>.
search-expression ::=      "/" <trading-context> {"/" <trading-context>}.
trading-context ::=    <identifier>.
```

As an example, a service offer should be searched in trading context E. This search constraint is specified as followes.

SELECT PRINTER **WITH**
  MINIMUM (queue_length/d) **AND** cost_per_page < 0.10 **AND** location =
  CompCent
IN /D/C/E **END SERVREQ**

This concept for specifying matching criteria and search constraints is included into the global ODP service management concept.

4.2 Defining Policies

In large distributed systems, changes of configuration are likely to appear quite often. Therefore, the behaviour of management and administration services must be flexible to cope with dynamicalLy changing configurations and requirements. In order to achieve a flexible object behaviour, it has to be determined by a policy. Changing the policy must enforce a change in the objects´ behaviour.

Referring to the definition given by the RM ODP, a policy is defined as a prescriptive relation between one or more objects and some reference behaviour. It can be expressed as an obligation, a permission and a prohibition. In the following, it will be focussed on restrictions of behaviour, interface and operations. Valid behaviour, operations and parameter value ranges are specified. A policy is valid as long as its policy maker, a role of an object defining policies, changes it.

```
<policy> ::= "POLICY" <name>
             "FOR AGENTS" <object_name> {<object_name>}
             ["ENFORCED BY MANAGER" <object_name>]
             "IS" <behaviour_specification>
             ["WHERE" <behaviour_specification> {<behaviour_specification>}]
             "END POLICY" .
```

A policy consists of a unique name, one or more objects to which it is applied called agents, and the prescribed behaviour for these objects. Optionally, another object can be determined as the manager, enforcing the policy on the agents. If an object with manager role is determined, the following describes the manager behaviour. If the manager role is omitted whereas it defines the agent behaviour. This behaviour specification might contain names of additional specifications, that are specified after the (optional) "where" statement. Thus behaviour can be split into blocks, increasing the readability of the specification and giving the designer a means for structuring.

```
<behaviour_specification> ::= "BEHAVIOUR" <behaviour_name> ":"
                              <behaviour>"END BEHAVIOUR".

<behaviour> ::= <behaviour_combination>
              | <restriction> | <activity> .

<behaviour_combination> ::= <behaviour> "AND" <behaviour>
                          | <behaviour> "OR" <behaviour>.

<restriction>       ::= "RESTRICT" <restriction_type>.
<restriction_type>  ::= <behaviour_restriction>
                      | <interface_restriction>
                      | <operation_restriction> .

<behaviour_restriction>  ::= "BEHAVIOUR TO" <behaviour>
                             ["AT" (<object_name>)] .
<interface_restriction>  ::= "INTERFACE TO"
                             <operation_name> {<operation_name>}
                             ["AT" <object_name> {<object_name>}] .
<operation_restriction>  ::= "SCOPE OF" <operation_name> {<operation_name>}
                             "TO" <scope_description>
                             ["AT" <object_name> {<object_name>}]
                             |
                             "ATTRIBUTE RANGE OF" <attribute_name>
                             "IN" <operation_name> {<operation_name>}
                             "TO" <range_description>
                             [" <object_name> {<object_name>}] .
```

With a policy specification, restrictions on behaviour, on interface and on operations of objects can be made. Restrictions on operations refer to their scope and parameter value ranges. If restrictions do not contain the object, they refer to, the object or objects determined by the policy header are assumed. In the following we focus on restricting the scope of operations. The next two production rules determine the way the scope of an operation can be described.

```
<scope_description>  ::=  "{" <scope> "}" .
<scope>              ::=  <scope> ";" <scope>
                          | "SUBGRAPH OF" <path> ["TO DEPTH" <depth>]
                          | <path> .
```

Range descriptions can be expressed as a single value or a set of valid values.

```
<range_description>  ::=  <value>  | "{"<value_set>"}" .
<value_set>          ::= <value> ";" <value_set> | ε .
```

The scope of an operation is expressed by a set of paths to objects that can be used during operation processing. Each TOD of a federated trader will result in one or more path or subgraph descriptions. Within the following text, a path leads to a context containing a number of service offers.

```
<activity>              ::=  "BEHAVIOUR" <behaviour_name>
                             | <action> {";" <action>} "." .
<action>                ::=  <interaction>
                             | <conditional_action> | <action_name> .
<interaction>           ::=  <operation_invocation>
                             | <operation_reception>
                             | <notification_sending>
                             | <notification_reception> .
<conditional_action>     ::=    "IF" <condition> "THEN" <activity>
                         ["ELSE" <activity>] "END IF" .

<operation_invocation>  ::= "INVOKE OPERATION" <operation_name>
                            "AT" (<object_name>)
                            ["WITH" (<value>)] ["RESULTS IN" (<variable>)] .
<operation_reception>   ::= "ON INVOCATION OF" <operation_name>
                            "FROM" <object_name>
                            ["WITH" (<variable>)]
                            "DO" <activity>
                            "RETURNING" (<value>) .
<notification_sending>  ::= "SEND NOTIFICATION" <notification_name>
                            "TO" (<object_name>)
                            ["WITH" (<value>)] .
<notification_reception> ::= "ON NOTIFICATION" <notification_name>
                            "FROM" <object>
                            ["WITH" (<variable>)]
                            "DO" <activity> .
```

The behaviour specification language should at least contain constructs to describe operation invocation and reception, notification sending und reception, conditional action and linear temporal ordering of actions, as given by the above described production rules. The following policy specification gives an example for the use of the above defined notation. The name structure refers to the context-relative naming scheme used in Fig. 5.

POLICY TraderSearchPolicy
FOR AGENTS LocalTrader

```
IS BEHAVIOUR LocalServiceImport :
        RESTRICT INTERFACE TO search select
    AND
        RESTRICT SCOPE OF search select TO {SUBGRAPH OF "/D/C"}
    AND
        RESTRICT ATTRIBUTE RANGE OF location
            IN search select TO rwth_aachen
    END BEHAVIOUR
END POLICY
```

With this policy, the search space for service import performed by the local trader is restricted to the context "/D/C" and all subcontexts. Service import includes the search and select operation of the trader service interface. In addition, all exporters must be located at the RWTH Aachen. Related work on the policy concept can be found in e.g. [RPB 93], where examples of policy specifications are provided, but nothing is said on syntax nor semantics for their specification language.

5. THE FOUR-STEP-CONCEPT FOR A FORMAL SEMANTICS

Now, the aim is to consider the formal semantics of a service request. This formal semantics is devided into four steps which have well-defined interfaces. Starting point of our semantics is the data base of the trader, i.e. the service direcotry, called service offer set (SOS), together with the service request and a policy specified formally in our Service Request Description Language, presented above. The result is the interpretation of a SRDL-text in zero, one or more service offers according to our service offer set, see Fig. 6.

The first step consists of limiting the search space of a service request with respect to a corresponding policy, and is performed by the function `restricting_scope`. The operations' search space is defined in a SRDL-text and will be restricted to the search space the requested trader is allowed to access, which is determined by the trader search policy. During the computation of this function, the SOS remains unchanged.

The second step of the formal semantics is denoted by the function `restricting_ TOD`. It limits the whole scope of service offers by mapping the search constraints onto a smaller set of service offers which are of interest. This step considers only the SRDL-text and the service offer set. The result is a new service offer set, which is the interface to the third phase of this semantics.

After having obtained a set of service offers which is of interest, the third step applies the `IT&SSP_check` function. During this phase the interface type is considered, and static service properties are evaluated in order to reduce the set of appropriate services. The result of `IT&SSP_check` is a subset of SOS' which includes services with demanded static service properties. This function requires the service offer set SOS' which is the result of the previous step and it demands the SRDL-text of the given service request. These two values are mapped by the `IT&SSP_check` function onto the power set of SOS', i.e. onto $\wp$(SOS'). The SRDL-text remains unchanged.

The fourth step of this formal semantics differents between `search` and `select` service requests. In case of a search service request it is important to prove if there are dynamic service properties included in the given SRDL-text. If no dynamic service properties are considered, the result of the third phase is equal to the result of our fourth phase. Otherwise, it is necessary to apply the `search` function onto an element of $\wp$(SOS') and the SRDL-text in order to get a subset of the previously considered service offer set. In case of a `select` function it is always necessary to apply this function. If there are no dynamic service properties requested by the given selection criteria, the most suitable service is selected. Otherwise, the corresponding dynamic service properties are evaluated and the most suitable service offer is selected. Result of the selection process is a single service offer.

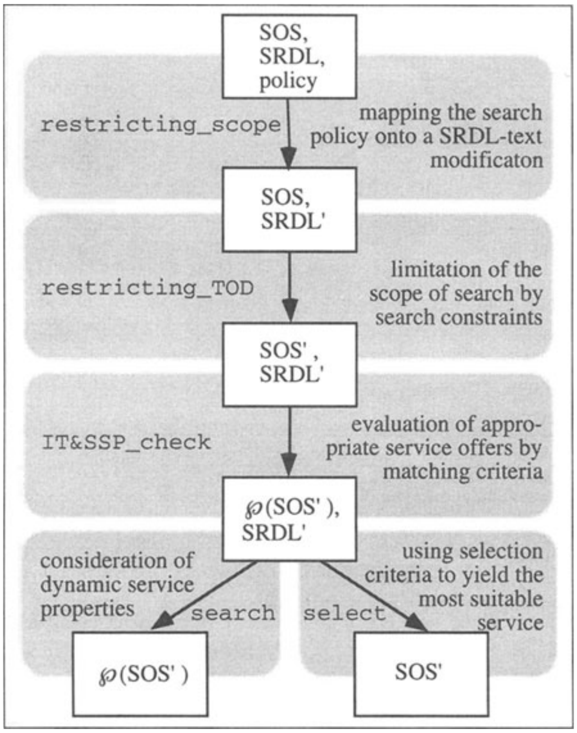
SOS,
SRDL,
policy
restricting_scope
mapping the search policy onto a SRDL-text modificaton
SOS,
SRDL'
restricting_TOD
limitation of the scope of search by search constraints
SOS',
SRDL'
IT&SSP_check
evaluation of appropriate service offers by matching criteria
℘(SOS'),
SRDL'
consideration of dynamic service properties
search
select
using selection criteria to yield the most suitable service
℘(SOS')
SOS'

Fig. 6: The four-step semantics for searching and selecting a service

The following subchapters explain these four steps of semantics in more detail. An exact formal description of each of these four steps is given.

5.1 The First Step: Restricting the Search Space

The first step of the semantics of service requests restricts the search space defined by the invoker, forming a part of the importer policy, in accordance with the search policy valid for the requested trader. Generally, the trader search policy has a higher priority than the importer policy. Thus,the search space used for a service request is an intersection of the search spaces of the trader search policy and the importer policy.

`restricting_scope`: policy, SRDL-text-> SRDL-text

The function `restricting_scope` extracts the operation_restriction for the SRDL-text from the policy specification. Since the scope_description of a policy and the search-constraint of a service-request do not have the same syntax, it also has to perform an adaption. Path specifications do not need to be changed, whereas subgraph specifications must be flatted to a set including a path for each context in a subgraph. Both, the operation_restriction and the unchanged SRDL-text serve as input for the function $[[\cdot]]_{ro}$. The semantics of an operation_restriction is given by a function $[[\cdot]]_{ro}$.

$[[\cdot]]_{ro}$: operation_restriction, SRDL-text-> SRDL-text

It maps a service request and an operation restriction onto a modified service request taking the restrictions into account. The result is a modification of the search space. It is out of the scope of this paper to present this semantics, for more detail see [PMK 94].

5.2 The Second Step: Mapping Search Constraints onto a Limited Service Offer Set

The second step of formal semantics performs a function

`restricting_TOD`: SOS, SRDL' →SOS'.

The formal semantics is given in the following. First, the SRDL-text is projected onto the search-expression.

```
if      SRDL-text = service-request
with    service-request = <service_request_operation> <search-constraint>
        "END SERVREQ",
        search-constraint = "IN" <search-expression>
then    RESULT = <search-expression>
```

A second step is to map the RESULT, i.e. the search-expression of our SRDL-text onto the corresponding set of service offers.
In our context relative naming model presented in chapter 4, unique names are guaranteed if each edge in the graph has a unique label relative to the source of the edge. For example, a search-expression "/D/B/A" denotes the subset obtained by taking the set D which contains the subset B which includes subset A. The subset relationships can be derived from the names. The context named "/D/B/A" is a subset of the context named "/D/B", which is a subset of the context named "/D". The search path is easy to derive and describes the set SOS' which is used in the following.

5.3 The Third Step: Formal Interpretation of Search and Select

In the following, we consider the third phase of our four-step-model. This step maps a set of service offers onto a smaller set of service offers according to special matching criteria.
The basic set of service offers which has to be considered is denoted by

$SOS' = (SO_1, SO_2, ..., SO_g)$,

SOS denotes the service offer set, and SO is a service offer. All together, we have a set of g elements. Considering the i-th service offer, it can be described as a 8-tupel

$SO_i =$ $(EId_i, SId_i, IT_i, SSP_i = \{sp_{i1}, sp_{i2}, ..., sp_{in_i}\}, DSP_i = \{dp_{i1}, dp_{i2}, ..., dp_{im_i}\},$
$SOP_i = \{so_{i1}, so_{i2}, ..., so_{ik_i}\}; tog_i, iog_i),$
where sp_{ij} := (<spn_{ij}> "=" <spv_{ij}>), dp_{ij} := (<dpn_{ij}> "=" <dpv_{ij}>), and
so_{ij} := (<son_{ij}> "=" <sov_{ij}>).

EId_i denotes the exporter identifier of the i-th service offer, SId_i denotes the service interface of the i-th service offer, IT_i describes the interface type of i-th service offer, SSP_i is the set of all static service properties and DSP_i is the set of dynamic service properties. The third phase of formal semantics maps all service offers in a suitable set of service offers SOS' and a matching criteria described as Service Request Description Language text into the power set of SOS', i.e., $\wp$(SOS'). That means:

`IT&SSP-check`: SRDL'-text, SOS' →$\wp$(SOS').

Beginning a search or selection process, a variable *res* denoting a result set has the value empty. The function KEEP <search> which is used in the following, stores simple BNF expressions which include dynamic service properties of type '<identifier>"/d"' for the 4th phase of formal semantics. Now, we have all means to describe the 3rd semantics phase of the SRDL.

if SRDL-text = search,
sti is a service-type-identifier,
cmc is basic-matching-criteria
with search = **SEARCH** sti **WITH** cmc,
cmc = <service-property-identifier><general-relation><real-number>
then <service-property-identifier> = <identifier>"/d" $\rightarrow$ KEEP <search>,
<service-property-identifier> $\neq$ <identifier>"/d" $\rightarrow$
$\forall$ i$\in$SOS: sti = $SId_i \wedge \exists$ x: service-property-identifier = $spn_{ix} \wedge spv_{ix} \in$ real
$\wedge$ spv_{ix} <general-relation><real-number>
$\rightarrow$ res = res $\cup$ SO_i

if SRDL-text = search,
sti is a service-type-identifier,
cmc is basic-matching-criteria
with search = **SEARCH** sti **WITH** cmc,
cmc = <service-property-identifier> <equal-or-not-symbol> <service-property-value-identifier>
then <service-property-identifier> = <identifier>"/d" $\rightarrow$ KEEP <search>,
<service-property-identifier> $\neq$ <identifier>"/d" $\rightarrow$
$\forall$ i$\in$SOS: sti = $SId_i \wedge \exists$ x: service-property-identifier = $spn_{ix} \wedge spv_{ix}$ <equal-or-not-symbol><service-property-value-identifier>
$\rightarrow$ res = res $\cup$ SO_i

if SRDL-text = search,
sti is a service-type-identifier,
cmc is a simple-matching-criteria
with search = **SEARCH** sti **WITH NOT** cmc,
cmc = <service-property-identifier><general-relation><real-number>
then <service-property-identifier> = <identifier>"/d" $\rightarrow$ KEEP <search>,
<service-property-identifier> $\neq$ <identifier>"/d" $\rightarrow$
$\forall$ i$\in$SOS: sti = $SId_i \wedge \exists$ x: service-property-identifier = $spn_{ix} \wedge spv_{ix} \in$ real
$\wedge \neg$ spv_{ix} <general-relation><real-number>
$\rightarrow$ res = res $\cup$ SO_i

if SRDL-text = search,
sti is a service-type-identifier,
cmc is simple-matching-criteria
cmc_1 and cmc_2 are simple-matching-criteria
with search = **SEARCH** sti **WITH** cmc,
cmc = cmc_1 **AND** cmc_2 .
then $\forall$ i$\in$SOS: if $\exists$ i: (**SEARCH** sti **WITH** cmc_1 $\rightarrow$ res = res $\cup$ SO_i
$\wedge$ **SEARCH** sti **WITH** cmc_2 $\rightarrow$ res = res $\cup$ SO_i)
$\rightarrow$ res = res $\cup$ SO_i

if SRDL-text = search,
sti is a service-type-identifier,
cmc is a conditional-matching-criteria
cmc_1, cmc_2 and cmc_3 are conditional-matching-criteria
with search = **SEARCH** sti **WITH** cmc,
cmc = **IF SUCCESS** cmc_1 **THEN** cmc_2 **ELSE** cmc_3 **END**
then $\forall$ i$\in$SOS: if $\exists$ i: (**SEARCH** sti **WITH** cmc_1 $\rightarrow$ res = res $\cup$ SO_i)
then search = **SEARCH** sti **WITH** cmc_2
else search = **SEARCH** sti **WITH** cmc_3

Note: So far, the "IF SUCCESS"-condition considers only static service properties.

```
if      SRDL-text = search,
        sti is a service-type-identifier,
        cmc is a conditional-matching-criteria
        $cmc_1$ and $cmc_2$ are conditional-matching-criteria
with    search = **SEARCH** sti **WITH** cmc,
        cmc = **IF SUCCESS** $cmc_1$ **THEN TAKE THAT ELSE** $cmc_2$ **END**
then    $\forall$ i$\in$SOS:    if $\exists$ i: (**SEARCH** sti **WITH** $cmc_1 \rightarrow res = res \cup SO_i$)
                        then search = **SEARCH** sti **WITH** $cmc_1$
                        else search = **SEARCH** sti **WITH** $cmc_2$
```

If SRDL-text = select with select = "**SELECT**" <service-type-identifier> "**WITH**" <conditional-selection-criteria> <search-constraint> "**END SERVREQ**", so the same semantics operations as above are valid, accepting the modification, that search is substituted by select. Until there is no rule which could be performed, these rules are applied, then the next step is performed.

Going in more detail, the above described structure is taken for all BNF-expressions, but here, only the structure of this semantics phase should be explained. The whole description could be found in an internal paper [PMK 94].

Finally, *res* = SOS'' $\in \wp$(SOS') and denotes the set of all service offers coresponding to the given matching criteria. If SRDL-text is a search request and there is a dynamic service-property-identifier so this service property is evaluated in the next step of formal semantics; if there is no dynamic service property involved in the request description, so *res* denotes the final result of the search. If SRDL-text is a select request then it is in every case necessary to consider the following step of formal semantics.

5.4 The Fourth Step: Performing `search` and `select`

First, the case of a search request should be considered. If KEEP <search> stores no expressions, so *res* denotes the final result. Otherwise, the same rules as above are applied to SOS'', accepting the extension, that $spn_i \rightarrow dpn_i$ and $spv_{ix} \rightarrow dpv_i$ are substituted, i.e.:

4th-semantics-phase(`search`) : SRDL-text, $\wp$(SOS') $\rightarrow$ $\wp$(SOS').

The result of applying these rules is an equal or smaller set SOS''' $\in \wp$(SOS'). The values for the dynamic service properties are updated.

If, in the second case, we have a select operation, it is possible to combine the rules given above and the following rules. The fourth step uses the preselected service offers included into SOS' and applies the whole set of rules, i.e.:

4th-semantics-phase(`select`) : SRDL-text, $\wp$(SOS') $\rightarrow$ SOS'

Similar, the variable *res* denoting a result set is set empty when applying all rules once more including the following rules:

```
if      SRDL-text = select,
        sti is a service-type-identifier,
        csc is basic-selection-criteria
with    select = **SELECT** sti **WITH** csc,
        csc = **MINIMUM** "(" <service-property-identifier> ")"
then    c=0, $\forall$ i$\in$SOS:   (sti = $SId_i \wedge \exists$ x: service-property-identifier = $spn_{ix} \wedge spv_{ix} \in$ real
                          $\wedge$ (c = 0 $\vee$ $spv_{ix} < s$)
                          $\rightarrow res := SO_i \wedge c := c+1 \wedge s := spv_{ix}$ )

if      SRDL-text = select,
        sti is a service-type-identifier,
        csc is basic-selection-criteria
with    select = **SELECT** sti **WITH** csc,
        csc = <choice-symbol> "(" <conditional-matching-criteria> ")"
then    ($\forall$ i$\in$SOS:   (sti = $SId_i \wedge \exists$ x: service-property-identifier = $spn_{ix} \wedge spv_{ix} \in$ real
                            $\rightarrow$ res := res $\cup$ $SO_i$ ));
                            res := <choice-symbol> (res)

if      SRDL-text = select,
        sti is a service-type-identifier,
        csc is a conditional-selection-criteria
        $csc_1$, $csc_2$ and $csc_3$ are conditional-selection-criteria
with    select= SELECT sti WITH cmc,
        cmc = IF SUCCESS $csc_1$ THEN $csc_2$ ELSE $csc_3$ END
then    $\forall$ i$\in$SOS:   if $\exists$ i: (SELECT sti WITH $cmc_1$ $\rightarrow$ res = res $\cup$ $SO_i$)
                        then select = SELECT sti WITH $cmc_2$
                        else select = SELECT sti WITH $cmc_3$

if      SRDL-text = select,
        sti is a service-type-identifier,
        csc is a conditional-selection-criteria
        $csc_1$ and $csc_2$ are conditional-selection-criteria
with    select= SELECT sti WITH cmc,
        cmc = IF SUCCESS $csc_1$ THEN TAKE THAT ELSE $csc_2$ END
then    $\forall$ i$\in$SOS:   if $\exists$ i: (SELECT sti WITH $cmc_1$ $\rightarrow$ res = res $\cup$ $SO_i$)
                        then select = SELECT sti WITH $cmc_1$
                        else select = SELECT sti WITH $cmc_2$
```

The other semantics expressions could be presented in a similar way. Now, *res* denotes the service offer, which is choosen by the selection criteria. This is the endpoint of our formal semantics.

6. CONCLUSIONS

To perform Open Distributed Processing it is important to enable a concept for searching and selecting service offers. The Service Request Description Language presented here in a BNF is a first step in that direction. It enables the specification of matching criteria, selection criteria, search constraints and policies. This language has not only a formal syntax, but also a formal semantics. Therefore it is possible to map the specification of a service request onto the offered entries given in the service directory. So far, the Service Request Description language does not consider service offer property values. It is difficult to find a general description for such properties corresponding to all exporters, but this subject is for further study. Separate examinations study several approaches to include Quality of Service (QoS)-aspects. However, at the moment there is no solution to cope with all requirements.

Another point for further work is to integrate openness into that approach. Therefore, the formal language has to be extended in a way, that the matching constraint also includes a transformation of data types. Going into more detail implies that the interface type of a service is further subdevided into signature, behaviour, constraints, and the role of the client, i.e. either client/server or provider/consumer. The signature of such an interface type includes data types. By mapping these data types it is possible to achieve openness within the well known transformations.

Further possibilities to enhance this formal language arise 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 interface type or properties which are requested. This possiblity should be includend into the language.

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.

REFERENCES

[ANSA] Architecture Projects Management Ltd.: *An Overview of ASNAware 4.1.* Document RM.099.02 February 1993

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

[Gei 92] Geihs, K.: *Trader Interaction Models and Infrastructure Implications.* In: Proceedings of IFIP TC6 International Conference on Information Networks and Data Communication IV, Espoo SF, North Holland, 1992

[Ke 93] Keller, L.: *From Name Server to the Trader - An Overview about Trading in Distributed Systems.* In : PIK 16(1993)3, pp. 122 - 133

[LOTOS] Information processsing systems - Open systems interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour. International Standard , ISO 1988

[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] ISO/IEC JTC1/SC21/WG 7 N885(preliminary): *Basic Reference Model of Open Distributed Processing - Part 1: Overview and User Model,* Nov. 1993

[ODP P2] ISO/IEC JTC1/SC21 Nxxxx: *WG7 DIS Basic Reference Model of Open Distributed Processing - Part 2: Descriptive Model,* Feb. 1994

[ODP P3] ISO/IEC JTC1/SC21 Nxxxx: *WG7 DIS Basic Reference Model of Open Distributed Processing - Part 3: Prescriptive Model,* Feb. 1994

[ODP P4] ISO/IEC JTC1/SC21 N7056: *Working Draft for the Basic Reference Model of Open Distributed Processing - Part 4: Architectural Semanitcs,* Aug. 1993

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

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

[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

[PMK 94] Popien, C.; Meyer, B.; Kuepper, A.: *A Formal Approach to Service Import in ODP Trader Federations.* In: Aachener Informatik-Berichte 94-6, ISSN 0935-3232, Aachen 1994, pp. 1-24

[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

[Ste 90] Stefani, J.: *Open Distributed Processing - The next Target for the Application of Formal Description Techniques.* In: Proceedings of FORTE III, Madrid 1990, North Holland