PIA - A Generic Model and System for Interactive Product and Service Catalogs

Florian Matthes and Ulrike Steffens

Software Systems Institute,
Technical University Hamburg-Harburg,
D-21073 Hamburg, Germany
{f.matthes, ul.steffens}@tu-harburg.de
http://www.sts.tu-harburg.de/

Abstract. This text motivates and defines a generic model for interactive (online or offline) product catalogs. Based on a detailed requirements analysis, the data model is defined using an object-oriented design notation and the query language for expressing customer interests on the catalog is defined using techniques from fuzzy set theory. The model provides the basis for the implementation of a generic, highly-interactive catalog management system which is designed to be interfaced with relational databases, information-retrieval engines and special-purpose index structures.

1 Introduction and Motivation

An ever increasing number of enterprises is using digital media like online web servers or offline CD-ROMs as part of their communication with (potential) customers. Interactive digital communication media have the potential to close the gap between two extreme communication modes. One extreme is the direct, one-to-one conversation between an individual customer and a well-trained sales person. The other extreme is the mass distribution of anonymous information to the customer via broadcast media like advertisements, product fact sheets or product catalogs. In this paper, we present a generic data, language and system model for interactive (online or offline) product catalogs that act as *active assistants* to help customers find items (products, services, information resources) matching their very personal interests and preferences. In particular, the system is capable of establishing and sustaining a long-term customer relationship which may lead to a valuable learning process involving customers, catalog maintainers and information providers.

We chose to name this model and system PIA (*personal information assistant*) emphasizing the active role of the system, driving a conversation from an initial (fuzzy) customer request towards a detailed mutual understanding of the (mis-)match between the customer's personal preferences and the available corporate services and products. This should be seen in contrast to a more traditional data-centered view of such product cataloges as index structures or search engines.

As a consequence, our work focuses on those objects and algorithms that are visible from the catalog users' perspective (customers, maintainers and providers) and that are relevant for the long-term cooperation between these users taking into account the inevitable evolution of customer interests and of the catalog contents and organization. At the present stage of our work, we are less interested in identifying a restricted query language with strict algebraic properties which can be exploited for clever query optimization, but we prefer to provide customers with a rich (but formally defined) language of predicates and connectives for the interaction with the catalog which can also help the catalog maintainer and the information provider to learn from these customer needs.

In this paper, we abstract from the specific terminology used in particular application domains (business, trade, library management, knowledge management, etc.) and focus on the strong commonalities in the information structures and interaction patterns with catalogs in these domains. We therefore talk generically of a *catalog* of *items* with *properties.* The catalog items do not need to be physical objects described by numeric or discrete values, they can as well be service descriptions (trips offered by a travel agency, financial and insurance services, ...) or descriptions of information resources (bibliographic data, news articles, image descriptors, software components, ...). Each property of an item is defined by a *value* for an *attribute.* An *attribute* is associated with a *domain* which can be a numeric, a discrete, a full-text or a special-purpose domain defined by the catalog maintainer. A *customer* expresses his/her *interest* interactively as *alternatives*, each of which combines several *criteria.* Each criterion is expressed by a parameterized *predicate* on a single attribute.

Our model is designed to be applicable to a rather homogeneous product catalog of a single producer (e.g., all cars produced by BMW, all books published by Springer) but also to a heterogeneous catalog integrating diverse products of many providers with no common property schema (e.g., all products at Harrods in London, all documents in a library).

An important assumption of our work is the fact that PIA only provides decision *support* for a customer. Therefore, we do not have to take into account the problem of *decision making* based on fuzzy requirements and partial information.

The main contributions of our work presented in this paper are

- the conceptualization and formalization of the generic objects and cooperative processes involved (like interest, alternative, criterion, attribute, domain, score, offer, decision, notification etc.), as described in Chapter 2 and in Chapter 3,
- the conceptualizition and formalization of generalized fuzzy query operators in a query language which permits combined similarity queries ranging over numeric, discrete and full-text domains and also supporting semi-structured data objects (see Chapter 4)
- the identification of supporting (commercially available) technologies and their consistent integration in an overall system architecture with well-defined component interfaces, demonstrated by a prototypical PIA implementation with an elaborate interactive front-end (see Chapter 5).

2 User Groups of Personal Information Assistents and their Requirements

As explained in the introduction, we can distinguish three classes of cooperating actors interacting with PIA: *Customers* approach the system to easily find products best suited for their interests. *Providers* employ the system as a show-case or electronic marketplace, publishing information on the products they offer. *Maintainers* act as mediators, keeping the catalog in a form that supports a long-term provider-customer relationship.

The goals of a *customer* are

– to gain an understanding of the items in the catalog and their properties by browsing, inspecting individual items, asking for representative samples, viewing summaries or ranked lists according to his/her personal criteria,
– to refine incrementally his/her personal interests from initial fuzzy qualitative requirements to more detailed quantitative criteria possibly structured into decision alternatives,
– to obtain specific offers from the provider taking into account his/her personal preferences,
– to decide on follow-up actions involving the matching items (buy a product, request a service, make use of a piece of information, enter a negotiation process using a different communication media, request notifications on *new* similar items becoming available in the future),
– to be able to return at a later point in time and to easily resume a previous interaction with the assistant and being recognized as an individual customer with a particular past conversation history.

The goals of a *provider* are

– to clearly arrange all items available, being able to capture and emphasize their diversity and not being forced to fit them into a rigid common property schema,
– to provide multiple access paths to the same item in order to give the customer maximum freedom in his/her decision process,
– to learn from and to recall the personal interests of individual customers,
– to identify and to quantify larger customer groups with shared preferences or interests,
– to learn from the reactions of customers to offers made.

The goals of a catalog *maintainer* are

– to integrate information from multiple providers,
– to detect and to resolve mismatches in attribute names, mesasuring units, or in the terminology used by multiple providers,
– to accomodate the evolution of domains, attributes and properties over time, with minimal disruption of existing customer relationships,
– to develop and enforce a consistent terminology for a smooth communication between customers and providers.

3 Design of a Generic Object Model for Personal Information Assistants

Our model makes use of two complementary formalisms: First, the central semantic concepts like *customer interest*, *criterion*, *item* or *property* are represented as objects in an object-oriented data model. They can be mapped to values in (standard relational) databases, copied as structured values across the network and be associated with each other. We use the unified modeling language UML [1] and the standardized base types of Java to formally describe the static semantics of the PIA model. The main PIA concepts are summarized in Fig. 1 and will be explained successively within this chapter.

The dynamic semantics of the model relies heavily on fuzzy set theory [2, 3], as a second formalism employed in our work. For example, an interest i is formalized as a matching method *i.score()* which assigns a *score* (a real number from the interval [0,1]) to a given item. A score of 1 expresses a perfect match while a score of 0 is assigned to items that are irrelevant w.r.t. to this interest.

3.1 Modeling the Catalog Users

The top of Figure 1 is a description of the three classes of actors cooperating with a PIA. *Customer*, *Maintainer* and *Provider* are subclasses of a common superclass *Actor*. A standard login procedure can be used to authorize and to identifiy maintainers and providers. However, immediate identification of customers may not be desirable and can be delayed until a later conversation phase, for example, when a subscription request is issued by a customer with a personal e-mail address. At the beginning of an anonymous interactive session, a new customer profile is created that can be merged with another customer profile, as soon as the identity of the customer can be determined (method *mergeWith* of class customer).

3.2 Modeling the Catalog Contents

A PIA catalog is a collection of items (see classes on the right of Fig. 1). For communication purposes between customer, provider and maintainer, each item has to be assigned a printable unique *catalogID*. In practice, the choice of a naming scheme for catalog items is very important for the long-term evolution of the catalog (e.g., to identify sucessive versions of a product), but in the examples of this paper, simple integer numbers will be used.

An item in the catalog does not need to capture all properties of an item but there may be further information on the item itself available elsewhere. In PIA, an attribute *details* is provided by objects of class *item* which can hold the URL of further information sources. This URL can also reference the item itself in case it is a purley digital artifact (a document, a piece of software, or a digital image).

In PIA, each item is described by a set of properties. A property is a pair consisting of an attribute and a value, for example:

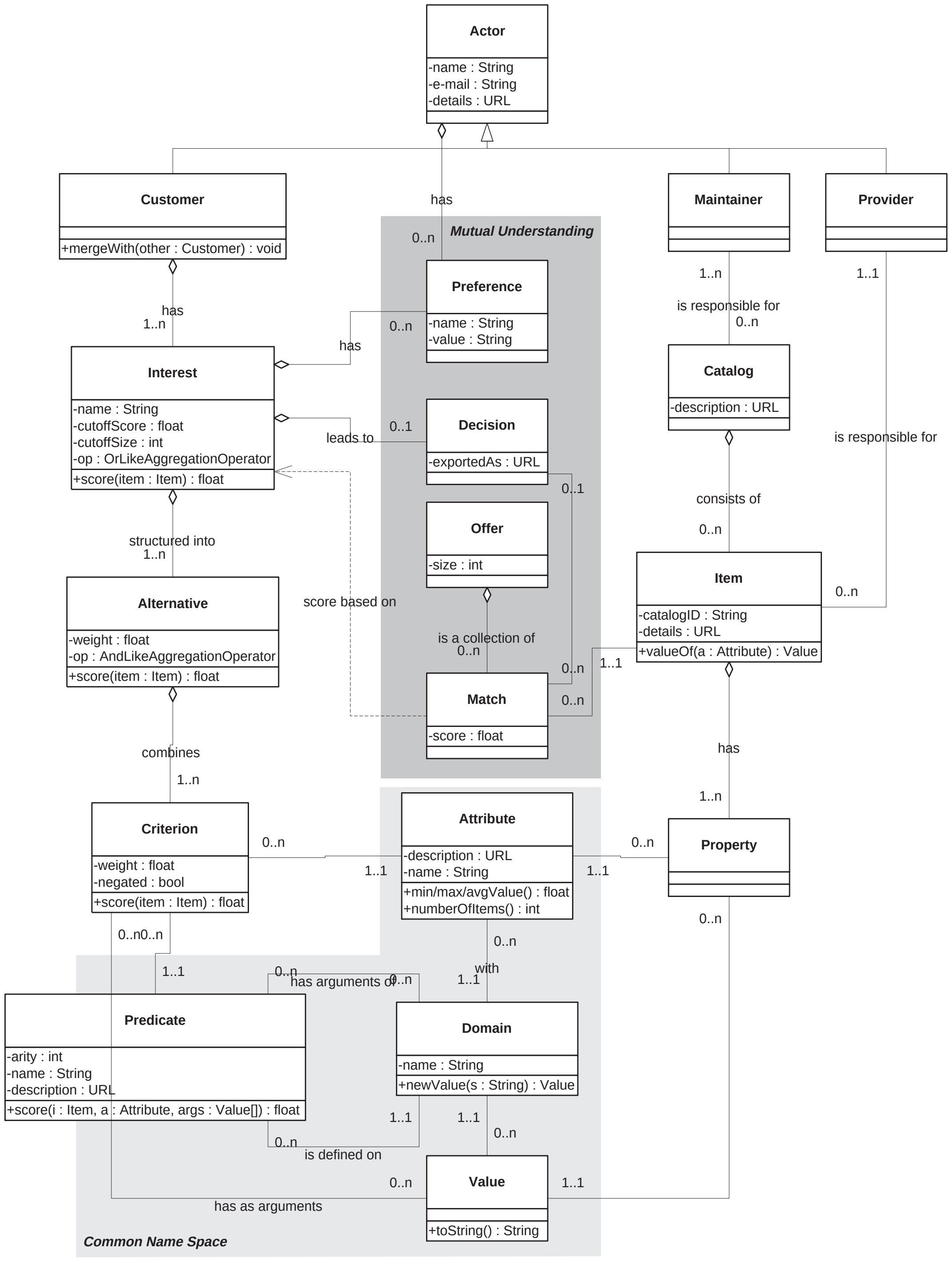


Fig. 1. Overview of the PIA Object Model

(1345, { (product name, BMW 328i), (product category, sports car), (price, (30000, USD)), (average fuel consumption, (11 l/100km)), (maximum speed, (230 km/h)), (SRS, present), (GPS, optional), ...})

Each attribute is identified by its name (e.g., *SRS*) and may have an optional link (URL) to a detailed description of the semantics of the attribute. For example, it should be explained what a SRS is or what measuring technique is used to determine the fuel consumption of a car, etc. Such descriptions play a central role in conversations between customers and sales persons. They contribute significantly to the customer's learning process and help her/him to get acquainted with the interactive catalog.

Each attribute is based on a domain that defines the universe of values available for the attribute and a set of predicates defined on values of the universe. Multiple attributes may share the same domain (e.g., *average fuel consumption* and *fuel consumption for city traffic* are both based on the domain *VolumePerLength*).

In our model, the product category of an item is defined as a regular property of the item. In particular, there is no typing or schema enforced on items of a certain product category. As a consequence, an information provider is free to attach arbitrary properties to an item or to omit properties for attributes where the values are unknown, unspecified, or simply not favorable.

3.3 Modeling Domains and Values

Despite the lack of schema constraints for product attributes, PIA enforces a large number of constraints on domains and values which are essential for the consistency and usability of the catalog. Domains and values exist indepent of concrete items, they define *possible* properties and they constitute the common language for the communication between customer and provider. The maintenance of domains and their associated objects (measuring units, discrete values, predicates, similarity functions etc.; see bottom of Fig. 1) is the task of the catalog maintainer.

Each domain has a unique name in the catalog and provides a *newValue* method which is similar to a constructor in object-oriented programming languages. The method returns a value of the domain identified by a textual description passed as an argument to the method. Such a value-identifying string is called a *literal* in programming language terminology and is required in PIA to convert textual descriptions submitted by customers, maintainers and providers into internal value representations supporting efficient storage and retrieval. For example, the domain *fuel consumption* defines a method *newValue* which given the string *"12.3 l/100 km"* returns a value that represents a fuel consumption of 12.3 liters per 100 kilometers.

PIA distinguishes three built-in classes of domains with distinct value representation. A concrete catalog assistant built with PIA contains a large number of domains each of which belongs to one of these three classes and has been defined and customized interactively by the catalog maintainer.

Numeric Domain A Numeric Domain describes numeric values represented internally as floating point numbers. A numeric domain may impose a (fixed) upper or lower bound on the admissible values. A numeric domain d can either describe ordinal numbers ($d.isOrdinal$, like the number of seats in a car) or it contains values that describe quantitative properties of items (like their length or their speed) which are expressed using a measuring unit (centimeter, inch, hour, second). Therefore, each numeric domain is associated with exactly one measuring unit (e.g. miles per hour). This unit can be regarded as a kind of reference unit for all other measuring units that measure the same physical phenomenon (e.g. meters per second or kilometers per hour). Values for a numeric domain expressed in these derived measuring units are converted by PIA via a sequence of offset and factor calculations involving intermediate measuring units to the desired reference unit. This modeling is motivated by the requirement to allow customers and providers to work with their preferred measuring units and to also provide active assistance during interactive user input.

Discrete Domain A Discrete Domain consists of a finite number of discrete values repesented by literals (or by aggreed-upon iconic visual representations), for example:

TV Norm = {*PAL, Secam, NTSC*}

In many cases, discrete values used by providers and customers describe overlapping concepts and over time the set of discrete values evolves, including specialization and generalization of individual concepts. For the purpose of catalogs, it is therefore highly desirable to be able to arrange the discrete values in a single rooted tree such that the children of a discrete value specialize the concept of their parent.

TV Norm = *(Any TV Norm,* {
(PAL, { *PAL B/G, PAL I, PAL L*}*),*
(Secam, {*Secam B/G, Secam D/K, Secam L*}*),*
(NTSC, {*...*}*)*}

The root of the hierarchy and the other inner nodes of the tree can be used by customers and providers as property values to denote approximations of existing values or discrete values not (yet) added to the domain by the catalog maintainer (e.g. after the release of a new PAL TV norm). This is again an example for a learning process where the catalog maintainer depends crucially on input (in this case notifications about exceptional values) from customers and providers to decide on ways how to improve the catalog.

Full-Text-Domain A Full-Text Domain consists of full-text values with no further restriction on the contents of these strings. A catalog may feature

distinct full-text domains to distinguish string values expressed in different languages (e.g. in English or in German). Strings and their domains are modeled as subclasses *FullTextValue* and *FullTextDomain* of the PIA classes *Value* and *Domain*, respectively (compare Fig. 1).
There is a tradeoff between the use of full-text domains and the use of discrete domains since the maintenance of a set of discrete values puts a burden on the catalog maintainer and the information providers who have to agree on this common vocabulary. On the other hand, customers are eager to gain a deeper understanding of the search space which is better supported by the organized structure provided by discrete domains. A run-time conversion (evolution) of a catalog domain from a full-text to a discrete domain and vice versa should therefore be supported.

For specific product catalogs, it may also be necessary to define additional refined domains, for example, to describe the domain of person name. By appropriate subclassing of the classes *FullTextDomain*, *FullTextValue* and *FullTextPredicate*, a catalog builder can choose a canonical internal representation for person names and can define specific predicates on person names (is similar to, sounds like, has family name, has first name, has low edit distance). Other examples of special-purpose domains are dates of the gregorian calendar or longitude/latitude coordinates on the globe.

The concept of domains could also be extended to support set-valued domains (colorset is a set of color) and aggregate domains (address is an aggregate of street, number, and address) by an appropriate subclassing of the classes *domain* and *value* to introduce set and record type and value constructors, respectively. This extension would lead to an orthogonal type system where type constructors can be nested to arbitrary depth (e.g. to define sets of records with set attributes). However, this generalization would add significant complexity to the model and its implementation.

3.4 Modeling Customer Interests

A customer may have several unrelated interests (e.g., buy a book, find a present for a friend) pertaining to the same catalog. Therefore, PIA maintains a set of interests per customer which are distinguished by name (see classes on the left of Fig. 1).

As a first cut, each interest can be described by a (non-empty) set of criteria, for example:

{*(product category, is, convertible car, very important),*
(price, is not much higher than, (15000, USD), normal),
(maximum speed, is more than, (160 km/h), normal),
(fuel consumption, is not too high, (), normal),
(color, not(is), (yellow), normal),
(number of seats, is between, (2, 4), normal)
(SRS, is, (present), normal),
(GPS, is, (optional), not so important) }

In our model, each criterion is expressed by a predicate on an attribute of the desired item(s) with an associated weight. Each attribute and predicate specified in a criterion has to be defined previously by the maintainer of the catalog (see Sec. 3.3 and Chapter 4). Each predicate can be negated expressing the fact that the customer is interested (most) in those items that do not satisfy the predicate.

The weight of a criterion is intended to capture the relative importance of a criterion w.r.t. to the other criteria contributing to the interest. Technically speaking, a weight is a numeric value in the interval [0..1]. In order o simplify the interaction with customers, weights are denoted by literals chosen from a fixed (small) set of names (e.g., *very important, important, normal, less important, not so important* or simply *essential, nice to have*).

Before the exact semantics and properties of predicates, criteria and of the dynamic combination of criteria are defined in Chapter 4, we have to further refine the notion of a customer interest.

A simple "conjunctive" combination of (negated) criteria is sufficient in the early stages of the customer's decision making process where he/she is exploring the space of items available and he/she is zooming in or out of the space by interactively adding or removing criteria or modifying predicate parameters. However, as a result of this refined understanding, customers tend to formulate *alternatives* each of which is in turn described by a combination of (negated) criteria. The decision which alternative is to be pursued further is made by a direct comparison of the items matching these alternatives. For example, a customer could formulate the alternative to either buy a light-weight laptop with a long battery lifetime *or* a fast desktop PC with a 3D graphics accelerator.

In PIA, a customer interest can therefore be described as a (non-empty) set of alternatives each consisting of a set of (negated) criteria. An item can match the customer's interest if it matches one (or several) of the alternatives. This "disjunctive" combination of alternatives and the semantics of weights assigned to individual alternatives are described in Chapter 4.

Especially when searching catalogs with large amounts of items, a customer may want to restrict the number of potentially suitable items returned by the system. Thus, the information assistant provides the possibility to equip a customer interest either with a cutoff score, returning only those items for which the evaluation results in a score higher than the cutoff score, or a cutoff size, returning no more than the specified number of items.

A possible generalization of our model is to add support for more customer-centered cumulative criteria, like the ones used in the following statement of interest:

{*(product category, is, car, normal),*
(performance, is very high, normal),
(comfort, is not too low, normal),
(lifetime, is high, important)}

These cumulative and qualitative criteria would be based on predicates on numeric domains defined by the catalog maintainer. The corresponding attribute values of items in the catalog could either be provided explicitly for each item or be derived via heuristic functions from other attribute values of the same item. Formal techniques for relating customer-centered qualitative criteria and product-centered quantitative criteria has been studied extensively since the mid-eighties in engineering sciences (quality function deployment, house of quality diagram) [4].

A distinguished property of our model is the fact that the product category is captured as an "ordinary" attribute on a discrete domain as described in Sec. 3.3. The intended positive effect of this uniformity is the fact that a customer can combine fuzzy predicates on the product category attribute with other predicates using weights, negation, conjunctive and disjunctive aggregation without being constrained by static schema information. Moreover, this uniformity makes it easy to implement catalog *brokers* that integrate the contents from several autonomous catalogs.

3.5 Modeling Mutual Understanding between Customer and Provider

The evaluation of a customer interest, as sketched in the previous section and detailed in the next chapter, leads to a set of *matches* which serve as a foundation for the establishment of a mutual understanding between the information provider and the customer (see classes in the center of Fig 1). Each match integrates an item and a score resulting from the scoring method of the interest applied to this item.

The evaluation may span one or more catalogs and produces a number of matches corresponding to the number of evaluated items or limited by some cutoff number or cutoff score defined by the user. The sequence of matches ordered by decreasing score is called an *offer*. A PIA offer may not only be displayed in the user interface, but it may also be saved by the customer for subsequent examination, or it may be re-created automatically after fixed time intervals as the result of a customer's subscription. It is also possible to compute cumulative figures (frequency, min, max, median, ..) over an offer or to perform statistical cluster analysis on the offer.

Examining an offer, the customer selects one or several items following her/his personal (subjective) rating. A *decision* with regard to a match can be observed only if the customer takes some action. As we abstract from any follow-up actions outside the field of decision support like e.g. putting a product in a shopping cart, we define that a decision is taken, if the customer expicitly marks the respective match as valuable.

Decisions are administered by the information assistant, because the may provide useful hints on the personal understanding that a customer may have of an item and on her/his habits formulating interests. The decisions are, therefore, recursively connected with their preceding interest where they can be used for supporting *relevance feedback* functionality [5] .

4 Fuzzy Matching between Customer Interests and Catalog Items

In Section 3.4 and in the left-hand part of the UML diagram of Fig. 1 we defined the (abstract) syntax for the customer interests (=fuzzy queries) by typed trees. In this section, we inductively define the semantics of customer interests by a query evaluation function which computes a numeric *score* for a given query tree (i.e., an object of class *Interest)* and for a given catalog item based on the item's properties (its attribute / value bindings).

In our object-oriented model, the score function is decomposed into (a hierarchy of) *score* methods defined in the three classes of the query tree nodes. The score for a given *Item* i and a given *Interest* x is thus defined as the value x.score(i) which is computed bottom-up from the scores of all predicates in the tree applied to i taking fuzzy negations, fuzzy and weighted conjunctions and finally fuzzy and weighted disjunctions into account.

The evaluation of fuzzy predicates involving negation, disjunction and conjunction is defined in Section 4.3. The underlying definition and evaluation of simple predicates, i.e. predicates involving only a single attribute of an item is explained in Section 4.2. Together, these sections define the generic semantics of an implementation of the PIA query engine which is parameterized by the particular choice of the basic domains and their predicates (technically defined as class files dynamically linked to the PIA kernel).

4.1 Capturing Similarity through Numeric Scores

A severe limitation of boolean predicates underlying classical (relational and object-oriented) database models is the difficulty to adequately model fuzzy customer interests. For example, a user interested in an item with a price of less than 100$ would certainly be willing to pay 101$ for an item that matches perfectly all other criteria imposed on the item. It is therefore common practice in information retrieval and multimedia databases to use numeric scores in the interval [0,1] to model user interests [6, 5, 7].

However, different research communities have associated different (partially incompatible) interpretations with the values returned from such score functions, such astThe *fuzzy set interpretation* [2, 8], the *spatial interpretation* originally used in text databases, the *metric interpetation* [9], or the *probabilistic interpretation* underlying advanced information retrieval systems [10].

Our goal in the design of the PIA model and system was to allow a maximum freedom in the formulation and combination of predicates while still preserving a minimum semantic consensus necessary to build a meaningful user interface, an efficient query evaluator, user profile manager, persistence manager etc. Therefore we constrained the system to a boolean combination of atomic fuzzy queries. More specifically, we allow n-ary (weighted) disjunctions of n-ary (weighted) conjunctions which leads to a model that is very similar to the work of [7].

4.2 Definition and Evaluation of Simple Fuzzy Predicates on Numeric, Discrete and Full-Text Domains

This section describes the semantics of simple predicates which are used by customers as building blocks in expressions to articulate their personal interests, as explained in Section 3.4.

If a customer defines a criterion on an attribute a with domain d, he/she can only make use of predicates defined on d by the catalog maintainer. For example, it is not allowed to evaluate the numeric predicate *is greater than* on an attribute a that takes values of a discrete domain like $TV\ Norm$ which does not define an order on its elements. This constraint is enforced already by the PIA front end through simple typing rules. As a consequence of this domain restriction, one can distinguish three kinds of predicates, called numeric, discrete and full-text predicates. Their superclass *Predicate* captures the *common properties* of all predicates (see Fig. 1).

For example, if *Length* is a *numeric domain,* a boolean equality predicate can be defined for that domain as follows using a Java-like syntax for the function definition:

```
(is exactly, 2, (Length, Length),
  float score(Item i, Attribute a, Value[ ] args) {
    Value val = i.valueOfAttribute(a);
    if (val ==  null)  return 0.0; // attribute a not defined for item i
    if (val.numericValue == args[0].numericValue)  return 1.0;
    return 0.0; } )
```

The predicate *is exactly* is defined on the domain *Length* and requires a second argument of domain *Length.* The function returns the value 0 if the item i does not have an attribute a or if the (numeric) attribute value does not exactly match the numeric attribute value specified as an argument. Otherwise, the value 1 is returned to indicate an exact match. This example also illustrates that standard boolean predicates ($=, >, \geq, \ldots$) can be captured in our model by restricting the set of admissible score values returned to elements from the binary set {0,1}.

As described in Section 3.4, a customer could then use this (parameterized) predicate to express his/her interest in items that have an exact width of 10 cm and an exact height of 15 cm (assuming that the attribute *width* and are both based on the domain *Length*):

{*(width, is exactly, (10 cm), normal),*
(, is exactly, (15 cm), normal)}

This way, a predicate can be applied to different attributes and it can also verify whether an item posesses the indicated attribute at all.

In the following, we illustrate the expressive power and uniformity of the PIA model by giving examples of predicates on numeric, discrete and full-text

Predicate Name	Vector of Argument Domains
is as large as possible; is (very) large; is average; is as small as possible; is (very) small	*(NumericDomain)*
is exactly; is approximately; is (strictly) larger than; is (strictly) smaller than	*(NumericDomain, NumericDomain)*
is exactly between; is between	*(NumericDomain, NumericDomain, NumericDomain)*

domains. We start with a list of generic *convex fuzzy predicates* on numeric domains:

On a *discrete domain* consisting of a single-rooted hierarchy of values the following predicates can be defined generically:

Predicate Name	Vector of Argument Domains
is an exact value; is vague	*(DiscreteDomain)*
is exactly; is subsumed by subsumes; is similar to	*(DiscreteDomain, DiscreteDomain)*
is one of; subsumes one of; is subsumed by one of	*(DiscreteDomain, DiscreteDomain[])*[1]

In order to define the exact semantics of these predicates we prefer to use the model of *fuzzy relations* and *fuzzy partitions* (as analogues of mathematical relations and partitions derived from mathematical sets) [11].

Modern information retrieval engines provide efficient index support to return scores for the following predicates on *full-text domains*:

Predicate Name	Vector of Argument Domains
is empty	*(Full-Text Domain)*
contains the word; contains a word starting with; contains the stemmed word	*(Full-Text Domain, Full-Text Domain)*
contains x near y; contains x followed by y not more than k words apart	*(Full-Text Domain, Full-Text Domain, NumericDomain)*
contains a passage of k words which contain the words	*(Full-Text Domain, NumericDomain, Full-Text Domain[])*

4.3 Definition and Evaluation of Complex Fuzzy Predicates

As explained already in the introduction of this chapter, the score function defining the semantics of a query is decomposed into (a hierarchy of) *score* methods defined in the three classes of the query tree nodes The score for a given *Item* i and a given *Interest* x is thus defined as the value x.score(i) which is computed bottom-up from the scores of all predicates in the tree applied to i taking fuzzy negations, fuzzy and weighted conjunctions and finally fuzzy and weighted disjunctions into account.

A criterion describes the (possibly negated) application of a predicate onto an attribute. A unary predicate (e.g.*isEmpty()*) does not require any further arguments while most predicates (e.g. isApproximately(a)) can be parameterized by the customer with additional values.

```
class Criterion {
  Predicate predicate; Attribute attribute; Value[ ] arguments;
  float weight;  bool negated;
  float score (Item i) {
    if (negated) { return 1.0 - predicate.score(i, attribute, arguments);}
    else { return predicate.score(i, attribute, arguments);}}
 }
```

The *score* method for a criterion is computed by evaluating the score method of the predicate with the given arguments on the given attribute. For a given score s a negated criterion returns the score 1-s.

A single alternative describes a weighted conjunction of criteria. In the literature one can find a large number of different aggregation functions that compute a score for a weighted "conjunction" of scores [12, 7]. Some of these aggregation operators are sensitive to the order of their subterms [13] so that we chose to represent the subterms by a vector and not by a set.

```
class Alternative {
  Criterion[ ] criteria; AndLikeAggregationOperator op;  float weight;
  float score (Item i) {
    float[ ] cs =   new float[criteria.size()] ;
    float[ ] cw =   new float[criteria.size()] ;
    for ( int j= 0, j ¡ criteria.size(), j++) {
       c=criteria[j]; cs[j]=c.score(i); cw[j]=c.weight;
    }
    return op(i, cs, cw);}
}
```

The *score* method for an alternative is computed by evaluating the score method of the criteria and then passing the results together with the weights of the subterms to the AndLikeAggregationOperator op. [7] describes desirable algebraic properties of weighted aggregation operators.

Finally, a customer interest is described as a weighted disjunction of alternatives.The score method for an interest is computed analogously by evaluating the score method of the alternatives and then passing the results together with the weights of the subterms to the OrLikeAggregationOperator op.

```
 class Interest {
   Alternative[ ] alternatives; OrLikeAggregationOperator op;
   float cutoffScore;  int cutoffSize;
   float score (Item i) {
     float[ ] as =  new float[alternatives.size()];
     float[ ] aw =  new float[alternatives.size()];
     for ( int j= 0, j ¡ alternatives.size(), j++) {
        a=alternatives[j]; as[j]=a.score(i); aw[j]=a.weight;
     }
     return op(i, as, aw);}
}
```

5 The PIA Software Architecture

The PIA object model serves as a framework for different query evaluation strategies, implementations of numeric scoring functions, and persistent storage technologies for semistructured data (see Fig. 2). In this paper, we intentionally do not target the implementation of these components, which are critical for both query efficiency and effectiveness, but instead rely on already existing solutions referred to in chapter 4. The object model itself defines a clear-cut interface to these components which enables the smooth integration of supporting technology.

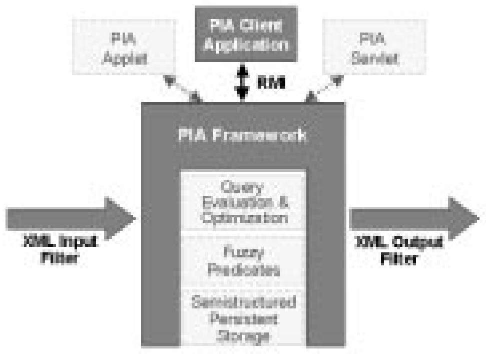
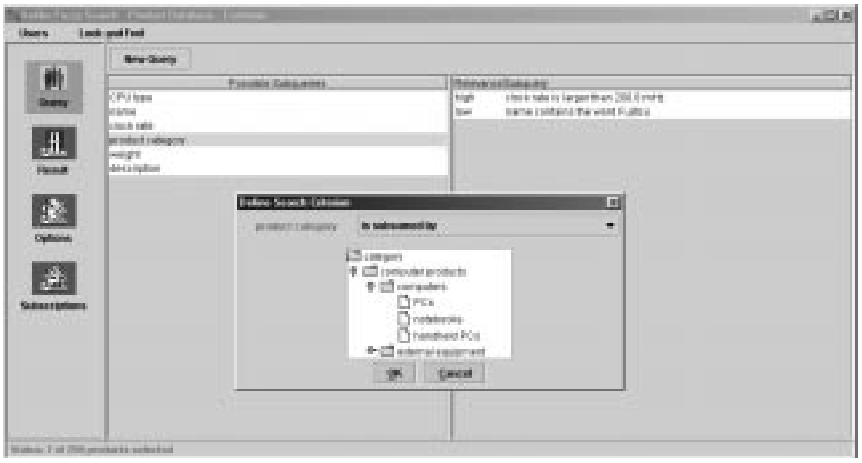


Fig. 2. Overview over the PIA Software Architecture

In the current state of our project, the PIA system is run as a Java Swing application communicating via RMI with the server containing the object-oriented catalog model. Other implementations based on Java applet or servlet technology are also imaginable and could be realized with little effort. Catalog data is imported via an XML import filter and can also be exported into XML files. Although, from our point of view, XML is suitable for the representation of objects in a personal information assistant, we still plan to realize filters for other well-accepted exchange formats, too.

5.1 Interaction with the System

The prototype's user interface is implemented by the use of the Java Foundation Classes (*JFC*, also known as *Swing*). The large number of graphical components available in Swing eases the task of giving the interface a clear, understandable structure which is essential for non-expert users of a personal information assistant.



high clock rate is larger than 200.0 mhz
low name contains the word Fujitsu

Status: 7 of 200 products selected

Fig. 3. The Inteactive Query Interface for PIA Customers

Taking the viewpoint of a PIA customer first, one can detect four large icons on the left side of the window, allowing the user to decide between four different activities. He/she can formulate an interest, evaluate this interest, change his/her personal preferences, or subscribe for periodical notifications on special offers according to his/her interests. These icons remain visible throughout the customer's session. Providers and maintainers are faced with similar activity panels representing their specific activities with the PIA system.

During a session, the information assistant stays in close interaction with the customer (and all other users). It provides e.g. tool tip information for the different graphical components that can be used. Furthermore, a status line at the bottom of the window displays messages important for the current state of usage (e.g. number of items matching the current query).

Formulating the query (see Fig. 3), the customer is being equipped with information on the catalog. In the table next to the activity panel, all the attributes currently available in the catalog are displayed together with the frequency by which they appear. In this way, the customer is able to estimate how effective the choice of a certain property in the formulation of his/her interest can be with respect to the current catalog contents. The current interest can be seen on the right of the window, where the desired criteria are shown together with their weight. The interest can be extended by clicking on one of the attributes in the attribute table on the left. As a result, an editor window opens. The customer now can choose one of the predicates available for the attribute's domain and afterwards enter the necessary argument values. As soon as he/she has finished, the new property will appear in the panel on the right. The status line is updated, now stating the number of items which would be retrieved from the catalog for the current interest, so that the customer knows how effective the query would be. The customer may now decide to refine the interest or to evaluate it. Evaluation is triggered by pressing the respective button in the activity panel.

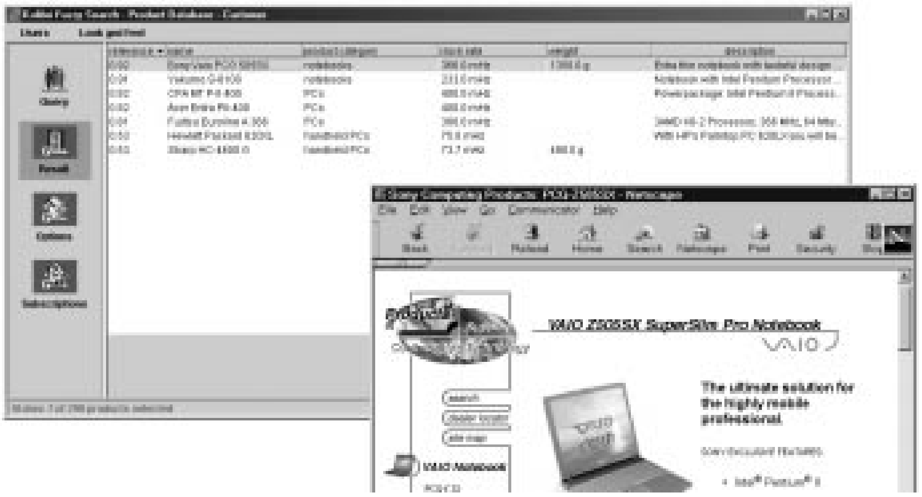

Query
Result
Options
Subscriptions
File Edit View Go Communicator Help
Back Reload Home Search Netscape Print Security
VAIO Z505SX SuperSlim Pro Notebook
The ultimate solution for the highly mobile professional.

Fig. 4. The Interactive Evaluation Interface for PIA Customers

The evaluation returns a list of items ranked by their relevance for the interest (see Fig. 4). Each item is connected with a link leading to the URL containing the detailed item description. In this way, the customer is offered a quick access to generic decision support as expected from a personal information assistant. Using URLs here guarantees that the descriptions can also be stored and updated by the information provider, who, in many cases, owns the deeper understanding of the objects' semantics. Having inspected the list of results, the customer may decide to reformulate his/her interest, which he can simply do by pressing the query button on the left again. The program then returns to the query window, still showing the interest formulated before. Of course, both the formulation and the evaluation of interests are also accessible for catalog maintainers and information providers.

The window for browsing and updating personal preferences can be reached by pressing the third button in the activity panel. Here, the customer is offered a tabbed pane consisting of different cards for different kinds of preferences. Among others, it is possible to choose preferred measuring units for domains or to view, refine or delete stored interests.

Modalities for periodical notification on new offers can be settled by pressing the last button on the activity panel. The customer has the choice between two different kinds of subscription. On the one hand, he/she can receive a general list of special offers generated from time to time for all PIA customers. On the other hand, he/she can subscribe to offers according to his own special interests stored witin the system.

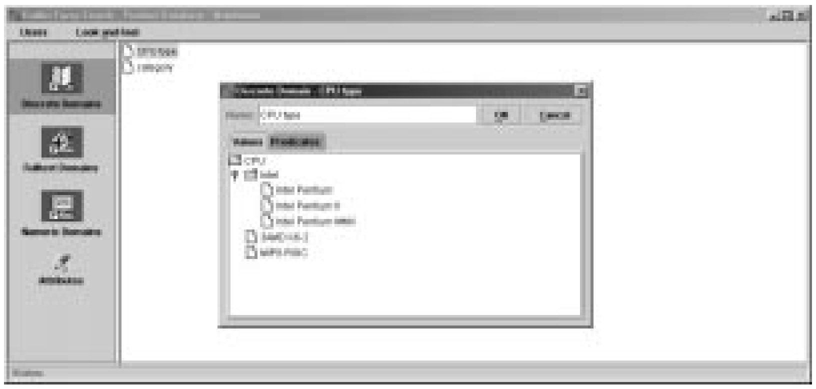


Fig. 5. The Interactive Interface for PIA Maintainers

The major task of PIA catalog maintainers is to keep the domains and attributes within the system in a state adequate to the needs of customers and providers. Thus, the maintainer's activity panel includes three buttons for browsing and updating discrete, full-text and numeric domains, as well as one button for the maintenance of attributes (see Fig. 5). The structure of the windows hiding behind these buttons is always very similar. The maintainer can choose one domain or attribute respectively from the table in the upper part of the window. Clicking on the desired line opens an editor window. This editor window allows to change the characteristics of domains and of attributes. For example, the maintainer can change the name of a domain, rearrange its value space, or add or delete predicates. He/she hereby is restricted to updates allowed by the system. In this way, a kind of type checking according to the PIA object model is realized.

Although PIA information providers would usually not type in product information item by item, they are still offered an interface of their own, which can be especially practical in cases, when some incorrect information has come into the catalog before, via some import filter. The provider then has the opportunity to bring the catalog back to a consistent state using this interface.

6 Concluding Remarks

This paper does not propose yet another fuzzy query model or fuzzy predicate semantics but it defines the syntax and semantics of a generic framework for the implementation of interactive catalogs which do not require information providers to adhere to a strict data model and which allow customers to combine in an intuitive and flexible way fuzzy predicates over numeric, discrete and full-text domains.

With our initial implementation of the query engine we target small to medium-sized corporate catalogs (up to 10^4 items with up to 10^2 attributes) where a semi-naive evaluation strategy suffices to yield acceptable interactive system performance in single-user mode. In this scenario, usability and flexibility of the interactive query interface are more important than raw query execution speed. We closely follow recent developments in database research on optimized access to semi-structured data [14, 15], on similarity indexing [9] and on fuzzy query evaluation techniques [7] which may provide significant potential for query evaluation improvements.

Our model and system architecture has been influenced heavily by our experience gained in building interactive web catalogs (e.g. electronic marketplaces for classified ads [16]) and we are currently transfering results of this research back into industrial projects with partners from the German media industry.

Acknowledgements

This work has been carried out as part of the *Kolibri* project funded by the DFG programme V3D2 under grant DFG Schm 450/7-1. Additional funding has been provided by *Pastel* (Esprit Project 22552).

References

1. M. Fowler and K. Scott. *UML Distilled - Applying the Standard Object Modeling Language.* Addison-Wesley Publishing Company, 1997.
2. Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.
3. Siegfried Gottwald. *Fuzzy Sets and Fuzzy Logic: Foundations of Application – from a Mathematical Point of View.* Verlag Vieweg, Wiesbaden, Germany, 1993.
4. Y. Akao. *Quality Function Deployment: Integrating Customer Requirements into Product Design.* Productivity Press, Cambridge, 1990.
5. G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Tokio, 1983.

6. C. J. van Rijsbergen. *Information Retrieval.* Butterworths, 1979.
7. R. Fagin. Fuzzy queries in multimedia database systems. In J. Paredaens, editor, *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Seattle, Washington*, pages 1–10. ACM Press, 1998.
8. H. J. Zimmermann. *Fuzzy Set Theory and its Applications.* Kluwer Academic Press, Boston, MA, 2nd edition, 1993.
9. Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Databases, Athens, Greece*, pages 426–435, 1997.
10. H. R. Turtle and W. B. Croft. A comparison of text retrieval models. *Computer Journal*, 35(3):279–290, 1992.
11. H. L. Larsen and R. R. Yager. The use of fuzzy relational thesauri for classificatory problem solving in information retrieval and expert systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(1):31–41, 1993.
12. Sadaaki Miyamoto. *Fuzzy Stes in Information Retrieval and Cluster Analysis.* System Theory, Knowledge Engineering and Problem Solving. Kluwer Academic Press, 1990.
13. R. Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):183–190, 1988.
14. S. Abiteboul. Querying semi-structured data. In F. Afrati and P. Kolaitis, editors, *Proceedings ICDT '97, 6th International Conference, Delphi, Greece*, Lecture Notes in Computer Science, pages 1–18. Springer-Verlag, 1997.
15. Sophie Cluet. Modeling and querying semi-structured data. In *SCIE 1997*, pages 192–213, 1997.
16. Higher Order GmbH. Reference list of Tycoon Adbase installations, 1999. http://www.higher-order.de/Produkte/AdBase/Referenzen.html.