International Journal of Cooperative Information Systems

IMPROVING THE AUTOMATIC PROCUREMENT OF WEB SERVICES USING CONSTRAINT PROGRAMMING*

ANTONIO RUIZ–CORTÉS, OCTAVIO MARTÍN–DÍAZ, AMADOR DURÁN AND M. TORO
Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla
E.T.S.I. de Informática, Avda. Reina Mercedes s/n
41012 Sevilla, Spain

Received (Day Month Year)
Revised (Day Month Year)

Software solutions to automate the procurement of web services are gaining importance as the technology evolves, the number of providers increases and the needs of the clients become more complex. There are several proposals in this field, but they all have important drawbacks, namely: many of them are not able to check offers and demands for internal consistency; selecting the best offer usually relies on evaluating linear objective functions—which is quite a naive solution—; the language to express offers is usually less expressive than the language to express demands; and, last but not least, providers cannot impose constraints on their clients. In this article, we present a solution to overcome these problems that relies on constraint programming; furthermore, we present a run–time framework, some experimental results, and a comparison with other proposals.

Keywords: Cross–Organizational Systems; Web Services; Procurement; Quality–of–Service; Matchmakers; Constraint Programming.

1. Introduction

The Web is an environment in which service providers, communication links, and traders may be set up or set down unpredictably. Furthermore, the e–society is becoming tightly dependent on web services[1], which argues for solutions to automate their procurement. Thus, an effective, automated search and selection of the best services is essential for administrators and matchmakers.

Many authors argue that the foundations for a set of criteria to select among different software packages must lie on user requirements[2,3], and that web services are just a particular case of software packages[4,5]. These user requirements, to which we refer to as *demands*, are usually specified using boolean expressions, i.e. *conditions*, on parameters describing the desired features of a service, for example PRICE < 90. On the other hand, providers usually describe the features

*This work has been funded by the Spanish Central Government under grant TIC 2003–02737–C02–01 (AGILWEB).

of the services they provide, i.e. their *offers*, using (*parameter*, *value*) pairs, for example (PRICE, 75). Notice that the expressiveness of this way of describing offers, known as *parameter–value offers*, is very limited because only the *equality* operator is allowed, i.e. (*parameter*, *value*) pairs are simply a short form of the condition *parameter* = *value*, for example PRICE = 75.

Procurement is the process of finding the best offer for a given demand and it consists of the following major steps[6]: (1) a provider advertises its offers in a repository, e.g., UDDI[7] or *SalCentral*[8]; (2) a customer asks its matchmaker for an offer to meet its demands; (3) the matchmaker searches for matching offers and returns a result which may be a set of matching offers, the optimal matching offer according to a given customer criterion, or a failure message if no matching offers are found.

Apparently, this is a simple process, but the first experiences in this field have unveiled many open issues that are not well–supported by current proposals. Unfortunately, extending current proposals to overcome these drawbacks is not easy without changing their formal foundations. Below, we briefly report on some major problems and their implications, namely:

(1) Prior to advertising an offer and issuing a demand, they both should be checked for *consistency*, i.e. to check that they do not have any internal contradictions. It is usually assumed that this checking is very simple or even unnecessary, but we do not agree with this idea. For instance, not every current proposal is able to detect an inconsistency in a very simple demand containing conditions such as MTTF < 90 and MTTF > 120[a].

(2) Checking whether an offer fulfills a given demand, i.e. checking them for *conformance*, is usually performed simply by substituting parameters names in the expressions in the demand for the parameter values the provider guarantees in its offer. If after the substitution, all the resulting conditions are evaluated as true, the demand and the offer are considered to be *conformant*. This form of checking conformance is limited by the fact of having different languages for demands and offers; any condition is allowed in demands but only equality conditions are allowed in offers. Thus, a matchmaker is said to be *expressively symmetric*, or simply *symmetric*, if it accepts offers and demands written in the same language and that language includes other logical operators apart from the equality operator; otherwise, a matchmaker is said to be *asymmetric*. Unfortunately, most current matchmakers are asymmetric so they are not able to deal with offers expressed in a more expressive language other than parameter–value pairs; for example, offers asserting something like MTTF $\in$ [90..120].

[a]In our examples, MTTF denotes *mean time to failure*, MTTR denotes *mean time to repair*, MEDIA denotes communication links, e.g., modem, ISDN, or ADSL, COUNTRY denotes the country where a service is requested from, and PRICE denotes its price. They are all examples of parameters describing the features of web services.

(3) The process to find the optimal offer out of a set of offers conformant to a given demand, hereafter *optimal selection*, may range from using an objective function to solving a linear optimization problem. In any case, current matchmakers have not been devised to deal with non–linear objective functions or symmetric models. Thus, current matchmakers are not able to find the offer that maximizes an objective given by a function like $\frac{\mathrm{MTTF}}{\mathrm{MTTF+MTTR}}$.

(4) A matchmaker that accepts conditions not only on providers but also on clients, i.e. conditions that must be satisfied by clients in order to be served, is said to be a *two–way matchmaker*; otherwise, it is said to be a *one–way matchmaker*. Situations in which two–way matchmakers are necessary are very frequent in practice. For instance, if a web service that offers 128–bit cryptographic functions is hosted in the USA, then demands coming from other countries must not be accepted due to current USA laws. Most current proposals do not take these situations into account.

In this article, we present a proposal to improve the automatic procurement of web services which overcomes the above limitations. Adopting *Constraint Programming* as a formal basis has been the key point to achieve this improvement, because: (1) constraints allow customers and providers to state their demands and offers declaratively, endowing the symmetric model with a very powerful expressiveness; and (2) both customers and providers do not have to write specific procedures for consistency and conformance checkings, and optimized search and selection; instead, these operations are implemented by checking properties on demands and offers by means of a constraint solver in order to get a solution automatically.

In addition, we present a proof–of–concept implementation which allows to evidence that our proposal is feasible as well as to find out to what extent the matchmakers that can be built according to our model have a clear practical interest from a computational complexity standpoint. We also present a comparison with other proposals, remarking their solutions to the limitations pointed out at the beginning of this section.

The rest of the article is structured as follows. First, Section 2 introduces the theoretical basis for interpreting offers and demands by means of constraints. Section 3 presents our proposal to model the matchmaking process by means of constraint satisfaction problems. Next, a proof–of–concept of our model is shown in Section 4, and experimental results are documented in Section 5. Then, Section 6 provides a review of similar proposals. Finally, Section 7 concludes the article and presents future work.

2. Constraint Programming in a Nutshell

In general, checking a set of constraints for consistency, conformance or finding an optimal solution are well–known combinatorial problems that are difficult to solve, not only from a computational complexity standpoint—they are NP–hard problems

in general—but also from a programming standpoint, since they require expertise in applied mathematics, algorithms, and software engineering[9,10]. These problems have usually been tackled using brute–force and *Mathematical Programming* (MP)[11]. We propose using *Constraint Programming* (CP) as an alternative.

2.1. *Generalities*

CP is the study of computational models and systems based on constraints. CP has recently attracted the attention of many experts from distant areas because of its potential to solve hard, real–life problems. Currently, it is becoming the method of choice for modeling many optimization problems. Not only it is based on a strong theoretical foundation, but it is also attracting widespread commercial interest[9].

A constraint is a relation among several variables, each of which ranges over a given domain. Thus, a constraint restricts the values of its variables. Their most important feature is their declarative nature, i.e., they specify what relationships must hold without specifying a computational procedure to enforce them. The idea of CP is to solve problems by stating constraints about the problem area and, consequently, finding a solution that satisfies all of the constraints. This task is carried out by so–called *solvers*.

The earliest ideas that led to CP date back to 1960s, and they arose from the field of Artificial Intelligence. The main step towards CP was achieved when Gallaire[12] and Jaffar and Lassez[13] noted that *Logic Programming* (LP) was just a particular kind of CP. In LP the user states what needs to be solved instead of how to solve it, which is very close to the idea of constraints. Therefore, the combination of constraints and LP is quite natural, and it is referred to as *Constraint Logic Programming* (CLP). Furthermore, there are libraries to work with CP in languages such as C++[14,15], Java[16,15] or C#[15].

A problem expressed as a set of constraints is formalized as a *Constraint Satisfaction Problem* (CSP). A CSP is defined as a set of variables and a set of constraints specifying which combinations of variables and values are acceptable. Although most techniques to solve CSPs deal with constraints in which the variables range over the set of real numbers, there exist well–known transformation schemata to deal with integer, boolean, enumerated or even powerset domains.

2.2. *Definitions*

The core of our proposal is a set of definitions by means of which we can rigorously define the consistency and conformance checks as well as optimal selection.

Definition 2.1. (CSP) A CSP is a three–tuple of the form (V, D, C) where $V \neq \emptyset$ is a finite set of variables, $D \neq \emptyset$ is a finite set of domains (one for each variable) and C is a set of constraints defined on V.

For instance, the following tuple denotes a simple CSP that is used to illustrate several concepts throughout the paper:

$$(\{x, y\}, \{[0..2], [0..2]\}, \{x + y < 4, x - y \geq 1\})$$

Constraints in CP are generally expressed in a rich language that includes, for instance, linear and nonlinear constraints, the ability to index arrays with variables, or logical combinations of constraints. In practice, its expressiveness is only limited by the capabilities of the underlying solver.

A solution to a CSP consists of an assignment in which each variable gets a value from its corresponding domain, as long as it satisfies each constraint. In the previous example, the assignment $\sigma = \{x \mapsto 2, y \mapsto 0\}$ is a solution since it satisfies $2 + 0 < 4$ and $2 - 0 \geq 1$.

Definition 2.2. (Solution space) Let ψ be a CSP of the form (V, D, C), its solution space, denoted as $sol(\psi)$, is composed of all its possible solutions.

$$sol(\psi) = \{\ \sigma \in V \rightarrow D \mid \sigma(C)\ \} \tag{2.1}$$

where $\sigma(C)$ holds iff each assignment in σ satisfies every constraint in C.

In the previous example the solution space is $\{\{x \mapsto 1, y \mapsto 0\},\ \{x \mapsto 2, y \mapsto 0\},\ \{x \mapsto 2, y \mapsto 1\}\}$.

Definition 2.3. (Satisfiability) Let ψ be a CSP of the form (V, D, C), ψ is said to be satisfiable, denoted as $sat(\psi)$, iff its solution space is not empty.

$$sat(\psi) \Leftrightarrow sol(\psi) \neq \emptyset \tag{2.2}$$

In the previous example, if the second constraint is replaced with $x + y < -1$, then there are no solutions, and the CSP is therefore not satisfiable.

Given a CSP, there are three possible goals to be achieved:[17] i) just one solution, with no preference regarding which one out of the solution space is selected; ii) all of the solutions, if any; iii) an optimal solution according to an objective function defined in terms of the set of variables of CSP. In the last case, such problems are referred to as *Constraint Satisfaction Optimization Problems*[18].

Definition 2.4. (Minimum space) Let ψ be a CSP of the form (V, D, C), its minimum space with regard to an objective function O, denoted as $min_S(\psi, O)$, is composed of all of the solutions of ψ that minimize O.

$$min_S(\psi, O) = \{\ s \in sol(\psi)\ \mid\ \forall\, s\prime \in sol(\psi)\ \cdot\ O(s) \leq O(s\prime)\ \} \tag{2.3}$$

For instance, consider the CSP in the previous example and an objective function defined as $O(x, y) = x^2 y$. In this case, $min_S(\psi, O) = \{\{x \mapsto 1, y \mapsto 0\},\ \{x \mapsto 2, y \mapsto 0\}\}$.

Definition 2.5. (Minimum value) Let ψ be a CSP of the form (V, D, C), its minimum value with regard to an objective function O, denoted as $min_V(\psi, O)$, is the value the objective function takes on $min_S(\psi, O)$.

$$min_V(\psi, O) = m \Leftrightarrow \forall\, s \in min_S(\psi, O) \cdot O(s) = m \tag{2.4}$$

In the previous example, the minimum value is 0.

2.2.1. *Graphical representation*

CSPs are usually represented by means of a graph, but we do not think this is adequate enough to illustrate the CSPs associated with the conformance checking and optimal solution problems. We prefer to use Venn Diagrams in which the universal set (S) represents the multidimensional space whose dimensions are given by the domains of the variables, and each Venn Diagram represents the solution space of a particular CSP. Fig. 1 shows a graphical representation of the CSP used in the example in Section 2.2.

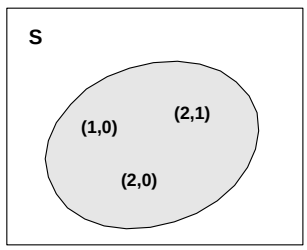


Fig. 1. Graphical representation of a CSP as a Venn Diagram.

For the sake of simplicity and understandability, the elements of the solution space are omitted when they are not relevant. Similarly, a constraint with only one solution is depicted as a point.

2.3. *Constraint programming versus mathematical programming*

The successful use of CP in areas such as planning, scheduling or optimization rises the question of whether the traditional field of *Operations Research* (OR) is a competitor or a partner of CP. For instance, it is worth mentioning that there is a significant overlap between CP and OR regarding NP–Hard combinatorial problems, where OR has been used successfully. Whereas OR has a long research tradition for solving problems using linear programming, CP emphasis is on higher–level modeling and solving methods that are easier to understand. Most recent advances[9] promise that both paradigms can benefit from each other. In particular, CP can be used as a platform for integrating several constraint–solving algorithms, including

those developed and checked to be successful in OR. The reason for choosing to represent and solve a problem as a CSP is twofold[17]:

- The representation as a CSP is often much closer to the original problem than the representation as an OR problem, since the variables of the CSP correspond directly to domain entities, and the constraints can be expressed without translating them into linear inequalities, as needed in OR. This makes the formulation simpler, the solution easier to understand, and the choice of good heuristics to guide the solution strategy more straightforward.

- Although CSP algorithms are essentially very simple, sometimes they can find a solution faster than more complex integer programming algorithms.

3. Procurement using Constraint Programming

In this section, we show how CP can help automate the *procurement tasks*, i.e. the checking for consistency and conformance, and the selection of optimal offers.

3.1. *Demands and offers*

The key to automating the procurement tasks is to map demands and offers onto CSPs. In order to do so, each parameter must be mapped onto a variable (with its corresponding domain), and each condition must be mapped onto a constraint. For instance, consider the demand "*The mean time to failure must be less than 100 minutes*" and the offer "*The mean time to failure is greater than 120 minutes*". Assuming that MTTF ranges over the natural numbers, the corresponding CSPs are $(\{\text{MTTF}\},\{[0,+\infty]\},\{\text{MTTF} < 100\})$ and $(\{\text{MTTF}\},\{[0,+\infty]\},\{\text{MTTF} > 120\})$ respectively.

This mapping needs to be extended for two–way matchmakers since both demands and offers may have complementary information. On the one hand, let δ denote a demand; in a two–way matchmaking context, we can consider δ to be composed of two parts to which we refer to as δ^γ, which asserts the conditions that the client meets, and δ^ρ, which asserts the conditions that the provider shall meet.[b] Similarly, any offer ω can also be considered to be composed of ω^γ (what it guarantees) and ω^ρ (what is required from its clients).

In general, with α being a demand or an offer, its corresponding CSP can be denoted as the pair of CSPs ψ^ρ_α and ψ^γ_α. In the case of one–way matchmakers, which can be considered a particular case of two–way matchmakers, the corresponding CSP is only ψ^ρ_α for demands and ψ^γ_α for offers, i.e. demands contain only requirements and offers contain only guarantees.

[b]In other words, δ^ρ represents the *requirements* a provider must fulfill in order to match the δ demand; that is why is denoted by the Greek letter rho (i.e. ρ*equirements*). On the other hand, δ^γ represents what a client *guarantees* about itself; that is why is denoted by the Greek letter gamma (i.e. γ*uarantees*). The same concepts are also applied to offers.

For instance, consider the demand "*The mean time to failure shall be less than 100 minutes*" (δ^ρ) "*and my host is in Spain*" (δ^γ) and the offer "*The mean time to failure is greater than 120 minutes*" (ω^γ) "*for USA and British clients only*" (ω^ρ). Assuming the previous definition of MTTF and that the COUNTRY variable ranges over the powerset of $\Lambda = \{\text{ES, US, UK, FR}\}$, i.e. $\mathcal{P}(\Lambda)$, their corresponding CSPs are defined as follows:

$$\delta^\rho = (\ \{\ \text{MTTF}\ \}, \{\ [0, +\infty]\ \}, \{\ \text{MTTF} < 100\ \}\ )$$
$$\delta^\gamma = (\ \{\ \text{COUNTRY}\ \}, \{\ \mathcal{P}(\Lambda)\ \}, \{\ \text{COUNTRY} = \{\ \text{ES}\ \}\ \}\ )$$
$$\omega^\gamma = (\ \{\ \text{MTTF}\ \}, \{\ [0, +\infty]\ \}, \{\ \text{MTTF} > 120\ \}\ )$$
$$\omega^\rho = (\ \{\ \text{COUNTRY}\ \}, \{\ \mathcal{P}(\Lambda)\ \}, \{\ \text{COUNTRY} \subseteq \{\ \text{UK, US}\ \}\ \}\ )$$

Note that interpreting demands and offers as CSPs is also valid for both symmetric and asymmetric matchmakers since the latter can be viewed as a particular case of the former in which the offers are expressed exclusively as *parameter–value offers*.

3.2. *Checking for consistency*

Checking an offer or a demand for consistency allows to unveil whether they have internal contradictions or not. In terms of CP, this amounts to verifying that their equivalent CSPs are satisfiable.

Definition 3.1. (Consistency) A demand or an offer α is said to be consistent iff its corresponding equivalent CSPs are satisfiable.

$$consistent(\alpha) \Leftrightarrow sat(\psi_\alpha^\rho) \wedge sat(\psi_\alpha^\gamma) \tag{3.5}$$

As an example, a demand that requires both MTTF < 90 and MTTF > 120 is inconsistent since it is not satisfiable. On the contrary, a demand that requires MTTF > 90 and MTTF < 120 is consistent since it is satisfiable (the range of its solution space is $\{91, \ldots, 119\}$).

3.3. *Checking for conformance*

Checking if an offer conforms to a demand allows to know whether the values guaranteed by a party meet the values required by the other party and viceversa. In terms of CP, this amounts to verifying that the solution space corresponding to the guarantees is a subset of the solution space corresponding to the requirements.

Definition 3.2. (Conformance) An offer ω and a demand δ are said to be conformant iff the solution space of ψ_ω^γ is a subset of the solution space of ψ_δ^ρ and the solution space of ψ_δ^γ is a subset of the solution space of ψ_ω^ρ.

$$conforming(\omega, \delta) \Leftrightarrow sol(\psi_\omega^\gamma) \subseteq sol(\psi_\delta^\rho) \wedge sol(\psi_\delta^\gamma) \subseteq sol(\psi_\omega^\rho) \tag{3.6}$$

For instance, an offer that guarantees MTTF $\in$ [100..120] is conformant to a demand that requires MTTF $\geq$ 100 since all of the values between 100 and 120 are greater or equal than 100, but it does not conform to a demand that requires MTTF $\geq$ 110 since the values between 100 and 109 are less than 110. Note that this conformance semantics might be branded as *pessimistic conformance*, because although *some* values in the guarantees could satisfy the requirements, only when *all* possible values in the guarantees satisfy the requirements, an offer and a demand are considered as conformant.

Fig. 2 depicts conformant and non–conformant situations. In Fig. 2.b, the non–conformance is due to the customer not being able to guarantee the conditions required by the provider, for example in a situation like the one described in Section 3.1, where the client of a host was neither from the USA nor British, but Spanish. In Fig. 2.c, the non–conformance is due to the provider not being able to guarantee the conditions required by the customer. Needless to say, this is the most usual non–conformance situation in practice.

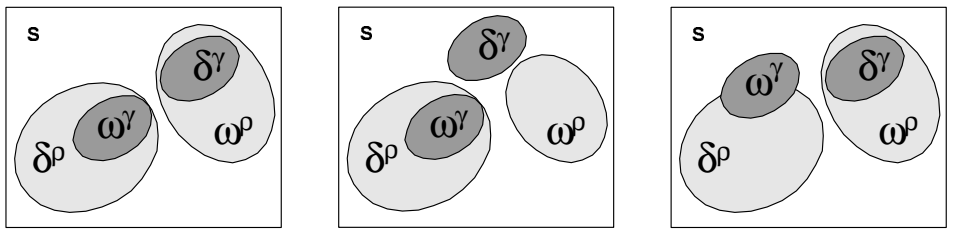


Fig. 2. Conformance in two–way, symmetric matchmakers.

3.4. *Finding the optimal*

The final goal of matchmaking is, given a demand, finding a conformant offer that is optimal from the customer's point of view. In the same way the two previous procurement tasks—checking for consistency and conformance—, finding the optimal offer can also be interpreted as a CSP, more specifically as a *Constraint Satisfaction Optimization Problems* (CSOP), which requires a preference order defined on the offer set. In order to establish a preference order, a *weighted composition of utility functions*[19,20] can be used, whose general form is as follows:

$$\mathcal{U}(p_1, \ldots, p_n) = \sum_{i=1}^{n} k_i \, U_i(p_i) \qquad k_i \in [0,1] \quad \sum_{i=1}^{n} k_i = 1 \tag{3.7}$$

where each p_i denotes a parameter, each k_i its associated weight, and each U_i its associated utility function ranging over $[0, 1]$ and describing how important the parameter values are for the client.

In Fig. 3, three utility functions for the parameters used in the example in the next section are shown. The utility function for MTTF is a piecewise linear function that defines a minimum utility for MTTF under 60 minutes; for MTTF between 60 and 120 minutes, the utility grows linearly, and for MTTF above 120, utility reaches its maximum value. The second utility function, which is a decreasing piecewise linear function, defines the utility for the MTTR parameter. The third utility function in Fig. 3 is a point–defined function for the MEDIA parameter in which the utility depends on the bandwidth of the different set of available communication links.

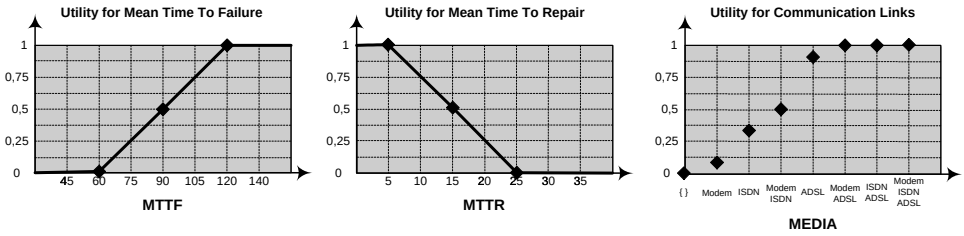


Fig. 3. Utility functions for MTTF, MTTR, and MEDIA.

As an example, consider an offer ω_1 that guarantees that MTTF $\in [75, 120]$ and an offer ω_2 that guarantees that MTTF $\in [90, 105]$. Which is it the optimal offer according to the utility function for MTTF in Fig. 3? At first sight, it could seem that ω_1 is the best one since its attainable maximum utility, i.e. its utility in the *best–case scenario*—which is 1 for a MTTF of 120—is greater than the attainable maximum utility of ω_2—which is 0.75 for a MTTF of 105.

Nevertheless, if a *pessimistic approach* were considered, the best offer would be ω_2 since its attainable minimum utility, i.e. its utility in the *worst–case scenario*—which is 0.5 for a MTTF of 90—is greater than the attainable minimum utility of ω_1—which is 0.25 for a MTTF of 75.

Definition 3.3. (Optimal offer) Let Ω_δ be a set of offers conformant to a demand δ, it is said that an offer $\omega \in \Omega_\delta$ is optimal with regard to a utility function U iff the minimum value of ψ_ω with regard to function U is the maximum of the minimum values of all offers in Ω_δ with regard to function U. We denote the set of optimal offers as $\Omega^*_{\delta,U}$.

$$\Omega^*_{\delta,U} = \{\ \omega \in \Omega_\delta \ \mid\ \forall \omega' \in \Omega_\delta \ \cdot\ min_V(\psi_\omega, U) \ \geq\ min_V(\psi_{\omega'}, U)\ \} \qquad (3.8)$$

3.5. *An illustrative example*

Consider that a company is interested in setting up a web portal specialized in movies offering a potentially infinite catalog of movies and the same functionality as a domestic video player. In order to achieve such a goal, the company should subcontract at least three kinds of web services: a service for streaming videos on the Internet, a service for managing catalogs and keeping them up–to–date, and a service for managing virtual shops. In this way, the portal would provide a service by means of the integration of other lower–level, third–party services.

Consider also that the interface *IVideoServer* abstracts the operations a web service for on–demand video streaming must have in order to be integrated into the web portal. Fig. 4 shows how the demand for such an interface, the offers and the parameters involved in our example can be formalized in QRL[21] (*Quality Requirements Language*), which is a language specifically devised for that purpose by one of the authors of this article as part of his PhD. thesis.[22]

In Fig. 4.a, two catalogs of parameters, the first one related to reliability and the second one to service hosting, are described. In Fig. 4.b, the demand from the web portal, including requirements, guarantees and preferences, is described. The preferences are specified in the *assessment* section by means of weighted utility functions, which in this case correspond to the utility functions depicted in Fig. 3. In Fig. 4.c, the offer from the *Velazquez* provider, which is only available for Spanish, British, American and Italian clients, is specified. In Fig. 4.d, the offer from a second provider, *Cervantes*, which imposes no requirements about the country where client must be located, is described.

In Fig. 5, the solution spaces of the demands and offers in the example are depicted. As can be seen, all solution spaces are not empty, implying that the corresponding demands and offers are all consistent. Taking into account that the two offers and the demand are conformant with respect to COUNTRY and MEDIA parameters—see Fig. 5.a for the space solutions related to the MEDIA parameter—, and that Fig. 5.b shows the inclusion of the solution spaces of both offers in the solution space of the demand, we can also say that the two offers and the demand are conformant.

As stated in Eq. 3.8, to select the optimal offer, the minimum value the weighted composition of utility functions (see Eq. 3.7) takes in the solution spaces of the offers must be computed. In order to do so, the minimum value of each parameter utility function must be computed and weighted. Applying Eq. 3.7 to the two offers, the results are the following:

$$min_V(\psi_{Velazquez}) = 0.9 * 0.25 + 0.05 * 0.85 + 0.05 * 1.0 = 0.32$$
$$min_V(\psi_{Cervantes}) = 0.9 * 0.50 + 0.05 * 0.75 + 0.05 * 0.5 = 0.51$$

so, the optimal offer is the offer from the *Cervantes* provider.

```
// A catalog of reliability parameters

catalog Reliability {
 MTTF {
   description : "Mean Time to Failure";
   domain : integer [0,10000] minute;
  }

 MTTR {
   description : "Mean Time To Repair";
   domain : integer [0,10000] minute;
  }
}


// A catalog of hosting parameters

catalog Hosting {
  MEDIA {
   description :
        "Communication links the service can be
         delivered over";
   domain : powerset { modem, ISDN, ADSL };
  }

 COUNTRY {
   description :
        "Country where the service is requested from";
   domain : powerset {ES, UK, IT, US, FR, BF};
  }
}
```

(a) Parameter catalogs

```
// Web service demand for IVideoServer

using Reliability, Hosting;
product IVideoServer;

requires {
 D1: MTTF/(MTTF+MTTR) >= 0.9;
  D2: MEDIA ⊇ {modem,ISDN};
}

assessment {
 MTTF { weight = 0.9,
   {(0,0), (60,0), (90,0.5), (120,1)}
  }
 MTTR { weight = 0.05,
   {(0,1), (5,1), (15,0.5), (25,0)}
  }
 MEDIA { weight = 0.05,
  { case MEDIA = {} : 0;
    case MEDIA = {modem} : 0.1;
    case MEDIA = {ISDN} : 0.3;
    case MEDIA = {ISDN, modem} : 0.5;
    case MEDIA = {ADSL} : 0.9;
    case MEDIA = {modem, ADSL} : 1;
    case MEDIA = {ISDN, ADSL} : 1;
    case MEDIA = {modem, ISDN, ADSL} : 1;
  }
}

guarantees {
 P1: COUNTRY = { ES };
}
```

(b) Customer's demand.

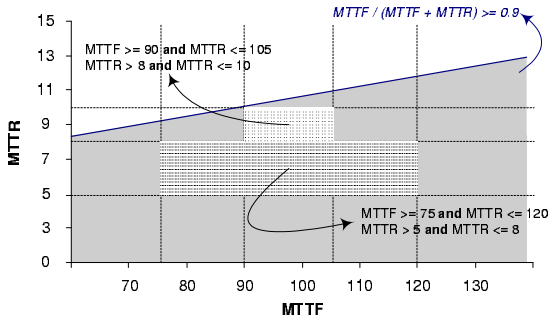
```
// Web service offer provided by Velazquez

using Reliability, Hosting;
product IVideoServer;

guarantees {
 O1: MTTF >= 75 and MTTF <= 120;
 O2: MTTR >= 5 and MTTR <= 8;
 O3: MEDIA = {ADSL,ISDN,modem};
}

requires {
 R1: COUNTRY ⊆ {ES, UK, US, IT };
}
```

(c) Velazquez's offer.

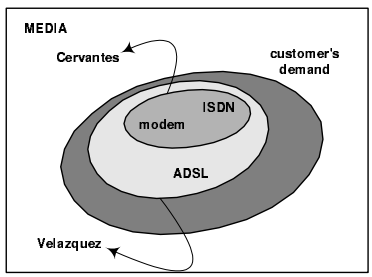
```
// Web service offer supplied by Cervantes

using Reliability, Hosting;
product IVideoServer;

guarantees {
 O1: MTTF >= 90 and MTTF <= 105;
 O2: MTTR >= 8 and MTTR <= 10;
 O3: MEDIA = {modem,ISDN};
}
```

(d) Cervantes's offer.

Fig. 4. Catalogues, demands, and offers written in QRL.



(a) Solutions regarding MTTF and MTTR.



(b) Solutions regarding MEDIA.

Fig. 5. Solution spaces of customer's demand, *Velazquez*'s and *Cervantes*'s offers.

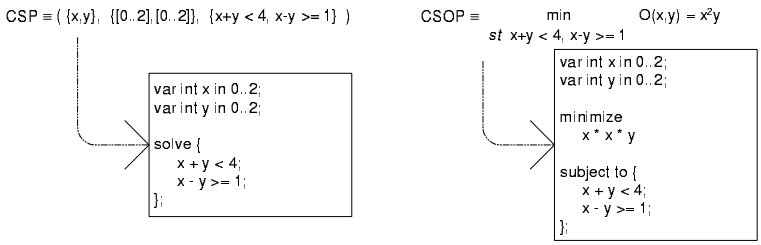
4. Proof–of–Concept Implementation

In this section, the most relevant aspects of the prototype of the symmetric, two–way matchmaker developed as a proof–of–concept are described. Firstly, the constraint solver used in the prototype is briefly commented. Then, we describe how procurement tasks can be performed using the solver capabilities. At the end of the section, the architecture of the prototype is presented[c].

4.1. *ILOG OPL Studio*

ILOG *OPL Studio*[15] version 3.6 has been the constraint solver chosen for the implementation of the proof–of–concept prototype. Apart of an *application programming interface* (API) which can be used from many programming languages, OPL Studio offers an *integrated development environment* (IDE) for modeling and solving CSPs and CSOPs with an intuitive graphical user interface.

[c]The interested reader can try a sample web application using the proof–of–concept prototype which is available at `http://www.tdg-seville.info/topics/procurement.html`.



(a) Mapping for a CSP. **(b) Mapping for a CSOP.**

Fig. 6. Mapping CSPs and CSOPs onto OPL models.

To solve a constraint problem in OPL Studio, a model of the problem specified in the *OPtimisation Language*[9] (OPL) is needed. Figure 6 depicts how to model in OPL[d] the CSP and the CSOP used in the examples in Section 2. In the case of a CSP, its equivalent OPL model consists of a declaration section where variables and their domains are declared, and a *solve* section where constraints to be solved are defined. In the case of a CSOP, its equivalent OPL model consists of a declaration section, a *maximize/minimize* section where the optimization function is defined, and a *subject to* section where constraints to which the optimization function is subject to are defined.

Once a model is defined in OPL, it is possible to query whether is satisfiable or not. If the model is satisfiable and all solutions are required, the OPL user must ask iteratively for the next solution. If the problem is a CSOP, solutions are returned ordered by their objective function value, which can also be delivered with the solution.

4.2. *Mapping procurement tasks onto OPL models*

QRL semantics assumes the existence of an ideal underlying solver. Thus, when translating offers and demands expressed in QRL into OPL, some details must be taken into account. As a case of QRL–to–OPL translation, the example in section 3.5 will be used for that purpose.

4.2.1. *Checking consistency*

According to Eq. 3.5, checking the consistency of an offer or a demand is equivalent to ask a constraint solver whether their corresponding CSPs are satisfiable or not. Fig. 7.a shows the OPL model for the consistency checking of the demand in Fig. 4.b. Notice that OPL does not allow to declare set variables, although they can be supported by means of boolean arrays indexed by an enumeration representing the set elements. A side–effect of this way of supporting set

[d]OPL also allows modeling more complex problems and specifying the solution search procedure. The details of such capabilities are out of the scope of this article.

```
enum TYPE_MEDIA {MEDIA_modem,MEDIA_ISDN,MEDIA_ADSL};
var int MEDIA[TYPE_MEDIA] in 0..1;

enum TYPE_COUNTRY {COUNTRY_ES, COUNTRY_UK, COUNTRY_IT,
                   COUNTRY_US, COUNTRY_FR, COUNTRY_BF};
var int COUNTRY[TYPE_COUNTRY] in 0..1;

range TYPE_MTTF 0..10000;
var TYPE_MTTF MTTF;

range TYPE_MTTR 0..10000;
var TYPE_MTTR MTTR;

solve {
   (MTTF * 100) / (MTTF + MTTR) >= 90;
   MEDIA[MEDIA_modem]=1 & MEDIA[MEDIA_ISDN]=1;
   COUNTRY[COUNTRY_ES] = 1
     & COUNTRY[COUNTRY_UK] = 0 & COUNTRY[COUNTRY_IT] = 0
     & COUNTRY[COUNTRY_US] = 0 & COUNTRY[COUNTRY_FR] = 0
     & COUNTRY[COUNTRY_BF] = 0;
};
```

(a) Checking consistency.

```
enum TYPE_MEDIA
          { MEDIA_modem,MEDIA_ISDN,MEDIA_ADSL};
var int MEDIA[TYPE_MEDIA] in 0..1;

range TYPE_MTTF 0..10000;
var TYPE_MTTF MTTF;

range TYPE_MTTR 0..10000;
var TYPE_MTTR MTTR;

solve {
  // CERVANTES'S IVIDEOSERVER OFFER
  90 <= MTTF <= 105;
  8 <= MTTR <= 10;
  MEDIA[MEDIA_modem] = 1
       & MEDIA[MEDIA_ISDN] = 1 & MEDIA[MEDIA_ADSL] = 0;

  not( // IVIDEOSERVER DEMAND
         ( (MTTF * 100) / (MTTF+MTTR) >= 90 )
       & ( MEDIA[MEDIA_ISDN] = 1
           & MEDIA[MEDIA_modem] = 1) );
};
```

(b) Checking conformance.

```
enum TYPE_MEDIA { MEDIA_modem,MEDIA_ISDN,MEDIA_ADSL};
var int MEDIA[TYPE_MEDIA] in 0..1;
var int UTILITY_MEDIA_VALUE in 0..111;

range TYPE_MTTF 0..10000;
var TYPE_MTTF MTTF;

range TYPE_MTTR 0..10000;
var TYPE_MTTR MTTR;

minimize
  0.90 * piecewise{0->60;1.67->90;1.67->120;0} MTTF+
  0.05 * (100 - piecewise{0->5;5->15;5->25;0} MTTR) +
  0.05 * piecewise{1->1;3.22->10;20->11;0.45->100;10->101;0}
              UTILITY_MEDIA_VALUE

subject to {
  UTILITY_MEDIA_VALUE
     = sum(AUX_MEDIA in TYPE_MEDIA)
         MEDIA[AUX_MEDIA] * pow(10,ord(AUX_MEDIA));

  // VELAZQUEZ'S IVIDEOSERVER OFFER
   75 <= MTTF <= 120;
   5 <= MTTR <= 8;
   MEDIA[MEDIA_modem] = 1
     & MEDIA[MEDIA_ISDN] = 1 & MEDIA[MEDIA_ADSL] = 1;
};
```

(c) Computing minimum value.

Fig. 7. OPL models for computing the procurement tasks.

variables is that constraint and utility function definitions become awkward. Also notice that the constraint $\mathrm{MTTF}/(\mathrm{MTTF} + \mathrm{MTTR}) \geq 0.9$ is translated into $100\,(\mathrm{MTTF}/(\mathrm{MTTF} + \mathrm{MTTR})) \geq 90$ due to the impossibility of OPL to deal with real non–linear constraints.

4.2.2. *Checking conformance*

According to Eq. 3.6, a possible way of checking conformance is computing the corresponding solution spaces and then checking whether they are subsets or not. Set inclusion is supported neither by OPL Studio nor—to the best of our knowledge—by any other constraint solver. Fortunately, there is an indirect way of checking

whether the solution space of a CSP is a subset of the solution space of another CSP by means of the *constraint implication* defined by Marriot and Stuckey[23]:

Definition 4.1. (Constraint implication) Let ψ_ω be a CSP of the form (V, D, C_ω) and let ψ_δ be a CSP of the form (V, D, C_δ). The solution space of ψ_ω is a subset of the solution space of ψ_δ iff the CSP defined as $(V, D, C_\omega \rightarrow C_\delta)$ is satisfiable, or iff its equivalent CSP $(V, D, C_\omega \wedge \neg C_\delta)$ is not satisfiable.

Thus, applying the constraint implication, both consistency and conformance checking can be considered as CSPs. In Fig. 7.b, the OPL model for checking whether the offer from the *Cervantes* provider and the demand in Fig. 4 are conformant, is shown. Notice the use of the *not* operator for expressing the negation of the demand, as indicated by the constraint implication definition.

4.2.3. *Finding the optimal offer*

According to Eq. 3.8, the optimal offer is the offer with the maximum of the minimum values of all conformant offers. The minimum value of an offer subject to an utility function (see Eq. 2.4) is obtained in OPL Studio requesting the solver for the first solution, which is the solution with the lowest utility value. The OPL model for finding the minimum value of the offer from *Velazquez* subject to the utility function in Fig. 4.b is shown in Fig. 7.c. Once the set of minimum values has been computed for all offers, the optimal offer is the one corresponding to the maximum value in the set.

Notice that in current version of OPL, piecewise linear functions are defined not as a sequence of points (x, y) but as a sequence of slopes at a point (s, x). Also notice that decreasing utility functions—like the utility function for MTTR in Fig. 3—are not allowed, so a translation into an equivalent increasing function is required.

4.3. *Architecture*

A prototype of the symmetric, two–way matchmaker has been developed using a component–oriented architecture, which is depicted in Fig. 8. The major components are briefly described below.

- *ST–Matchmaker.* In order to provide a unified interface to the matchmaker clients as well as minimize the communication and dependencies among components of the prototype, we have decided to apply the *facade* design pattern.[24] The ST–Matchmaker component plays this role.

- *Translator.* We assume that both demands and offers are specified in QRL, so a translation from QRL into OPL is needed. As an XML–based abstract syntax for QRL is available, this translation is carried out by using XSLT stylesheets[25] taking into account the mapping issues described in previous sections.

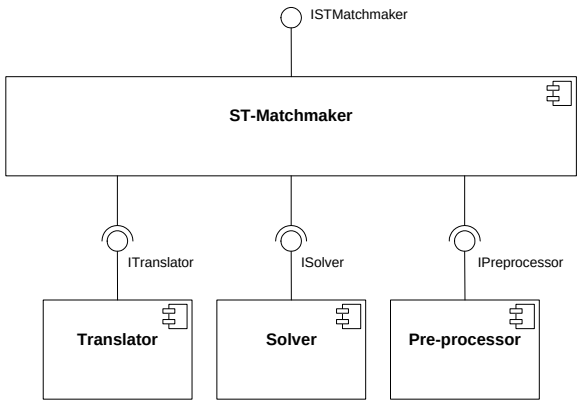


Fig. 8. Matchmaker architecture.

- *Pre–processor.* The goal of this component is to improve the solving process by using symbolic processing techniques very similar to the techniques used by the authors in previous works[10]. These techniques transform the OPL model into an equivalent model which can be solved more efficiently.

- *Solver.* This is the central component of the architecture. In the current prototype, this component is an *adapter*[24] of the OPL Studio API. Thus, this component shields the matchmaker from solver specific details, allowing the use of different solvers in the future.

The activities needed to support the matchmaking process are depicted in the self–explaining activity diagram in Fig. 9.

5. Experimental Results

In this section, the experiments carried out to empirically analyze the behavior of our symmetric, two–way matchmaker performing procurement tasks are presented. The empirical results have made evident that using a constraint solver is a viable technique from a practical point of view, provided several conditions on significant constraint factors are fulfilled.

5.1. *Execution environment*

The experiments were implemented using the *J#* programming language, which is an efficient Java dialect for the Microsoft .NET platform. The tests were performed on a computer with the Microsoft Windows 2000 Professional operating system, a 1.8 Ghz AMD Athlon microprocessor, and 512 megabytes of RAM memory. The background tasks of the operating system—known as *services* in Microsoft's terminology—except the .NET garbage collector were reduced to the minimum in

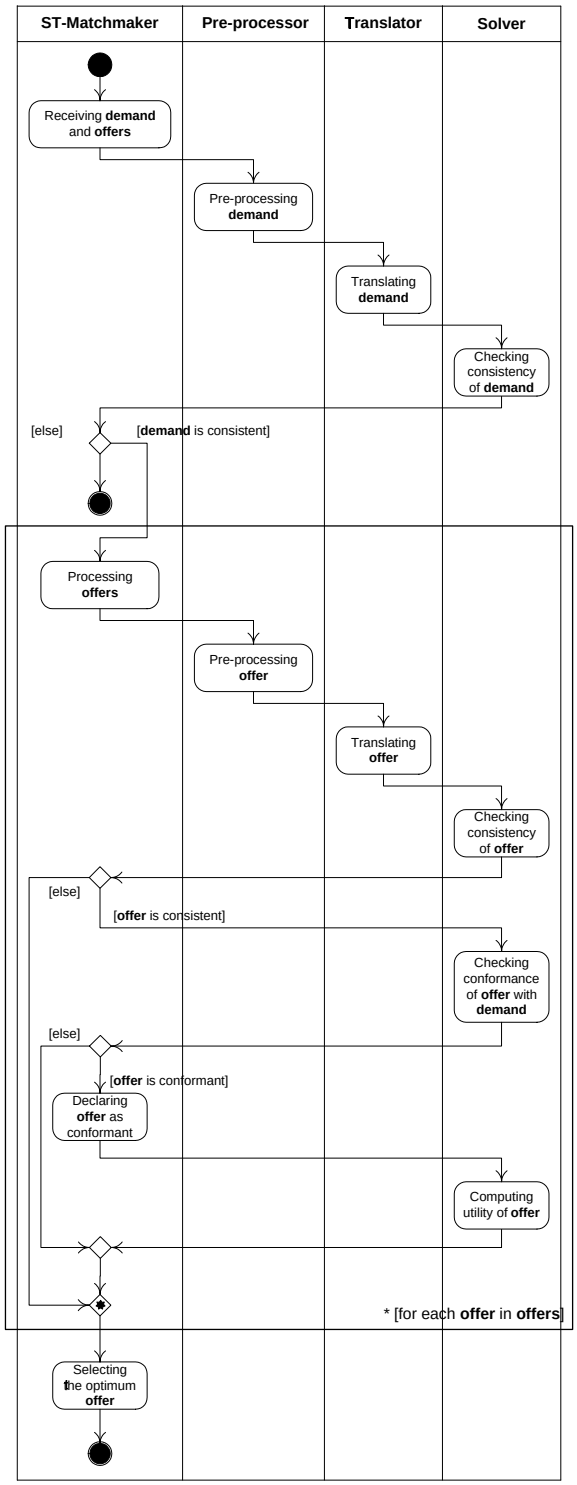


Fig. 9. Matchmaking activities.

order to avoid as much exogenous interferences as possible. In order to reduce significance of possible outliers produced by occasional interferences with the operating system or the network, averaged times in 30 runs were registered and the maximum and the minimum timings for each experiment were discarded.

5.2. *Significant factors*

Both from a theoretical and practical point of view, the analysis to determine the tendency of the performance of a constraint solver is a difficult task. Most experimental studies are based on identifying significant factors that determine the viability of CSP/CSOP resolution in the worst case. The validity of the results may be different between different solvers because not all of them implement the same solving heuristics. As an example, variables can be preprocessed and reordered in order to adjust the CSP to a concrete solver and get a better performance. Therefore, our experimental results are influenced by heuristics of the OPL Studio solver.

In our experiments, we have considered that the worst case is a CSP/CSOP whose constraints are defined so that they reduce the *initial search space* as less as possible. The significant constraint factors taken into account are the following:

(1) The number of variables (N) of the CSP which, together with their domains, determine the *initial search space* given by the Cartesian product of the domains. We distinguish between *small domains* (unsigned integers providing 256 solutions per variable), *medium domains* (unsigned short integers providing 65536 solutions per variable), and *large domains* (signed long integers providing $4.2 * 10^9$ solutions per variable).

(2) The number of constraints (C) of the CSP, which can affect solving time, especially in case of inequalities.

(3) The arity of a constraint (A), which is given by the number of bound variables in a constraint. In general, solvers are able to reduce the initial search space using unary constraints. For example, the unary constraint $x_0 \leq 10$ cuts off the initial search space, whereas the binary constraint $x_0 + x_1 \leq 100$ does not.

5.3. *Consistency checking experiments*

In the first experiment, the performance of consistency checking—of both consistent and non–consistent CSPs—was measured. In order to reduce the initial search space as less as possible, the tested CSPs contained the minimum number of constraints needed to bind all variables, which is given by the expression $C_{min} = N - A + 1$. For example, Fig. 10.a depicts the CSP used for $N = 3$, $A = 1$ and Fig. 10.c depicts the CSP used for $N = 5$, $A = 3$. In the case of non–consistent CSPs, a constraint to induce the inconsistency was explicitly added (see Fig. 10.b and 10.d).

```
var int x0 in 0..255;
...
var int x2 in 0..255;

solve {
  x0 > 10;
  x1 > 10;
  x2 > 10;
};
```

(a) Unary constraints (consistency)

```
var int x0 in 0..255;
...
var int x2 in 0..255;

solve {
  x0 > 10; x0 < 10;
  x1 > 10;
  x2 > 10;
};
```

(b) Unary constraints (non-consistency)

```
...

solve {
  x0 + x1 + x2 > 10;
  x1 + x2 + x3 > 10;
  x2 + x3 + x4 > 10;
};
```

(c) 3-ary constraints (consistency)

```
...

solve {
  x0 + x1 + x2 > 10; x0 + x1 + x2 < 10;
  x1 + x2 + x3 > 10;
  x2 + x3 + x4 > 10;
};
```

(d) 3-ary constraints (non-consistency)

Fig. 10. A number of OPL models for consistency checking experiments.

The first series of experiments aimed to study consistency checking in small and large domains. In the CSPs tested with small domains, N ranged from 1 to 1500 and A ranged from 1 to 10. The results of these experiments are shown in Fig. 11.a. As can be seen, the performance presents a linear behavior so that execution times slightly increase with respect to A and N. This was the expected behavior since the consistency checking is actually a satisfiability problem, so the solver only needs to find a solution for determining whether a CSP is satisfiable or not.

In CSPs tested with large domains, N ranged from 1 to 30 and A ranged from 1 to 10. Results are shown in Fig. 11.b. Note that for arities below 4, execution times are very similar in small and large domains, but for arities above 4, execution time gets considerably worse in large domains. For example, whereas in small domains it

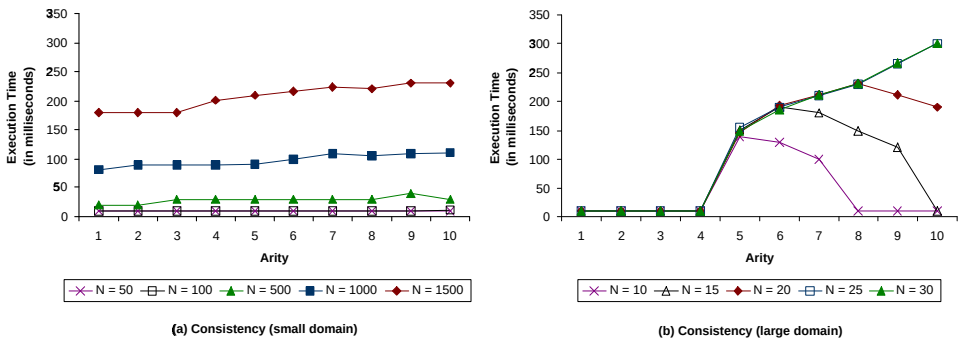


Fig. 11. Results from consistency checking experiments.

is possible to check the consistency of a constraint of 1500 variables with a maximum arity of 4 in 200 milliseconds, in large domains it is only possible to check the consistency a constraint of 30 variables with a maximum arity of 6 in the same period of time. As can be seen in Fig. 11.b, there are some arity values in which execution time suddenly decreases. Taking into account that the number of constraints in the tested CSP decreases as the arity increases, the results in Fig 11.b are not surprising. Notice that when the number of variables is equal to the arity, the number of constraints is only one.

The second series of experiments aimed to studying non–consistency checking in small and large domains. In the CSPs tested with small domains, N ranged from 1 to 20 and A ranged from 1 to 10. Results are shown in Fig. 12. As can be seen, the performance presents an exponential behavior so that execution times heavily increase with respect to A and N. This behavior is due to the fact that the solving process derives to an exhaustive search when no solutions can be found.

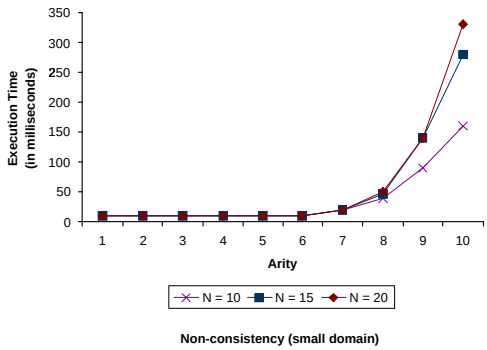


Fig. 12. Results from non–consistency checking experiments.

With respect to large domains, the OPL solver throws an *out–of–memory* exception when tries to solve a non–consistent CSP with a binary constraint. This behavior is due to the fact that the initial search space is 10^6 times bigger than in the small domain cases.

Summing up, the performance of consistency checking is acceptable for demands and offers using small domains and having an arity below 7. Since the usual arity in demands and offers is usually lower than 4 and large domains are seldom needed, we can conclude that consistency checking using our approach can be performed efficiently in most cases.

5.4. *Conformance checking experiments*

In the second experiment, the performance of conformance checking—again, of both consistent and non–consistent CSPs—was measured. Since conformance checking is also a satisfiability problem—and in order to avoid experiment repetition—in

```
var int x0 in 0..255;
...
var int x2 in 0..255;

solve {
  x0 > 4;  x1 > 2;  x2 > 10;
  not(x0 > 4 & x1 > 2 & x2 > 10);
};
```

(a) Binary constraints(conformance)

```
var int x0 in 0..255;
...
var int x2 in 0..255;

solve {
  x0 > 4;  x1 > 2;  x2 > 10;
  not(x0 > 4 & x1 > 2 & x2 < 10);
};
```

(b) Binary constraints (non-conformance)

```
...

solve {
  x0 + x1 + x2 > 10;
  not (x0 + x1 + x2 > 10);
};
```

(c) 3-ary constraints (conformance)

```
...

solve {
  x0 + x1 + x2 > 10;
  not(x0 + x1 + x2 < 10);
};
```

(d) 3-ary constraints (non-conformance)

Fig. 13. Some OPL models for conformance checking experiments.

this experiment we focused on the impact of the negation clause in Marriott and Stuckey's equation (see Section 4.2.2). Fig. 13 shows some examples of CSPs used in this experiment, with significant factors taking similar values than in the first experiment.

Experimental results of this experiment are shown in Fig. 14. In the case of satisfiable CSPs, i.e. non–conformant demands and offers, the performance shows a linear behavior with respect to the number of variables in both small and medium domains. In the case of non–satisfiable constraints, i.e. conformant demand and offers, the performance presents an exponential behavior with respect to the number of variables for medium domains and a linear behavior for small domains.

Notice that non–conformant cases perform better than conformant ones, making evident the strong impact on performance of the negation clause in Marriott's equation. In the worst case, the problem is viable when the number of variables is under 4.

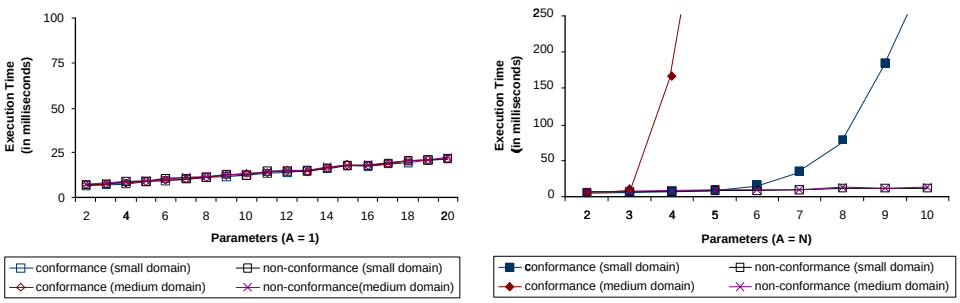


Fig. 14. Results from conformance checking experiments.

Summing up, the performance of conformance checking is acceptable for demands and offers using unary or binary constraints. For constraints with arity greater than 2, performance gets worse exponentially. Although conformance checking figures are worse than consistency checking figures, they can still be considered as being acceptable.

5.5. *Optimal selection experiments*

In the third experiment, the performance of optimal selection was measured. We were inspired by one of the experiments on *AgFlow* carried out by Zeng.[11] In Zeng's experiment, the execution time needed to find the optimal offer from a set of candidate offers is computed. The number of demands ranged from 10 to 80 in steps of 10 demands and the number of offers ranged from 10 to 40 in steps of 10 offers. As our matchmaker is symmetric, we performed this experiment for both parameter–value and non–parameter–value offers.

Fig. 15 shows the OPL model template for computing the optimal selection, where weights in objective functions (k_i) and literal values in inequality constraints (v_i) are randomly assigned.

```
var int x0 in 0..65535;
var int x1 in 0..255;
var int x2 in 0..255;

minimize
    k0 * (100 - piecewise{0.1->1000;0} x0)
  + k1 * piecewise{1->100;0} x1
  + k2 * piecewise{1->100;0} x2

subject to {
   x0 - x1 < v0;
   x1 + x2 > v1;
};
```

Fig. 15. OPL model template for optimal selection experiments.

Empirical results of this experiment are shown in Fig. 16. In the case of parameter–value offers, performance shows a linear behavior with respect to the number of variables. In the case of non–parameter–value offers, the result was a second–grade polynomial function—the same as in Zeng's experiment—so that performance increases with the number of demands and offers.

As expected, non–parameter–value offers cases performed better than parameter–value ones, making evident the strong impact on performance of computing the minimum value of a function.

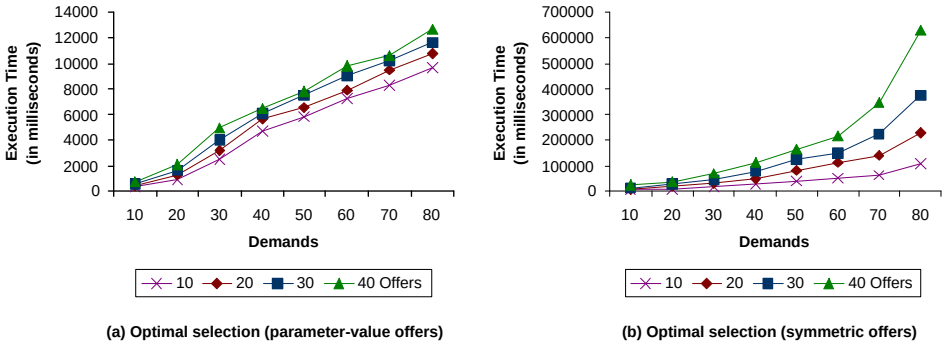


Fig. 16. Results from optimal selection experiments.

6. Related Work

In this section, some proposals that totally or partially perform the procurement tasks described in this article are reviewed. They have been grouped according to their underlying formalism, providing a summarized view of the current state–of–the–art. A detailed explanation of each proposal is available for the interested reader in a previous work[26] by the authors of this article.

6.1. *Proposals based on ad–hoc formalisms*

The first group is constituted by those proposals whose semantics is defined by using an *ad–hoc* formalism. The reviewed proposals are the following: (1) *ServiceGlobe*[27] and *UDDI Extension*[28] (UDDIe), both working on UDDI[7] repositories; (2) the IBM *Web Services Level Agreement*[29] language (WSLA), which uses a tree–based algorithm to check conformance; (3) the Hewlett–Packard *Quality-of-service Modeling Language*[30] (QML), whose semantics rely on the syntactic form of the constraints; and (4) the IBM *Web Services Matchmaking Engine*[31] (WSME), which uses a Java–based language to specify demands and offers.

None of the above proposals are focused in improving the expressiveness of demands and offers but in aspects related to matchmaker development in order to get a successful integration in current service–oriented platforms. Thus, regardless of specific aspects, all proposals can perform consistency and conformance checking applying their own *ad–hoc* algorithms, which are much more limited than—and not as efficient as—current CSP solvers, i.e. they are somehow *reinventing the wheel.*

6.2. *Proposals based on semantic web formalisms*

Proposals in this group are based on semantic web foundations[32,33]. We have reviewed two proposals from Hewlett–Packard and one proposal from the *Multichannel Adaptive Information Systems project* (MAIS).

The first proposal[34] from Hewlett–Packard, which is based on RDF[35], uses an

ad–hoc algorithm whereas the second[36], which is based on the DAML+OIL[37] ontology language, uses *description logics*[38,39] (DL) reasoners. The MAIS proposal[40] uses ontology–based classification techniques, which are supported by the ARTEMIS tool environment[41].

An advantage of using *ad–hoc* algorithms for carrying out the procurement tasks is that they can be tailored for specific–problem optimizations. On the contrary, there are a number of major disadvantages like development time, lower quality, debugging time, maintenance, and so on.

Proposals based on DL solvers usually rely on external constraint solvers to deal with numeric constraints.[42] Notice that DL solvers are devised to reason on relationships among objects not among numeric variables.

Summing up, in order to provide an expressiveness similar to our matchmaker, current proposals need to use CP as their formal basis. Otherwise, a lot of already known algorithms have to be reinvented.

6.3. *Proposals based on constraint/mathematical programming*

Apart from the one described in this article, only IBM *AgFlow*[11] can be considered as a proposal based on CP or MP. This proposal deals neither with consistency nor with conformance, but it provides a solution for optimal selection which could even deal with non–parameter–value offers. However, as commented in Section 2.3, MP is not as expressive as CP, so interpreting consistency and conformance checking as a MP problem may lead to a very artificial formulation.[e]

7. Conclusions and Future Work

In this article, we have shown how constraint programming can be used to improve the automation of the procurement of web services and therefore of the current matchmakers. We have also shown the soundness of our proposal from both theoretical and experimental points of view. The major advance of our proposal is to provide a semantics for the procurement tasks such that demands and offers can be modeled using a very expressive language, QRL, which allows inequalities and non–linear expressions. From the experimental study we can conclude that, in the domain of web service procurement, CP–based matchmakers are practically viable despite of its—theoretical—combinatorial nature.

As future work, we are considering to improve the expressiveness of QRL by allowing the association of temporal periods to constraints. Thus, it would be possible to state something like "*The* MTTF *should be greater than 120 minutes on weekends and greater than 60 minutes from Monday to Friday*". Another future improvement would be allowing that both offers and demands could be specified using different parameter catalogs, i.e. different ontologies of web service features. For

[e]The interested reader can see the article in which *AgFlow* is described[11] to compare the formulation of the optimal selection using Integer Programming with our formulation using CP.

that purpose, we are currently considering using description logics together with constraint programming, thus bringing together the best of both worlds.

Acknowledgements

The authors would like to thank Dr. Rafael Corchuelo, Mr. David Benavides and the Quivir Group members for their helpful discussions. We also thank to the reviewers of both the International Conference on Service Oriented Computing and the IJCIS Journal whose comments and suggestions improved the presentation substantially.

References

1. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architecture and Applications*. Springer–Verlag, 2004.
2. A. Finkelstein and G. Spanoudakis. Software Package Requirements and Procurement. In *Proc. of the 8th Int'l IEEE Workshop on Software Specification and Design (IWSSD'96)*. IEEE Press, 1996.
3. X. Franch and J.P. Carvallo. Using Quality Models in Software Package Selection. *IEEE Software*, 20(1):34–41, 2003.
4. O. Martín-Díaz, A. Ruiz-Cortés, D. Benavides, A. Durán, and M. Toro. A Quality-aware Approach to Web Services Procurement. In *Fourth International VLDB Workshop Technologies for E-Services*, volume 2819 of *Lecture Notes in Computer Science*, pages 42–53, Berlin, Germany, 2003. Springer Verlag.
5. O. Martín-Díaz, A. Ruiz-Cortés, A. Durán, D. Benavides, and M. Toro. Automating the Procurement of Web Services. In *1st Int.l Conf. on Service-Oriented Computing*, volume 2910 of *LNCS*, pages 91–103, Trento, Italy, 2003. Springer Verlag.
6. K. Sycara, M. Klusch, S. Widoff, and J. Lu. Dynamic Service Matchmaking among Agents in Open Information Environments. *SIGMOD Record*, 28(1):47–53, 1999.
7. UDDI web site, 2004. `http://www.uddi.org`.
8. SalCentral web site, 2004. `http://www.salcentral.com`.
9. P. Hentenryck. Constraint and Integer Programming in OPL. *Informs Journal on Computing*, 14(4):345–372, 2002.
10. R. Martínez, J.A. Ortega, and M. Toro. A Framework for Semiqualitative Reasoning in Engineering Applications. *Applied Artificial Intelligence*, 16(3):173–197, March 2002.
11. L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, May 2004.
12. H. Gallaire. Logic programming: Further developments. In *IEEE Symposium on Logic Programming*. IEEE, 1985.
13. J. Jaffer and J.L. Lassez. Constraint logic programming. In *14 ACM Symposium on Principles of Programming Languages*, pages 111–119. ACM, 1987.
14. G. Katsirelos. EFC web site. Class Library for Constraint Programming in C++, 2004. `http://www.cs.toronto.edu/ gkatsi/efc/`.
15. ILOG. ILOG optimization suite web site. `http://www.ilog.fr`.
16. N. Tamura. Cream web site. Class Library for Constraint Programming in Java , 2005. `http://bach.istc.kobe-u.ac.jp/cream/`.
17. B. Smith. A Tutorial on Constraint Programming. Research Report 95.14, School of Computing. University of Leeds, 1995.
18. E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1995.

19. J.J. Dujmovic. A Method for Evaluation and Selection of Complex Hardware and Software Systems. In *Proc. of the 22nd Int'l Conf. for the Resource Management and Performance Evaluation of Enterprise Computing Systems*, pages 368–378, 1996.
20. J. Koistinen and A. Seetharaman. Worth–based Multi–Category Quality–of–Service Negotiation in Distributed Object Infrastructures. In *Proceedings of the 2nd Int'l Enterprise Distributed Object Computing Workshop (EDOC'98)*, La Jolla, USA, 1998.
21. A. Ruiz-Cortés, R. Corchuelo, A. Duran, and M. Toro. Automated support for quality requirements in web-services-based systems. In *Proc. of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'2001)*, Bologna, Italy, 2001. IEEE Press.
22. A. Ruiz-Cortés. *A Semiqualitative Approach for the Automatic Management of Quality Requirements (in Spanish)*. PhD thesis, University of Seville, 2002.
23. K. Marriott and P.J. Stuckey. *Programming with Constraints: An Introduction*. The MIT Press, 1998.
24. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison–Wesley, 2005.
25. W3C. XSL Transformations Version 1.0, 1999. `http://www.w3.org/TR/xslt`.
26. O. Martín-Díaz, A. Ruiz-Cortés, R. Corchuelo, and M. Toro. A Framework for Classifying and Comparing Web Services Procurement Platforms. In *First International WISE Web Services Quality Workshop*, pages 156–164, Rome, Italy, 2003. IEEE Press.
27. M. Keidl, S. Seltzsam, and A. Kemper. Reliable Web Service Execution and Deployment in Dynamic Environments. In *Fourth International VLDB Workshop Technologies for E-Services*, volume 2819 of *Lecture Notes in Computer Science*, pages 104–118, Berlin, Germany, 2003. Springer Verlag.
28. A. ShaikhAli, O. Rana, R. Al-Ali, and D. Walker. UDDIe: An Extended Registry for Web Services. In *Proc. of the IEEE Int'l Workshop on Service Oriented Computing: Models, Architectures and Applications at SAINT Conference*. IEEE Press, January 2003.
29. P. Grefen, H. Ludwig, and S. Angelov. A Three-Level Framework for Process and Data Management of Complex E-services. *International Journal of Cooperative Information Systems*, 12(1):455–485, December 2003.
30. S. Frolund and J. Koistinen. Quality–of–service specification in distributed object systems. *Distributed Systems Engineering Journal*, 5(4), 1998.
31. Y. Hoffner, S. Field, P. Grefen, and H. Ludwig. Contract-driven Creation and Operation of Virtual Enterprises. *Computer Networks*, (37):111–136, 2001.
32. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web: A New Form of Web Content that is Meaninful to Computers will Unleash a Revolution of New Possibilities. *The Scientific American*, 284:34–43, 2001.
33. S. McIlraith, T. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems. Special Issue on the Semantic Web*, 16(2):46–53, March/April 2001.
34. D. Trastour, C. Bartolini, and J. González-Castillo. A Semantic Web Approach to Service Description for Matchmaking of Services. Technical Report HPL-2001-183, Hewlett-Packard, 2001.
35. O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Technical report, W3C Recommendation, 1999.
36. J. González-Castillo, D. Trastour, and C. Bartolini. Description Logics for Matchmaking of Services. Technical Report HPL-2001-265, Hewlett-Packard, 2001.
37. Joint US/EU Agent Markup Language Committee. DARPA Agent Markup Language. Technical report, US's DARPA Defense Advance Research Projects Agency and EU's IST Information Society Technologies, 2000. `http://www.daml.org`.

38. I. Horrocks. Description of the RACER System and its Applications. In *Proc. of the Int'l Workshop on Description Logics (DL'99)*, 1999.
39. V. Haarslev and R. Moller. Description of the RACER System and its Applications. In *Proc. of the Interntional Workshop on Description Logics (DL-2001)*, 2001.
40. A. Maurino, S. Modafferi, and B. Pernici. Reflective Architectures for Adaptive Information Systems. In *First International Conference on Service-Oriented Computing*, volume 2910 of *Lecture Notes in Computer Science*, pages 115–131, Trento, Italy, 2003. Springer Verlag.
41. D. Bianchini and V. De Antonellis. Ontology-based Integration for Sharing Knowledge over the Web. In *Proc. of the* 3^{rd} *International Workshop on Data Integration over the Web*, 2004.
42. V. Haarslev and R. Moller. Practical Reasoning in RACER with a Concrete Domain for Linear Inequations. In *Proc. of the Int'l Workshop on Description Logics (DL-2002)*, 2002.