Agent-Based Semantic Service Discovery for Healthcare: An Organizational Approach

César Cáceres, Alberto Fernández, Sascha Ossowski, and Matteo Vasirani,
University Rey Juan Carlos

This semantic-service-discovery mechanism considers relevant parts of the organizational context in which e-health services are used, to improve a service-discovery system's usability in medical emergencies.

Information and communication technologies offer great potential for society to quickly adopt e-services for economic and social development. Healthcare activities based on these technologies (e-health) are probably the most prominent of these e-services. However, e-health is evolving into such entities as m-health (mobile) or u-health (ubiquitous), which focus on applications that provide healthcare to people anywhere, anytime using broadband and wireless mobile technologies.

To adapt the technologies to this new scenario, a service-oriented approach is gaining popularity. In this context, services are software entities that can be described, published, discovered, orchestrated, and invoked by other software entities. A service-oriented approach to e-health must consider semantics, because in healthcare, every description must have a unique, clear meaning. So, defining and maintaining expressive ontologies for e-health are crucial.

Healthcare applications are usually based on interactions between people playing different roles in diverse organizational contexts. Agent technology provides the means to capture this structure because it proposes an interaction-oriented way of designing open software systems.[1] Current agent-oriented methodologies support this claim by treating organizational abstractions as first-class citizens in the design process.[2,3] So, a promising approach would be to combine Semantic Web services[4] and agent technologies[5,6] for advanced e-health applications.

Here, we concentrate on dynamic healthcare service discovery in open multiagent systems. Most service discovery techniques aim at Web services and base their search on the service's inputs and outputs.[7–9] Some approaches also consider preconditions, effects, and other parameters that describe the service. However, for agent-based service discovery mechanisms to be successful for healthcare, they must also use the interactions' contextual information and, in particular, the underlying organizational structure. So, we've developed an approach that extends existing mechanisms by considering the types of interactions in which services can be used and the roles the services play. Using this approach, we've built Rowls, a role-based service matchmaker, and integrated it into the software framework developed by the European Information Society Technologies project Cascom (Context-Aware Business Application Service Coordination in Mobile Computing Environments, www.ist-cascom.org).[10]

Modeling healthcare interactions

A typical healthcare emergency assistance scenario illustrates our approach to interaction modeling based on organizational abstractions.

Frans, a businessman from Finland, is in Austria when he suddenly experiences serious stomach pain.

1541-1672/06/$20.00 © 2006 IEEE
Published by the IEEE Computer Society

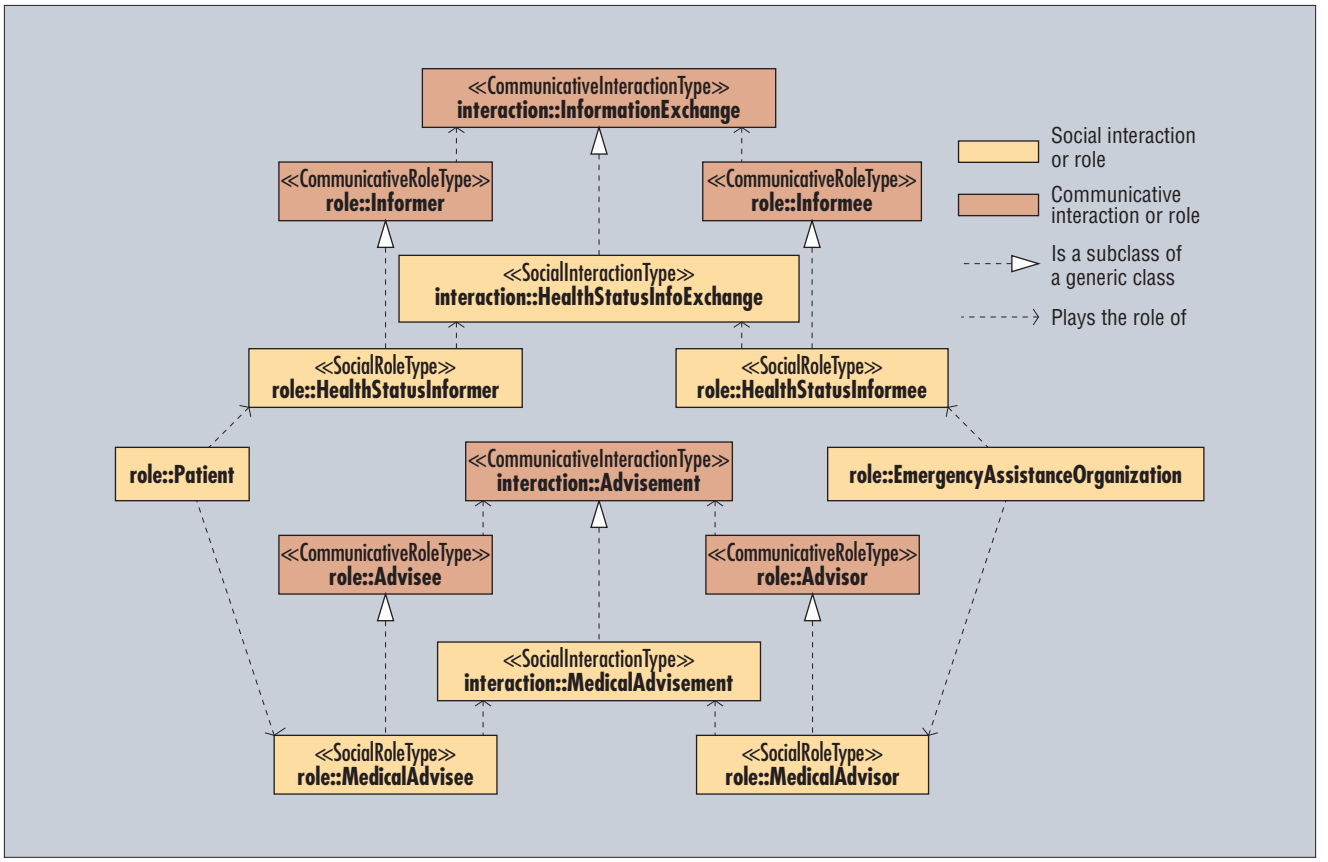


Figure 1. A UML class diagram of a taxonomy showing the refinement of specific roles and interactions to more general roles and interactions, for an emergency-medical-assistance scenario.

Frans doesn't know what to do, so he activates his PDA with the CASCOM mobile-agent suite installed and contacts an Emergency Medical Assistance service center in Finland to ask for advice. (EMA is a Finnish emergency healthcare provider.) The EMA operator asks for his symptoms, if necessary requesting that he detail them and list the medications he's taking. The operator then recommends that Frans visit the nearest healthcare center participating in the CASCOM network. The CASCOM agent installed in the EMA service center can automatically inform the healthcare center that Frans is on his way, to allow for any necessary preparation. Frans receives a detailed map showing how to get to the center; his CASCOM agent could even have called a taxi for him if he needed one.

After arriving at the healthcare center, Frans uses his PDA during check-in, first to announce his arrival to the local network system (login) and then to transfer general (nonsensitive) information to the local hospital information system. This can be either his personal data certified by an electronic signature (certificate) or his social security number, his blood type, and possible allergies.

Shortly after an initial examination, the local physician realizes he must access some laboratory results and ultrasonic images taken before the symptoms appeared. Frans's CASCOM agent locates the home medical record. After Frans gives permission, the requested information is transferred to his PDA, which translates it and forwards it to the local information system. Even with this additional information, the physician still isn't sure about the diagnosis and wants a second opinion. So, using a CASCOM agent at the healthcare center, he finds a cardiologist's contact information and establishes a connection. The cardiologist asks the physician for Frans's symptoms and medical records. After assessing the situation, he recommends that Frans should be transferred soon to a hospital with advanced cardiac life support. All arrangements for that transfer are also made using the CASCOM agent, so that Frans receives the proper medical care and will hopefully recover soon at home.

Interaction analysis and modeling

In this scenario, persons or organizations play five main social roles:

- the patient,
- healthcare professionals in local healthcare institutions who provide on-site medical care for the patient,
- other healthcare professionals who provide additional information or a second opinion about the patient,
- external emergency assistance organizations, and
- administrators of local healthcare institutions.

We've identified several possible interactions between this scenario's participants; figure 1 summarizes some of them. The first part of the scenario is basically a dialogue between Frans and the EMA operator. In this dialogue, Frans is playing the generic social role of **Patient**, while EMA is playing the generic social role of **EmergencyAssistanceOrganization**. By asking for medical assistance, Frans

is starting a MedicalAdvisement interaction, in which he plays the MedicalAdvisee social role and EMA plays the MedicalAdvisor social role. To successfully complete this interaction and give a recommendation, EMA needs certain information about the patient's symptoms, medication, and so on. So, EMA starts a new nested interaction, in which it asks for the patient's health status. In this subinteraction, EMA plays the HealthStatusInformee social role and Frans plays the HealthStatusInformer social role. When this subinteraction finishes (its protocol should be modeled after the real protocol that emergency-assistance organizations follow in these situations), EMA can advise Frans about the best course of action—for example, how to reach the nearest hospital.

To model these interactions, we first obtain a taxonomy of the social roles and interactions. The next step is an abstraction process that generalizes the social roles and interactions into communicative roles and interactions. This process refines the taxonomy to make it more generic, so that we can reuse individuated concepts for other scenarios and application domains. In our case, MedicalAdvisement is a particular case of an Advisement interaction, MedicalAdvisee is generalized to the Advisee role, and MedicalAdvisor is generalized to Advisor. Similarly, we can generalize HealthStatusInfoExchange to InformationExchange, HealthStatusInformer to Informer, and HealthStatusInformee to Informee. Figure 1 depicts this abstraction.

We can similarly extract additional interactions and roles from the scenario. For example, the second opinion in the scenario is a type of medical advisement, in which a specialist diagnoses the situation. The remote health professional (the cardiologist), before providing his opinion, asks the local physician for medical information, such as symptoms, medications, and past exams (in the same way EMA asked Frans about his health status before deciding what he should do). Finally, the physician can require an explanation of the second opinion. So, here we can isolate three different interactions: two that also occurred in the dialogue between EMA and Frans (MedicalAdvisement and HealthStatusInfoExchange) and a new one—SecondOpinionExplanation (see figure 2).

Role and interaction ontologies

On the basis of our analysis and the application of our method to other healthcare scenarios, we've devised a more general ontology containing a taxonomy of interaction types and a taxonomy of the roles in those interactions. Figure 3 shows a UML class diagram of part of the interaction-type ontology. There are two kinds of interactions: *social-interaction types* are domain interactions (EmergencyMedicalAssistance, MedicalAdvisement, and so on), while *communicative-interaction types* are generic reusable interaction patterns. The latter constitute abstract communication interactions that can be instantiated for different scenarios. For instance, a MedicalAdvisement is a specialization, in the medical domain, of the generic Advisement interaction type. The UML diagram represents generic (communicative) interaction types as interfaces and domain (social) interactions as classes.

The ontology's top-level concept is Communication, which represents the most generic interaction type. Any interaction type is a specialization of this concept. The first level of specialization is the ClosedActionPerforming interaction type, which represents any interaction that implies performance of an action. The tag Closed means the interaction has a fixed number of participants (usually two), unlike OpenActionPerforming, where that number isn't predetermined (for example, a call for proposals). Explanation, Advisement, Admission, Assistance, and InformationExchange are specializations of ClosedActionPerforming. For example, in InformationExchange, an Informee asks an Informer to perform the action of informing about a certain fact. We can apply a similar analysis to the domain-interaction types that specialize communicative-interaction types into social-interaction types.

> The abstraction process refines the taxonomy to make it more generic, so that we can reuse individuated concepts for other scenarios and application domains.

Similar to the interaction taxonomy, the role taxonomy includes a hierarchy of the roles participating in the interactions. Each concept in the interaction ontology will have at least two related participant concepts in the role ontology. For example, InformationExchange has Informer and Informee as participants. We use these ontologies, especially their communicative part, when extending service descriptions and the matchmaking process.

Verifying and extending the ontology

We can easily extend our ontology to include new interaction types and roles, making it more complete and reusable. To illustrate this, we use another healthcare scenario involving telemonitoring and "e-inclusion" of patients. (E-inclusion means involving people in the benefits of information and communication technologies.)

Fred suffers from congestive heart failure. His primary-care physician has equipped him with a health-monitoring system consisting of a smart shirt, a sensor embedded in a ring, an accelerometer, and a PDA for computation and wireless communication. This setup allows unobtrusive monitoring of electrocardiograms, heart and respiratory rates, blood pressure, oxygen saturation, and Fred's motion. The PDA wirelessly communicates with Fred's home automation system to exchange data. If Fred's health changes, the system automatically notifies his physician and retrieves all important medical data. Additionally, the system offers the physician a feedback channel that lets her interact with Fred.

The system also lets healthcare service providers attend large numbers of patients efficiently. Extending the communication pattern to one-to-many communication lets caregivers build virtual groups of patients with similar characteristics. This opens up new possibilities for medical care, such as publishing tasks or goals to a group, or informing patients that they're part of a group that can communicate with each other and share their experiences if they want.

We can model some of this scenario's roles and interactions as specializations of existing ones; for example, BiomedicalSignalNotifier and BiomedicalSignalNotification are derived from the Notifier role and Notification interaction. But some new roles, such as VirtualGroupReceiver, and interactions, such as VirtualGroupCommunication, have no direct match in the initial ontology. So, we extend the initial ontology with a new generic role (Proxy) and interaction (ProxyPerforming) to consistently accommodate these new elements.

Role-based service description and matching

In a service-oriented architecture, services

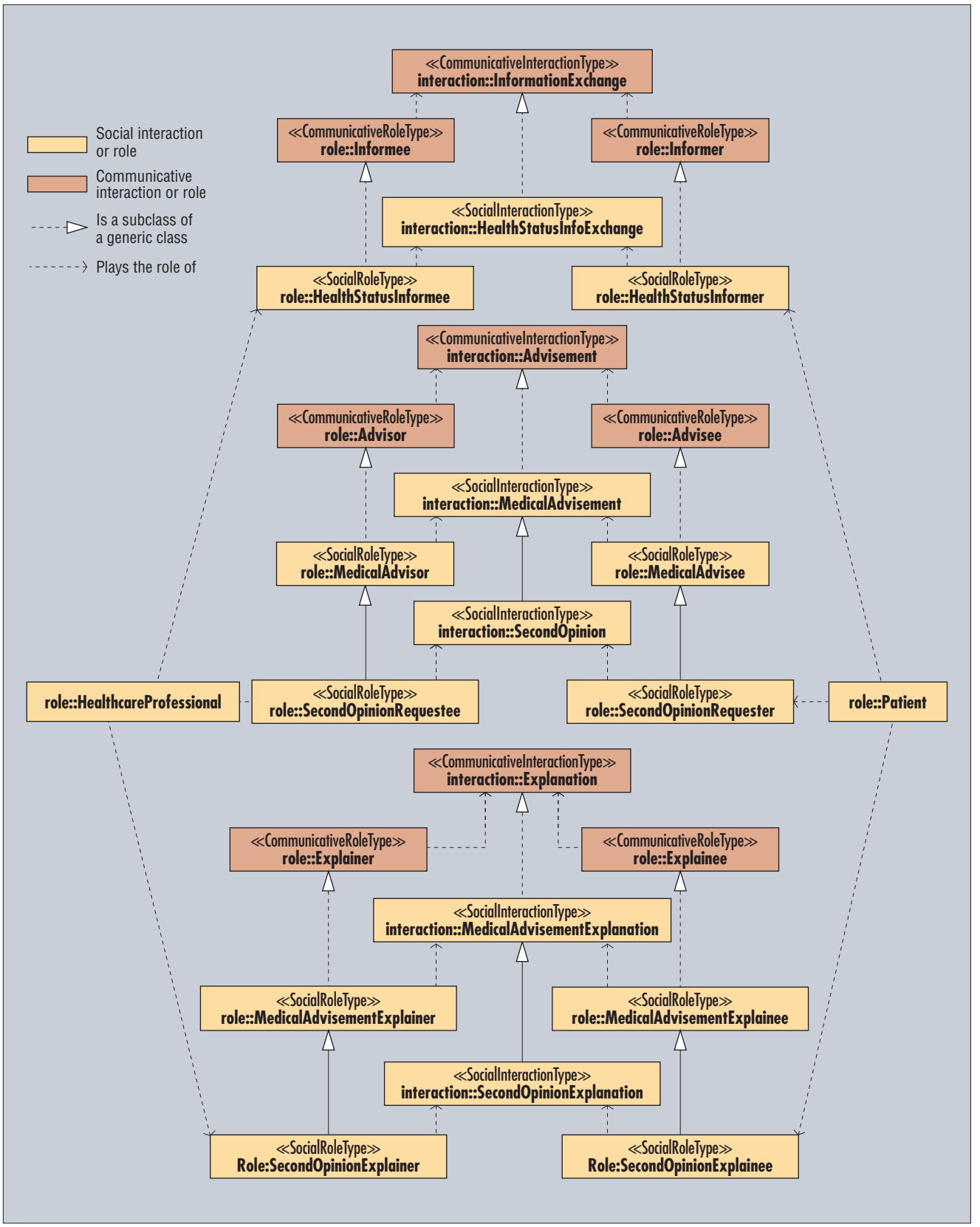


Figure 2. A taxonomy of roles and interactions for a second-opinion medical advisement.

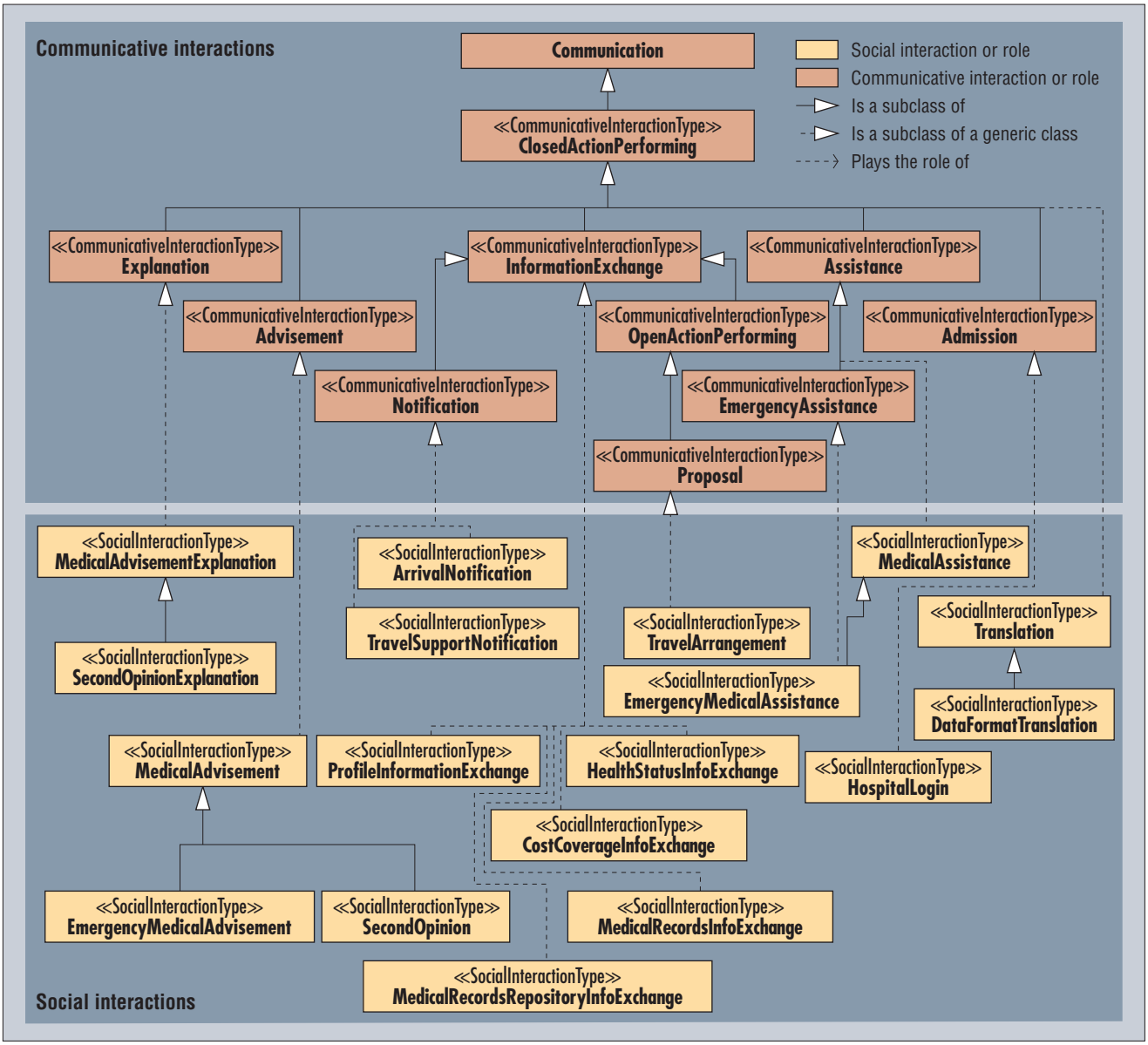


Figure 3. A partial representation of the interaction-type ontology.

are described and registered in some kind of directory (also know as *yellow pages*). When a service is required, these descriptions are used for service discovery. Here, we show how we describe services by using their role-based information and show a matching algorithm based on that description.

Service advertisements

To illustrate our approach, we use the second-opinion service we mentioned earlier. In this example, the service provider (the remote healthcare professional) and the requester (the local healthcare professional who asked for a second opinion) engage in a conversation during the medical advisement. The remote healthcare professional plays the Advisor role. However, to correctly execute the service, the remote healthcare professional must initiate another type of interaction (InformationExchange) to ask the local healthcare professional for additional information. That means the provider requires the requester to play the Informer role.

This situation, also present in many other types of conversations, leads us to specify two fields in the service advertisement that are related to the interactions in which the service provider can engage. First, the *provider role* field indicates the role the service provider plays in an interaction—for example, Advisor in the second-opinion service.

Second, the *depending roles* field indicates the set of roles that the requester must play to correctly execute the service. We represent these roles as a logical formula in disjunctive normal form—that is, a disjunction of conjunctions of roles. This formulation lets us express, for instance, that to correctly execute the service the requester must be able to play an Explainer role or at least an Informer role.

	Provider role	Depending roles
Interaction type 1	Advisor	Explainer ∨ Informer
Interaction type 2	Explainer	
Interaction type 3	Informer	

(a)

Searched provider roles: Advisor ∧ Explainer
Requester capacity roles: Informer, Explainer

(b)

Figure 4. A role-based (a) service advertisement and (b) service request for a second-opinion service.

```
<profile:ServiceParameter rdf:ID="SERVICE_ROLES">
  <profile:sParameter>
    <roles:ServiceRoles rdf:ID="ROLE_DESCRIPTION_LIST">
      <roles:interactiveRole>
        <roles:InteractiveRoleDescription rdf:ID="INTERACTIVE_ROLE_1">
          <roles:providerRole rdf:resource="http://.../roles.owl#AdvisorRole"/>
          <roles:dependingRoles rdf:resource="#DEPENDING_ROLES_1"/>
        </roles:InteractiveRoleDescription>
      </roles:interactiveRole>
      <roles:interactiveRole>
        <roles:InteractiveRoleDescription rdf:ID="INTERACTIVE_ROLE_2">
          <roles:providerRole rdf:resource="http://.../roles.owl#ExplainerRole"/>
        </roles:InteractiveRoleDescription>
      </roles:interactiveRole>
      <roles:interactiveRole>
        <roles:InteractiveRoleDescription rdf:ID="INTERACTIVE_ROLE_3">
          <roles:providerRole rdf:resource="http://.../roles.owl#InformerRole"/>
        </roles:InteractiveRoleDescription>
      </roles:interactiveRole>
    </roles:ServiceRoles>
  </profile:sParameter>
</profile:ServiceParameter>
<roles:RoleExpression rdf:ID="DEPENDING_ROLES_1">
  <roles:item>
    <roles:ConjunctiveRoleList rdf:ID="CONJUNCTIVE_ROLE_LIST_1_2">
      <roles:roleEntry rdf:resource="http://.../roles.owl#InformerRole"/>
    </roles:ConjunctiveRoleList>
  </roles:item>
</roles:RoleExpression>
```

Figure 5. A partial OWL-S description of the second-opinion service advertisement in figure 4a.

These two fields apply to each role the service provider can play.

We can graphically represent a service advertisement as a table with two columns, one for the provider role and the other for the depending roles. Each row represents a different type of interaction with which the provider can provide the service. Figure 4a shows a role-based service advertisement for the second-opinion example.

Service requests

For a service request, we consider queries comprising two elements. First, *searched provider roles* are the roles for which the requester is searching. Although one role will be enough in most cases, we allow for more complex search patterns in which the provider can play more than one role. For example, in the service request in figure 4b, an agent is looking for a second-opinion provider (the Advisor role) that also explains advisements (the Explainer role). As in the case of service advertisements, we require an expression in disjunctive normal form here.

Second, the set of *requester capability roles* define the requester's capabilities regarding the roles the requester can play. This information is important if the provider requires interaction from the requester. For example, in figure 4b, the service request specifies that the requester can play the Informer and Explainer roles if necessary.

Extending OWL-S with roles

Several ontology languages have been developed during the past few years. The most popular is OWL (Web Ontology Language, www.w3.org/2004/OWL), a W3C recommendation that's well supported by commercial and free tools, such as editors, parsers, reasoners, and Java libraries.

To describe services, we use OWL-S (www.daml.org/services/owl-s), a service description language based on OWL. An OWL-S service description has three parts:

- The *service profile* is used for advertising and discovering services.
- The *process model* details how the service operates.
- The *service grounding* provides details on how to interoperate with the service.

We include the role and interaction information in the service profile. Because the service profile describes what the service does, it's an adequate location for role descriptions. A service profile basically includes information about inputs, outputs, preconditions, and effects, as well as the service category and a set of service parameters. However, it doesn't include a specific field for describing organizational information such as roles or interactions, so we must incorporate that information in the available fields. We include the role description as a service parameter called ServiceRoles for service advertisements and one called QueryRoles for service requests. Figure 5 illustrates a partial OWL-S description of the second-opinion service advertisement in figure 4a.

A role-based service-matching algorithm

We've developed a role-based matching algorithm that takes as inputs a service request (R) and a service advertisement (S) and returns the degree of match (*dom*) between them, which is a real number between 0 and 1. Essentially, the algorithm searches for the role in S that best matches the one in R.

Figure 6 shows pseudocode for the Match function. As we described before, the service request might include not only a role but also an expression (a disjunction of conjunction of roles). The loops in lines 4 and

6 decompose that expression, using the minimum as a combination function for the values in a conjunction and the maximum for disjunctions.

MatchAtomicRequest returns the degree of match between a role in the service request and in a service advertisement, given the requester's set of capabilities. It compares the requested role with every other given role and returns the maximum degree of match. For each role in the advertisement, the algorithm makes the match with the provider roles; it also makes the match between the depending roles and the requester's capabilities. The degree of match is the minimum of both values. Again, we represent the depending roles as a logical expression, which the algorithm must evaluate by decomposing it within the **MatchRoleExpr** function.

The algorithm calculates the degree of match between roles on the basis of the role ontology. For this, we use the four classes of degrees of match that Massimo Paolucci and his colleagues proposed: *exact, plug-in, subsumes*, and *fail*.[9] For determining the degree of match between an advertisement role r_A and a query role r_Q, we use these definitions:

- Exact match: $r_A = r_Q$; $dom(r_A, r_Q) = 1$
- Plug-in match: r_A is a subclass of r_Q; $dom(r_A, r_Q) \in [0.5, 1)$
- Subsumes match: r_Q is a subclass of r_A; $dom(r_A, r_Q) \in (0, 0.5)$
- Fail: no subclass relation exists between r_A and r_Q; $dom(r_A, r_Q) = 0$

Furthermore, we want a *dom* that's independent of the ontology tree's overall depth but takes into account the semantic distance $||r_A, r_Q||$ between r_A and r_Q in the role ontology. The following equation describes an intuitive function that fulfills our requirements:

$$dom(r_A, r_Q) = \begin{cases} 1 & \text{if } r_A = r_Q \\ \frac{1}{2} + \frac{1}{2 * e^{||r_A, r_Q||}} & \text{if } r_A \text{ is a subclass of } r_Q \\ \frac{1}{2} * e^{||r_A, r_Q||} & \text{if } r_Q \text{ is a subclass of } r_A \\ 0 & \text{otherwise} \end{cases}$$

With this *dom* function, the smaller the distance between concepts (either in the case of a plug-in or subsumes match), the more influence a change of distance in the degree of match will have.

```
Match(R: service request, S: service advertisement)
{
   dom = 0
   FOR ALL CRLi IN R.ProviderRole {
      dom′ = inf
      FOR ALL rj IN CRLi {
         dom′ = min(dom′,MatchAtomicRequest(rj,R.RequesterCapabilities,S))
      }
      dom = max(dom, dom′)
   }
   return dom
}
MatchAtomicRequest(role: Role, RequesterCapabilities: SET OF ROLES,
                   S: service advertisement)
{
   dom = 0
   FOR ALL IRDi IN S.InteractiveRoleDescription {
      dom1 = MatchRole(role,IRDi.ProviderRole)
      dom2 = MatchRoleExpr(IRDi.RoleExpression, RequesterCapabilities)
      dom = max(dom, min(dom1,dom2))
   }
   return dom
}
MatchRoleExpr(RExpression: SET OF ConjunctiveRoleList,
              RequesterCapabilities: SET OF Roles)
{
   dom = 0
   FOR ALL CRLi IN RExpression {
      dom′ = inf
         FOR ALL rj IN CRLi {
            dom′ = min(dom′,MatchRoleInList(rj, RequesterCapabilities))
         }
      dom = max(dom, dom′)
   }
   return dom
}
MatchRoleInList(role: Role, RL: SET OF Roles)
{
   dom = 0
   FOR ALL ri IN RL {
      dom = max(dom, MatchRole(role,ri))
   }
   return dom
}
```

Figure 6. The main function of a role-based service-matching algorithm. This function returns the degree of match between a request *R* and a service advertisement *S*.

Consider, for instance, the service advertisement in figure 4a and a service request that specifies the searched provider role **SecondOpinionRequestee** and the requester capability role **Informer**. (That is, an agent is looking for a second-opinion provider and can play the **Informer** role if the provider requires it.) Assuming that the semantic distance between the advertisement and the query role is the length of the path between both concepts in the ontology tree, we calculate the degree of match between both services as follows: On the basis of the ontology in figure 3, for interaction type 1, the match between the provider role **Advisor** and the searched role **SecondOpinionRequestee** is

$$\|Advisor, SecondOpinionRequestee\| = 2$$

$$dom(Advisor, SecondOpinionRequestee) = \frac{1}{2} + \frac{1}{2 * e^2} = 0.5677$$

Furthermore, the provider declares the depending role **Explainer** ∨ **Informer**, and the requester includes that role in its capability, so there's an exact match ($dom = \max(dom(Explainer, Advisor), dom(Advisor, Advisor)) = \max(0, 1) = 1$) for the depending-roles section. As the matching algorithm (see figure 6) states, the degrees of match of the provider and depending roles sections are combined by the minimum. So, in our example, interaction type 1's degree of match is $\min(0.5677, 1) = 0.5677$.

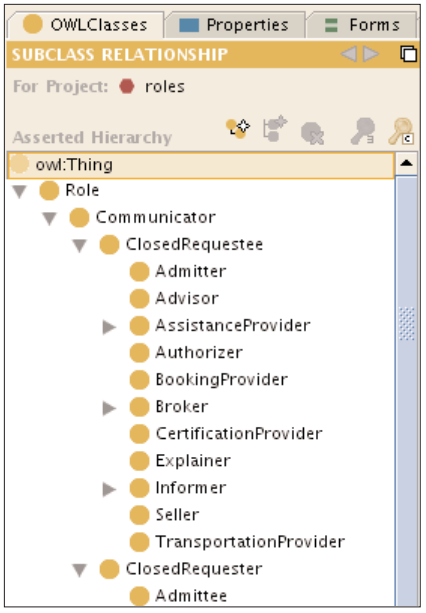


Figure 7. A screenshot of the role ontology developed with Protégé.

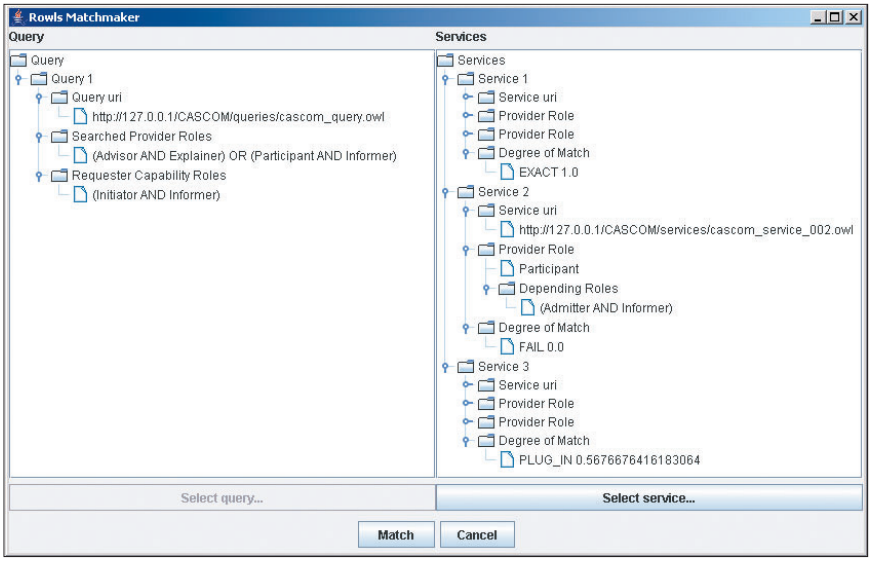


Figure 8. The matchmaker GUI, matching a query with three services.

The same reasoning holds for the other two interaction types of the advertised service. In these cases, no subsumption relation exists between the provider role (Explainer and Informer, respectively) and the searched role (Advisor), so the degree of match is 0. Finally, the degree of match between both services is the best (maximum) of the three possible interactions, which is 0.5677.

Implementing role-based service discovery

As we mentioned before, we've implemented our approach in the Rowls matchmaker, which we integrated into the CASCOM software framework.

Implementing the ontology

To create our role ontology (see figure 7), we used Protégé (http://protege.stanford.edu), a free, open source ontology editor and knowledge base framework. Protégé ontologies can be exported into a variety of formats including OWL, and Protégé is supported by a strong community of developers and academic, government, and corporate users. Furthermore, Protégé has several extensions, including an OWL-S plug-in.

Implementing Rowls

Our role-based matchmaker

- reads user queries and service descriptions,
- extracts role information from the service profiles,
- accesses the role ontology to perform matching between queries and service descriptions, and
- returns service descriptions ordered by the degree of match.

The basic requirements include parsing OWL-S service descriptions and managing the role ontology, with some reasoning support. For this purpose, we chose the Mindswap Java library (www.mindswap.org/~mhgrove/kowari), developed by the University of Maryland's Semantic Web Research Group. For managing OWL ontologies, we opted for the well-known Jena (http://jena.sourceforge.net) framework for building Semantic Web applications. Jena provides a programming environment for RDF, RDF Schema, and OWL; a SPARQL (www.w3.org/TR/rdf-sparql-query) query engine; and a rule-based inference engine. Furthermore, Jena is used also by the Mindswap OWL-S library, letting us reduce our reliance on additional external libraries.

The matchmaker takes an OWL-S service profile as a query, and returns an ordered set of relevant services that match the query, each annotated with its match degree. Because the role ontology's generic part is supposed to be rather static over time, we provide the option of using a matching matrix that stores the degree of match between every pair of roles as new matches are performed.

When asked to perform a match, the matchmaker receives a list of URIs (uniform resource identifiers) of service advertisements (usually from a service directory) and a full-text OWL-S service profile representing the query. It first reads the service advertisement at every URI and extracts the role structure. It then extracts the query's role structure and applies the matching algorithm against the services stored in the engine. Figure 8 shows a screenshot of the matchmaker GUI, matching a query with three services.

Integrating and validating Rowls

In CASCOM, the *Service Matchmaking Agent* provides extended matchmaking functionality. The SMA performs a match between the set of OWL-S service profiles retrieved from distributed directories and the user query, which is also an OWL-S service profile. The SMA has three main components, which focus on different parts of OWL-S service profiles:

- OWLS-MX performs a hybrid syntactic and semantic match of inputs and outputs.[7]
- Rowls looks for similarities between the roles of a query and service advertisement.
- PcEM (Preconditions and Effects Matchmaker) analyzes the logical expressions in the precondition and the effects parts of the OWL-S service profile.

You can combine the three components in several ways. Because they offer similar interfaces, besides the trivial standalone

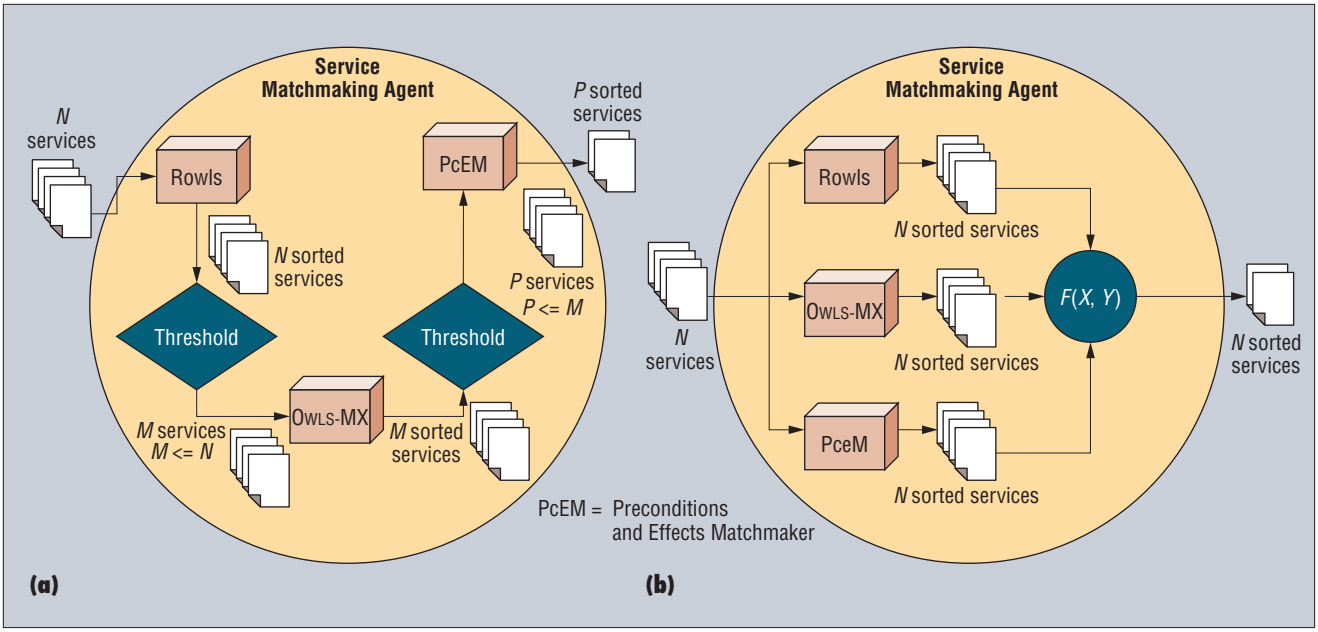


Figure 9. Two matchmaker agent configurations: (a) using each component sequentially; (b) using each component simultaneously.

configurations, we've considered two interesting setups. First, we've used the three components sequentially, where the first two act as prefilters. Figure 9a shows such a configuration, where Rowls possibly reduces the number of input services to OWLS-MX, which in turn possibly reduces the number of input services to PcEM.

The second setup runs the three matchmakers with the same set of services and then combines the returned degrees of match with an aggregation function (see figure 9b).

The only capability the SMA offers to other CASCOM agents is **getMatchingServices**. This capability requires as parameters a URI or full-text OWL-S service profile as a query (service request) and one or more service profiles (services). It can also take an optional parameter defining the type of requested match (exact, plug-in, subsumes, or fail). If **getMatchingServices** executes successfully, the SMA returns a ranked set of services for a query.

For basic evaluation of the algorithm, we've developed a test collection of approximately 300 OWL-S Semantic Web service descriptions, annotated with roles. Our results confirm that the algorithm can obtain ranked matching results in negligible response time (less than 1 second). In addition, we're quantitatively evaluating the matchmaker's performance as part of the SMA, using the different matchmaker combinations we described earlier. Furthermore, the matchmaker, as part of the overall CASCOM infrastructure, is being validated in a field trial concerning real-world medical-emergency assistance, supported by the Austrian Tyrolean Hospital Consortium and EMA.

We plan to vary several aspects of our algorithm to improve its precision. In particular, we'll study how the matchmaker behaves if we substitute our aggregation functions with more complex triangular norms and conorms from fuzzy logic theory.

We also plan to explore how compatible our approach is with the WSMO (Web Service Modeling Ontology) service description approach. ■

Acknowledgments

This work has been partially supported by the European Commission under grant FP6-IST-511632 (CASCOM) and by the Spanish Ministry of Education and Science, project TIC2003-08763-C02-02.

References

1. M. Luck et al., eds., *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing)*, AgentLink, 2005; www.agentlink.org/roadmap/al3rm.pdf.

2. S.A. DeLoach, M.F. Wood, and C.H. Sparkman, "Multiagent Systems Engineering," *Int'l J. Software Eng. and Knowledge Eng.*, vol. 11, no. 3, 2001, pp. 231–258.

3. M. Wooldridge, N.R. Jennings, and D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design," *Autonomous Agents and Multi-Agent Systems,* vol. 3, no. 3, 2000, pp. 285–312.

4. M. Burstein et al., "A Semantic Web Services Architecture," *IEEE Internet Computing,* vol. 9, no. 5, 2005, pp. 72–81.

5. M. Huhns et al., "Research Directions for Service-Oriented Multiagent Systems," *IEEE Internet Computing*, vol. 9, no. 5, 2005, pp. 65–70.

6. L. Cavedon et al., *Extending Web Services Technologies: The Use of Multi-Agent Approaches*, Springer, 2004.

7. M. Klusch, B. Fries, and K. Sycara, "Automated Semantic Web Service Discovery with OWLS-MX," *Proc. 5th Int'l Conf. Autonomous Agents and Multi-Agent Systems* (AAMAS 06), ACM Press. 2006, pp. 915–922.

8. L. Li and I. Horrock, "A Software Framework for Matchmaking Based on Semantic Web

Technology," *Proc. 12th Int'l World Wide Web Conf. Workshop E-Services and the Semantic Web* (ESSW 03), ACM Press, 2003, pp. 331–339.

9. M. Paolucci et al., "Semantic Matching of Web Services Capabilities," *Proc. 1st Int'l Semantic Web Conf.* (ISWC 02), Springer, 2002, pp. 333–347.

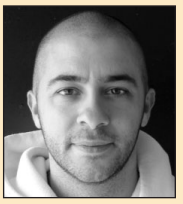
10. C. Cáceres, A. Fernández, and S. Ossowski, "CASCOM—Context-Aware Health-Care Service Coordination in Mobile Computing Environments," *ERCIM News*, no. 60, 2005, pp. 77–78.

The Authors

César Cáceres is an assistant professor of computer science at the University Rey Juan Carlos. He's completing his PhD thesis in e-health, where his primary interests are disease modeling and evaluating how telemedicine affects the care of chronic patients. He obtained his MSc in telecommunications engineering from the Technical University of Madrid. Contact him at Escuela Superior de Ciencias Experimentales y Tecnología (ESCET), Univ. Rey Juan Carlos, C/ Tulipán, s/n. 28933 Mostoles, Madrid, Spain; cesar.caceres@urjc.es.

Alberto Fernández is an assistant professor of computer science at the University Rey Juan Carlos and is finishing his PhD thesis at the university. His main research interests are multiagent systems and service coordination. He obtained his MSc in informatics from the Technical University of Madrid. Contact him at Escuela Superior de Ciencias Experimentales y Tecnología (ESCET), Univ. Rey Juan Carlos, C/ Tulipán, s/n. 28933 Mostoles, Madrid, Spain; alberto.fernandez@urjc.es.

Sascha Ossowski is associate professor of computer science at the University Rey Juan Carlos, where he leads the Artificial Intelligence Research Group. His main research interests include Web intelligence and agent systems. He obtained his PhD in AI from the Technical University of Madrid. Contact him at Escuela Superior de Ciencias Experimentales y Tecnología (ESCET), Univ. Rey Juan Carlos, C/ Tulipán, s/n. 28933 Mostoles, Madrid, Spain; sascha.ossowski@urjc.es.



Matteo Vasirani is a PhD student at the University Rey Juan Carlos and a member of the university's Artificial Intelligence Research Group. He obtained his MSc in informatics from the University of Modena. Contact him at Escuela Superior de Ciencias Experimentales y Tecnología (ESCET), Univ. Rey Juan Carlos, C/ Tulipán, s/n. 28933 Mostoles, Madrid, Spain; matteo.vasirani@urjc.es.