Dipartimento di Informatica
Università del Piemonte Orientale “A. Avogadro”
Via Teresa Michel 11, 15100 Alessandria
http://www.di.unipmn.it

On-line Product Configuration using Fuzzy Retrieval and J2EE Technology

Authors: Luigi Portinale, Maurizio Galandrino
(luigi.portinale@di.unipmn.it)

TECHNICAL REPORT TR-INF-2009-05-04-UNIPMN
(May 2009)

The University of Piemonte Orientale Department of Computer Science Research Technical Reports are available via WWW at URL `http://www.di.unipmn.it/`. Plain-text abstracts organized by year are available in the directory

Recent Titles from the TR-INF-UNIPMN Technical Report Series

2009-03 *GSPN Semantics for Continuous Time Bayesian Networks with Immediate Nodes*, Portinale, L., Codetta-Raiteri, D., March 2009.

2009-02 *The TAAROA Project Specification*, Anglano, C., Canonico, M., Guazzone, M., Zola, M., February 2009.

2009-01 *Knowledge-Free Scheduling Algorithms for Multiple Bag-of-Task Applications on Desktop Grids*, Anglano, C., Canonico, M., February 2009.

2008-09 *Case-based management of exceptions to business processes: an approach exploiting prototypes*, Montani, S., December 2008.

2008-08 *The ShareGrid Portal: an easy way to submit jobs on computational Grids*, Anglano, C., Canonico, M., Guazzone, M., October 2008.

2008-07 *BuzzChecker: Exploiting the Web to Better Understand Society*, Furini, M., Montangero, S., July 2008.

2008-06 *Low-Memory Adaptive Prefix Coding*, Gagie, T., Nekrich, Y., July 2008.

2008-05 *Non deterministic Repairable Fault Trees for computing optimal repair strategy*, Beccuti, M., Codetta-Raiteri, D., Franceschinis, G., July 2008.

2008-04 *Reliability and QoS Analysis of the Italian GARR network*, Bobbio, A., Terruggia, R., June 2008.

2008-03 *Mean Field Methods in performance analysis*, Gribaudo, M., Telek, M., Bobbio, A., March 2008.

2008-02 *Move-to-Front, Distance Coding, and Inversion Frequencies Revisited*, Gagie, T., Manzini, G., March 2008.

2008-01 *Space-Conscious Data Indexing and Compression in a Streaming Model*, Ferragina, P., Gagie, T., Manzini, G., February 2008.

2007-05 *Scheduling Algorithms for Multiple Bag-of-Task Applications on Desktop Grids: a Knowledge-Free Approach*, Canonico, M., Anglano, C., December 2007.

2007-04 *Verifying the Conformance of Agents with Multiparty Protocols*, Giordano, L., Martelli, A., November 2007.

2007-03 *A fuzzy approach to similarity in Case-Based Reasoning suitable to SQL implementation*, Portinale, L., Montani, S., October 2007.

2007-02 *Space-conscious compression*, Gagie, T., Manzini, G., June 2007.

2007-01 *Markov Decision Petri Net and Markov Decision Well-formed Net Formalisms*, Beccuti, M., Franceschinis, G., Haddad, S., February 2007.

2006-03 *New challenges in network reliability analysis*, Bobbio, A., Ferraris, C., Terruggia, R., November 2006.

2006-03 *The Engineering of a Compression Boosting Library: Theory vs Practice in BWT compression*, Ferragina, P., Giancarlo, R., Manzini, G., June 2006.

2006-02 *A Case-Based Architecture for Temporal Abstraction Configuration and Processing*, Portinale, L., Montani, S., Bottrighi, A., Leonardi, G., Juarez, J., May 2006.

Contents

Abstract

Flexible retrieval of configurable products is one of the most challenging task for e-commerce applications, since user requirements are usually imprecise or approximate, in order to be interpreted in the right way by the application, and in such a way that a set of relevant products is finally retrieved. In many cases products are stored as a database of components and an automatic configuration process has to be provided. In the present paper, we propose an approach to intelligent retrieval and configuration of component-based products, starting from a set of possibly fuzzy user requirements provided at different levels of detail. A conceptual product model is introduced and its use during the configuration process is discussed. The proposed approach exploits a fuzzy generalization of SQL and a bottom-up (from basic to complex components) configuration process. Finally, a 3-tier system architecture based on J2EE and standard RDBMS technology is presented and its use for on-line fuzzy configuration is illustrated through a simple example based on a PC assembly task.

1 Introduction

The wide use of e-commerce applications has produced a remarkable interest in approaches able to provide flexible retrieval of specified items, in such a way that the final user is able to deal with a limited but very significant amount of information. When searching an on-line electronic catalog for a target product, the user is asked to specify a set of requirements the product has to fulfill [1]; however, very often either the user overspecifies the product (and then no item is returned) or he/she underspecifies the product (and then a very large amount of items is returned). Moreover, several products having a reasonable market on the WWW are configurable products (e.g. personal computers, travels, music compilations, personal gifts, etc...). This means that the final target product is composed by a set of components meeting a given set of (possibly imprecise) requirements. However, a limitation of several WWW applications to e-commerce is the lack of automatic configuration facilities. Very often the deployed application is not able to provide a reasonable help in configuring the product, leaving the user alone in trying to merge sub-components, in such a way of fulfilling every needed requirements.

In addition, even if web architectures offer flexible interfaces towards relational databases (where products are usually stored) standard SQL retrieval on a database suffers of the so-called “boolean limitation”: either a tuple satisfies a given retrieval condition or not. This is essentially the major source producing the undesired behavior described above. In general, even if a reasonable number of products is returned, only those products that completely satisfy the search parameters are returned, missing those items that only partially match the query,

but that may be potentially interesting to the user. This problem is particularly worth in B2C (Business to Customer) applications, where the final user is a customer who is usually not an expert in the product field and is not usually able to precisely specify every feature of the desired product or to correctly combine elementary components when a configurable product is the target. On the other hand, RDBMS provides efficient data management and query facilities, and real-world applications cannot leave aside the fact that the data of interest are usually stored in a relational database.

Configuration problems are traditionally dealt with in AI either within the constraint satisfaction paradigm [2, 3, 4, 5] or by structured logic approaches [6, 7, 8, 9]. Common to every knowledge-based approach to configuration are the following items ([10, 11])

- a set of concepts or domain objects, possibly organized in classes;
- a set of relations between domain objects, in particular taxonomical and compositional relations;
- a configuration objective or task, specifying the demand and the constraints a created configuration must fulfill;
- control knowledge for the configuration process.

As recognized in [11], both declarative models and inference machineries must be defined, in order to adequately accomplish the final task. In other words, operational processable models are needed. In order to achieve such a goal, different methodologies can be adopted like compilation of models [12], the use of the case-based reasoning (CBR) paradigm [13, 14, 15], the automatic decomposition of the problem during state space search [16].

However, all the above mentioned approaches do not directly address the issue of flexibly exploiting, during the configuration process, the presence of the data of interest in standard relational databases, and the corresponding data management and query facilities. Moreover, most of the proposed frameworks (an exception is provided by [14]) do not take into consideration the approximate nature of the user requirements, that very often cannot be assumed to be totally precise. In the present paper, we propose an approach for the definition of an architecture for on-line searching and configuration of products based on the following characteristics:

- a conceptual model of the configurable products, representing the structural decomposition of the modeled product;
- a set of approximate, i.e. fuzzy, user requirements on the desired target product;

- a fuzzy knowledge base providing the semantics for the possible user requirements;
- a fuzzy extension to the standard SQL query language, through which to implement the retrieval and the composition strategy of the product components, directly on top of a RDBMS.

In particular, the last point concerns the extension of the standard retrieval features of SQL in such a way that vague and imprecise specifications could be used to return items. We introduce an extended version of SQL, able to deal with fuzzy predicates and conditions, defined over standard attributes of a table. This approach is based on the **SQLf** language proposed in [17], extending standard SQL in order to deal with *fuzzy queries*. We will show how to exploit the conceptual model and the user requirements and constraints, in order to automatically derive a set of fuzzy SQL queries able to retrieve the suitable components. A bottom-up strategy on the conceptual model is then introduced with the aim of guiding the application to suitably combine the results of such queries, within a standard RDBMS framework. The advantage of this approach is that the whole power of an SQL engine can be fully exploited, with no need of implementing specific retrieval algorithms. Moreover, the use of SQL and of standard DBMS allows us to obtain an efficient retrieval in very large product catalogs.

We exploit the J2EE framework (JSP, Java Servlets and JDBC) for the implementation of the proposed approach.

The paper is organized as follows: section 2 introduces a short review of fuzzy queries on a database, section 3 discusses the conceptual model used for the configuration process while in section 4 the proposed architecture and the configuration algorithm are presented and discussed through a specific example. Finally, conclusions are drawn in section 5.

2 Fuzzy Queries in Databases

It is well-known that standard relational databases can only deal with precise information and standard query languages, like SQL, only support boolean queries. Fuzzy logic provides a natural way of generalizing the strict satisfiability of boolean logic to a notion of satisfiability with different degrees [18] [1]; this is the reason why considerable efforts has been dedicated inside the database community toward the possibility of dealing with fuzzy information in a database. We are here only interested in *fuzzy queries* on an *ordinary (non-fuzzy) database*. In particular, in [17] standard SQL is adopted as the starting point for a set of extensions able

[1] We assume here the reader familiar with the basics of fuzzy logic (see [18] for a survey).

to improve query capabilities from boolean to fuzzy ones. The implementation of the SQL extensions can be actually provided on top of a standard relational DBMS, by means of a suitable module able to transform a fuzzy query into a regular one, through the so called *derivation principle* [17]. In the fuzzy SQL language we consider in this paper, the WHERE condition can be a composite fuzzy formula involving both crisp and fuzzy predicates (i.e. linguistic values defined over the domains of the attributes of interest), as well as crisp and fuzzy operators (e.g. operator *about* over the attribute `price` to implement conditions like `product.price about $100`);

By allowing fuzzy predicates and operators to form the condition of the `WHERE` clause, the result of the `SELECT` is actually a fuzzy relation, i.e. a set of tuples with associated the degree to which they satisfy the `WHERE` clause. Such a degree can be characterized as follows: let

`SELECT` A `FROM` R `WHERE` fc

be a query with fuzzy condition fc; the result will be a fuzzy relation Rf with membership function $\mu_{Rf}(a) = \sup_{(x \in R) \wedge (x.A=a)} \mu_{fc}(x)$. The fuzzy distribution $\mu_{fc}(x)$ relative to fc must then be computed by taking into account the logical connectives involved and their fuzzy interpretation (usually $\min$ for conjunction and $\max$ for disjunction). In order to process a query using a standard DBMS, we have to devise a way of translating the fuzzy SQL statement into a standard one. The most simple way is to require the fuzzy query to return a boolean relation R_b which tuples are extracted from the fuzzy relation R_f, by considering a suitable threshold (or confidence level) on the fuzzy distribution of R_f. We consider, as in [17], the following syntax

`SELECT (`λ`)` A `FROM` R `WHERE` fc

which meaning is that a set of tuples with attribute set A, from relation set R, satisfying the fuzzy condition fc with degree $\mu \geq \lambda$ is returned. In fuzzy terms, the λ-cut of the fuzzy relation R_f resulting from the query is returned. The interesting point is that, when the $\min$ operator is adopted as *t-norm* for conjunction and the $\max$ operator is adopted as *t-conorm* for disjunction, then it is possible to derive from a fuzzy SQL query, an SQL query returning exactly the λ-cut required (see [17] for details).

Example. Consider a generic relation `PRODUCT` containing the attribute `price` over which the linguistic term `medium` is defined. Figure 1 shows a possible fuzzy distribution for `medium` as well as the distribution of a fuzzy operator $\ll$ (much less than), defined over the difference ($a - b$) of the operands, by considering the expression $a \ll b$. Let C be a generic condition and $D(C)$ the fuzzy degree of C; $D(\texttt{price} = \texttt{medium} \wedge \texttt{price} \ll 100) \geq 0.8$ will hold iff $\min(D(\texttt{price} = \texttt{medium}), D(\texttt{price} \ll 100)) \geq 0.8$, iff $D(\texttt{price} = \texttt{medium}) \geq 0.8 \wedge D(\texttt{price} \ll 100) \geq 0.8$ iff $(110 \leq \texttt{price} \leq 180) \wedge (\texttt{price} - 100) \leq -18$.

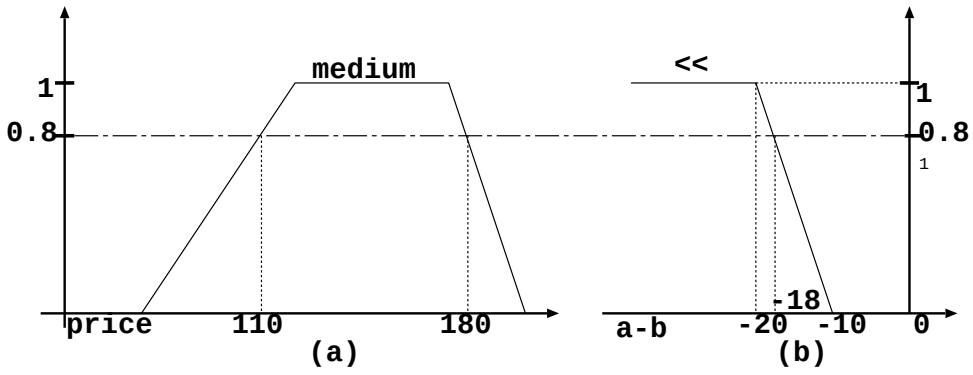

Figure 1: Fuzzy Distributions

The latter condition can be easily translated in a standard WHERE clause of SQL.

3 Conceptual Modeling

As already mentioned, any knowledge-based approach to product configuration needs to define a suitable conceptual model [19], where both taxonomic (*is_a*) and partonomic (*composed_by*) relations are of fundamental importance. The model we adopt in our approach is based on the FPC model by Magro and Torasso [7], which is in turn based on the KL-ONE framework [20].

We propose to adopt a conceptual framework based on a hierarchy of components with the following basic ontology: *entities* divided into *categories* representing generic components (i.e. generic classes of components) of the final product, and *basic categories* representing classes of components having specific instances associated with them; *composed_by links* connecting a category A to an entity B, meaning that B is a component of A; *is_a links* connecting a category A to an entity B, meaning that B is a subclass of A (i.e. that the components in B are a particular typology of the components in A). If B is a component of A or if B is a sub-class of A we say that B is a child of A (i.e. *composed_by* links are from parent to child, while *is_a* link are from child to parent). We also consider cardinality information with respect to a *composed_by* link: in particular every such a kind of link is provided with a so-called *number restriction* ([20]) representing the minimal and maximum number of sub-components that may occur in the link.

An example of a conceptual model for a (simplified) personal computer system is reported in figure 2[2] (similar examples in the FPC framework are reported in [7, 16, 9]). For instance, we read from the model that the PC system is composed by exactly one computational system and exactly one storage system; the

[2]Dashed squares around some entities will be explained in section 4.

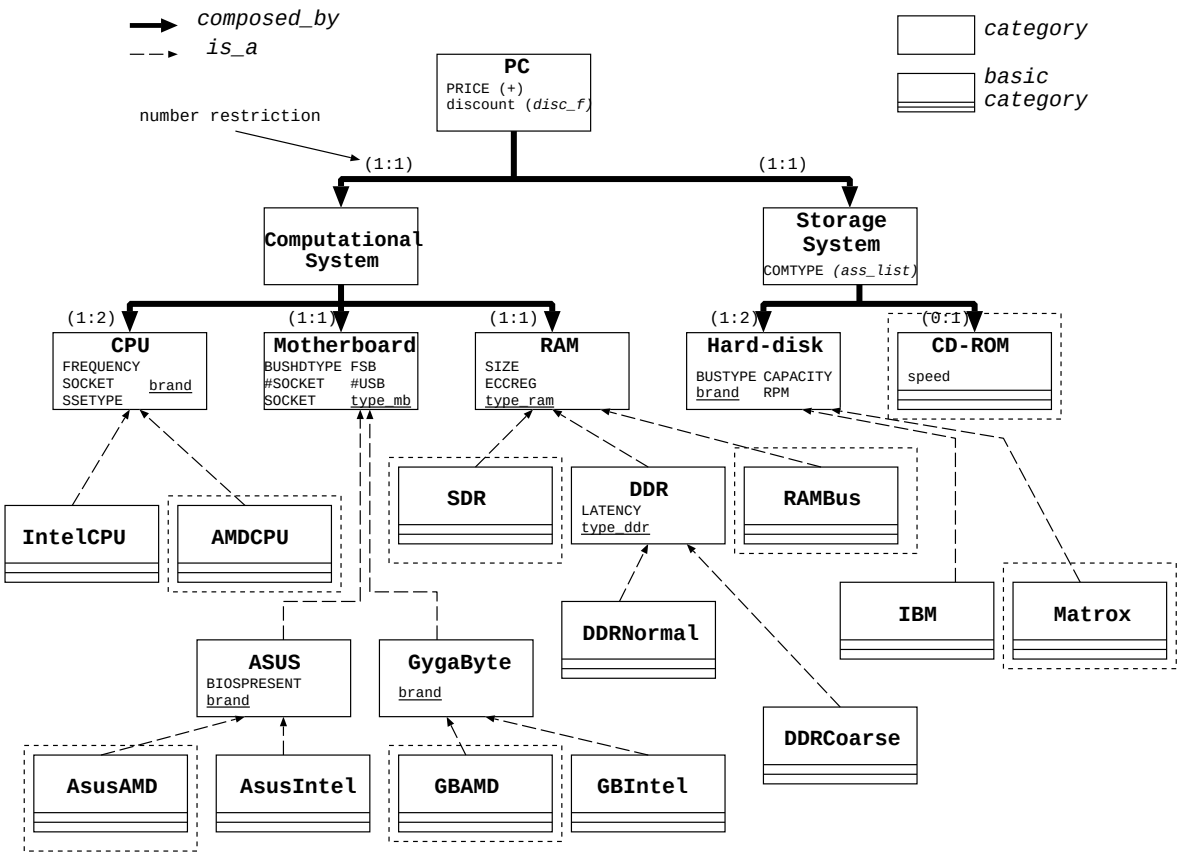

Figure 2: Example of a conceptual model for a PC

computational system can be composed by one or two CPUs, one motherboard (MB) and one set of RAM slots; the storage system of interest is composed by one or two hard-disks and by one (optional) CD-ROM; Different typologies of CPU, MB or RAM are possible, as well as different typologies of hard-disks. For example MBs are differentiated into ASUS and GigaByte MBs, which are in turn both distincted into Intel or AMD MBs. The ASUS entity is a category (i.e. there are no specific stored instances of such category), while ASUSAMD is a basic category with specific stored instances associated to such component. As we will see, instances of non-basic categories are built, during the configuration process, from instances of basic categories. Similar consideration hold for the other entities in fig. 2.

Entities can have attributes. Attributes may be be specific to a given entity (i.e. non-inheritable) or inherited from source to target in a *composed_by* link chain and from target to source in a *is_a* link chain. Inheritable attributed are then associated only to categories; in case they are defined on a category having

sub-components, an *aggregation function* must be defined; the meaning is that the value of the attribute is determined by combining the values of the inherited attributes of its sub-categories (combination provided by the aggregation function specified). For example the attribute `PRICE` is defined on the category `PC` and inherited by every sub-component; however the price of the final product (i.e. the assembled PC) is composed by summing the prices of each component, so the aggregation function `sum` ($+$) is defined over `PRICE` on category `PC`. The aggregation function is inherited along a *composed_by* link chain (the inheritance of the function is stopped when a category has not sub-components). Aggregation function can be standard mathematical function (as `sum` for `PRICE`) or any user-defined function with a specific aggregation task (e.g. the function *assoc_list* on attribute `COMTYPE` of the storage system, building an associative list with the possible communication standards of the storage devices present in the final product). In figure 2 inheritable attributes are indicated in capital letters (possibly with the associated function).
Besides aggregation functions, arbitrary functions can be attached to attributes in order to compute their value; this is necessary in case the attribute is a non-inheritable attribute defined on a (non-basic) category. In the example of figure 2, function $disc\_f$ ia associated with the attribute `discount` of `PC` to compute tha available discount for the assembled computer (this may be a simple percentage on the attribute `PRICE` or a more complex procedure taking into account the kind of involved components). In categories being a target of an *is_a* link, a special *discriminant attribute* is introduced, which aim is to distinguish the different typologies the category has (underlined attributes in fig. 2). For example attribute `brand` in the category `CPU` determines the brand of the corresponding component (and so identify the typology below)[3].

In our framework, a fuzzy semantics can be associated to any attribute, by considering its type (see [1]). The definition of suitable fuzzy sets or operators can then be exploited to model a given level of approximation in the user requirements for the target configured product. This means that the final user can specifies both crisp (i.e. precise and well-defined) requirements over an attributes (e.g. "I need an Intel CPU and an hard-disk with a capacity greater than 60Gb "), as well as imprecise (i.e. fuzzy) requirement ("I want a PC with a large memory size and with a price of about $500 ").

The definition of the fuzzy sets associated with the attributes can be made at different levels of details. Indeed, the fuzzy semantics of a linguistic value

[3]It is worth noting that the introduction of a typology is a modeling choice, often depending by efficiency reasons: for example the introduction of basic categories IntelCPU and AMDCPU (with *is_a* links to CPU) can be justified when a large number of user queries make a distinction on the brand of the CPU; an alternative could be to make CPU a basic category, with a standard attribute *brand* of type `enum('Intel', 'AMD')`.

depends in general from the entity we are considering; for instance, a linguistic value "cheap" for attribute `PRICE` of category `PC` has a different semantics (and so a different fuzzy set definition) than the same value defined on the same (inherited) attribute on category `CPU`, since the price of a component may a have a different order of magnitude than the price of the whole product. This means that, if fuzzy user requirements are allowed on a given attribute of an entity, possible different fuzzy semantics for the linguistic values are allowed for each entity that defines or inherit the attribute, if the entity is part of a *composed_by* chain. On the other hand, entities in a *is_a* chain sharing the same attribute (by inheritance), also share the fuzzy semantics for that attribute. For instance, the fuzzy definition of the concept "high capacity" is shared by entities `Hard-disk`, `IBM` and `Matrox`.

To sum up, user requirements represent a set of user constraints on the required configuration that can be finally expressed through a complex possibly fuzzy condition involving both the set of considered entities, as well as the set of attributes.

Moreover, as the notion of configuration assumes, the conceptual model is also augmented with a set of *constraints*. We can identify two different kinds of constraints: (1) *model constraints* (MC) representing general constraints imposed on the model and that must be satisfied by any allowed configurations; (2) *user constraints* (UC) which are specified by the user only for the current configuration process (user's requirements). We assume that a MC is an expression according to the following grammar:

```
<mc> ::= (<mc>) | not <mc> | <mc> and <mc> |
   <mc> or <mc> | <expr>
<expr> ::= <expr1> <rel> <expr2>
<expr1> ::= <entity>.<attribute> | #<entity>
<expr2> ::= <entity>.<attribute> | #<entity> | <value>
<rel> ::= <relational_symbol>
```

where `<entity>` is the name of an entity, `<attribute>` the name of an attribute, `<value>` represents any allowed value on the range of an attribute, `<relational_symbol>` represents any standard relational symbol like equality, diversity, etc..., and `#E` represents the cardinality (number of occurrences) of entity E.

User constraints can be either of the same form of MCs or specific condition of selection over component attributes. In the latter they can be either crisp/boolean conditions or fuzzy conditions (see [1] and section 2). In the next session we will show some examples.

Finally, in case a user requires $n > 1$ occurences of a given entity (as allowed by the corresponding number restriction), then such an entity E is supposed to be replicated n times ($E(1), \ldots E(n)$).

4 Implementation Framework and Architecture

Given the conceptual model described in the previous section, a natural choice is to associate a relation (table) with each basic category; they represent the basic components to be used for the configuration, stored as tuples of the corresponding relation. Data structure corresponding to inner nodes will be created during the configuration process. In the example of figure 2 we will have the following tables (table names represent the path in the hierarchy):

```
PC-CS-CPU-INTELCPU(id1,frequency,socket,ssetype,price)
PC-CS-CPU-AMDCPU(id2,frequency,socket,ssetype,price)
PC-CS-MB-ASUS-ASUSINTEL(id3,biospresent,bushdtype,
  #sockets,socket,fsb,#usb,price)
PC-CS-MB-ASUS-ASUSAMD(id4,biospresent,bushdtype,
  #sockets,socket,fsb,#usb,price)
PC-CS-MB-GB-GBAMD(id5,bushdtype,#sockets,socket,fsb,
  #usb,price)
PC-CS-MB-GB-GBINTEL(id6,bushdtype,#sockets,socket,fsb,
  #usb,price)
PC-CS-RAM-SDR(id7,size,eccreg,price)
PC-CS-RAM-RAMBUS(id8,size,eccreg,price)
PC-CS-RAM-DDR-DDRNORMAL(id9,latency,size,eccreg,price)
PC-CS-RAM-DDR-DDRCOARSE(id10,latency,size,eccreg,price)

PC-SS-HD-IBM(id11,bustype,capacity,rpm,comtype,price)
PC-SS-HD-MATROX(id12,bustype,capacity,rpm,comtype,price)

PC-SS-HD-CDROM(id13,speed,comtype,price)
```

where fields $\texttt{id}_i$ work as primary keys. Such fields are then supposed to be available to any entity in the model to identify their components (see below).
Let us suppose to have the following Model Constraints (MC): `MC1`:"ASUS MBs are incompatible with DDRCOARSE memories"; `MC2`:"the socket type of CPU and MB must be the same"; `MC3`:"the number of CPUs must not be greater than the number of sockets in MB"; `MC4`:"if there are more CPUs they must be the same product"; `MC5`:"the communication standard of storage devices (HDs and CDROMs) must be the same". Their formalization is reported below:

```
MC1:  not(Motherboard.type='Asus' and DDR.type='Coarse')
MC2:  CPU.socket = Motherboard.socket
MC3:  #CPU ≤ Motherboard.#sockets
```

```
MC4:  CPU(1).id_cpu = CPU(2).id_cpu (iff #CPU=2)
MC5:  Hard-disk.comtype = CDROM.comtype
UC1:  CPU.brand = Motherboard.brand
```

These are considered as model constraints that must be satisfied by any configuration; we also add a specific user constraint `UC1`:"CPUs and MB must be of the same brand". Such a constraint can be added to the set of model constraints, but only for the current configuration process.
Once the model and user general constraints are defined, the configuration can start, by collecting specific user requirements as well. We have implemented a bottom-up configuration algorithm based on the following steps:

1. following initial user requirements, user constraints and model constraints, a part of the general conceptual model is identified and instantiated;
2. starting from basic categories, we ask the user to specify (possibly fuzzy) requirements for the current entity and a fuzzy SQL query is automatically generated;
3. when every child of a given entity has been queried, a view is generated on the parent in the following way: in case of *is_a* links an SQL UNION operation is performed, otherwise an SQL JOIN (checking for constraints) is performed;
4. the procedure is iterated (going to step 2), until a view is generated on the top of the model's hierarchy.

Let us show some details of the above procedure with the model of fig. 2. First of all the user is asked to specify which parts of the final product he/she is interested in; this will provide an instantiation of the whole model with specific cardinalities and whith only some of the whole set of entities involved (i.e. those the user is interested in). Let us suppose that the initial user requirements are the following: "I'm interested in a single CPU system, in IBM disks, in Intel components and I don't want a CDROM"; The first requirement states that `#CPU=1`, so no replication is needed and constraint `MC4` is not activated. Moreover, the system caches the value `#CPU=1`, since it will use it later on to check `MC3` when needed. The remaining requirements prune from the model all the entities surrounded by dashed squares in fig. 2. Such user constraints can be easily implemented by allowing the user to select the entities he/she's interested in.

Now a bottom-up process starts from basic categories, asking the user to specify (possibly fuzzy) requirements over categories, by producing suitable (fuzzy) SQL queries. For example the user may require an IBM hard-disk with high capacity with a confidence of 0.7: the following view, representing the possible hard disks the user is looking for is generated:

```
CREATE VIEW PC-SS-HD AS
SELECT (0.7) id11 as id_hd, brand='IBM', price,
comtype FROM PC-SS-HD-IBM WHERE capacity=high
```

where `high` is a fuzzy linguistic values suitably defined over the attribute of interest (capacity). Notice that only the id field (needed to retrieve the components with all their inherited attributes) and the (non-inherited) discriminant attribute are explicitly necessary in such a view. For the sake of convenience, we also store in the view inherited attributes associated to an aggregation function; this avoids the need of a further join condition when *composed_by* links are dealt with (see below). The configuration process switches now to basic categories for RAM and the user is asked to specify requirements over the RAM (remember that an inherited attribute keeps its semantics along an *is_a* chain); suppose the user decides that the price of the RAM has to be not very expensive with a confidence of 0.7, the following query (building category DDR) is generated:

```
CREATE VIEW PC-CS-RAM-DDR AS
(SELECT (0.7) id9 as id_ddr, type_ddr='ddrnormal',
price FROM DDRNormal WHERE price<>very_expensive) UNION
(SELECT (0.7) id10 as id_ddr, type_ddr='ddrcoarse',
price FROM DDRCoarse WHERE price<>very_expensive)
```

Next step generates the RAM view:

```
CREATE VIEW PC-CS-RAM AS
SELECT id_ddr as id_ram, type_ram='ddr', price
FROM PC-CS-RAM-DDR
```

Concerning the CPU, suppose the user wants a CPU with high speed of the front side bus (confidence 0.9); the generated query is:

```
CREATE VIEW PC-CS-CPU AS
SELECT (0.9) id1 as id_cpu, brand='Intel', price FROM
PC-CS-CPU-INTELCPU
WHERE fsb=high
```

For the MB, the user asks (confidence 0.9) a high speed front side bus and a large number of USB ports; these are the generated views:

```
CREATE VIEW PC-CS-MB-ASUS AS
SELECT (0.9) id3 as id_asus, brand='Intel' , price FROM
PC-CS-MB-ASUS-ASUSINTEL
WHERE fsb=high AND #usb=large
CREATE VIEW PC-CS-MB-GB AS
SELECT (0.9) id6 as id_gb, brand='Intel', price FROM
PC-CS-MB-GB-GBINTEL
WHERE fsb=high AND #usb=large
CREATE VIEW PC-CS-MB AS
```

```
   (SELECT id_asus as id_mb, type_mb='Asus', price FROM
PC-CS-MB-ASUS) UNION
   (SELECT id_gb as id_mb, type_mb='gb', price
FROM PC-CS-MB-GB)
```

At this stage, all the *is_a* links have been processed and the configuration algorithm starts to deal with compositional links. Concerning the storage system a simple SELECT is sufficient[4]:

```
   CREATE VIEW PC-SS AS
   SELECT id_hd, price, comtype FROM PC-SS-HD
```

Regarding the computational system a JOIN merging the child components and checking for the available constraints is necessary as follows:

```
   CREATE VIEW PC-CS AS
   SELECT id_cpu, id_mb, id_ram, (c.price+m.price+r.price)
as price
   FROM PC-CS-CPU c, PC-CS-MB m, PC-CS-RAM r
   WHERE MC1 AND MC2 AND MC3 AND UC1
```

Constraints can be implemented as follows[5]:

```
   MC1:  NOT(m.type_mb='Asus' AND r.id_ram IN
(SELECT id_ddr FROM PC-CS-RAM-DDR WHERE type_ddr='ddrcoarse'))
   MC2:  c.socket=m.socket
   MC3:  m.#sockets >= 1
   UC1:  c.brand=m.brand
```

Finally, the last view representing the possible PC configurations is generated, possibly imposing further requirements as, for instance, a medium final price with confidence 0.9:

```
   CREATE VIEW PC AS
   SELECT (0.9) id_cpu, id_mb, id_ram, id_hd,
(cs.price+ss.price) as price, disc_f as discount,
(price-discount) as disc_price
   FROM PC-CS cs, PC-SS ss
   WHERE disc_price = medium
```

The framework and the configuration algorithm we have described have been implemented in a J2EE 3-tier architecture with JDBC interface. The resulting architecture is shown in figure 3. We tested the system using ORACLE XE, MySql and

[4]Notice that if `price` and `comtype` would not been stored in the `PC-SS-HD` view, a join on id attributes with `PC-SS-HD-IBM` would have been necessary to retrieve such aggregated attributes.

[5]Notice that if the user had chosen `#CPU=2`, then constraint `MC4` would have also been added here. In addition, constraint `MC3` is, in this example, trivially true (any motherboard has at least 1 socket), while user constraint `UC1` could be safely removed, since from the initial user require-

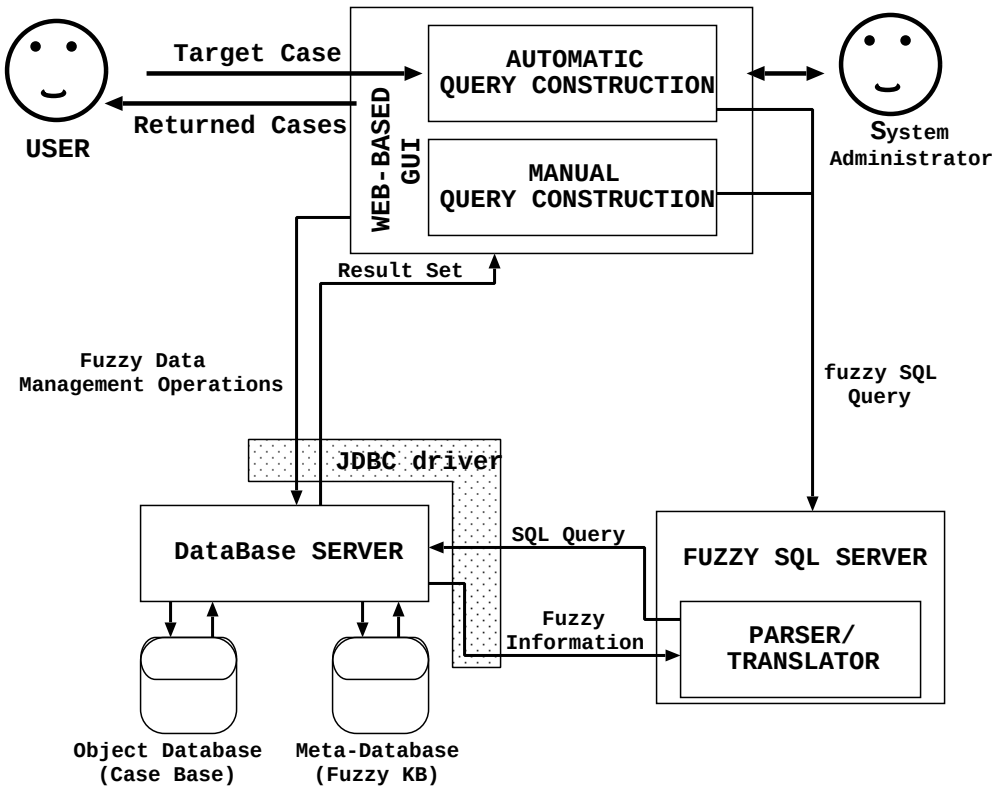

Figure 3: J2EE Architecture

SQL Server DBMS, while the adopted web container has been Apache TomCat (version 5.5.7). Fig. 4 shows a screenshot of a prototypical web application we developed using the architecture of fig. 3 and the approach described in the present work (http://caribe.mfn.unipmn.it:8080/mgalandr/store). Concerning the configuration example discussed above, we have tested it on an Oracle XE database; table 1 reports the cardinalities of the tables in the DB (i.e. the number of stored instances of basic categories).

Given the above number of instances, the PC configuration task discussed in this section, would produce, without considering constraints, **66640** possible configurations (i.e. 1CPU+1MB+1RAM+1HD=$35 \cdot 14 \cdot 17 \cdot 8 = 66640$). By considering the given model and user constraints, the number of possible solutions reduces to **4560**.

User requirements collected during the configuration process furtherly reduce such a number. In particular, table 2 reports the cardinalities of the generated views. We can notice that user requirements, even if approximate, can reduce

ments (user's exclusive interest in Intel devices) it is definitely satisfied.

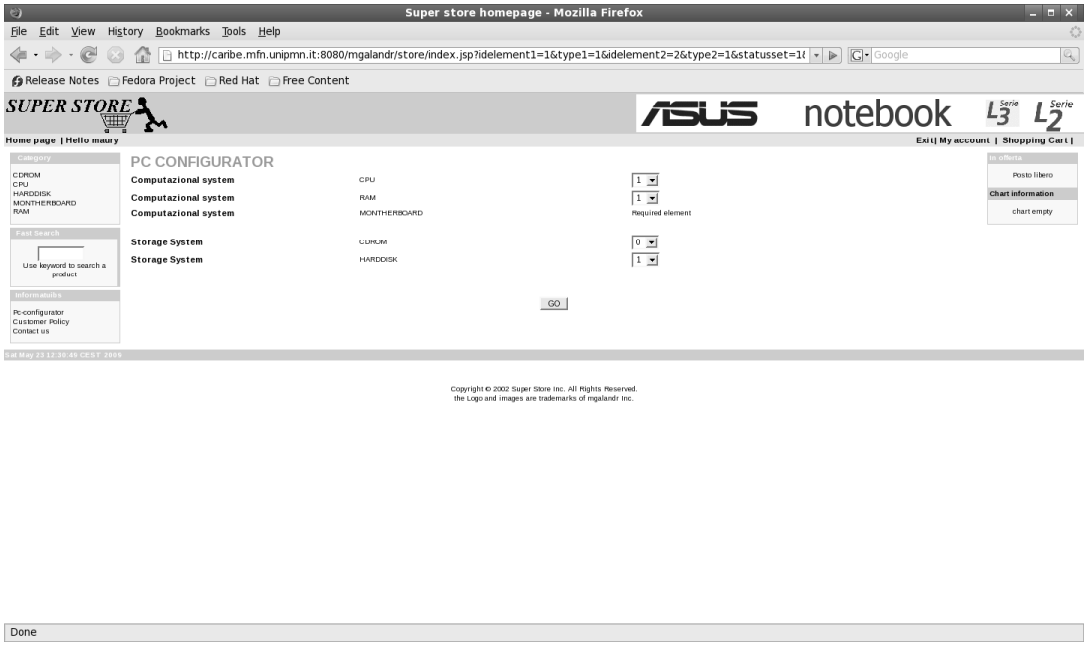

Figure 4: A screenshot for the PC assembly web application.

Table	Cardinality
PC-CS-CPU-INTELCPU	15
PC-CS-CPU-AMDCPU	10
PC-CS-MB-ASUS-ASUSAMD	3
PC-CS-MB-ASUS-ASUSINTEL	3
PC-CS-MB-GB-GBINTEL	5
PC-CS-MB-GB-GBAMD	3
PC-CS-RAM-DDR-DDRNORMAL	7
PC-CS-RAM-DDR-DDRCOARSE	4
PC-CS-RAM-SDR	3
PC-CS-RAM-RAMBUS	3
PC-SS-CDROM	4
PC-SS-HD-IBM	4
PC-SS-HD-MATROX	4

Table 1: Number of basic components (occurrences of basic categories) in the PC assembly example application.

View	Requirement (conf. lev.)	Cardinality
PC-SS-HD	capacity=high (0.7)	1
PC-CS-RAM-DDR	price<>very_expensive (0.7)	3
PC-CS-RAM	none	3
PC-CS-CPU	fsb=high (0.9)	11
PC-CS-MB-ASUS	fsb=high∧#usb=large (0.9)	5
PC-CS-MB-GB	fsb=high∧#usb=large (0.9)	3
PC-CS-MB	none	8
PC-SS	none	1
PC-CS	none	88
PC	price=medium (0.9)	5

Table 2: Cardinalities of the views generated during configuration for the PC assembly example application.

in a significant way the number of possibility to be considered. Of course, the confidence level used for a given fuzzy requirement is also relevant for selecting a reasonable number of results. In particular, in the above example, when building the view concerning to the computational system (CS), the corresponding join operation produces 88 tuples (they results from 3 RAMs modules, 11 CPUs and 8 MBs, pruned out by the available constraints); by requiring a medium price for the final PC (with a confidence level of 0.9), only 5 tuples survive and are finally presented to the user. It is worth noting that it would be really important to have the possibility of measuring the sensitivity of the confidence threshold with respect to the number of returned tuples, in such a way of defining suitable, possibly interactive, configuration policies able to tune such a threshold in a flexible and efficient way.

5 Conclusions and Future Works

We have presented an approach to product configuration, based on a hierarchical conceptual model and on fuzzy user requirements on the product features. The approach can be implemented on top of a relational database, exploiting the whole power of an SQL engine to implement both retrieval of product components, component composition and constraint checking. The proposed framework is a first step towards the definition of a flexible configuration architecture, where suitable strategies of system-user interactions can be defined. Indeed, future works will concentrate on such strategies, in such a way of providing the user with the guarantee of getting a reasonable set of acceptable configurations, by checking when

constraint are too strict or too large, by defining suitable policies on the definition of either hard or soft requirements, by tuning the sensitivity of the fuzzy semantics of attributes and by allowing a suitable ranking (possibly exploiting fuzzy membership) of the obtained solutions.

Acknowledgments

We are grateful to Gianluca Corazza who implemented the original fuzzy SQL compiler.

References

[1] L. Portinale and S. Montani. A fuzzy case retrieval approach based on SQL for implementing electronic catalogs. In *LNAI 2416*, pages 321–335. Springer, 2002.

[2] G. Fleischanderl, G. Friedrich, A. Haselbock, H. Scheiner, and M. Stumptner. Configuring large systems using generative constraint satisfaction. *IEEE Intelligent Systems*, 13(4):59–68, 1998.

[3] D. Sabin and E. Freuder. Configuration as composite constraint satisfaction. In *Proc. Artificial Intelligence and Manufacturing Research Planning Workshop*, pages 153–161, Albuquerque, NM, 1996. AAAI Press.

[4] M. Veron and M. Aldanondo. Yet another approach to CCSP for configuration problem. In *Proc. ECAI'00 Workshop on Configuration*, pages 59–62, Berlin, GE, 2000.

[5] D. Magro. Using constraint optimization to enhance the diversity in the set of computed configurations. In *Proc. ECAI'06 Workshop on Configuration*, Riva del Garda, IT, 2006.

[6] G. Friedrich and M. Stumptner. Consistency-based configuration. In *Proc. AAAI Workshop on configuration*. AAAI Press, 1999.

[7] D. Magro and P. Torasso. Supporting product configuration in a virtual store. In *LNAI 2175*, pages 176–188. Springer, 2001.

[8] T. Soininen, I. Niemela, J. Tiihonen, and R. Sulonen. Representing configuration knowledge with wieight constraint rules. In *Proc. AAAI Spring Symposium on Answer Set Programming*. AAAI Press, 2001.

[9] D. Magro and P. Torasso. Decomposition strategies for configuration problems. *AI for Engineering Design, Analysis and Manufacturing*, 17(1):51–73, 2003.

[10] A. Gunter and C. Kuhn. Knowledge-based configuration: survey and future directions. In *LNAI 1570*. Springer, 1999.

[11] T. Krebs, L. Hotz, and A. Gunter. Knowledge-based configuration for configuring combined hardware/software systems. In *Proc. PUK 2002*, Freiburg, GE, 2002.

[12] C. Sinz. Knowledge compilation for product configuration. In *Proc. ECAI'02 Workshop on Configuration*, pages 23–26, Lyon, FR, 2002.

[13] Eyke Hullermeier. Case-based search techniques for solving configuration problems. citeseer.ist.psu.edu/79018.html.

[14] L. Geneste and M. Ruet. Fuzzy case-based configuration. In *Proc. ECAI'02 Workshop on Configuration*, pages 1–10, Lyon, FR, 2002.

[15] H-E. Tseng, C-C. Chang, and S-H. Chang. Apllying case-based reasoning for product configuration in mass customization environments. *Expert Systems with Applications*, 29(4):913–925, 2005.

[16] L. Anselma, D. Magro, and P. Torasso. Automatically decomposing configuration problems. In *LNAI 2829*, pages 39–52. Springer, 2003.

[17] P. Bosc and O. Pivert. SQLf: a relational database language for fuzzy querying. *IEEE Transactions on Fuzzy Systems*, 3(1), 1995.

[18] L. Zadeh. Fuzzy logic. *IEEE Computer*, 21(4):83–93, 1988.

[19] J.R. Wright, D.L. McGuiness, C.H. Foster, and G.T. Vesonder. Conceptual modeling using knowledge representation: configurator applications. In *Proc. of Artificial Intelligence in Distributed Information Networks*, 1995.

[20] R. Brachman and J. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.