Implementing Agent-based Web Services

Jonathan Dale
Fujitsu Laboratories of America
595 Lawrence Expressway
Sunnyvale, CA 94085, USA
jonathan.dale@fla.fujitsu.com

Luigi Ceccaroni
Fujitsu Laboratories of America
595 Lawrence Expressway
Sunnyvale, CA 94085, USA
luigi.ceccaroni@lycos.com

Youyong Zou*
CSEE Department
U. Maryland, Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250, USA
yzou1@cs.umbc.edu

Avigail Agam*
Department of Computing
Imperial College
180 Queen's Gate
London, SW7 2BZ, UK
aa1199@doc.ic.ac.uk

ABSTRACT
As part of the Agentcities project, we have developed a prototype of an Evening Organiser application which allows users to flexibly and dynamically schedule activities within an itinerary. The Evening Organiser and the Web-accessible restaurant and cinema services which it uses have been developed within a generic service environment and the implementation of this has been built using the April Agent Platform, the DAML+OIL ontology language, the DAML Query Language and the Java Theorem Prover. This service environment is populated with agents of different natures, such as *service instances* and *service finders*. Service instances represent individual business entities, such as restaurants and cinemas. Service finders represent aggregated views over service instances, such as *Yahoo!*-hosted restaurants or *Citysearch*-hosted cinemas. The details of the implementation of these Web Services are described through the use of a motivating scenario.

General Terms
Algorithms, Design, Implementation, Experimentation, Languages.

Keywords
Agents, Ontologies, Agentcities, FIPA, DAML+OIL, Web Services, April.

1. INTRODUCTION
The Agentcities.RTD project, funded by the European Commission, is part of a worldwide initiative [12] intended to realize and advance the potential of agent-based applications by constructing an open, distributed network of platforms hosting diverse agents and services. The ultimate aim of Agentcities is to enable the dynamic, intelligent and autonomous composition of services to achieve user and business goals. Communication among these services has part of its semantic grounding in a series of utility ontologies, which model common general concepts, and in several ontologies for specific domains, such as Travel, Food, Lodging and Entertainment.

In the context of the Agentcities.RTD project, an Evening Organiser (EO) is an agent-based Web Service which helps the user to plan an evening, composing services, such as cinemas, theatres, restaurants, ratings and reviews. As of April 2003, two EOs have been presented in different phases of development, within the Agentcities.RTD project: the Fujitsu EO Prototype and the Motorola EO, which are both non-refined Agentcities demonstrators.

In this paper, we present the design and implementation of the Fujitsu EO Prototype, version 2, and of the associated Web Services which reside in a complex, distributed, agent-based environment called the Agentcities Network.

2. THE EVENING ORGANISER
The Fujitsu EO Prototype 2 refines, improves and extends a previous prototype developed for the Agentcities.RTD project. Prototype 2 implements an architecture, previously presented in [2], into an entertainment-based B2C environment. Agents and services of Prototype 2 are based on various technologies and standards, primarily: the April Agent Platform (AAP) [1], the Foundation of Intelligent Agents (FIPA) software standard and various DAML-based languages.

2.1 General Description
The motivating scenario behind the development of our EO is a user who would like to organize an evening out and, moreover, would like to have the system resolve most of the details on their behalf. Using predefined templates and preferences, an itinerary is composed by the personal assistant agent (PAA) and passed to the EO for resolution using the services that are available in the environment. If the EO cannot solve the itinerary, then it plans alternative-itinerary suggestions and returns these to the PAA for the user to either approve or modify. Upon approval by the user, the individual events in the itinerary are booked and confirmed with the appropriate services instances.

The actors in Prototype 2, besides the human user, are the PAA, the EO agent, restaurant and cinema service-finders and service-agents (see Figure 1). The roles and capabilities of the two main agents in the environment are as follows. The **PAA** gathers personal information about the user; composes itineraries, which are derived from a collection of templates, such as *Pizza and a Movie*, *Oysters and the Opera*, *Burger and a Rollercoaster*, or, alternatively, are entirely new itineraries from the user; customizes the itineraries with the preferences of the user, for example, Japanese restaurants or Mozart operas; manages the outstanding itineraries of the user (of which there may be many).

In turn, the **EO** interrogates the local service environment; attempts to resolve the itineraries by selecting specific service instances (found by querying service finders); interacts with the PAA in the process of itinerary refinement.

We believe that by moving towards an interaction metaphor of *indirect manipulation* [11], with agents as the autonomous actors performing actions on behalf of the user, systems such as our prototype EO can help users to plan most, if not all, of an itinerary.

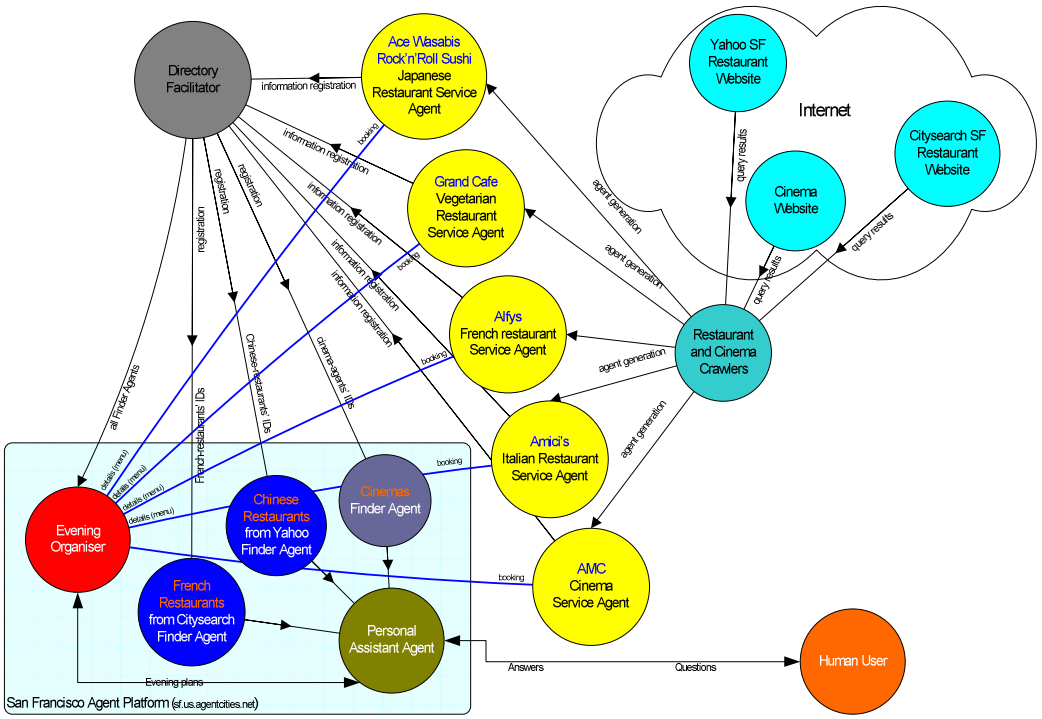

Figure 1: Agent Roles in Prototype 2

2.2 Component Implementation

The Prototype 2 is mainly implemented in the April programming language [9] and is built on top of a FIPA-compliant agent platform called the April Agent Platform (AAP) [1]. This prototype also uses the ICM [10] (an asynchronous communications system for multi-agent systems), Webstream [7] (which allows April programs to easily collect repetitive information from Web pages, such as the results pages of search engines), and MySQL (a popular, open-source database). Some of the technologies we use and their interactions can be seen in Figure 2.

In this section we describe in details how Prototype 2 works and what it can achieve, by describing the functionality of the main agent components and following an accompanying scenario. Details for running Prototype 2 can be found in [3].

2.2.1 Service Instances

An April program generates a number of AAP-based restaurant service instances (SIs) with information being gathered from the Internet through a Webstream crawler. A second program similarly generates a number of cinema SIs. These instances represent real restaurants and cinemas (for example, *Ace Wasabis Rock'n'Roll Sushi* restaurant or *AMC* cinema) and are modelled as persistent parts of the environment. Each of these SIs registers its set of available services with the Directory Facilitator (DF) of the platform through a FIPA ACL message:

```
(request
  :sender restSI@sf.us.ac.net
  :receiver df@sf.us.ac.net
  :language fipa-sl0
  :ontology fipa-agent-management
  :protocol fipa-request
  :content
    "((action register df@sf.us.ac.net
       :df-agent-description (
         :name restSI@sf.us.ac.net
         :services (set
           :service-description (
             :name BookTable
             :type RestaurantInstance)
           :service-description (
             :name QueryMenu
             :type RestaurantInstance)
           ...))))")
```

After registering, the SI generates a MySQL database with the data organised in the tables containing all of the information necessary to model most of the functions of a restaurant business entity. The data modelling for the restaurant and the cinema is derived from the Restaurant and the Shows ontologies, which were developed collaboratively by the authors and other partners in the Agentcities.RTD project.

The MySQL database is used to provide the persistence of the restaurant or cinema business entities. If the service agent is terminated, it can quickly and automatically be restarted and reconnected to its database.

Finally, the SI models all of the pertinent information about the restaurant or cinema as a DAML-S service profile, which is published on the San Francisco Agentcity Web server. This allows other Semantic Web-enabled software to find out information about the SI and use it for their needs (for example, a search engine that supports semantic querying).

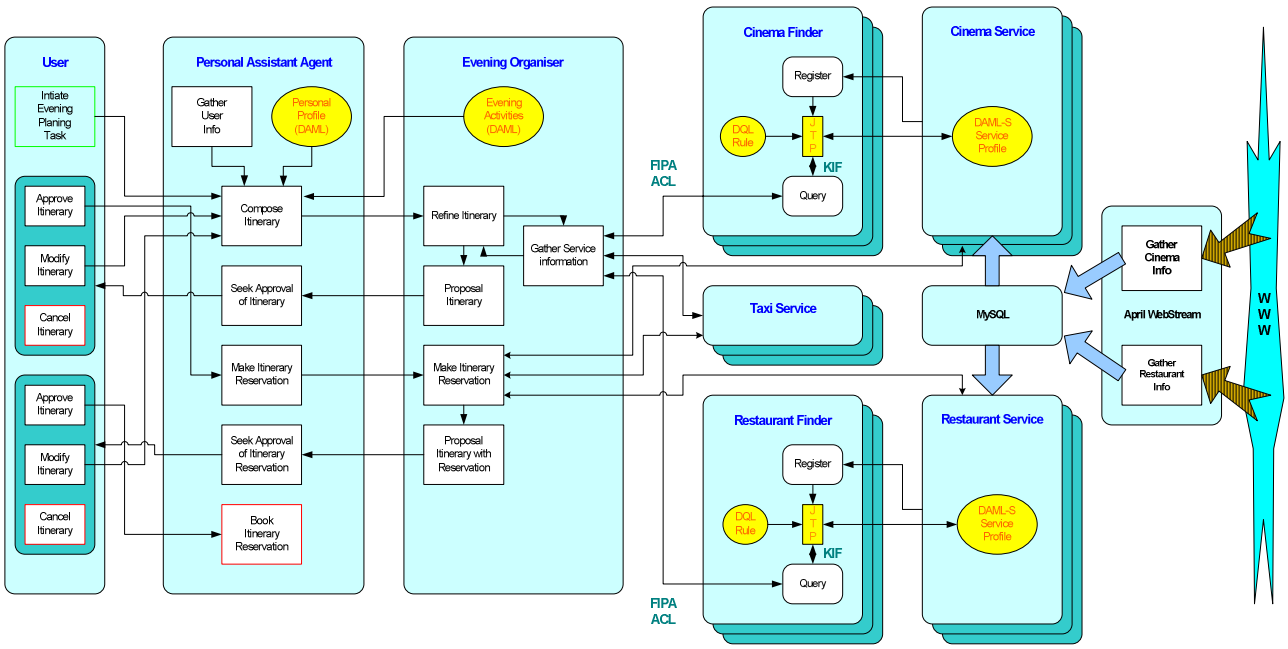

Figure 2: Technologies and Information Flow in Prototype 2

2.2.2 *Service Finders*

Another April program generates a number of aggregated views (or finders) of service instances. These finders represent Internet portals or other composite service providers who deal in information from multiple restaurants and cinemas, for example, French restaurants in San Francisco found through *Yahoo!* or cinemas in San Francisco found through *Citysearch*. In a real-world scenario, these service finders would run on the computer systems of the portals and possibly be accessed directly by the user via an HTML interface. Finders are characterized by a pattern expressed in the DAML Query Language (DQL) [7], such as:

```
<dql:Query>
  <dql:queryPattern>
    <dql:conjuction>
      <dql:triple
        dql:subject="profile:serviceType"
        dql:predicate="rdf:value"
        dql:object="RestaurantInstance"/>
      <dql:triple
        dql:subject="restprofile:CuisineType"
        dql:predicate="rdf:value"
        dql:object="restaurant:Italian"/>
      <dql:triple
        dql:subject="restprofile:InfoSource"
        dql:predicate="rdf:value"
        dql:object="restaurant:yahoo"/>
    </dql:conjuction>
  </dql:queryPattern>
</dql:Query>
```

This example expresses a search pattern over a number of properties in a DAML-S description, namely, any service instance which is a Restaurant, which offers Italian Cuisine and is part of the *Yahoo!* portal. It is important to be able to determine the information source of business entities such as restaurants and cinemas since business models can be based upon reputation and advertisement. The restrictions on the properties of service instances are passed as a DQL expression to the service finder when it is started. The finder uses part of this information (the *profile:serviceType*) to send a FIPA-SL [5] query to the DF, asking for a restricted set of service instances, for example, only for restaurant services. The DF returns a list of service instances which match this request and the service finder iterates over this list, sending a message to each service instance and inviting it to register its DAML-S profile. If the service instance agrees, then it replies with a message which contains the URI of its DAML-S profile, which is retrieved by the service finder through an HTTP call.

Each received DAML-S profile is compared to the DQL expression that the service finder was started with and only if it matches it is added into a Java Theorem Prover (JTP) engine [8] as a fact, which can then be used to answer queries that use either DQL or KIF expressions.

2.2.3 *Evening Organiser Agent*

The EO agent is an April program which receives and fulfils evening plans (called *itineraries*) using the services available in the environment. The EO is a meta-service for the entertainment domain; examples of other Web-based meta-services with similar functionality are *Expedia*, *Travelocity* and *Orbitz*.

2.2.4 *Personal Assistant Agent*

The PAA is the representative of the user and is started with a user personal profile, in DAML+OIL [6], such as:

```
<Profile>
  <FirstName>Joe</FirstName>
  <LastName>Smith</LastName>
  <Organization>Fujitsu</Organization>
  <Email>j.smith@fujitsu.com</Email>
  <ZipLocation>94085</ZipLocation>
  <PhoneNumber>15551234</PhoneNumber>
  <Restaurant.typeOfCuisine>
    VegetarianCuisine</Restaurant.typeOfCuisine>
  <Performance.genre>SciFiMovie</Performance.genre>
</Profile>
```

This profile contains personal information about the user, such as his favourite cuisine and movie types, and is designed to simulate the information that a personal profiling agent might capture

either by watching the user to determine behaviour patterns, or by being explicitly told.

2.3 Scenario

In our scenario, the user Joe Smith wants to arrange to go first to a pizza restaurant and then to see a science fiction movie at a cinema. We consider this to be a task to be achieved and Joe is guided by the PAA in how to specify this task by first selecting a template evening-plan from a template library; in this instance, he selects *Pizza and a Movie* which has the following events pre-specified:

```
(Restaurant
  (typeOfCuisine:Cuisine = Pizza)
 Shows = CinemaShow
  (show:Performance
    (genre:string = "")))
```

Through interactions with the PAA, Joe can customise this template and refine it to his particular needs. The PAA can use the personal profile information of Joe to augment partially specified events; for example, if Joe does not specify a particular genre of movie, the PAA may insert a restriction on science fiction movies from Joe's personal preferences and also introduce new events, such as booking taxis to and from each event. The PAA sends the partially completed evening-plan to the EO for resolution:

```
(EveningPlan
  :plan (set
    (PlanStep
      :step-id step1
      :related-event (Event :related-to Restaurant)
      :initial-state (State :time "" :location "")
      :end-state (State :time "" :location ""))
    (PlanStep
      :step-id step2
      :related-event (Event :related-to Shows)
      :initial-state (State :time "" :location "")
      :end-state (State :time "" :location "")))
  :preferences (set
    (Preference
      :value (any (sequence ?step1)
        (exists ?x1
          (and (= (value ?step1 step-id) step1)
               (and (= ?x1 (value (value ?step1 related-
event) typeOfCuisine))
                    (= ?x1 Pizza)))))
      :weight 0.7)
    (Preference
      :value (any (sequence ?step2)
        (exists ?performance
          (and
            (= (value ?step2 step-id) step2)
            (and
              (= ?performance (value (value ?step2
related-event) show))
              (= (value ?performance genre) Sci-Fi)))))
      :weight 0.7)
    (Preference
      :value (any (sequence ?step1 ?step2)
        (and
          (= (value ?step1 step-id) step1)
          (and (= (value ?step2 step-id) step2)
               (and
              (< (value (value ?step1 end-state) time))
                 (value (value ?step2 initial-state)
time)))))))
      :weight 0.9)))
  :reservation-information (set (RI
    :service ""
    :bookingLocator "")))
```

The EO tries to complete the itinerary by searching the service finders (which it finds from the DF) for SI-matches to each event. Where it finds service instances that match, it sends a query message to the SI to see if a booking can be made. If it can, then that event in the itinerary is finalised.

If the EO can find each event, then the completed itinerary is returned to the PAA for final approval by the user. If it cannot, for example there are no restaurants available for the time chosen by Joe, then the EO can make minor changes to the itinerary, such as choosing another time segment or restaurant type. In this instance, the EO generates multiple alternative-itineraries, which are prioritized, and Joe can either select the most appropriate one or modify one and resubmit it to the prototype for resolution. Once Joe confirms an itinerary, the EO makes all of the necessary bookings with the SIs. When a booking is made by the EO with an SI, it makes an actual reservation entry in a table of the MySQL database; this allows us to maintain a persistent state of the SIs and also to enhance the realism of the prototype simulation.

3. CONCLUSIONS

In this paper, we have presented research that relates to the Agentcities.RTD project, which is part of a worldwide experiment aiming to use agents in large-scale environments. We have described a prototype which implements and extends a design for service composition developed within the Agentcities Network. The system, an Evening Organiser, allows users to create and manage plans composed of entertainment events. Our prototype is one of the first deployments within Agentcities and uses different technologies and software standards together to build a large-scale, agent-based service. An important aspect of our prototype is that it allows users to have the underlying agent system resolve their tasks asynchronously and autonomously, providing a model for task delegation.

4. ACKNOWLEDGMENTS

The authors wish to extend their thanks to John Knottenbelt (Imperial College) for the help in the implementation of the prototype. The research described in this paper is partly supported by the EC project Agentcities.RTD (IST-2000-28385). The opinions expressed in this paper are those of the authors and are not necessarily those of the EU Agentcities.RTD partners.

5. REFERENCES

[1] Dale, J. April Agent Platform Reference Manual. Fujitsu Laboratories of America, 2002. http://sf.us.agentcities.net/aap/

[2] Dale, J. and Ceccaroni, L. Pizza and a Movie: A Case Study in Advanced Web Services. In Proceedings of the workshop AAMAS 2002 – W04: Challenges in Open Agent Systems, Bologna, Italy, 2002. http://www.nar.fujitsulabs.com/documents/fla-nartm02-03.pdf

[3] Dale, J., Ceccaroni, L., Zou, J., Agam, A. and Knottenbelt, J. Fujitsu Evening Organiser Prototype. FLA-NARTM02-13, Fujitsu Laboratories of America, 2002. http://www.nar.fujitsulabs.com/documents/fla-nartm02-13.pdf

[4] DAML Query Language, August 2002. DAML, 2002. http://www.daml.org/2002/08/dql/

[5] FIPA SL Content Language Specification [FIPA00008], Foundation for Intelligent Physical Agents, November, 2002. http://www.fipa.org/specs/fipa00008/

[6] Hendler, J. and McGuinness, D. L. The DARPA Agent Markup Language, IEEE Intelligent Systems 15 (6), pages 67-73, November/December 2000. http://www.daml.org/2001/03/daml+oil-index.html

[7] Hong, T., Concurrent Programming with Webstream. Imperial College, 2002.

[8] Java Theorem Prover: An Object-Oriented Reasoning System. Knowledge Systems Laboratory, Stanford University, 2001. http://www.ksl.stanford.edu/software/JTP/

[9] McCabe, F. April Programming Language Reference Manual. Fujitsu Laboratories of America, 2002. http://www.nar.fujitsulabs.com/april/

[10] McCabe, F., InterAgent Communications Model Reference Manual. Fujitsu Laboratories of America, 2002. http://www.nar.fujitsulabs.com/icm/

[11] Negroponte, N., Hospital Corners. In: The Art of Human-Computer Interface Design, Laurel, B., Ed., pages 347-353, Addison-Wesley, 1990.

[12] Willmott, S., Dale, J., Burg, B., Charlton, P. and O'Brien, P. Agentcities: A Worldwide Open Agent Network. In: The Agentlink Newsletter 8, 13-15, 2001. http://www.agentcities.org/

---

* Youyong Zou and Avigail Agam carried out the research related to this paper at Fujitsu Laboratories of America.