RZ 3178 (# 93224) 10/25/99
Computer Science/Mathematics 11 pages

Research Report

Advanced Dynamic Property Evaluation for CORBA-Based Electronic Commerce

A. Schade, C. Facciorusso, S. Field and Y. Hoffner

IBM Research
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

Email: {san,cfa,sif,yho}@zurich.ibm.com

IBM Research
Almaden · Austin · Beijing · Haifa · T.J. Watson · Tokyo · Zurich

Advanced Dynamic Property Evaluation for CORBA-Based Electronic Commerce

A. Schade, C. Facciorusso, S. Field and Y. Hoffner

IBM Research, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

Abstract

Trading in distributed systems and match-making in electronic commerce systems are similar activities since both purport to evaluate some client constraint against a set of available offers, described in terms of properties. E-commerce systems, however, require additional flexibility to cope with frequently changing offers. Many properties of those offers are dynamic, i.e. their value has to be evaluated by an associated algorithm at match-making time.

The CORBA Trading Service specification provides the concept of dynamic properties. Maintenance and execution of dynamic property algorithms, however, are outside the scope of the standard. Hard-coding these algorithms does not provide a flexible enough method for dynamic property management. This paper presents the concept and design of a generic engine to represent and execute dynamic property evaluation algorithms. This concept brings about a number of benefits: An improved development environment, a simulation test-bed, improved management and maintenance, and structured way of linking to legacy systems.

1 Introduction

Electronic commerce systems, until recently part of the Internet's strategic outlook, are already becoming web services of every day use. Although many different kinds of such services exist, ranging from books purchasing, grocery shopping to travel route planing, ticket and hotel room booking, at their core e-commerce systems are quite similar.

One important aspect of electronic commerce systems concerns the comparison of customer requirements with available service or product offers. This task of finding suitable offers is required by any kind of e-commerce system, regardless whether its purpose is merely offer browsing or the actual purchasing of goods. Match-making in e-commerce systems relates to the widely known concept of *trading* [APM93], [OMG98a] in distributed systems.

Electronic commerce systems are highly dynamic environments in which the availability of offers varies over time and existing offers are changed frequently. To make match-making relevant to e-commerce systems, up-to-date descriptions of offers are necessary. The dynamic property facility of the CORBA Trading Service specification [OMG97] provides a suitable mechanism in the form of dynamic properties. Contrary to static properties which are fixed at advertising time, the resolution of dynamic property values is deferred until match-making time.

Being a set of interface definitions, the CORBA trading specification fails to provide any guidelines as to how to create, use, modify and manage dynamic property evaluators. The problem of integrating existing back-office systems, which often provide the necessary up-to-date values, is also outside the scope of the standard.

This paper describes an approach to realizing a dynamic property evaluation service on top of CORBA trading. The core features of our approach consists of execution engines, an accompanying powerful but easy to use scripting language, script exits to other programming languages, a simulation environment for testing and a special purpose editor. Using these concepts, the approach is flexible enough to be deployed in CORBA-based electronic commerce systems where flexible offer management is an essential requirement.

The remainder of this paper is organized as follows. In Section 2 we will give a brief overview about the CORBA Trading Service specifications. The evaluation process of dynamic properties is explained in more detail in Section 3. Section 4 presents the general concept of the dynamic property evaluator solution, describes the scripting language in use and summarizes the advantages of our solution. Section 5 discusses how core concepts of the evaluator solution can be reused to implement constraint match-making. Section 6 puts our solution in perspective to recent CORBA developments, and Section 7 concludes the paper.

2 The CORBA Trading Service

In order to issue a request to an object in CORBA, a client has to know the identity of this object in terms of its object reference. The CORBA Trading Service [OMG97] can be used if the client does not know the precise identity of the server object, but knows the set of properties (e.g., name, QoS parameters, etc.) it requires from a suitable service. The Trading Service tries to find a service matching the client's requirements.

In CORBA, objects are described by the specification of the interface which is implemented by this object. This specification represents the *interface type*. For a given interface type different implementations can exist, but they cannot be distinguished by their interfaces. A *service type* is formed by extending the interface type with a set of properties of the service described. Properties are typed name-value pairs which are either *static properties* or *dynamic properties*. The value of a static property is known at export time. The value of a dynamic property is computed at match-making time (query). Different implementations of the same interface type can be distinguished by their service type (different property sets) and/or, in the case of identical service types, by different property values.

Service trading works in two steps (see Fig. 1). In a first step, an object offering some service registers itself with the CORBA Trading Service. This is done using an *export* operation. The parameters of this export operation are the object reference of the registering object, the name of the service type the registering object implements, and a set of properties that further describe the service. In a second step, a client wishing to use a service fulfilling certain requirements inquires the Trading Service using a *query* operation. Besides additional parameters, the query operation has to provide the service type the client is looking for, and the client's service requirements given as a constraint expression to the CORBA Trading Service. The operation performed by the Trading Service is also referred to as *match-making* which involves the following steps:

1. searching for a matching service type using *type conformance* rules given by the standard, and

2. applying the constraint expression to the properties of those service descriptions matched by the first step.

The query operation returns an *offer sequence* data structure which contains the object reference and the property values of the matching service implementations. The client can then iterate through the offers returned, select the most suitable one and invoke the service object.

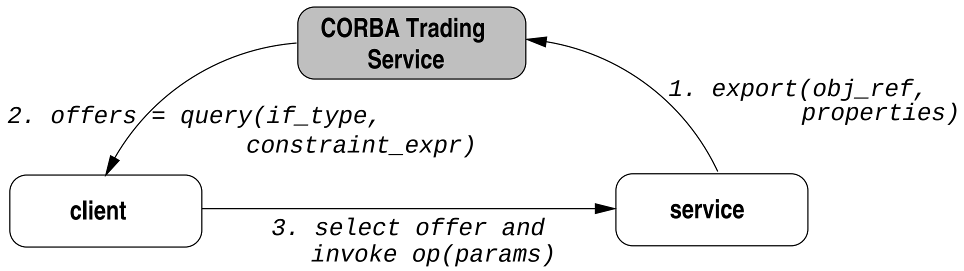


Figure 1: The CORBA Trading process

The process described above is the classical uni-directional trading mechanism as specified in [OMG97] where offer properties are matched against client constraints. If clients also submit some descriptive properties and offers also contain constraints on a potential client, the process can be made symmetric. *Symmetric trading* is well adapted to non-commodity markets where providers do not want to do business with all clients. For a detailed description of symmetric trading see [HS98].

Objects are registered with the CORBA Trading Service (export) by their service type which describes both the service interface type *and* the service properties. A client using the Trading Service therefore searches (query) for a matching service implementation. The constraint expression passed with the query operation is a logical expression to be applied to service properties. The evaluation of the constraint expression triggers the evaluation of the referenced properties. Evaluating a static property is trivial as only its corresponding value must be returned. The evaluation of dynamic properties is described in the following section.

3 Evaluation of Dynamic Properties

Contrary to static properties whose value is known at export time, the value of dynamic properties must be computed at query time. For a dynamic property of an exported service, an object reference, a return type, and optionally parameters are specified. This object reference identifies the object to be called when evaluating the dynamic property. The CORBA Trading Service standard just specifies the interface of this dynamic property evaluator object (see Fig. 2); its implementation is outside the scope of the standard.

```
// IDL

#include <orb.idl>

typedef string PropertyName;

exception DPEvalFailure {
  PropertyName name;
  TypeCode     returned_type;
  any          extra_info;
};

interface DynamicPropEval {
  any evalDP ( in PropertyName name,
               in TypeCode     returned_type,
               in any          extra_info)
  raises (DPEvalFailure);
};

struct DynamicProp {
  DynamicPropEval eval_if;
  TypeCode        returned_type;
  any             extra_info;
};
```

Figure 2: CORBA Dynamic Property Evaluator interface

A dynamic property of a service is specified using the `DynamicProp` structure. The `eval_if` member represents the object reference of the corresponding dynamic property evaluator. The return type is specified by the `returned_type` member, and required formal parameters are encoded in the `extra_info` member. The `any` type is used here to obtain universally deployable specification.

Whenever a dynamic property must be evaluated the CORBA Trading Service invokes the `evalDP` operation at the associated evaluator object. The parameters of this operation specify the name of the dynamic property to be evaluated, the expected type of the returned value, and any actual parameters as required by the dynamic property specification. The returned value represents the current value of the dynamic property.

The `evalDP` operation shown in figure 2 submits the name of the dynamic property to be computed to the evaluator object. This means that in theory the evaluator object is capable of handling multiple dynamic properties. We will make use of this fact in the next section.

4 The Dynamic Property Engine Concept

This section describes an approach to realizing a dynamic property evaluation service on top of CORBA trading which is flexible enough to be deployed in e-commerce systems by addressing the problems of executing and managing dynamic property evaluation algorithms left open by the CORBA trading service specification. Some benefits of this approach are described in Section 4.2.

We have developed a flexible method for managing dynamic property evaluation algorithms which is based on evaluation engines and scripts. Dynamic property evaluation algorithms are specified using a dedicated scripting language which is designed to relieve the programmer of heavy ordinary programming language efforts. The evaluator object is replaced by a so-called Dynamic Property Engine (DPE) which implements the `DynamicPropEval` interface as required by the CORBA Trading standard.

For each dynamic property there must exist a corresponding evaluation algorithm in the form of a DPE script. When the dynamic property is specified as part of the export operation then the name of the handling DPE Engine, the DPE script name, the return type, and the set of actual parameters (which may themselves reference other properties) must be provided. Evaluating a dynamic property triggers the corresponding DPE Engine to interpret the associated DPE script (see Fig. 3).

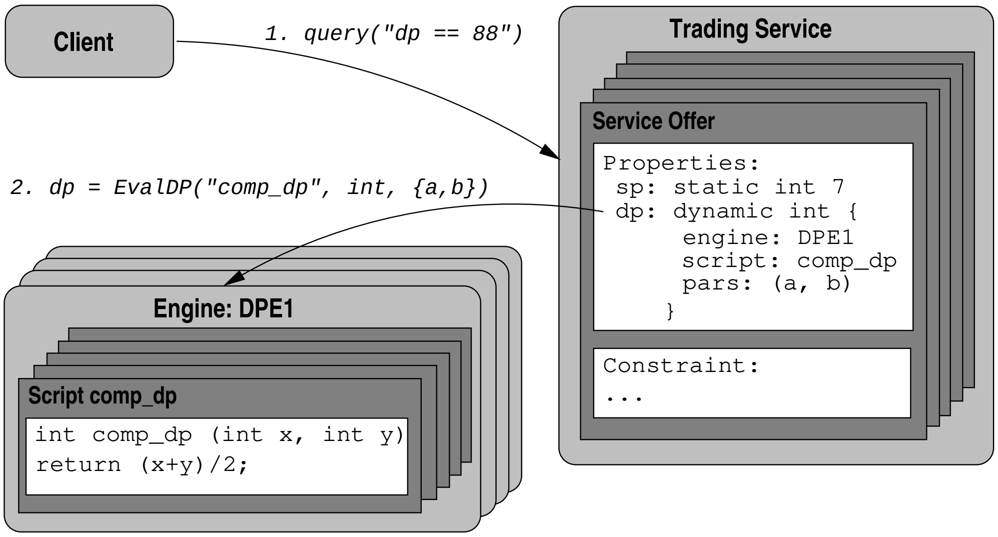


Figure 3: Evaluation of dynamic properties using the DPE Engine

The call to the DPE Engine requesting the evaluation of a script communicates the name of the script and the actual parameters to be used. The script selection method uses both the name of the DPE script and the supplied actual parameters to select the DPE script to interpret, thus allowing evaluation algorithms to be used with different parameter sets. The matching DPE script is chosen and evaluated. Upon termination the evaluation returns a value whose type is compliant with the dynamic property type as specified in the export operation. This value represents the current dynamic property value.

The DPE Engine implements the interface `DynamicPropEvalEngine` which is an extension of the standard `DynamicPropEval` interface. The additional operations are used to manage the script base in a DPE engine. The extended interface is shown in Fig. 4.

A DPE Engine maintains and executes DPE scripts through a well-defined interface. A tool, called the DPE Editor, is used to write DPE scripts and to submit them to the DPE Engine in textual form. The interaction between the DPE Engine, the DPE Editor and the CORBA Trading Service is illustrated in Fig. 5.

The Engine contains a two-pass compiler which validates syntax and static semantics of the script. Once compiled successfully, the script is then registered with the DPE Engine. Syntactic and semantic errors are indicated using CORBA exceptions. Using the DPE Editor, a registered script can be modified, deleted, and executed for test purposes.

4.1 DPE scripting language

The DPE language provides a scripting mechanism which can be used to define evaluation algorithms for dynamic properties in trading environments. The general idea is to store these DPE scripts at a DPE Engine which implements the `DynamicPropEval` interface as defined in the CORBA trading service specification. Whenever a trader has to evaluate a dynamic property, it invokes the `evalDP` operation at the DPE Engine, providing the name of the dynamic property, the

```
// IDL

interface DynamicPropEvalEngine : DynamicPropEval {
        exception DPEParseException {
                string reason;
        };

        exception NoSuchScript {};

        string submit(in string dpeScript)
                raises (DPEParseException);

        void   update(in string name, in string dpeScript)
                raises (NoSuchScript, DPEParseException);

        typedef sequence<string> stringSeq;
        long getScriptHeaders(out stringSeq scripts);

        string getScript(in string name)
                raises (NoSuchScript);

        void deleteScript(in string name)
                raises (NoSuchScript);
};
```

Figure 4: Dynamic Property Engine interface

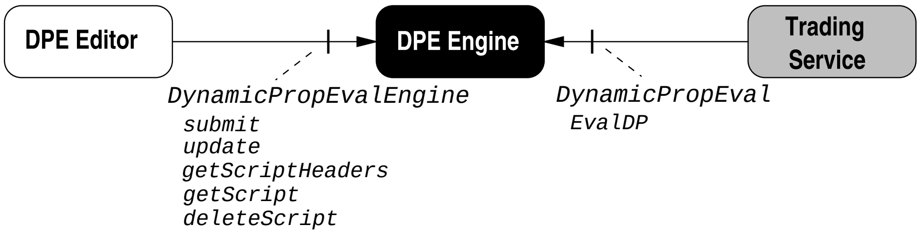


Figure 5: Interactions with the DPE Engine

expected return type and any optional parameters required by the script. For space considerations the description of the language features is kept briefly.

The DPE scripting language is a strictly typed language. It supports six primitive types (integer, float, string, character, boolean, and date), and allows the construction of aggregate types such as records and sequences. While record types can contain other record and sequences, multi-dimensional arrays (sequences of sequences) are not supported similar to the CORBA constraint language proposed by the standard.

A DPE script can be considered like a function in any other programming language. It consists of a signature (name, return type, formal parameters) and a body. The return type must comply with the expected return type of the DPE script as specified in the export step. The script's body consist of a set of statements which are interpreted top-down.

A statement may be the definition of an auxiliary variable, an assignment of some value to a variable, a control-flow statement, or an expression. The following control-flow statements are implemented:

- *if*-conditions,
- *while*-loops,
- *do-while*-loops, and
- *for* loops.

All expressions return a typed value. The script's return value is generated by either a return statement or the last statement of the body. The language is not scoped, that is all variables as well as for-loop counters remain valid throughout the script.

Unless there is an additional type mapping module, the DPE type system also determines the type system to be used throughout the trading infrastructure. This implies that properties must be defined using the same types that can also be specified in a DPE script.

The DPE scripting language supports a number of different operations for different types. Most of these operations are polymorphic, that is they work on different types of operands with different semantics.

In addition to a number of built-in functions for frequently needed operations such as string or date manipulation, a DPE script may invoke other DPE scripts as sub-routines. Direct recursion is also possible. DPE scripts can also call (static and non-static) public methods in any Java class. Each DPE type has a corresponding representation in Java file. All methods can be called provided that their parameters and return values are type-compliant to the DPE typing model.

Example: To conclude the overview of the scripting language we shall give a short example of a dynamic property evaluation algorithm. Suppose there is an offer of some product with a dynamic property "price". The value of the price property is computed by a script "discount_price". The price per piece shall be $5. The price of the offer depends on the number of items ordered: If more than ten pieces are ordered a discount of 5 percent is granted, and for more than 100 pieces there is a 15 percent discount. Additionally, if the order is placed before November 1st, the calculated price is reduced by another 3 percent.

While the current date is available to the evaluation engine, the number of ordered pieces must be passed as a parameter. Therefore, the script header contains only one formal parameter. The dynamic property algorithm represented in the DPE script language may appear as illustrated in Fig. 6.[1]

```
float discount_price (int noOfPieces)          # script header

float price = noOfPieces * 5.0;  # normal price for this order
if (noOfPieces > 100)            # for more than 100 pieces
  price = price * 0.85;          # grant a 15% discount
else if (noOfPieces > 10)        # for more than 10 pieces
  price = price * 0.95;          # grant a 5% discount
if (<> < <2000/11/01>)           # if the current date <> is
                                 # before November 1st, 2000
  price = price * 0.97;          # grant another 3% discount
return price;                    # return the calculated price
```

Figure 6: A sample price calculation algorithm

4.2 Benefits of the DPE Mechanism

Using the concepts of execution engines and scripting for dynamic property evaluation has some important advantages when deploying a trading solution in electronic commerce systems.

[1]Note that script body is not enclosed in braces. A DPE script always starts with the header immediately followed by the body statements.

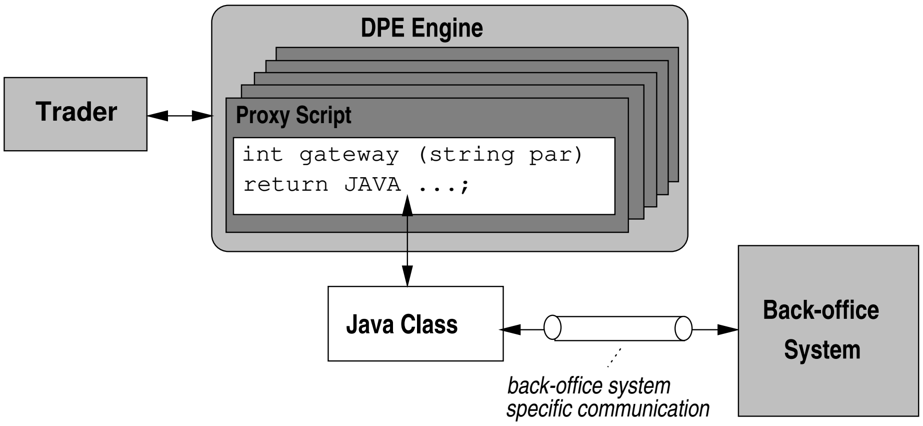


Figure 7: DPE scripts as gateways between trading and back-office systems

Improved development environment: Different versions of scripts for both testing and production can be kept in separate engines. The transition form the testing phase to the production phase can then simply be accomplished by changing the reference to the DPE Engine in the dynamic property description advertised in the trader.

The same effect can be achieved if only one engine is used by maintaining two scripts and changing the script name in the dynamic property description. While the existing solution uses one script the new version can be tested simultaneously.

Gradual transition from one script version to another is possible in a similar manner: An intermediary script can be used which invokes both the old and the new version and compares the results raising notifications if the results differ.

Script simulation and testing: DPE scripts being developed can be tested using the DPE Editor. The editor invokes the same script evaluation method that is also used the by Trading Service. Actual parameters can be supplied simulating usage cases in real trading.

Access to legacy systems: DPE scripts can call methods in external Java classes. This indirection provides great flexibility and allows easy bridging to existing back-office systems. Legacy systems implemented in C or C++ can be accessed using the Java Native Code Interface. Exits to other programming language can also be accomplished in this manner. The external Java code can also act as a gateway implementing CORBA client functionality for accessing CORBA-enable back-end systems, or stub functionality for communication based on message queueing. Finally, back-office data base systems can be tied in using ODBC/JDBC connectivity. A schematic illustration is given in Fig. 7.

Advanced manageability: The trading system can use multiple DPE Engines distributed in a network. Load balancing becomes feasible then by distributing dynamic property scripts among these engines ensuring that frequently called scripts reside in different engine running on separate hosts.

Script organization: Since a DPE script can call other scripts either locally or remotely, existing scripts can be re-used to accomplish tasks that have often to be performed when dynamic properties are evaluated. This feature allows also to build hierarchies of scripts which reflect the organizational structure of the provider. The responsibility for maintaining these scripts can then be delegated, too.

Simplified tracing and error detection: DPE Engines can log their activity either at a central place or to different authorities in the provider's organizational structure. It is thereby possible to collect statistics information about DPE script usage. Another important aspect is to detect errors during script execution which can be hard to identify as they often result in trading mismatches. In case of abnormal situations DPE Engines can emit alert notifications to managers.

Improved maintenance: The modularity of the engine concept allows dynamic property evaluation algorithms to be modified while the trading infrastructure is up and running.

5 An Extended Constraint Language

Parts of the DPE Engine code which is used to execute a DPE script can be reused in the match-making facility (trading service object) for interpreting constraint expressions. The concept of a constraint expression can be specialized to a constraint script under the assumption that the script always returns a boolean value indication match or mismatch.

Contrary to a DPE script which relies on the concept of parameter passing to represent external values, a constraint script refers directly to the properties over which it operates. The execution environment for the constraint script is the CORBA Trading Service itself which possesses all the necessary information about offer properties. A constraint script, thus, does not require a signature.

In addition to DPE script evaluation, two different aspects must be considered:

- As the trading service attempts to match all existing offers against a given constraint script, the script can refer to property values which do not exist in every offer.
- A constraint script can refer to a dynamic property value more than once.

5.1 Unknown Property Values

The first problem relates to so-called unknown property values. The problem can escalate to the point where a constraint script cannot be evaluated with certainty to a boolean value. To cope with this case, an additional pseudo-value "unknown" for expression results in constraint scripts has been introduced. This special "unknown" value is propagated in a constraint script as follows. Any expression, except for logical AND and OR, in which one of its operands is "unknown" yields "unknown" itself. For logical conjunction and disjunction special value tables are used as shown in Fig. 8.

AND

operands	*true*	*unknown*	*false*
true	*true*	*unknown*	*false*
unknown	*unknown*	*unknown*	*false*
false	*false*	*false*	*false*

OR

operands	*true*	*unknown*	*false*
true	*true*	*true*	*true*
unknown	*true*	*unknown*	*unknown*
false	*true*	*unknown*	*false*

Figure 8: Value tables for logical AND/OR operations using unknown values

If the entire constraint script yields the special "unknown" value two narrowing policies could be applied to re-map the constraint script result to the classical boolean domain:

- Optimistic approach: Returning "true" for an "unknown" value indicates a match for all offers which could not be excluded with certainty.

- Pessimistic approach: Returning “false” for an “unknown” value indicates a mismatch for all offers which could not be matched with certainty.

5.2 Repetitive Dynamic Property Evaluation

The second problem of multiple dynamic property references in a constraint script can lead to the disadvantage of unnecessary re-evaluation of a dynamic property, or, more dangerously, return different results for each execution (if the script has changed in the meantime). This problem can be handled by introducing *caching policies* for already computed dynamic property values.

A caching policy associated with a dynamic property determines the period of time a computed dynamic property value remains valid for (*validity interval*). A dynamic property value which is still valid does not need to be re-computed, and the script execution can be saved. The caching policy is expressed in a certain specification languages which allows the specification of the validity interval in different ways.

The validity interval can begin or end during the time a DPE script is executed. Parts of the script would be executed outside the validity interval, and dynamic property access in this period of time is not protected against re-evaluation. To overcome this undesirable effect, the validity interval specified by a caching policy is extended using the following procedure:

1. While a constraint script is executed all computed dynamic property values will be cached.

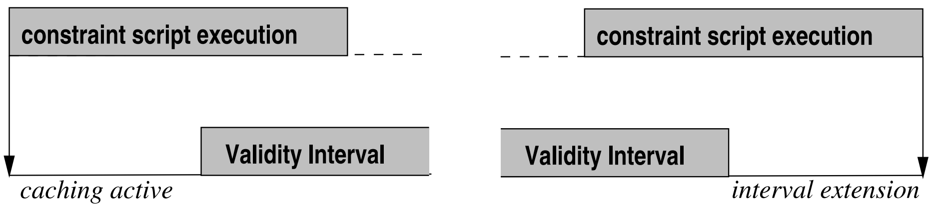
2. If so indicated by the caching policy, the computed values remain valid after the script execution has terminated. They become invalid at the end of the corresponding validity interval.

3. If a validity interval ends during script execution the interval is extended until the script terminates.

This leads to two possible extensions of a validity interval.

a. **Front-end extension:** The caching of property values starts before the validity interval begins (see Fig. 9a).

b. **Back-end extension:** The validity interval ends before the constraint script evaluation terminates (see Fig. 9b).



(a) front-end extension (b) back-end extension

Figure 9: Possible extensions of the validity interval

6 Applying new CORBA Features

According to the CORBA Trading Service standard, the interface of the dynamic property evaluator object makes use of the `any` type. On the one hand, this is necessary since no assumptions can be made on the specific type returned by the `evalDP` operation and on the types of the parameters. On the other hand, the `any` type is one of the biggest obstacles in CORBA programming. Since an `any` value is intrinsically structured according to the specific type of the value represented as `any`, this value cannot be mapped directly (type-cast) to a corresponding target programming language type. Therefore, most CORBA implementations provide dedicated marshaling and unmarshaling methods for basic scalar types to store and retrieve value as "anys". For structured types, however, this approach obviously does not work.

The concept of passing structured type definitions *by-value* between CORBA objects has recently been introduced to the CORBA standard [OMG98b]. To reflect this concept in OMG IDL, the new keyword *valuetype* has been added to the set of IDL keywords. Value type definitions represent a cross between a struct and an interface [Vin98], and can be defined using the same inheritance rules as for interface types. Besides normal member variables they can also contain operations similar to an interface type. When used as parameters or return types of operations, the semantics of the call becomes different to a call on an interface type since entire values will be passed instead of object references. Any further operation invoked by a receiving object on a value definition will therefore result in a local operation call.

This concept is well adapted to specifying dynamic property evaluator interfaces. A universal container type *Value* could be defined to represent the set of different DPE types. The resulting interface definition for the `DynamicPropEval` interface would then look very similar to an interface used in a DPE implementation on top of Java-RMI. The evaluator can then safely receive and return values of all possible types as defined by the CORBA constraint language. The data would be transferred by value instead of merely passing references to remote CORBA objects. Since the value objects would be represented as objects in the target programming language, explicit marshaling and unmarshaling is no longer required.

7 Conclusions

The DPE Engine concept has been implemented and successfully used in the Virtual Market Place (ViMP) project [HFF+99]. ViMP is a toolkit which supports design, development, creation and maintenance of market places for non-commodity offers. At the heart of such a market place is a match-making facility based on the CORBA Trading Service. Offers described using dynamic properties (e.g., price, availability) occur frequently in electronic commerce scenarios such as ViMP.

DPE Engines offer some fundamental advantages. The DPE scripting language was designed as a simple, yet powerful notation for evaluation algorithms. The person responsible for designing offers is no longer required to have in-depth distributed systems and programming language knowledge. Distribution issues (encoding, failure) are completely shielded from the programmer. Another obvious key benefit is the simplified testing and manageability of dynamic property evaluation algorithms. Property evaluation algorithms can be changed while the trading infrastructure is up and running. Existing back-office systems (quotation engines) implemented in form of dynamic link libraries or external databases can be easily integrated using the Java method call feature and native code or ODBC/JDBC connectivity, respectively.

By exploiting the scripting method for dynamic property evaluation algorithms to represent constraint expressions, large parts of the code base can be shared. Unspecified property values are represented using a pseudo value. The narrowing of constraint script results indicating match or mismatch permits an optimistic or a pessimistic approach. Referring directly to dynamic property values in constraint script can lead to unnecessary re-evaluation which is avoided by caching policies.

References

[APM93] APM Ltd.: The ANSAware Reference Manual. Architecture Project Management Ltd., Poseidon House, Cambridge, CB3 0RD UK, 1993.

[HFF+99] Hoffner Y., Facciorusso C., Field S., Schade A.: Distribution Issues in the Design and Implementation of a Virtual Market Place, *Proceesings of the Second IFIP WG6.1 International Working Conference of Distributed Applications and Interoprable Systems* (DAIS'99), Chapman & Hall, London, 1999.

[HS98] Hoffner Y., Schade A.: Co-operation, Contracts, Contractual Match-Making and Binding. *Proceedings of the Second Enterprise Distributed Object Computing Workshop* (EDOC'98), pp. 75-86, La Jolla, California, 1998.

[OMG97] Object Management Group: CORBAservices: Common Object Services Specification - Trading Object Service Specification v1.0. OMG Document #97-12-23, 1997.

[OMG98a] Object Management Group: The Common Object Request Broker: Architecture and Specification. Revision 2.3., 1998.

[OMG98b] Object Management Group: Objects by Value. OMG TC Document orbos/98-01-01, 1998.

[Vin98] Vinoski S.: New Features for CORBA 3.0. *Communications of the ACM*, 41(**10**), 44–52, October 1998.