Modeling Web Service Selection for Composition as a Distributed Constraint Optimization Problem (DCOP)*

Diego Anabalon[1,2], Martín Garriga[1,2], and Andres Flores[1,2]

[1] GIISCo Research Group, Facultad de Informática, Universidad Nacional del Comahue, Neuquén, Argentina.
[diego.anabalon,martin.garriga,andres.flores]@fi.uncoma.edu.ar
[2] CONICET (National Scientific and Technical Research Council), Argentina

Abstract. During development of a Service-oriented Application, some software pieces could be fulfilled by the connection to Web Services. A list of candidate Web Services could be obtained by making use of any service discovery registry, which are then selected and integrated into the application. However, when it comes to a distributed system, multiple functional and non-functional constraints arise from the interaction between several service requesters and providers, particularly when composing different services. To overcome with such constraints, in this work we propose to model service selection and composition scenarios as Distributed Constraints Optimization Problems (DCOP). We propose different modeling approaches and develop representative examples to be solved through different DCOP algorithms. Also, we analyze the impact of possible extensions to the model in the computability of the problem.

Keywords: Web Services, Service Selection, Service Composition, DCOP, DCSP, Constraints Optimization

1 Introduction

Service Oriented Computing (SOC) is a computing paradigm whose main objective is the development of distributed applications in heterogeneous environments, which are built by assembling or composing existing functionality called *service*. Services are published through a network and accessed by specific protocols [18, 12]. A Service-oriented Architecture (SOA) is composed of three actors: a provider, a consumer and a service registry. The registry is used by the provider to publish the description of their services, and also for consumers looking for services that meet their needs. Once a service is selected, it will be invoked from the client application [24]. In general, service-oriented applications are implemented using Web Services technology. A Web Service is a program with a well-defined interface which can be localized, published and invoked using the standard Web infrastructure [17]. SOA provides a promising solution for the seamless integration of business applications to create new value-added services, i.e., composite services [8]. Practice witnesses a growing interest in the ad-hoc service

* This work is supported by projects: ANPCyT–PICT 2012-0045 and UNCo – Service-oriented Reuse (04-F001).

composition in the areas of supply chain management, accounting, finances, eScience as well as in multimedia applications.

However, with the growing number of alternative Web Services that provide similar functionality, the composition issue becomes a decision problem on the selection of component services with regard to functional and non-functional requirements and constraints [2]. This is particularly true in the context of a distributed system [10], where these constraints arise from the interaction between several service requesters (probably competing for various services) and providers (probably competing for the market share). Additionally, developers have to manually search for suitable services in Web registries showing poorly relevant information. Even with a wieldy candidates list, a developer must be skillful enough to determine the most appropriate service for the consumer application. All of this implies a large effort to manage the Web Service lifecycle, i.e., service discovery, selection, integration and composition.

Meanwhile, Distributed Constraint Optimization Problems (DCOP) is a promising approach for modeling distributed reasoning tasks that arise in multiagent systems. Multiagent systems are composed of multiple, autonomous and interacting agents [25]. DCOP generalizes Constraint Satisfaction Problems (CSP), whose states and goals conform to a standard, structured, and very simple representation. Search algorithms can take advantage of the structure of states and use general-purpose rather than problem-specific heuristics to enable the solution of large problems. Perhaps most importantly, the standard representation of the goal test reveals the structure of the problem itself [23].

In previous work [11, 6], we defined an approach for service selection, based in an interface compatibility assessment of the candidate Web Services and the (potentially partial) specification of the required functionality. In this work, we propose to apply DCOP as an approach to model service selection and composition problems to be solved in a decentralized way. We developed representative scenarios where distributed, multiagent service requesters have to coordinate to select the best candidate services without any centralized control. Also, we analyze the impact of possible extensions to the model in the computability of the problem.

The rest of the paper is organized as follows. Section 2 briefly introduces DCOP and Service Selection and Composition basics. Section 4 describes our proposal for Service Selection and Composition modeled as a DCOP. Conclusions and future work are presented afterwards.

2 Background

2.1 Distributed Constraints Optimization (DCOP)

Several researchers have proposed the Distributed Constraint Optimization Problem (DCOP) for modeling a wide variety of multiagent coordination problems such as distributed planning, distributed scheduling, distributed resource allocation and others [27]. Multiagent systems are composed of multiple interacting agents, with two important capabilities [25]. First, they are at least to some extent capable of *autonomous action* – i.e., deciding for themselves what they need to do in order to satisfy their

design objectives. Second, they are capable of interacting with other agents, not only by exchanging data but also by coordinating, cooperating and negotiating with each other. DCOP provides a useful framework for investigating how these agents can coordinate their decision-making in a wide variety of domains. A DCOP includes a set of variables, each variable is assigned to an agent who has control of its value, and agents must coordinate their choice of values so that a global objective function – the *utility function* – is optimized. The global objective function is modeled as a set of *constraints*, and each agent knows about the constraints in which its variables are involved. DCOP significantly generalizes the Distributed Constraint Satisfaction Problem (DisCSP) framework [26] in which problem solutions are characterized with a designation of "satisfactory or unsatisfactory", according to degrees of quality or cost [16].

Formally, a Distributed Constraint Optimization Problem (DCOP) consists of n variables $V = x_1, x_2, ..., x_n$, each one assigned to an agent, where the values of the variables are taken from finite, discrete domains $D_1, D_2, ..., D_n$, respectively. Only the agent who is assigned a variable has control of its value and knowledge of its domain. The goal for the agents is to choose values for its variables such that a given global objective function is maximized (or minimized) [16].

The objective function is described as the summatory over a set of utility (cost) functions. A cost function for a pair of variables x_i, x_j is defined as $f_{ij} : D_i \times D_j \rightarrow N$. The utility (cost) functions in DCOP are the analogue of constraints from DisCSP and are sometimes referred to as "valued" or "soft" constraints. For convenience in this paper, we will refer to these functions simply as constraints. The objective is to find an assignment A of values to variables such that the aggregated utility (cost) function F is maximized (or minimized).

2.2 Service Selection through Interface Compatibility Assessment

In previous work [6, 11], we defined an approach for service selection based in an Interface Compatibility Assessment. This procedure is focused in structural and semantic aspects from candidate services. Given a required operation op_R and an operation op_S from a candidate service, similarity cases are typified through the structural/semantic similarity degree of the following elements: Return type [`R`], Exceptions [`E`], Operation Name [`N`], and Parameters [`P`]. Structural aspects involve data types equivalence (mainly subtyping), while semantic aspects involve analysing identifiers similarity by using WordNet as a semantic basis [15, 9].

The structural assessment is based in structural conditions for data type equivalence, involving the subtyping relationship. It is expected that types on operations from a candidate service have at least as much precision as types on operations from a required functionality. The `String` type is a special case, which is considered as a wildcard supertype since it is generally used in practice by programmers to allocate different kinds of data [19]. Complex types and ad-hoc exceptions imply a special treatment in which the comprising fields must be equivalent one-to-one with fields from a counterpart complex type.

The semantic assessment compares terms and identifiers from service interfaces – e.g., operation and parameter names, or complex types. We adopted the lightweight semantic basis WordNet, a widely accepted lexical database for the English language

that is structured as a lexical tree. WordNet groups terms in *synsets* (synonym sets) that represent the same lexical concept. Several relations connect different synsets, such as hypo/hyperonymy, and antonymy. To determine the semantic similarity between two identifiers, two terms lists are used as input [6]. First, a term separation step (considering typical naming conventions) extracts terms from identifiers. Then, a stopwords removal step filters meaningless words (articles, pronouns, prepositions), and a semantic stemming step reduces derived words to their base or root form. Finally, the identifiers compatibility is calculated accounting the number of exact (identical) terms between both terms lists; number of synonyms (words with the same meaning); number of hyperonyms (parents); and number of hyponyms (children).

Adaptability Value The structural and semantic assessment of interfaces outlines the adaptation effort of integrating a candidate service into a client application. Hence, an appraisal value named *adaptability value* (Formula 1) was defined to synthesize the achieved compatibility between compared interfaces. For details of this procedure we refer the reader to [11, 6].

$$adapVal = \frac{\sum_{i=1}^{N}(Max(AdapMap(op_{Ri}, op_{Sj})))}{N} \quad (1)$$

Where N is the size of the required interface, and $AdapMap$ is the best value among the possible mappings for a required operation op_{Ri} with regard to the candidate operations op_{Sj}, calculated according to Formula 2.

$$adapMap(op, op_S) = R + E + N + P \quad (2)$$

Where `R`, `E`, `N`, `P` are the equivalence values between op_R and op_S for return type, exceptions, operation names and parameters respectively.

3 Related Work

Artificial Intelligence has contributed significantly to the Web Services field, either in the form of planning [4, 21], case based reasoning [13, 7], or constraints satisfaction/optimization [22, 1, 28, 14].

A foundational work of DCOP/CSP applied to service selection and composition is [1]. This work presents a constraint driven Web service composition tool, which allows the process designers to bind Web services to an abstract process, based on business and process constraints and generate an executable process. It uses a multiphase approach for constraint analysis. This work was done as part of the METEOR-S (Managing End-To-End OpeRations for Services) framework. This framework manages the complete lifecycle of semantic Web processes, which represent complex interactions between semantic Web services.

Later, the work in [14] presents a consistency-based service composition approach, where composition problems are modeled in a generative constraint-based formalism. This approach is suitable for dynamic composition scenarios, where pre-planning or prediction of the number of required services is not possible. Also, the work presents the underlying domain-independent algorithm to find valid compositions, and shows that it scales to non-trivial problems.

More recently, the work in [28] presents a constraint satisfaction method for configuring (adapting) non-local service interfaces. This interface configuration approach for distributed components assumes closed-source, black-box services (i.e., the behavioural protocol is unknown). Thus, authors introduce a generic Message Definition Language, which can extend the existing interfaces description languages, such as WSDL, with support of subtyping, inheritance and polymorphism. Based in such language, an algorithm solves the interface reconciliation (i.e., adaptation) problem using constraint satisfaction.

Finally, the approach in [22] presents a constraint-driven dynamic RESTful service composition. RESTful services follow a resource-centric approach, where resource representations are accessed through standardized HTTP operations (GET, PUT, POST and DELETE). Automatic RESTful composition is still unexplored as compared to SOAP/WSDL-based services. Thus, the work proposes a goal-based constraint-driven composition model, with automatic workflow generation, with support for nested composition – i.e., where a dynamic composition is created from other compositions.

4 Service Selection and Composition with DCOP

In this section, we detail our proposal of applying DCOP to model service selection and composition problems. Also, we analyze the impact of possible extensions to the model in the computability of the problem. We assume that the service selection and composition scenarios to be modeled present certain challenging characteristics. First, service requesters define complex functionality – i.e., they require at least two functional categories. Also, these service requesters have to coordinate among them to select the better candidate services in a decentralized way – i.e., as a choreography. Choreography tracks the message interchange among multiple parties and sources from a third-party point of view, rather than a specific business process that a single party executes [20]. Finally, we assume the conversation and coordination among service requesters and providers without any centralized control.

In the context of this work, each service requester defines required interfaces for different functionalities. Also, several service providers may exist for a given functionality, providing different interfaces. Thus, the utility function to be maximized is the Adaptability Value (Formula 1) which accounts the difference between a requested functionality (expressed through a required interface) and a candidate service interface. Initially, we will assume that two service requesters are not allowed to select the same service provider for a given functionality – for the sake of service availability and simplicity of the model. Then we propose extensions to this simple model to capture more realistic scenarios.

Let us consider D the adaptability value for each pair of interfaces in the form (required, candidate). The value D represents the similarity between a required interface and the interface of a candidate service. Then, the local utility function F is calculated in each service requester as the sum of the D values for their requested functionalities. The global utility function UF to be maximized is the sum of all F values in each service requester. Thus,

$$local\,utility: F_{Ri} = \sum D_{Ri} \;\Rightarrow\; global\,utility: UF = \sum F_{Ri}$$

4.1 Proof-of-Concept

To illustrate our proposal, we introduce a simple proof-of-concept example, consisting in four service requesters R_1, R_2, R_3, R_4 which require different composite functionalities from three functional categories: *Chat*, *Temperature Conversion* and *Calculator*. Different service providers are listed for each functionality and category, as shown in Table 1. Finally, the complex required functionalities are defined as follows:

- R1 = {Chat, Temperature Conversion}
- R2 = {Chat, Calculator}
- R3 = {Temperature Conversion, Calculator}
- R4 = {Temperature Conversion}

The required interfaces are assumed to be the identical for each functional category and requester – e.g., the *Calculator* functionality required by R3 is identical to the *Calculator* functionality required by R2.

Table 1: Service providers by functional category for the example

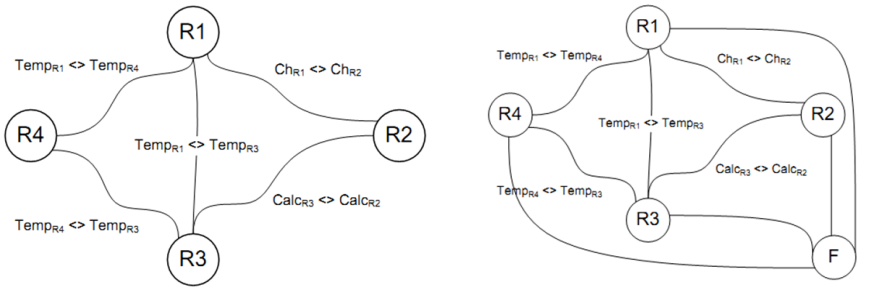
Functional Category	Service Name	URI
Chat (instant messaging)	OMS*	`www.nims.nl/soap/oms.wsdl`
	OMS2_Simple	`www.nims.nl/soap/oms2simple.wsdl`
	OMS2	`www.nims.nl/soap/oms2.wsdl`
Temperature Conversion	TConversions	`webservices.daehosting.com/services/TemperatureConversions.wso?`
	TempConvServ	`www.w3schools.com/xml/tempconvert.asmx?`
	CelsFar	`http://www.elguille.info/Net/WebServices/CelsiusFahrenheit.asmx?`
Calculator	CalcServ	`http://soatest.parasoft.com/axis/calculator.wsdl`
	SimpleCalc	Local Sample Repository

*Online Messenger Service

4.2 Simple Model – Modeling Service Requesters

Considering the scenario presented in Section 4.1, the restrictions graph for this simple model is shown in Figure 1a, where:

- requesters are represented as nodes,
- functional requirements are represented as variables, and
- providers are represented as the domain values for these variables.



(a) Restrictions Graph for the example scenario (b) Restrictions Graph with Fake Node

Fig. 1: Restriction Graphs for the example scenario (simple model)

Restrictions (e.g., $Temp_{R1} \neq Temp_{R4}$) represent, for this particular domain, the assumption "two service requesters are not allowed to simultaneously select the same service provider for a given functionality". In this model, local utility values are also represented as constraints, particularly unary constraints – i.e., constraints that only involve the agent that chooses that value. However, as the algorithms accepts only binary restrictions, a fake node F is introduced to represent unary restrictions as binary ones, as shown in Figure 1b. Agent in fake node F owns a variable but never chooses its value over the "real" domain agents.

For the example scenario, the utility values between each pair of required and candidate service interfaces are shown in Table 2, where the higher utility values are better. Parenthesized values in the "Provider" column correspond to the codification of each provider as a value in the domain of the variables. For simplicity, utility values were rounded to the nearest integer.

This model was used as input for two DCOP solving algorithms: the original ADOPT (Asynchronous Distributed Constraint Optimization with Quality Guarantee) algorithm [16], and the MCAdopt (Multiple Constrained Adopt) algorithm [5], which includes a graphic interface showing the restrictions graph and allows multiple, simultaneous constraints.

Table 2: Utility Values (Adaptability Values) for Providers by Functional Category

Functional Category	Service Provider (domain value)	Utility Value
Chat (instant messaging)	OMS (0)	13
	OMS2 (1)	06
	OMS2_Simple (2)	11
Temperature Conversion	TConversions (0)	14
	TempConvServ (1)	12
	CelsFar (2)	10
Calculator	CalcServ (0)	14
	SimpleCalc (1)	16

Results The output corresponding to the ADOPT algorithm is shown in Listing 1. Variables 1 to 7 represent functional requirements of service requesters. Variable 0 represents the fake node. The mapping of Values to service providers are codified in Table 2. According to such codification, this solution is optimal as it maximizes the utility function when selecting service providers, and complies with the defined constraints (each provider can be assigned to only one requester). Broken constraints indicate when a provider assignment is not optimal but is the best one available (since the optimal provider was assigned to another agent).

Listing 1.1: Execution results for ADOPT algorithm in the simple scenario

```
Solution --------------------
agent1-var1 = 2; agent1-var2 = 2
agent2-var3 = 0; agent2-var4 = 1
agent3-var5 = 0; agent3-var6 = 0
agent4-var7 = 1
agent5-var0 = 0
quality: 90
broken constraints:
(agent5-var0, agent1-var1)(agent5-var0, agent1-var2)
(agent5-var0, agent2-var3)(agent5-var0, agent3-var5)
(agent5-var0, agent3-var6)(agent5-var0, agent4-var7)
```

For the MCAdopt algorithm, Figure 2 depicts the graphical output of the algorithm, with the restrictions graph and the assignment results. Table 3 summarizes the results for both algorithms, with the utility of assigned providers for each requested functionality and total utility of both solutions. It is noticeable that both algorithms provide different solutions, albeit optimal ones, as both solutions maximize the utility function.

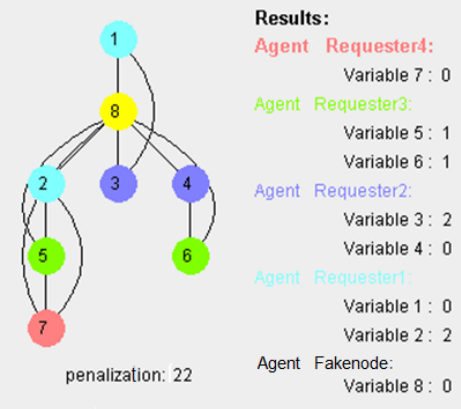


Fig. 2: Restrictions Graph for the MCAdopt algorithm

Table 3: Comparative Results with ADOPT and MCAdopt algorithms

Agent	Variable (category)	ADOPT value (provider)	Utility	MCAdopt value (provider)	Utility
Requester1	Var1 (Chat)	2 (Oms2Simple)	11	0 (Oms)	13
	Var2 (Temperature)	2 (CelsFar)	10	2 (CelsFar)	10
Requester2	Var3 (Chat)	0 (Oms)	13	2 (Oms2Simple)	11
	Var4 (Calculator)	1 (SimpleCalc)	16	0 (CalcServ)	14
Requester3	Var5 (Temperature)	0 (TConversions)	14	1 (TempConvServ)	12
	Var6 (Calculator)	0 (CalcServ)	14	1 (SimpleCalc)	16
Requester4	Var7 (Temperature)	1 (TempConvServ)	12	0 (TConversions)	14
FakeNode	Var8	-	-	-	-
Total Cost			90		90

Another dimension to compare both algorithms is the average execution time. We run the experiment 10 times for each algorithm: the average execution time for the ADOPT algorithm was 2 seconds, while the average execution time for the MCAdopt algorithm was 14 seconds.

4.3 Realistic Model

A possible extension for the simple model of the previous section arises from considering the following statement: “A provider could manage several requests simultaneously”. As the base case we consider up to two simultaneous requests.

Modeling n-ary Restrictions This model is similar to the simple model in the sense that service requesters are nodes and service providers are the domain values for the variables. Thus, the assumption “A provider could manage several requests simultaneously” can be stated from a requester’s point of view as “Among all the requesters of a given functional category, up to two can select the same service provider”. In a scenario with n requesters for a certain functional category, the aforementioned assumption is modeled as a n-ary restriction among all requesters of such functional category.

To illustrate this situation, we consider a subset of the example presented in Section 4.1, for the sake of simplicity. Let us consider $R1temp$, $R3temp$ and $R4temp$ the variables for requesters R1, R3 and R4 respectively, while the values for these variables represent the selection of a provider for the Temperature category. Figure 3a depicts the ternary restriction *2equals* (a particular case of the n-ary restriction) that allows up to two variables to select the same value simultaneously.

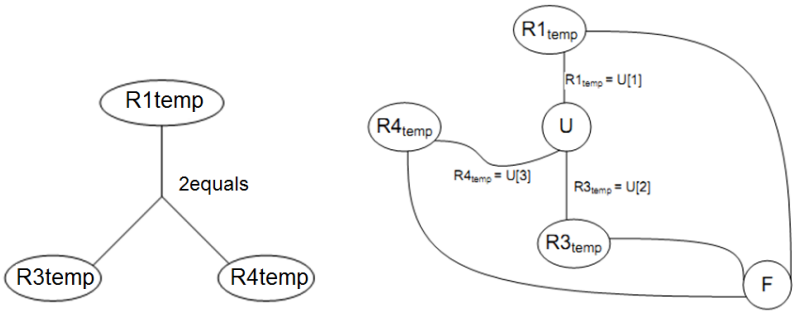
As stated earlier, current DCOP algorithms only accept as input binary restrictions. Thus, we follow the guidelines given in [3], where an n-ary constraint is represented as binary constraints with regard to a new encapsulated variable U, which encapsulates the variables involved in the n-ary constraint. The domain of the variable U is the cartesian product of each encapsulated variable, restricted to the valid combinations of values for such variables according to the defined constraint.

In our example, the variable U will have a domain in the form of triples (r_1, r_3, r_4) where r_1, r_3 and r_4 are the values assigned to $R1temp$, $R3temp$ and $R4temp$ respectively, that satisfy the defined constraint (up to two variables can assume the same value simultaneously). Then, if the domain for variables R_iTemp is $0, 1, 2$, the domain of the variable U will be:

$$\{(0,0,1);(0,0,2);(0,1,0);(0,1,1);(0,1,2)....(2,2,1)\}$$

Finally, the constraint between each original variable and the variable U states that each value in the triple equals to the value of the corresponding variable – e.g., if $U = \{0,0,1\}$ then $R1_{temp} = 0$, $R2_{temp} = 0$, $R3_{temp} = 1$. Also, this model includes a fake node F (as described in Section 4.2) to describe unary restrictions – as shown in Figure 3b.

This example run successfully with the original ADOPT algorithm. Results are shown in Listing 2, where var-1, var-2 and var-3 corresponds to $R1temp$, $R3temp$ and $R4temp$ respectively; $var0$ is the fake node and $var4$ is the encapsulated variable U. Domain values are {0,1,2} with utility value {14,12,10}, corresponding to service providers *TConversions*, *TempConv* and *CelsFar* respectively. Thus, the quality of the solution is 40, being the optimal solution as two service requesters select the provider TConversions (0) with maximum utility 14, and the remaining requester selects the provider TempConv (1) with utility 12.



(a) Schematic Ternary Restriction (b) Graph with Fake Node F and Encapsulated Variable U

Fig. 3: Modelling n-ary Restrictions

Listing 1.2: Execution results for ADOPT algorithm with n-ary constraints

```
Solution -------------
agent1-var0 = 0; agent1-var1 = 0; agent1-var2 = 0
agent1-var3 = 1; agent1-var4 = 0
quality: 40; broken constraints: none
```

5 Conclusion

This work proposed different models to represent service selection and composition as a Distributed Constraint Optimization Problem (DCOP). This type of model is particularly suitable in decentralized contexts, where conversational agents owned by service requester have to coordinate to select the most suitable candidate services among a limited subset of providers. We have shown the feasibility of this approach with the *adaptability value* as utility function, which compares required interfaces and candidate service interfaces. Through a simple example, we reached optimal solutions using alternative models and algorithms, namely ADOPT and MCAdopt algorithms. This implies that service requesters could coordinate to maximize the assignment of optimal providers without any centralized control. As future work, we are planning to address the scalability of the model to support more realistic scenarios.

Limitations QoS features should be considered for the calculation of the utility function. However, the models can be easily generalized for any QoS metric. Besides, the models are not scalable in a straightforward way. A simple assumption such as "a provider may handle several requests at the same time" implies an exponential grow of nodes and restrictions. Another limitation of this solution is the inconvenience of adding new service providers for a given functional category on-the-fly, as this implies re-calculating variable domains.

References

1. R. Aggarwal, Kunal Verma, J. Miller, and W. Milnor. Constraint driven web service composition in meteor-s. In *Services Computing, 2004. (SCC 2004). Proceedings. 2004 IEEE International Conference on*, pages 23–30, Sept 2004.
2. M. Alrifai, D. Skoutas, and T. Risse. Selecting skyline services for qos-based web service composition. In *Proceedings of the 19th international conference on World wide web*, pages 11–20. ACM, 2010.
3. Roman Barták. Principles of constraint processing. *Artificial Intelligence for Advanced Problem Solving Techniques*, pages 63–106, 2008.
4. Piergiorgio Bertoli, Marco Pistore, and Paolo Traverso. Automated composition of web services via planning in asynchronous domains. *Artificial Intelligence*, 174(3):316–361, 2010.
5. Emma Bowring, Milind Tambe, and Makoto Yokoo. Multiply-constrained distributed constraint optimization. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1413–1420. ACM, 2006.
6. A. De Renzis, M. Garriga, A. Flores, A. Zunino, and A. Cechich. Semantic-structural assessment scheme for integrability in service-oriented applications. In *Latin-american Symposium of Enterprise Computing, held during CLEI'2014*, September 2014.
7. Alan De Renzis, Martin Garriga, Andres Flores, Alejandra Cechich, and Alejandro Zunino. Case-based reasoning for web service discovery and selection. *Electronic Notes in Theoretical Computer Science*, 321:89–112, 2016.
8. J. Erickson and K. Siau. Web Service, Service-Oriented Computing, and Service-Oriented Architecture: Separating hype from reality. *Journal of BD Management*, 19(3):42–54, 2008.
9. Mark Alan Finlayson. Java libraries for accessing the princeton wordnet: Comparison and evaluation. In *Proceedings of the 7th Global Wordnet Conference, Tartu, Estonia*, 2014.

10. I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, 35(6):37–46, Jun 2002.
11. M. Garriga, A. Flores, C. Mateos, A. Zunino, and A. Cechich. Service selection based on a practical interface assessment scheme. *International Journal of Web and Grid Services*, 9(4):369–393, October 2013.
12. M. Huhns and M. Singh. Service-Oriented Computing: Key Concepts and Principles. *IEEE Internet Computing*, 9(1):75–81, January-February 2005.
13. Soufiene Lajmi, Chirine Ghedira, and Khaled Ghedira. Cbr method for web service composition. In *Advanced Internet Based Systems and Applications*, pages 314–326. Springer, 2009.
14. Wolfgang Mayer, Rajesh Thiagarajan, and Markus Stumptner. Service composition as generative constraint satisfaction. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 888–895. IEEE, 2009.
15. George Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller. Introduction to Wordnet: An On-line Lexical Database. *International Journal of Lexicography*, 3(4):235–244, 1990.
16. Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal*, 161(1):149–180, 2005.
17. Ramesh Nagappan, Robert Skoczylas, and Rima Patel Sriganesh. *Developing Java web services: architecting and developing secure web services using Java*. John Wiley & Sons, 2003.
18. M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 40(11):38–45, November 2007.
19. J. Pasley. Avoid XML Schema Wildcards For Web Service Interfaces. *IEEE Internet Computing*, 10(3):72–79, 2006.
20. C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, 2003.
21. J. Rao and X. Su. A survey of automated web service composition methods. In *International Workshop on Semantic Web Services and Web Process Composition (SWSWPC)*, pages 43–54, 2004.
22. Digvijasinh Rathod, Satyen Parikh, and M. S. Dahiya. *Proceedings of International Conference on ICT for Sustainable Development: ICT4SD 2015 Volume 1*, chapter Goal-Based Constraint Driven Dynamic RESTful Web Service Composition Using AI Techniques, pages 745–754. Springer Singapore, Singapore, 2016.
23. Stuart Russell and Peter Norvig. *Artificial Intelligence: A modern approach*. Prentice-Hall EUA, Egnlewood Cliffs EUA, 1995.
24. S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005.
25. Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.
26. Makoto Yokoo. *Distributed constraint satisfaction: foundations of cooperation in multi-agent systems*. Springer Science & Business Media, 2012.
27. Makoto Yokoo and Katsutoshi Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2):185–207, 2000.
28. Pavel Zaichenkov, Olga Tveretina, and Alex Shafarenko. A constraint satisfaction method for configuring non-local service interfaces. *CoRR*, abs/1601.03370, 2016.