A System for Querying and Viewing Business Constraints

Mizuho Iwaihara†, Masayuki Kozawa†, Jun Narazaki‡ and Yahiko Kambayashi†
†: Department of Social Informatics,
Kyoto University, Sakyo-Ku, Kyoto, 606-8501 Japan
Email: iwaihara@i.kyoto-u.ac.jp
‡: Accenture Corporation, Tokyo, Japan.

Abstract

In E-commerce processes, various rules and constraints regarding product specifications, pricing, terms and conditions are exchanged between vendors and buyers. Developing a formal model for those rules and constraints gives a foundation for new E-commerce applications. Our approach is to develop a constraint database holding these conditions as dynamic constraints and provide a query language, called dynamic constraint algebra (DCA), for retrieving and matching business constraints. In this paper, we describe a system for processing DCA queries, its system architecture, constraint language, and graphical user interface for interactively querying over a database containing pricing and combination rules of products.

1 Introduction

Formalizing various rules and constraints appearing in business processes has a great potential toward advanced intelligent business automation. E-commerce processes involve matchmaking and negotiation between buyers and vendors. Such processes involve exchange of a wide variety of conditions such as price, discount, and delivery.

RuleML[4] is a standardization project aiming at providing XML interfaces for rules and formulas as a part of Semantic Web. RuleML incorporates research results on business rules from [11] and IBM Common Rules[12]. EContracts[17][18] provides a formal model for machine-interactible and analyzable E-commerce information, and provides Commerce Automata for matchmaking buyer and vendor conditions.

To formally represent and handle business constraints, we have introduced the notion of *dynamic constraints*[14]. A *dynamic constraint* is a condition attached to an object or a set of objects, and it constrains values an object can take, or constrains relationships among a set of objects. We use quantifier-free Boolean formulas on equalities as dynamic constraints, which are suitable for representing E-commerce negotiation constraints such as choices, combinations, and dependencies.

Constraint databases[22][8][7][19] have been studied for storing and querying constraints. Any decidable constraints having quantifier elimination can be chosen as database objects, and first-order queries on constraints in effect perform quantifier elimination to obtain quantifier-free constraints as query results[19].

In our previous work[14], we defined a new query algebra, called the dynamic constraint algebra (DCA), which has special operations for querying a variety of properties of dynamic constraints. Those properties include satisfiability, cardinality of constraint solutions, and dependencies between constraints and variables.

The following new features can be realized by utilizing the DCA and databases of dynamic constraints:

- Intelligent matching of multiple buyers and suppliers is possible, instead of today's hard-coded online sites or natural language-based order descriptions.
- Even if a buyer's purchasing plan includes incomplete informations such as unspecified or candidate values, the buyer can search potentially- or partially-matching suppliers.

- Although buyers and suppliers can freely describe dynamic constraints, the same query can be executed over such diversified constraints.

In this paper, we report our prototype system of a DCA database for business constraints. We describe the system architecture, constraint language, and especially its graphical user interface for interactively accessing the database.

2 Dynamic Constraint Algebra

In this section, we describe the Dynamic Constraint Algebra (DCA)[14][15] as our underlying formalism for modeling business constraints. Constraint databases[19] have been studied as a powerful approach for extending querying power of databases from the traditional relational database model. The key idea of constraint databases is that instead of storing a collection of tuples as data representing the real world, constraint databases store collections of *constraints*. Constraints are logical formulas having truth values for a given (ordinarily) tuple. The facts a constraint database represents is the set of tuples satisfying one of the constraints. By this way, constraint databases truly extend the expressive power of relational databases. Constraint representation of data also allows flexible logical operations on constraints, while deductive databases have restrictions on evaluation order of Horn clauses. Constraint databases are suitable for E-commerce applications, since business rules can be directly represented as database objects, and constraint-solving approach is necessary for matching complicated buyer's and vendor's constraints.

In the following, in stead of giving full formal definitions, we describe the DCA to be sufficiently concise for following discussion. A (constraint) tuple $[a_1, \ldots, a_k]/\phi$ consists of values $a_1, \ldots, a_k$ and constraint ϕ. The values are either constants or variables. Any class of constraints can be used for ϕ if the satisfiability problem is decidable in the class. We particularly choose the class of equivalence constraints such that equalities of the form $x = \text{“}120''$ or $x = y$ are connected by logical operations $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT), $\rightarrow$ (IMPLICATION). An example of constraint relations is shown in Figure 1. The class of equivalence constraints can represent Boolean combinations of possible constants, which are frequent in selling and buying constraints.

pid	**Item**	**Model**	**Price**	**Buy**	**Condition**
p0	MPU	m_1	p_1	b_1	$(m_1 = 500 \vee m_1 = 750) \wedge (m_1 = 500 \rightarrow p_1 = 249) \wedge (m_1 = 750 \rightarrow p_1 = 399)$
p0	RDRAM	m_2	p_2	b_2	$m_2 = 128\text{MB} \vee m_2 = 256\text{MB}$
p0	SDRAM	128MB	100	b_3	T
p0	Fan	Twin	60	b_4	$m_1 = 750$
p0	Fan	Single	p_3	b_5	$(b_1 = 1 \rightarrow p_3 = 30) \wedge (b_1 = 0 \rightarrow p_3 = 40)$

Figure 1: Constraint relation

In addition to the general framework of constraint databases, our approach of the DCA has the following unique features:

- We do not suppose a predefined set of variables which appear in constraint relations. Most of existing constraint algebras such as [2] use a simple schema structure such that each attribute is equivalent to a constraint variable. On the other hand, our constraint relation rather originates from conditional tables[1][10]. This is based on the observation that business rules of E-commerce have different formulations among products and vendors, so it is difficult to define a fixed schema/signature of variables, as constrains in the database can be added or updated as time goes on.
- First-order queries on constraints can be formulated by Boolean operations of constraints and by quantifications over variables. We need to specify argument variables, although we do not assume the existence of a predefined variable schema. To support querying on such variable schema-free constraints, we have introduced *dynamic constraint operations* to quantify variables without

explicitly specifying them. These operations use meta information on occurrence of free-variables in constraints. Dynamic constraint operations also have cardinality queries, which is represented by extended monadic second-order logic. These features are not covered by the constraint algebras of [2][1][10].

In the following, we describe the operations of the DCA. The standard operations of the traditional relational algebra can be extended to the constraint database version, where constraint of each tuple can be manipulated by these operations.

- **Union, Difference, Cartesian Product:** For constraint relations r and s, the union, difference and Cartesian product are denoted as $r \cup s$, $r - s$, and $r \times s$, respectively. Boolean operations on constraints are associated to those set operations. For $r \times s$, one tuple of r and one tuple of s are joined and AND of their constraints becomes the constraints of the new tuple. The difference $r - s$ is more complicated but we do not use this operation in this paper.

- **Selection, Projection, Substitution:** For a constraint relation r, $\sigma_F(r)$ gives the selection of r by the condition formula F, where each tuple's constraint is combined with F by AND, and the tuple will be in the result if the new constraint is satisfiable. The projection $\pi_A(r)$ is similar to the traditional projection. The substitution $S_\theta(r)$ gives the constraint relation such that θ is a mapping from variables to constants, and each variable v in r is replaced by $\theta(v)$, and the new tuples whose constraints are satisfiable will be in the result.

The following three operations are dynamic constraint operations, and they are unique to the DCA. The Boolean operation gives universal or existential quantification over all the variables of each tuple's constraint. The cardinality and dependency operations are specialized on the class of equivalence constraints, where number of satisfying constants of a certain variable is examined.

- **Boolean:** The Boolean operation has two forms: B_T and B_M. $B_T(r)$ returns tuples whose constraints are true, while $B_M(r)$ returns tuples whose constraints are unsatisfiable, i.e., there exists a constant assignment which makes the constraint false.

- **Cardinality:** The cardinality operation $C_{|A|\alpha k}$ finds tuples satisfying the cardinality predicate $|A|\alpha k$ such that A is an attribute, $|A$ is the number of constants such that the tuple's constraint is satisfiable if the constant is assigned to A, α is a comparison operation in $\{<, =, >\}$, and k is a non-negative number where k can be the infinity symbol $\mathtt{inf}$.

- **Dependency:** The dependency operation has two forms: $D_{cv(A)\beta Dep}$ and $D_{x\beta Dep}$. Here, Dep is the set of variables such that a tuple's constraint is dependent on. For each tuple, Dep is calculated and the predicate $cv(A)\beta Dep$ or $x\beta Dep$ is evaluated. Symbol $cv(A)$ is a variable on attribute A, if the value for A is actually a variable; otherwise the predicate is evaluated as true. Symbol x is a variable; this form is used for directly specifying a variable. Symbol β is a membership operator in $\{\in, \notin\}$. Dependency operation can be used to find tuples satisfying a given dependency relationship between variables and constraints.

Adopting constraint databases has drawback of increased complexities in query evaluation. Testing satisfiability of equivalence constraints is NP-complete. However, in practical situations we have to pay attentions on what subclasses of constraints we are actually dealing with. We could find a number of polynomial-time solvable constraint subclasses.

3 A Constraint Language

We have defined an XML-based language for constraint databases. Its DTD (document data type) is shown in Figure 2. The syntax structure of the constraint language is rather simple: A table element has a set of tuple elements, tuple elements have value elements and a condition element (cond and condxml tags; where cond is a text version and condxml is a tagged-version). A condition element contains constraints formed as a list of Boolean functions, where each function is defined as a formula over equality predicates and function symbols.

Translating the constraint language to RuleML[4] shall be straightforward except the point that RuleML needs to be extended to include constraint clauses in rule bodies, for testing and modifying constraints during query evaluation.

```
<!ELEMENT table (tuple)*>
<!ATTLIST table name CDATA #IMPLIED>
<!ELEMENT tuple ((value)*,(cond|condxml))>
<!ATTLIST tuple tid NMTOKEN #IMPLIED>
<!ELEMENT value (cons|var)>
<!ATTLIST value atrid NMTOKEN #IMPLIED atrname CDATA #IMPLIED>
<!ELEMENT cond (#PCDATA)>
<!ELEMENT condxml (expr)+>
<!ELEMENT expr (par|eq|neq|fdef|and|or|xor|imply|not|func|dva|bool)*>
<!ELEMENT fdef (func,expr)>
<!ATTLIST fdef delay (0|1) #REQUIRED>
<!ELEMENT eq ((var|cons),(var|cons))>
<!ELEMENT neq ((var|cons),(var|cons))>
<!ELEMENT par (par|eq|neq|fdef|and|or|xor|imply|not|func|dva|bool)*>
<!ELEMENT and EMPTY>
<!ELEMENT or EMPTY>
<!ELEMENT xor EMPTY>
<!ELEMENT imply EMPTY>
<!ELEMENT not EMPTY>
<!ELEMENT func EMPTY>
<!ATTLIST func name CDATA #REQUIRED>
<!ELEMENT var EMPTY>
<!ATTLIST var name CDATA #REQUIRED>
<!ELEMENT bool EMPTY>
<!ATTLIST bool value (0|1) #REQUIRED>
<!ELEMENT cons (#PCDATA)>
<!ELEMENT dva (func,(var|(bool|(var|bool)*)))>
<!ATTLIST dva type (bool|card|dep) #REQUIRED>
```

Figure 2: The DTD of the constraint language

4 System Architecture

We have developed a system for querying and manipulating business constraints using the DCA. The architecture of the system is depicted in Figure 3. The DB Interface parses input constraint databases, where each table is formated using the constraint language of Figure 2. The Query Parser parses input query files and performs top-level query processing in the processing hierarchy. The Query Processor splits dynamic constraints and inserts intermediate operations and hands those to the Constraint Translator. Here, intermediate operations are generated from DCA operations and used to evaluate chunks of dynamic constraints.

The Constraint Translator transforms dynamic constraints into propositional formulas and also generates Boolean operations expanded from intermediate operations. The translation algorithm is described in [14]. These outputs are sent to the Propositional Formula Processor, where model checking for testing requested properties is carried out. Binary decision diagrams are used for propositional formula manipulation[21][6]. The results are handed to the Query Processing Engine, where resultant relations are produced.

The system has an Web-based user interface, which we will describe in Section 5.

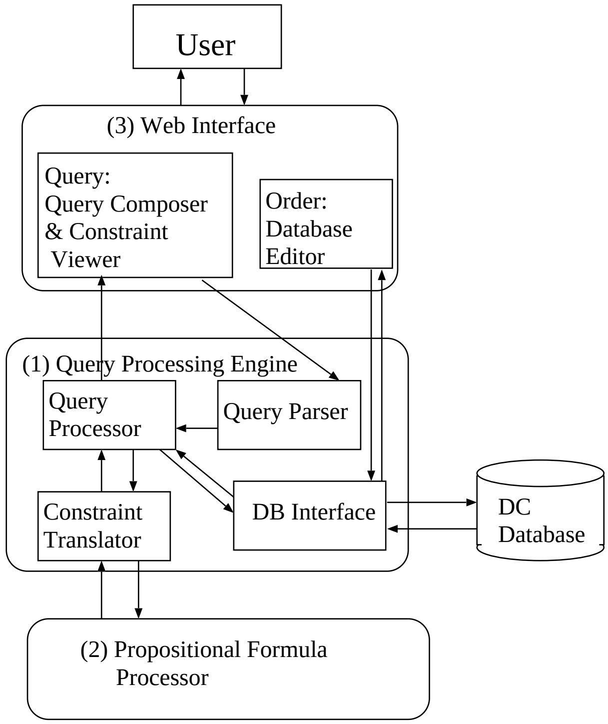

Figure 3: System architecture

5 Web Interface

In this section, we describe the Web Interface of the system. The Web Interface consists of two components: (1) Order and (2) Query. Order (Figure 4) is intended to be used by a buyer to conform a constraint relation representing the buyer's requests. Query (Figure 5) is used to compose a query, using the constraint relation of Order. Query also displays query results.

As application data, we use product specifications of personal computers(PCs). One PC model consists of a list of parts, such as MPU, Memory, Video Card, and each part has price information and constraints such that one part can or cannot be selected together with another parts. There are also rules for discounts such as giving discounts for specific combinations of parts. The buyer can choose parts and configure his/her own PCs.

Existing PC vendor sites require buyers to fill all the choices to check price and validity of choices. It is also a hard task to visit many sites and try a large number of combinations. On the other hand, in our system the buyer can describe his/her requests using a constraint relation, and by joining the constraint relation with the database of product constraints, PCs satisfying given specifications are found. It is possible to search by partial specification and to search multiple vendors by a single query.

The following requirements are considered to be necessary for the Web Interface.

- Guiding the user to enter constraints and queries, especially without knowing the syntax of the query and constraint languages.
- Assisting the user to verify whether entered queries and constraints are what he/she is intended. Also assist the user to correct wrong parts.
- Displaying query results (in the form of constraint relations) in an easily understandable form.

The Web Interface is implemented using Java and its GUI components Swing. We describe the functionalities of the interface below.

1. Order presents the user a list of constraint templates so that the user can fill the templates with variables and constants (Figure 6). Currently four types of templates are provided.

2. Order provides a list of variables and constants necessary for entering a constraint relation. The variables and constants are extracted from the product constraints, where one variable has a list of constants as a subset of its domain. The user can select one variable and a set of constants from a pull-down list, or add a new variable/constant(Figure 7 and Figure 8). Checking the syntax of entered constraints is carried out immediately.

3. Query provides templates of operations and a list of possible arguments for assisting the user to enter a query (Figure 9). The user's constraint relation entered using Order can be joined with the product constraints by a predefined query macro. The user can also write a query in the text area below.

4. Query displays a constraint relation as a query result. One constraint tuple may have complex constraints, so that constraints for a selected tuple can be displayed separately in the lower box (Figure 10). Constraints may be displayed in natural language sentences to help reading the constraints.

The Web Interface realizes basic functionalities for interactively matching and searching buyer and vendor's constraints, and examining properties regarding how one's constraints are satisfied. We briefly summarize our experience and observation using the system below:

- The process of finding satisfying products under given constraints was realized using the system, where interactive sessions of gradually adding constraints were effective.

- The interface functionalities of viewing constraints of a selected tuple in a separated window and summarizing constraints using natural language texts were effective. However, constraints sometimes become lengthy, and in such cases we believe that more advanced interfaces, such as reordering and navigating within constraints and introducing graphical visualization, are necessary.

6 Conclusion

In this paper, we described our prototype of a constraint database system for business constraints, including the user interface functionalities for querying business constraints.

It is natural to assume that different ontologies are used among different vendors. Hence it is necessary to incorporate semantic web technologies into query processing, to bridge the semantic gap. However, we must point out that even if a common ontology is used, business constraints can be still diversified in logical structures. In that respect, our approach of building databases of business constraints and offering querying functionalities is effective. Building more application-specific web interfaces is another future plan.

References

[1] S. Abiteboul, P. Kanellakis, and G. Grahne. On the Representation and Querying of Sets of Possible Worlds. *Theoretical Computer Science*, 78:159-187, 1991.

[2] A. Belussi and E. Bertino. An Extended Algebra for Constraint Databases. *IEEE Trans. Knowledge and Data Eng.*, 10(5): 686-705, 1998.

[3] E. Bertino, B. Catania and B. Chidlovskii. Indexing Constraint Databases by Using a Dual Representation. *Proc. Int. Conf. Data Engineering,* pp. 618–627, 1999.

[4] H. Boley, S. Tabet and G. Wagner, "Design Rationale of RuleML – A Markup Language for Semantic Web Rules," Semantic Web Working Symposium, July/August 2001, Stanford, 2001.

[5] A. Brodsky, J. Jaffer, and M. J. Maher. Toward Practical Constraint Databases. *Proc. 19th Int. Conf. on Very Large Databases*, 1993.

[6] R. E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams," *ACM Computing Surveys,* 24(3): 293–318, September 1992.

[7] J. V. den Bussche. "Constraint Databases: A Tutorial Introduction," *SIGMOD Record*, Vol. 29, No. 3, pp. 44–51, September 2000.

[8] V. Gaede, et al (eds.) *Constraint Databases and Applications.* (Proc. CDB'97 & Proc. CP'96 Workshops), Lecture Note in Computer Science, Springer, 1997.

[9] D. G. Goldin and P. C. Kanellakis, "Constraint Query Algebras," *Constrains J.,* 1(1-2): 45-83, 1996.

[10] G. Grahne, *The Problem of Incomplete Information in Relational Databases*, Springer-Verlag, Berlin, 1991.

[11] B. N. Grosof, Y. Labrou and H. Y. Chan. A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML. *Proc. ACM E-Commerce 99*, pp.68–77, Denver, 1999.

[12] IBM Common Rules, http://alphaworks.ibm.com.

[13] T. Imielinski and W. Lipski. The Relational Model of Data and Cylindric Algebras. *J. Computer and System Sciences*, 28(1): 80–102, 1984.

[14] M. Iwaihara, "Supporting Dynamic Constraints for Commerce Negotiations," *2nd Int. Workshop in Advanced Issues of E-Commerce and Web-Information Systems* (WECWIS), IEEE Press, 12–20, June 2000.

[15] M. Iwaihara. "Matching and Deriving Dynamic Constraints for E-Commerce Negotiations," *Workshop on Technologies for E-Services*, (informal proceedings), Cairo, Sep. 2000.

[16] P. C. Kanellakis and D. Goldin. Constraint Programming and Database Query Languages. *Proc. Int. Symp. Theoretical Aspects of Computer Software*, Lecture Notes in Computer Science 789, Springer, 96-120, 1994.

[17] D. Konopnicki, L. Leiba, O. Shmueli, and Y. Sagiv, "Toward Automated Electronic Commerce," First IAC Workshop on Internet Based Negotiation Technologies, 1999.

[18] D. Konopnicki, L. Leiba, O. Shmueli, and Y. Sagiv, "A Formal Yet Practical Approach to Electronic Commerce," Proc. CoopIS, pp.197-208, 1999.

[19] G. Kuper, L. Libkin and J. Paredaens, eds. "*Constraint Databases*," Springer Verlag, 2000.

[20] W. Lipski. On Databases with Incomplete Information. *J. ACM*, 28(1): 41–70, 1981.

[21] S. Minato, N. Ishiura, and S. Yajima. "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation," *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 52-57, June 1990.

[22] R. Ramakrishnan and P. J. Stuckey (eds.) "*Constraints and Databases*," Kluwer Academic Publishers, December 1997.

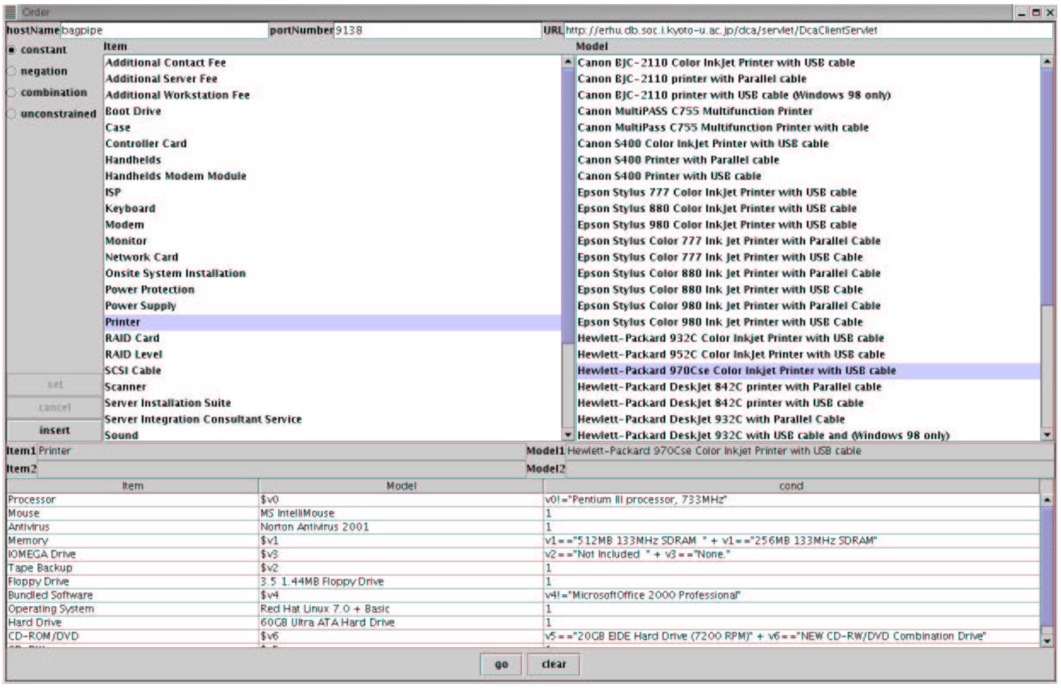

Figure 4: Order: web interface for entering constraint relations

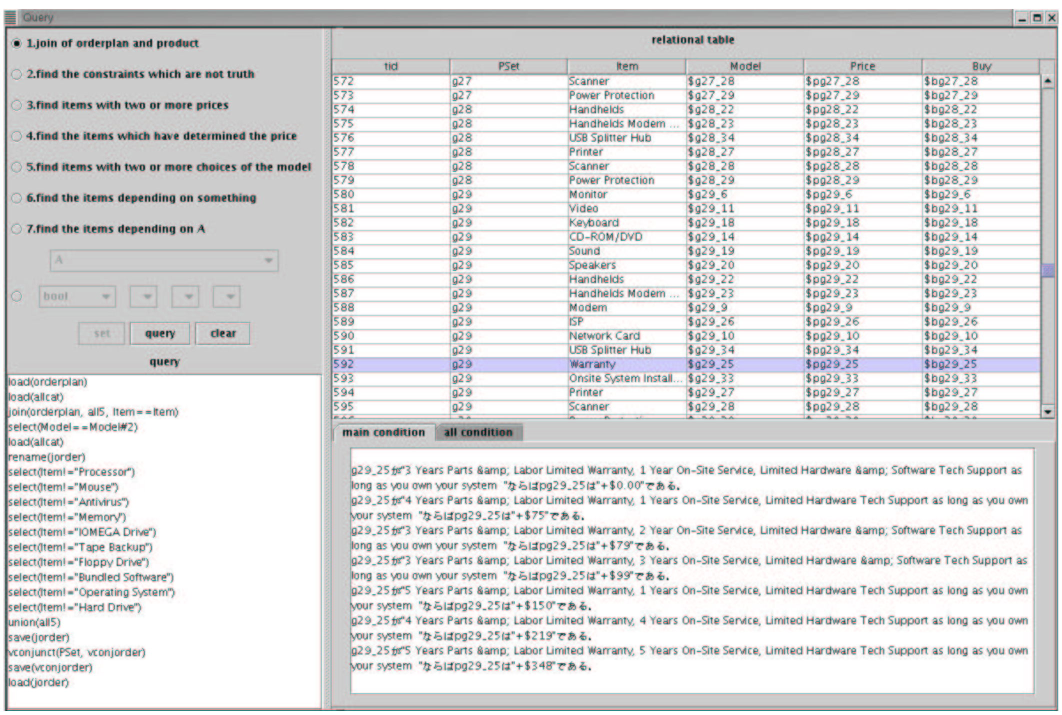

Figure 5: Query: web interface for interactive querying

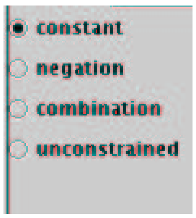

Figure 6: Order: specifying constraint templates

Figure 7: Order: choosing a variable in building constraints

Figure 8: Order: choosing a constraint in building constraints

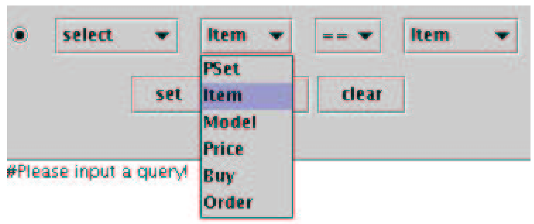

Figure 9: Query: choosing a query operation and arguments

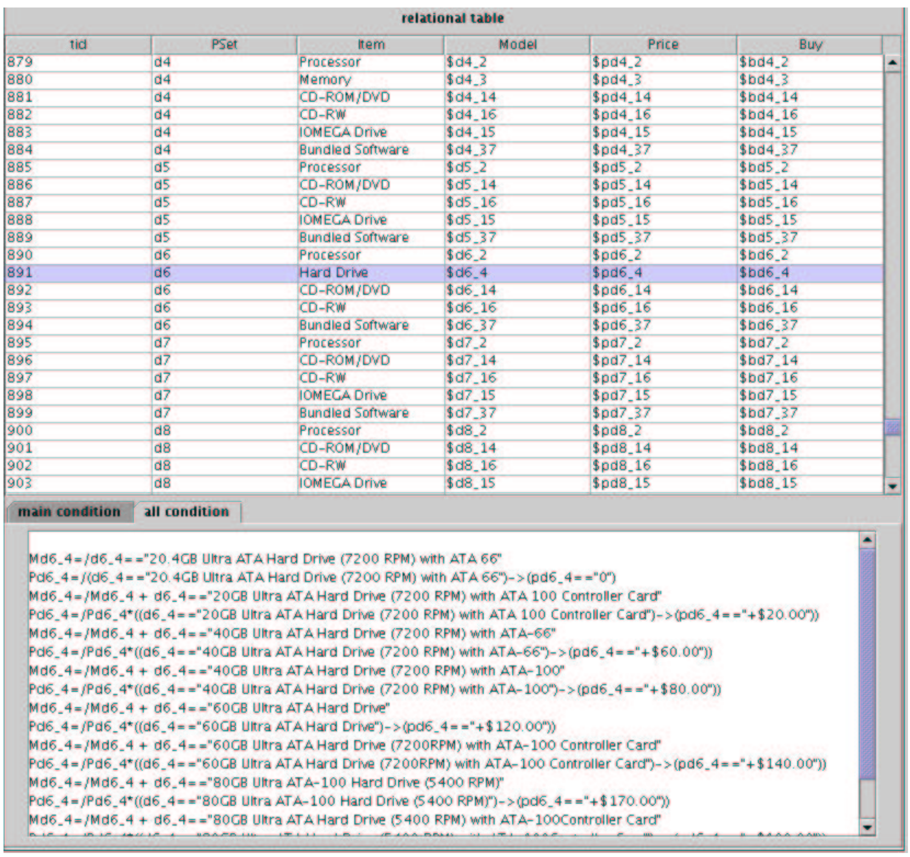

Figure 10: Query: viewing constraints of a query result