# The C-Cube Framework: Developing Autonomic Applications through Web Services

Gerardo Canfora[1], Piero Corte[2], Antonio De Nigro[2],
Debora Desideri[2], Massimiliano Di Penta[1],
Raffaele Esposito[1], Amedeo Falanga[1], Gloria Renna[1], Rita Scognamiglio[1],
Francesco Torelli[2], Maria Luisa Villani[1], Paolo Zampognaro[2]

[1] RCOST — Research Centre on Software Technology
Department of Engineering - University of Sannio
Viale Traiano - 82100 Benevento, Italy
{canfora, dipenta, amedeo.falanga, gloria.renna, r.esposito, ritasco, villani}@unisannio.it

[2] Engineering SpA - Research & Development Lab

Via San Martino della Battaglia, 56 - 00185 Roma, Italy
{piero.corte, antonio.denigro, debora.desideri, francesco.torelli, paolo.zampognaro}@eng.it

## ABSTRACT

Web services constitute a promising technology to support autonomic computing. Automatic discovery of new services, their composition and binding based on Quality of Service (QoS) are just some of the most promising features that can be provided using web services. In other words, a service oriented system is able to automatically discover, bind, and use, at run time, the services that, among those available, offer a given piece of functionality with a QoS compatible with the system non–functional requirements.

This paper describes our work–in–progress related to the development of an electronic marketplace, named $C^3$ (Creation, Certification and Classification of Services) to enable the publication, semantic discovery, service buying, SLA negotiation and QoS-aware composition and replanning. The marketplace is mainly targeted to corporate intranets, although its technologies and approaches can be easily exported to a wider scenario.

**Keywords:** Service-oriented systems, automatic service discovery, run–time binding, automatic service negotiation, service replanning

## 1. INTRODUCTION

Web services are rapidly changing the landscape of software engineering, introducing interesting challenges and new, promising scenarios. A relevant piece of this technology is represented by semantic web services, that provide the capability of automatically binding, even at run–time, a service specification with any service capable to realize a given piece of functionality. In other words, semantic web services allow us to (semantically) specify in our system (or service) the requested feature (e.g., booking an hotel, getting the temperature of a given city). Then, using a proper mechanism, it is possible to automatically select and bind the services that fulfill the query.

The set of retrieved services (we refer to them as *concrete services*) will be all functionally equivalent with respect to our query (we refer to it as an *abstract service*). However, they could exhibit different Quality of Service (QoS) attributes: there may be faster services, cheaper ones, or a compromise. The choice of the service to be invoked is therefore based on some QoS constraint or even optimization criteria. Moreover, when selecting the service, the SLA is automatically determined applying a negotiation protocol to provider offerings and user QoS constraints.

A complex service oriented system, or even a composite service (i.e., a service obtained by orchestrating more services using a language like BPEL4WS [2]) can contain one or more abstract services. It is therefore necessary to determine the set of bindings that meet some constraints (often established by means of a Service Level Agreement (SLA)) and optimize some criteria (e.g., the cost). Finally, at run–time, a service may not be available, or the overall QoS may deviate from the estimated one, thus leading to some constraint violation. This makes necessary some form of run–time replanning of bindings between abstract and concrete services.

All of the above described scenarios make web services a promising technology for building autonomic systems, in particular systems able to:

- discover services (or a composition of services) able to provide a specified piece of functionality needed for the system itself;
- select, among the available services those able to satisfy QoS requirements, possibly negotiating their SLA; and


DEAS 2005, May 21, 2005, St. Louis, Missouri, USA

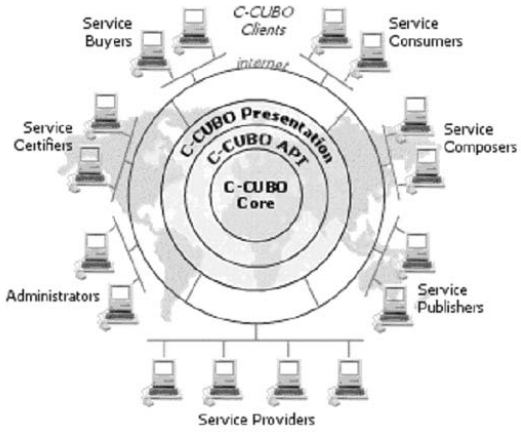


**Figure 1: $C^3$ Physical Architecture**

- replanning the binding between abstract and concrete services whenever needed.

The existing Service Oriented Architecture (SOA) [17] and means to describe service interfaces (WSDLs) [18] poses serious limits to that. It is therefore necessary to introduce mechanisms to enable late binding and replanning, and a means to semantically describe services and then discover them.

This paper describes a service marketplace, originally targeted to corporate intranets, but that can be possibly opened to a wider community. The marketplace, named $C^3$ (Creation, Classification and Certification of Services) permits the publication, certification, discovering, buying of services. The paper is organized as follows. Section 2 briefly introduces the architecture and the features of $C^3$. Then, Section 3 and Section 4 focus on the feature enabling autonomic computing, namely semantic discovery, SLA negotiation and, finally, QoS aware composition and replanning. Section 5 concludes.

## 2. ARCHITECTURE

This section describes the architecture of $C^3$. As shown in Figure 1, $C^3$ is basically a distributed system composed of several nodes communicating via SOAP through the Internet.

From a functional perspective, the $C^3$ system is composed of three software layers, *Core*, *API* and *Presentation*, depicted in Figure 2 and described in the following subsections.

### 2.1 Core

The *Core* layer realizes the basic set of features of the $C^3$ marketplace, as described below. Clearly, the *Core* layer is built upon a data layer not shown in this architecture for sake of simplicity.

#### *2.1.1 Administration*

The administration module provides all the features concerning the system configuration, such as handling users, privileges, contracts between users and the system, defining monitoring policies and performing backup operations.

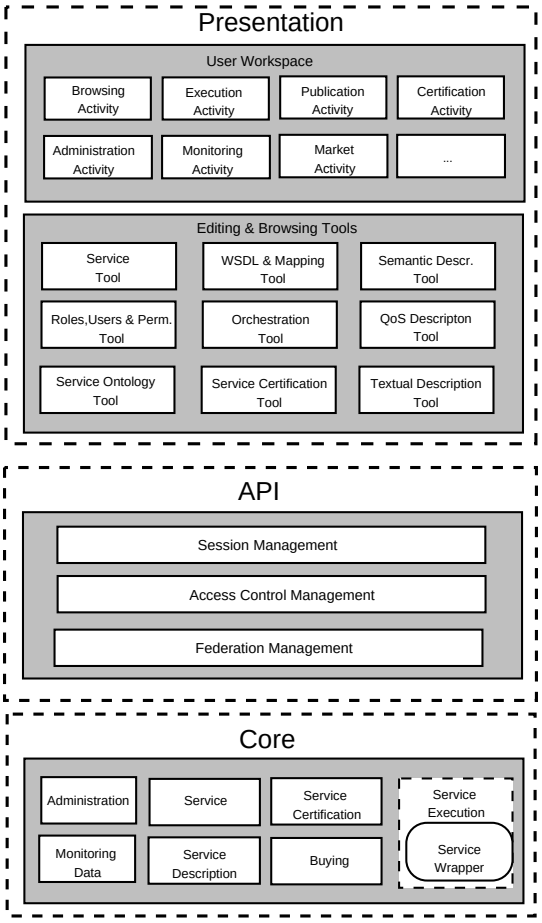


**Figure 2: $C^3$ Logical Architecture**

#### *2.1.2 Service Description*

This module manages the service descriptions. In particular, it handles the following description categories:

- *natural language descriptions*, used to early querying the service repository (i.e., searching for services using free-text, requirements or use cases as queries);
- *functional descriptions*, that semantically describe the services. As detailed in Section 3, these descriptions enable the run–time automatic service discovery.
- *QoS descriptions*, that are: i) related to simple services; ii) used to compute the QoS of composite services; and iii) used to handle additional, user–defined QoS attributes.

Also, the service description module handles WSDLs associated with both simple and composite services, workflow descriptions of composite services, and service SLA Templates [12].

#### *2.1.3 Service*

This module contains all the core features to support service publishing, maintenance and removal from the marketplace. Also, it provides a support for the service discovery features.

### 2.1.4 Service Certification

This module handles *service certificates*, declaring the degree of trust for service QoS and functional behavior. In particular, the module allows the users to request QoS certifications and the certifiers to grant or revoke certificates.

### 2.1.5 Buying

This module manages the service acquisition process, and the SLA negotiation between a service buyer and a service provider. As described in Section 4.1, the service provider offers several possible SLA Template for a given service. At buying time, the buyer may instantiate his/her own SLA from one of the possible templates, or even negotiate a customized SLA.

### 2.1.6 Monitoring

This module provides monitoring capabilities. In particular, it collects service monitoring data (QoS values, paths followed inside composite services, statistics on bindings) that can be used to enable QoS-aware composition and replanning (see Section 4), but also to continuously check the consistency and correctness of the different system activities (discovery, composition, binding, etc.).

Also, the monitoring module keeps track of user accesses to the system and to services. The latter information can be used to compute the amount a user should pay for to use some services.

### 2.1.7 Service Execution

Service execution is managed and controlled through special services, or wrappers, with the aim of:

- monitoring each service execution. In particular, the QoS values of some attributes, like response time or availability, for that execution of the service are updated.
- binding between abstract and concrete services. For each abstract service, the wrapper service allows to apply the automatic discovery mechanisms, described in Section 3, to determine the list of services matching that description and that respect some quality constraints. Whenever invoked, for example in the context of a composition, the wrapper service will forward the invocation message to the chosen service.
- realizing the QoS-aware composition and replanning described in Section 4.

The main advantage of using wrapper services for compositions is that there is no need to perform any change in the BPEL4WS language nor in the workflow engine to support the use of abstract services in the workflow and to extend it with monitoring facilities.

## 2.2 API

The *API* layer provides the system business logic to the upper *Presentation* layer. Therefore, all the $C^3$ features are available as services accessible through SOAP, thus they can be potentially used by external systems without relying on the *Presentation* layer.

In addition to that, the *API* layer guarantees some additional features, horizontally to all the specific-purpose modules. In particular, the following features are provided:

- *Session management:* this module permits the concurrent access of users to the system. For each access, a user session is created. The session retains all the privileges the user has with respect to the system and to the published services.
- *Authentication:* the system provides support for two different levels of authentication: i) a *basic authentication* level, based on username and password and ii) a *strong authentication* level, based on digital certificates that are provided by a proper *Certification Authority* aiming to identify the user.
- *Authorization:* it handles the policies that grant or revoke a user's access to specific resources. The authorization module has been developed according to the RBAC [8] model.
- *Federation:* as stated above, the $C^3$ marketplace has been conceived as a network of distributed nodes communicating via the SOAP protocol. The federation module realizes the policies of task distribution and delegation to the different nodes. This aims to make the whole system both scalable and fault–tolerant.

## 2.3 Presentation

The *Presentation* layer provides a set of tools accessible to users to interact with the system. These tools are basically web–applications, and offer the set of features also exported as API by the underlying layer. In particular, the *Presentation* layer provides:

1. A set of *tools* that enable the editing and browsing of different service–related artifacts. For example, there exists a tool to browse and edit service semantic descriptions, a tool to edit and browse ontologies, a tool to design composite service orchestrations, a tool to describe a service QoS, etc. Figure 2 reports a complete list of the available tools. Clearly, more tools can be easily added to extend the system.
2. A user workspace, in which the user can access to specific *activities*. Basically, each activity defines an interaction scenario between the user and one or more tools to realize the service marketplace activities.

The remainder of this section briefly describes the main *activities* provided by the $C^3$ presentation layer.

### 2.3.1 Publication Activity

The publication activity supports the description and the publication of services, with the aim to make their discovery possible. As also mentioned in Section 2.1, each service needs to be annotated with three different types of description:

- *The Textual Description* consists of any textual document that can be attached to a service. These documents are indexed to support the service free-text discovery.
- *The Functional Description*, i.e., semantic annotation, based on ontologies, describes the service from a functional point–of–view (see Section 3 for further details).
- *The QoS Description*, useful to support discovery activities and, above all, to enable QoS–aware composition and binding (see Section 4).

Finally, the publication module permits to specify/change all the other information related to the service, namely:

- the WSDL associated with simple and composite services;
- workflow descriptions (orchestration) associated with composite services;
- SLA Templates;
- QoS certificates.

All the information tied to a service is summarized in Figure 3.

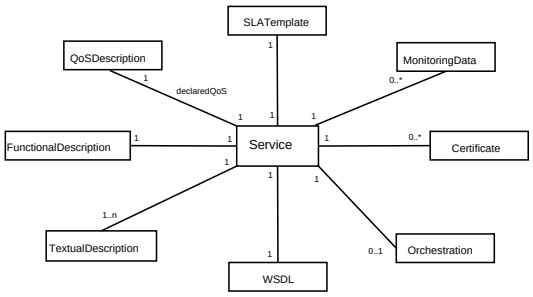


**Figure 3: Service information model**

### 2.3.2 Browsing Activities

The browsing activity offers various ways to search for a service, i.e.:

- textual search;
- semantic search based on precise queries (see Section 3);
- QoS search (i.e., search for services exhibiting QoS values in a given range); and
- a combination of the above.

### 2.3.3 Certification Activity

This activity allows the customers to ask for service certification, and the certifier to grant or revoke certificates.

### 2.3.4 Market Activity

This activity constitutes the way a buyer interacts with the $C^3$ marketplace to acquire services. Interestingly, a buyer can also supply descriptions of simple or composite services that have not been published yet. This may raise a sort of demand that some service provider can decide to fulfill. Finally, in the context of the market activity, the buyer negotiates the SLA with the service provider.

### 2.3.5 Execution Activity

As said in Section 2.1, the execution activity is supported by means of service wrappers. Despite that, the *Presentation* layer provides some web based facilities for invoking services (for testing purposes) during the discovery phase.

### 2.3.6 Monitoring Activity

The monitoring activity provides a user–interface support to easily access monitoring data, allowing the user to query the monitoring repository and compute some statistics on it.

### 2.3.7 Administration Activity

This activity basically constitutes a sort of front–end for the administration API. It has been conceived to easily manage the $C^3$ configuration.

## 3. SEMANTIC SERVICE DISCOVERY

As stated in the introduction, one of the most interesting features of $C^3$ is the possibility to automatically search for, bind to, and execute services. To this aim, it is necessary to unambiguously describe services, and carry out service discovery with a precision of 100%. This possibility can be supported by the new standards for the Semantic Web [1], and languages based on formal logic and with a precisely–defined semantic (e.g., OWL and OWL-S [7]). Amazingly, various projects (e.g., [10], [11], [13], [14], and [15]) seem to suggest that the adoption of these standards is not sufficient to achieve a 100% precision anyway.

Similarly, $C^3$ introduces a new discovery approach aiming to achieve a 100% precision in the search phase (unless of human mistakes in the services description or in the ontology definition). The semantic description of a service represents the functional aspects of the service expressed in a logical language and by means of formal ontologies.

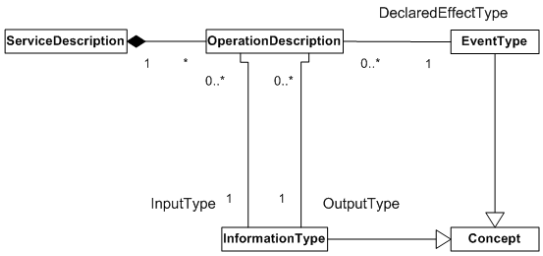


**Figure 4: $C^3$ Service semantic description**

As shown in Figure 4, each service operation is described in terms of types of information for each input and output. In particular, we distinguish between ontological categories that we call *conceptual types* (or *entity types*) whose instances represent entities of the application domain (e.g., "Person") and categories that we call *information types*. Instances of *information types* represent information. Intuitively, an *information type* can be thought of as a collection of attributes chosen from an ontology. For example, the *information type* `AnagraphicInformation` could be composed of the attributes `firstName`, `lastName` and `birthDate`. A *conceptual type* may correspond to different *information types*. For example, a person can be described through personal details, but also by information related to the person's health. Clearly, two services can be semantically equivalent, but they may differ in their implementation or in the data format adopted to represent the information. For example, a month may be represented, as a date input parameter, using an integer or strings like "Jan", "Feb", etc. Therefore, the $C^3$ discovery engine requires to specify the representation format to adopt for each service, in order to support the matching/integration of services adopting different representations of the same information.

The effect of an operation is modeled in terms of events ([6], [16]) produced after its execution. The pre and post conditions, used in other approaches for service description, constitute a special case of event category. The use of events ensure a better scalability than

the use of pre and post conditions. For instance, it may suffice to say that a service produce a *flightTicketBooking* (that is a primitive category of events). Clearly, writing the corresponding pre and post conditions may result as by far more difficult. The capability of publishing services is limited to the possibility of developing a complete ontology thoroughly describing the application domain. During the service publication, a service provider can, if necessary, update the ontology following a collaborative process that involves the ontology administrator.

The service semantic description is expressed through a formal language based on description logic [3], extended in order to distinguish between the concepts of *entity* and *information* associated with the *entity* itself. The language supports the distinction between subsumption (i.e., generalization between *conceptual types*) and specialization (i.e., generalization between *information types*). This distinction constitutes the basis for the $C^3$ service discovery algorithm.

The $C^3$ reasoning algorithms used to discover services constitute a further point of strength. Most of the recent systems aim to guarantee the completeness of the reasoning mechanisms, sacrificing in some measure the expressiveness of the description language. On the contrary, the $C^3$ approach tends to privilege the query expressiveness. While this could reduce the discovery algorithm recall, it does not constitute a serious problem. In fact, having a more expressive query mechanism permits to adequately describe services and to guarantee a precision of 100% (that is a necessary condition to enable the run–time service discovery and binding).

# 4. QOS-AWARE COMPOSITION AND RE-PLANNING

Automatic service discovery and late binding features require that the current SOA be also extended with QoS management mechanisms, that is, components to estimate, monitor and control the quality of the services being provided. In fact, the service selection is usually determined by some quality constraints that are part of the query by the user, such as a limit for the cost of the service or its security level. For composite services, their overall QoS needs to be computed from that of the component services, according to how they are orchestrated. Also, as the availability of services may change over time, and different services may exist with the same functionality (i.e. corresponding to the same abstract service), replanning mechanisms are used to reach and maintain the target QoS-level. Monitoring tools allow to control whether the actual QoS is compliant with what formalized in SLAs with the customers, so that risks of penalties may be prevented through recovery actions, e.g. a service substitution in case of non-availability, or service re-planning.

## 4.1 SLA negotiation

The QoS guarantees of the service provider to a buyer are formalized in a SLA document, as part of the contract. This document is usually created out of a schema, named SLA Template, that is submitted by the provider at service publication time. A SLA Template document contains a clear QoS description of the service offered and some customizable parts to be finalized after negotiation with potential buyers. A negotiation process with a customer would essentially lead to a compromise between the cost of the service and its quality. For example, higher costs may be asked for better performances or service availability.

To support SLA negotiation in $C^3$, a protocol has been defined, whose main steps are described below.

1. The SLA negotiation is usually opened by the service buyer, who expresses his/her QoS requests by providing QoS values (or ranges of values) to the negotiable attributes of the SLA Template.
2. The service provider may: i) either accept the buyer's proposal, and in this case a SLA is automatically created out of the customized SLA Template; ii) or propose a new SLA Template just for that customer on the basis of his/her indications.
3. If a new SLA Template is created, the buyer may: i) either accept; ii) or make a new proposal on that template, so that a new cycle of the process starts.
4. The negotiation process ends if an agreement is reached on some version of the SLA Template, and so a SLA is signed out of it, or no agreement is reached between the parts.

Clearly, different SLAs may co-exist with respect to the same service, and so they have to be managed singularly. Instead, in the context of a composite service, SLAs of the component services are, most probably, automatically created out of the respective SLA Templates without negotiation.

## 4.2 QoS-aware composition

The QoS-aware composition requires that the following two problems be solved: i) determine the QoS of a composite service as a function of the QoS of its components; and ii) determine the set of concrete services that maximize the QoS of the composite service.

To estimate the QoS of a composite service, we use the aggregation formulae proposed by Cardoso [5] for each pair QoS attribute/control statement (e.g., *sequence*, *switch*, *loop* or *flow*). Thus, the QoS is computed by recursively applying these formulae to compound nodes of the service workflow. The model uses stochastic information indicating the probability of transitions being fired at run-time, in the case of a *switch*, and estimations on the number of iterations for *loops*. This information may be initially set by the designer, wherever possible, and then periodically adjusted, based on data on previous executions stored in the monitoring database.

Determining the best concretization of a composite service is an optimization problem, aiming to i) maximize a fitness function of the available QoS attributes; and ii) meet the constraints specified for some of the attributes. For this, different strategies can be adopted, for example integer programming [19] or Genetic Algorithms (GA) [4]. Clearly, the fitness function may need to maximize some QoS attributes (e.g., reliability), while minimizing others (e.g., cost). When user–defined, domain–specific QoS attributes are used, the specification of the fitness function is left to the workflow designer.

## 4.3 Replanning

At run-time, the actual QoS values of the individual services may deviate from the estimations and the execution path in the workflow may not be the one foreseen. These changes could have a drastic impact on the expected overall QoS of the composite service. Thus, the risk of breaking SLAs and obtaining a poor QoS

could be avoided by re-planning the service bindings of the workflow part still to be executed. In fact, the new binding could compensate the observed workflow QoS loss, or simply lead to a suboptimal solution still within the SLAs. To this aim, we have defined a re-planning triggering algorithm (described in Canfora et.al [4]), to try to detect re-planning needs as soon as possible during execution. The idea is to re-estimate the workflow QoS whenever new information is available, e.g. the number of times a *loop* will be iterated, by observing the variables of the conditional statement, or the *case* followed after a decision point, or the QoS values measured when a service is invoked. In fact, changes of these parameters may cause a big deviation from the QoS initially predicted and/or lead to the non satisfaction of some global constraints. Whenever replanning is triggered, the slice of the workflow that still has to be executed must be computed, and the optimization problem be solved again on that slice. Once (acceptable) new bindings are identified, the service execution may restart.

## 5. CONCLUSION AND WORK IN PROGRESS

This paper has described the work–in–progress in the development of $C^3$, a service marketplace to enable the publication, discovery, certification, buying and execution of services. Such a marketplace aims to provide to corporate intranets, and possibly to a wider population, many features that will enable the building of autonomic systems. In particular, a system developed in this scenario will be able to discover, at run–time, the services realizing a given piece of functionality. This implies the development of a complex semantic service discovery mechanism, based on ontologies. Then, it will be possible to automatically select services offering the desired level of QoS, and even to negotiate them. For composite services, a mechanism based on Genetic Algorithms will automatically determine the best combination of service bindings. Finally, at run–time, a replanning mechanism adjusts the bindings whenever a service is not available or the QoS level deviates from the required one.

Work–in–progress is devoted to complete the development of the marketplace, refining at the same time the algorithms and approaches that constitute its main added value, possibly adopting some techniques inspired to autonomic computing [9].

## 6. ACKNOWLEDGMENTS

This work is partially supported by the project “C-CUBO” prot. n. 10941, co–founded by the Italian “Ministero dell’Instruzione, dell’Università, e della Ricerca”.

## 7. REFERENCES

[1] Semantic Web http://www.semanticweb.org/.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, S. T. D. Smith, I. Trickovic, and S. Weerawarana. Business process execution language for web services. *http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/*.

[3] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[4] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. A lightweight approach for QoS-aware service composition. In *Proc. 2nd International Conference on Service Oriented Computing (ICSOC’04) - short papers*, New York, USA, Nov. 2004. IBM Technical Report.

[5] J. Cardoso. *Quality of Service and Semantic Composition of Workflows*. PhD thesis, Univ. of Georgia, 2002.

[6] R. Casati and A. Varzi. Events. *The Stanford Encyclopedia of Philosophy*, 2002.

[7] M. B. David Martin, G. Denker, J. Hobbs, L. Kagal, O. Lassila, D. McDermott, S. McIlraith, M. Paolucci, B. Parsia, T. Payne, M. Sabou, E. Sirin, M. Solanki, N. Srinivasan, and K. Sycara. OWL-S: Semantic markup for web services.

[8] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.

[9] J. Kephart and D. Chess. The vision of autonomic computing. *IEEE Computer*, 2003.

[10] M. Klein and A. Bernstein. Towards high-precision service retrieval. *IEEE Internet Computing*, 8(1):30–36, January 2004.

[11] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *WWW ’03: Proceedings of the twelfth international conference on World Wide Web*, pages 331–339. ACM Press, 2003.

[12] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck. Web service level agreement (WSLA) language specification. *http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf*.

[13] M. Mecella, B. Pernici, and P. Craca. Compatibility of e -services in a cooperative multi-platform environment. In *TES*, pages 44–57, 2001.

[14] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *ISWC02*, volume 2348 of *Lecture Notes In Computer Science*, pages 333–347. Springer-Verlag, June 2002.

[15] A. A. Patil, S. A. Oundhakar, A. P. Sheth, and K. Verma. METEOR-S web service annotation framework. In *WWW ’04: Proceedings of the 13th international conference on World Wide Web*, pages 553–562. ACM Press, 2004.

[16] S. Russell and P. Norvig. *Artificial Intelligence: A modern approach (2nd edition)*. Prentice-Hall, Upper Saddle River, NJ, USA, 2003.

[17] W3C Working Group. Web services architecture. *http://www.w3.org/*.

[18] W3C Working Group. Web services description language (WSDL). *http://www.w3.org/*.

[19] 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), May 2004.