GraniteNights - A Multi-Agent Visit Scheduler Utilising Semantic Web Technology

Gunnar AAstrand Grimnes, Stuart Chalmers, Pete Edwards, and Alun Preece

Dept. of Computing Science, University of Aberdeen
Aberdeen, AB24 5UE, Scotland.
{ggrimnes, schalmer, pedwards, apreece}@csd.abdn.ac.uk

Abstract. This paper describes a multi-agent system, GraniteNights, modelled on the Agentcities project "evening agent" scenario. GraniteNights allows a user to plan an evening's entertainment in the city of Aberdeen, Scotland. The application fuses agent and Web technology, being viewed as an agent-based Web service. In particular, Semantic Web standards are used to a great extent in delivering the service. The paper argues that, in fact, the Semantic Web standards are more important for this type of application than the agent standards. A key feature of the application is component re-use: GraniteNights attempts to reuse without modification existing ontologies wherever possible; it also is comprised of a number of generic and thus wholly-reusable agents, including a user profiling agent and a constraint-based scheduler. The system is open in the sense that most of the individual agents can be invoked directly by external agent platforms, without going through the Web interface.

1 Introduction

GraniteNights is a multi-agent application which allows a user to schedule an evening out in Aberdeen (aka the "Granite City"). Intended users of the GraniteNights application are visitors to Aberdeen or anyone else requiring assistance in identifying suitable dining and entertainment venues, as part of an evening schedule. The scenario is closely modelled on the Agentcities evening-planning agent scenario [7]. We view this type of application as a form of (intelligent) Web service, and have therefore employed Web standards wherever possible. In particular, we have committed to the Resource Description Framework [11] as the main content language in order to align our work as closely as possible with the Semantic Web [2]. The cost of creating RDF to describe the resources of interest in our application domain - restaurants, public houses, etc. is relatively low (a form-filling interface facilitates the process). Moreover, we are able to utilise for the most part existing ontologies represented in DAML+OIL [10], and we retain forward compatibility with the emerging OWL standard [12].

Agents within the application infrastructure are organised according to a series of predefined roles: information agents (wrappers for RDF resources - either static or dynamically generated from existing WWW sources); profile agent (manages user data, such as id, password and preferences); constraint-solver

agent (maps RDF data to finite domain constraints and produces valid instantiated schedules); evening-agent (receives user queries and controls invocation of other agents to generate a solution); user-interface agent. The application has been constructed using the Java Agent DEvelopment framework (JADE) agent platform[1], SICStus Prolog (+ Jasper[2]) as well as BlueJADE [6] running under the JBoss[3] J2EE application server.

The GraniteNights application was conceived as a vehicle for exploring a number of issues: integration of agent and Semantic Web standards; utilisation of off-the-shelf ontologies; re-use of components of information agent systems (at a number of levels); use of constraint-based scheduling; semantic user profiling; and not least the deployment of a real and substantial Agentcities application. We believe that a number of novel contributions arise from this work. Firstly, we assert that (at least for this type of information agents application) most of the power comes from the content standards, specifically the Semantic Web standards. Essentially all of the FIPA[4] standards employed in GraniteNights could easily be replaced by Web services technology, for example SOAP and WSDL [5]. Perhaps controversially, we discarded FIPA-SL altogether as we took the view that it brought nothing to our type of application.

Secondly, we believe that GraniteNights demonstrates that currently available, relatively weak ontologies expressed for the most part simply in RDF Schema, are the appropriate level of Semantic Web technology for useful applications. In fact, our view is that stronger ontologies would have been harder to re-use, because they would have imposed overly-restrictive commitments on their applicability. Thirdly, GraniteNights embodies several strategies for component reuse. Several of the agents are wholly general and would serve as useful components of other agent systems — the information agents, the ProfileAgent, and the SchedulerAgent (the latter *is* in fact already being used in another application). Moreover, within the various agents, there is considerable reuse of code, notably in the information agents that all share the same "shell".

Finally, we aim to show in this paper that GraniteNights includes a number of novel technologies, in particular its approach to semantic profiling (allowing users to incrementally create profiles that are meaningful to them, and portable across a variety of applications), and the RDF Query By Example language.

The remainder of this paper is organised as follows: Section 2 provides an overview of the system architecture, and the relationships between the various components; Section 3 describes the lowest level agents which manage a series of information resources (encoded using RDF); Section 4 discusses the operation of the scheduling component, including the constraint solver; Section 5 presents the user-profile management aspects of GraniteNights; we conclude with a discussion of lessons learned and related work in Section 6; and discuss suggested extensions to the architecture in Section 7.

[1] http://sharon.cselt.it/projects/jade/
[2] http://www.sics.se/sicstus/docs/latest/html/sicstus.html/Jasper.html
[3] http://www.jboss.org
[4] Foundation for Intelligent Physical Agents, http://www.fipa.org

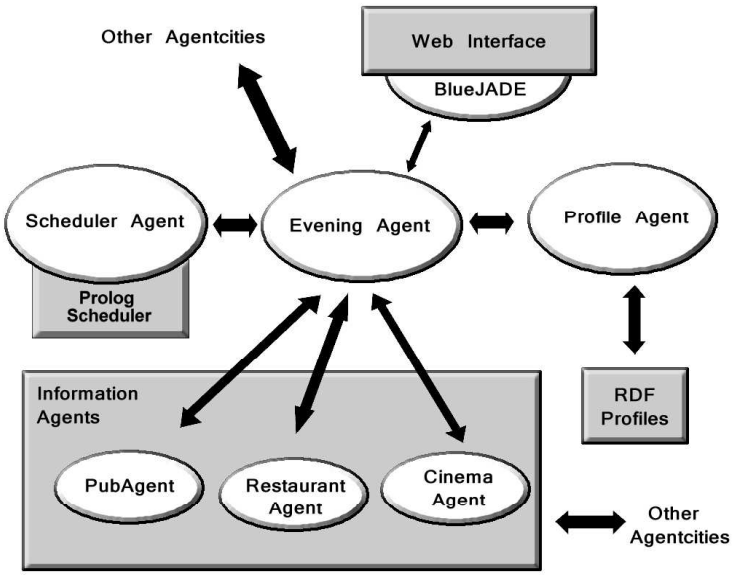

Fig. 1. GraniteNights - Architecture Diagram.

2 GraniteNights System Overview

The GraniteNights architecture is presented in Figure 1. The system consists of a collection of service providing agents, and a central agent which coordinates the process of delivering a plan, called the EveningAgent. The decomposition of services into the collection of agents shown (white ovals) aims to maximise the reusability of the components; where possible, each discrete independent sub-service is embodied in a separate agent.

The application can be accessed by another software agent via communication with the EveningAgent, or by a user directly through a Web interface. For space reasons we are unable to present full details of interactions with the EveningAgent here, but these are available from our Web site[5].

The EveningAgent receives a partially instantiated evening-plan containing a list of events, possibly including start times and durations. Each event is either a visit to a public house, a restaurant or a cinema, with optional constraints on that event. Example constraints include: *This Pub must serve Guinness*, or *I want to see the film "The Pianist"*. The EveningAgent queries the appropriate InformationAgent to get a list of possible events which match the given constraints. Once all the possible events have been assembled they are passed to the SchedulerAgent along with the partial plan. The SchedulerAgent then generates a set of possible complete plans for the evening, choosing times and events according to the user specified criteria. One finished plan will then be

[5] http://www.csd.abdn.ac.uk/research/AgentCities/GNInfo/

returned to the user, who has the option of accepting the plan or asking for the next solution. The EveningAgent uses a hard-coded collaboration strategy; it knows about the three information agents available, and makes no attempt at discovering new information sources. It also works closely with the ProfileAgent and SchedulerAgent, but is only tied to the external interfaces of these agents, which makes them pluggable modules of GraniteNights. It would thus be trivial to deploy a more sophisticated scheduler or profiling mechanism in the future.

The GraniteNights Web interface[6] is implemented using Java Server Pages (JSP), Java Servlets and BlueJADE. BlueJADE allows Web applications to send and receive agent messages through a *gateway agent*, which in turn allowed us to write an EveningAgent client that generates the RDF messages from simple Web forms completed by the user. Screenshots from the Web interface are presented in Figure 2. On the left hand side is the input interface, showing that a user has specified that (s)he wants to go to a public house that serves *Hoegaarden* at 18.00, followed by a visit to a cinema showing the film *The Pianist* and ending their evening with a meal in a restaurant serving Italian cuisine. The right hand side of Figure 2 shows one possible plan for their evening generated by GraniteNights. Note how the user has also specified that the cinema should be a 15 minute walk (or less) away from the pub, and the restaurant a 15 minute walk away from the cinema. These proximity constraints are implemented by positioning each venue available in GraniteNights within a two dimensional grid overlaid on a map of Aberdeen (see Figure 4 for a section of the full map). There are also some venues that are outside the map. Three different location constraints are available: 15 minutes walk, which means within an adjacent square; 30 minutes walk or a short cab-ride, meaning within two squares; and long taxi-ride which is everything else. The sections that follow deal with the various agents in more detail.

Name	**Developers**	**URI**
Pubs	Aberdeen	http://www.csd.abdn.ac.uk/research/... ...AgentCities/ontologies/pubs
Beer	Aberdeen	— " —/beer
Query By Example	Aberdeen	— " —/query
Eveningplan	Aberdeen	— " —/eveningplan
Restaurants v.4	Agentcities	— " —/restaurant-v4.daml
Shows v.2.7	Agentcities	— " —/shows-v27.daml
Address	Agentcities	http://sf.us.agentcities.net/ontologies/address.daml
Calendar	Agentcities	http://sf.us.agentcities.net/ontologies/calendar.daml
Price	Agentcities	http://sf.us.agentcities.net/ontologies/price.daml
Contact Details	Agentcities	http://sf.us.agentcities.net/ontologies/contact-details.daml

Table 1. Ontologies Used By GraniteNights.

[6] http://www.csd.abdn.ac.uk/research/AgentCities/GraniteNights

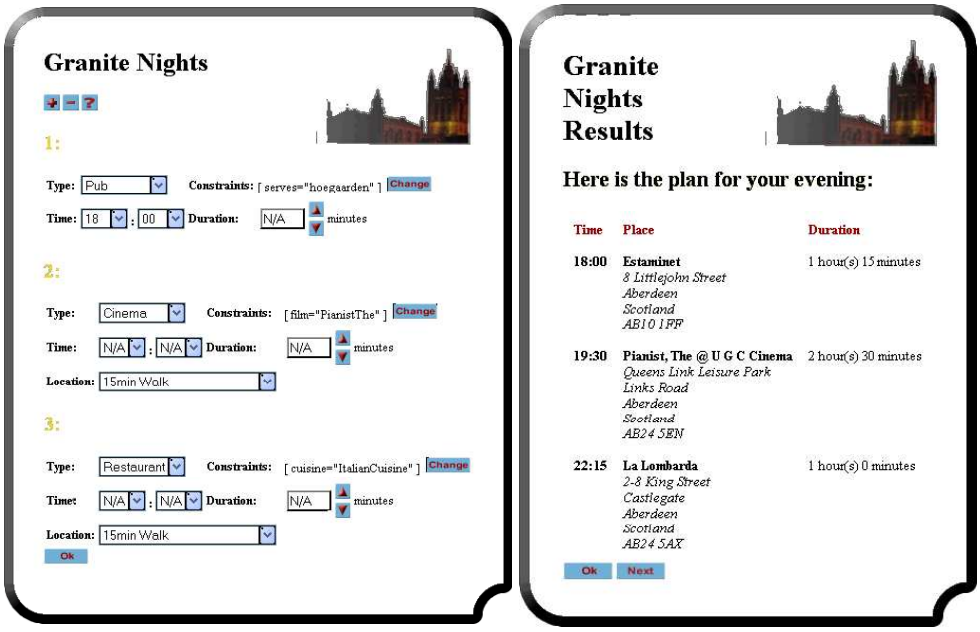

Fig. 2. GraniteNights - Web Interface.

3 Information Agents

GraniteNights uses three information agents, all of which act as simple wrappers for RDF data-sources. Each has a simple, consistent query interface based upon Query By Example[7] (QbEx), discussed in Section 3.2. The three agents facilitate access to information about public houses, cinemas and restaurants in Aberdeen. The PubAgent and RestaurantAgent both wrap static RDF files, while the CinemaAgent wraps RDF which is dynamically generated from a conventional Web page supplying up-to-date cinema listing information. Although the information agents form the basis of GraniteNights, their usefulness is not restricted to this application, as they are all externally accessible over the Agentcities network, and were designed with re-use in mind.

All the information agents run off the same code-base with configuration files used to specify the input source to wrap. As far as possible our information agents use standard Agentcities ontologies, such as the utility ontologies for addresses, contact details and calendar information (Table 1).

3.1 Static vs. Dynamic Information Agents

The static information agents facilitate access to RDF data that has been manually generated, containing information about public houses and restaurants in

[7] http://www.csd.abdn.ac.uk/research/AgentCities/QueryByExample

Aberdeen. Ontology support for the PubAgent is provided by pub and beer ontologies developed at Aberdeen (Table 1). The RestaurantAgent uses the Agentcities restaurant ontology. Examples of instances from these ontologies are presented in Figures 3 and 5. Instances contain information about the contact details and services available in each pub or restaurant; in addition they contain location information, expressed as coordinates within a map of Aberdeen city centre, as described in Section 2.

```
<pub:EnglishPub rdf:ID="#estaminet">
  <pub:liveEntertainment
    rdf:resource="pubs#Sometimes"/>
  <pub:location>
    <add:Address rdf:ID="address">
      <add:publicPlaceName>Little John St.
        </add:publicPlaceName>
      <add:cityName>Aberdeen</add:cityName>
      <add:countryName>Scotland
        </add:countryName>
      <add:doorNumber>8</add:doorNumber>
      <add:zipCode>AB10 1FF</add:zipCode>
      <add:locationId> 7, 7
        </add:locationId>
    </add:Address>
  </pub:location>
  <pub:onLicense>true</pub:onLicense>
  <pub:pubName>Estaminet</pub:pubName>

  <pub:openingPeriods
    rdf:resource="times#sun_thu_midnight"/>
  <pub:openingPeriods
    rdf:resource="times#weekend_three"/>

  <pub:servesBeer
    rdf:resource="beer#budweiser"/>
  <pub:servesBeer
    rdf:resource="beer#leffeblonde"/>
...
  <pub:servesFood>on</pub:servesFood>
  <pub:telNumber>01224 622657
  </pub:telNumber>
</pub:EnglishPub>
```

Fig. 3. Example RDF Pub Instance.

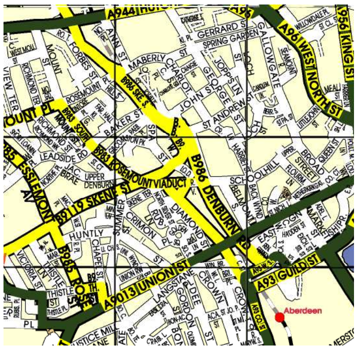

Fig. 4. Location Map Example.

```
<res:Restaurant rdf:about="#lalombarda">
  <res:name>La Lombarda</res:name>
  <res:averageMealDuration>2
    </res:averageMealDuration>
  <res:address>
    <add:Address rdf:about="rest#lombardaaddr"/>
  </res:address>
  <res:atmospheres rdf:resource="res#CasualAtmosphere"/>
  <res:atmospheres rdf:resource="res#RelaxedAtmosphere"/>
  <res:caterings rdf:resource="res#ALaCarte"/>
  <res:caterings rdf:resource="res#HomeDelivery"/>
  <res:facilities rdf:resource="res#SmokingFacility"/>
  <res:typeOfCuisine rdf:resource="res#ItalianCuisine"/>
</res:Restaurant>
```

Fig. 5. Example RDF Restaurant Instance.

GraniteNights also contains a dynamic information agent, which wraps a conventional HTML Web page giving information about films showing at cinemas in Aberdeen (Figure 6). This information is extracted and converted into RDF conforming to the Agentcities Shows ontology (Table 1). The extraction and conversion is done via simple regular expression pattern matching, and would

need to be rewritten if the structure of the HTML source were to change. For performance reasons, the RDF is extracted once a day and cached locally.

Original HTML Page.

```
<s:Shows rdf:ID="ugc_PianistThe">
  <s:time>
    <s:ShowScheduleCollection> <!–THUR-6:30PM–>
      <s:consistsOf><s:ShowSchedule>
        <s:startTime><c:Calendar><c:calendarDate>
          <c:Date>
          <c:dateDayOfWeek
             rdf:resource="cal#Thursday" />
          <c:year>2003</c:year>
          <c:month>1</c:month>
          </c:Date>
        </c:calendarDate><c:calendarTime>
          <c:Time>
          <c:format rdf:resource="cal#24h"/>
          <c:timeHour>19</c:timeHour>
          <c:timeMinute>30</c:timeMinute>
...
  <s:location rdf:resource="cinemas#ugc" />
  <s:description>Certificate: 15</s:description>
  <s:show>
    <s:CinemaPerformance rdf:ID="PianistThe">
    <s:title>Pianist, The</s:title>
    <s:wwwInformation>http://scoot.co.uk/info?a=5
       </s:wwwInformation>
    <s:origin>Canada</s:origin>
    <s:duration><c:Duration>
       <c:durationHour>2</c:durationHour>
       <c:durationMinute>28</c:durationMinute>
    </c:Duration></s:duration>
    <s:artist><s:ArtistCollection>
      <s:consistsOf><s:Artist>
        <s:artistName>Adrien Brody</s:artistName>
        <s:rolevalue rdf:resource="shows#Actor"/>
...
```

Resulting RDF.

Fig. 6. Dynamic Information Agent Example.

3.2 RDF Query By Example

RDF Query By Example (QbEx) is the query language used throughout Granite-Nights. It was developed by the authors to fulfil the need for a higher-level RDF query language than existing triple-based solutions such as RDQL [14], used in the Jena Semantic Web toolkit[8] and RDFStore[9]. We felt that RDQL, while a useful tool for the application developer, is not suitable for end-user queries, as constructing RDQL queries requires detailed insight into the workings of RDF

[8] http://www.hpl.hp.com/semweb/jena.htm
[9] http://rdfstore.sourceforge.net/

and the RDF schemas used. Our solution is RDF Query By Example (QbEx) which allows the user to express queries in RDF, partially describing the RDF instance(s) to be returned. Figure 7 shows a simple example of a QbEx query, requesting that all pubs serving *Guinness* be returned. RDF statements can be constructed using either RDF literals or variables constrained using an RDF constraint language [9]. Internally the RDF QbEx structure is converted to RDQL (with automated RDF schema subclass inference). Figure 9 shows a more sophisticated example using a variable to return all restaurants open after 7 PM, while Figure 8 shows the same query when converted to RDQL by the QbEx engine.

4 The Scheduling Agent

The purpose of the SchedulerAgent is to take information on available films, restaurants and pubs (provided by the EveningAgent), together with the preferences expressed by the user on type, time, duration and location (provided by the user through the Web interface) and create a set of valid schedules, showing the order, start time and duration of the events requested. To achieve this the agent implements a finite domain scheduling algorithm using the SICStus finite domain constraint library [4] to match the user requirements to the given information on the available venues, creating sets of valid schedules that satisfy the request.

The information given to the SchedulerAgent consists of a piece of RDF containing the possible venues for the evening (plus the information on those venues) together with the user requirements. On receipt of this the agent extracts the set of possible events, which are parsed and asserted as simple prolog facts. These are then matched with the schedule created from the user's request, with the requirements provided by the user constraining the items in the schedule.

As an example, Figure 10 shows such an RDF fragment detailing two movies and three pubs, along with their equivalent prolog representation (the `data/6` constructs)[10]. In this example, suppose the user asks for a schedule containing three items: a pub, a movie and a restaurant (in that order). To represent these we begin by creating two finite domains, one for the start time of each event requested and one for its duration. The example below shows the internal representation of the three events along with their respective domains:

```
% domain ( events, start-time, stop-time).
domain([START1,START2,START3],0,104),
domain([DUR1,DUR2,DUR3],4,24).
```

These predicates set up three tasks each with a start time between 0 and 26hrs (as some pubs are open till 2 AM) and durations of 1 to 6hrs[11]. Depending on

[10] Space does not permit us to show the full RDF representation, which also contains a number of restaurants

[11] The measurement scale shown means that the duration and start time can be measured at a granularity equivalent to 15 minute intervals.

```
<q:Query>
  <q:template>
    <p:EnglishPub>
      <p:servesBeer
        rdf:resource="beertypes#guinness"/>
    </p:EnglishPub>
  </q:template>
</q:Query>
```

Fig. 7. Simple QbEx Example.

```
SELECT ?x WHERE (?x, ?y, ?z),
   ( ?x, <r # open-time>, ?v_x ),
   ( ?x, <rdf # type>, <r # restaurant> ),
   ( ?x, <r # type>, <r # Tandoori> ) AND ( ?v_x > 1900 )
```

Fig. 8. RDQL Generated From the QbEx Query in Figure 9.

```
<q:Query>
  <q:template>
    <r:Restaurant>
      <r:type rdf:resource=?r#Tandoori" />
        <r:open-time>
          <cif:variable rdf:ID="x">
            <cif:varname>x</cif:varname>
          </cif:variable>
        </r:open-time>
    </r:Restaurant>
  </q:template>
  <q:constraints>
    <cif:comparison>
    <cif:comparison_operator>&gt;
        </cif:comparison_operator>
    <cif:comparison_op1>
      <cif:variable rdf:about="#x"/>
    </cif:comparison_op1>
    <cif:comparison_op2>
      <cif:integerconst>
        <cif:constant_value>1900
          </cif:constant_value> .. . .
```

Fig. 9. QbEx Example with Variables.

```
<ep:EveningPlan>
 <ep:events>
  <rdf:Seq>
   <rdf:_1>
    <ep:Event>
     <ep:duration>
      <c:Duration>
       <c:durationHour>2</c:durationHour>
       <c:durationMinute>0</c:durationMinute>
      </c:Duration>
     </ep:duration>
     <ep:place>
      <rdf:Alt>
       <rdf:li><pub:EnglishPub rdf:about="pubs#estaminet" /></rdf:li>
       <rdf:li><pub:EnglishPub rdf:about="pubs#wildboar" /></rdf:li>
       <rdf:li><pub:EnglishPub rdf:about="pubs#eastneuk" /></rdf:li>
      </rdf:Alt>
     </ep:place>
   </rdf:_1><rdf:_2>
    <ep:Event>
     <ep:place>
      <rdf:Alt>
       <rdf:li><cin:Shows rdf:about="films#ugc_PianistThe" /></rdf:li>
       <rdf:li><cin:Shows rdf:about="films#lighthouse_PianistThe" /></rdf:li>
      </rdf:Alt>

...

% data(<name>,<type>,<open>,<close>,<location>).
data('ugc_PianistThe',movie,2020,2248,9,6).
data('lighthouse_PianistThe',movie,1815,2043,5,6).
data('estaminet',pub,1000,2700,7,7).
data('wildboar',pub,1000,2400,5,5).
data('eastneuk',pub,1000,2400,7,5).
```

Fig. 10. An RDF Fragment and its Equivalent Prolog Representation.

the requirements set by the user, these domains are then constrained with the following information:

- Order & Location Constraints: The initial constraint on the schedule domains comes from the ordering of the events and their relative locations (described in Section 2). For example, if the three tasks mentioned were to be ordered as `START1, START3, START2`, all within 15 minutes walking distance of each other we would set the following constraints on the domains:

```
START1 + DUR1 + 1 #=< START3,
START3 + DUR3 + 1 #=< START2
```

- Time Constraints: These can be provided by the user or left unspecified. This constrains the time that the user wants to visit the venues, and the amount of time spent in each location. If specified then we make the `START` and `DUR` variables equal to the values given.

Once constrained, the remaining domain values are matched with the given possibilities (held in the `data/6` format). From this we create a set of possible evening schedules. Figure 11 shows one such valid schedule. The scheduler creates as many of these schedules as required (or as many as is possible) and returns these as a set of possible evening plans (in RDF format).

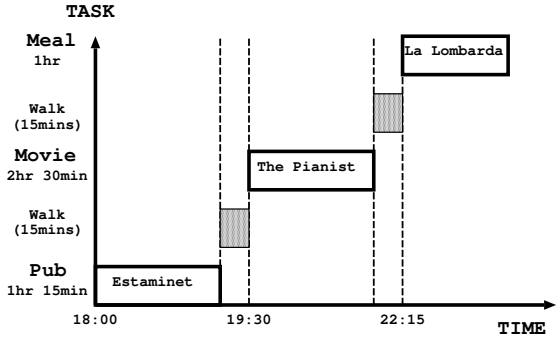

Fig. 11. An Example Schedule Created by the SchedulerAgent.

5 Profile Agent

The ProfileAgent has two tasks within GraniteNights: The first is handling initial user registration and user authentication on return visits; registration simply involves setting up an account with a username and a password. The second task of the ProfileAgent is to keep track of the user's interests; the interest model is generated through analysis of previous interactions with the system, so there is no model available for new users. If a user's preferences are available, they will be included in the reply to a successful login request.

To generate the user interest model the ProfileAgent has access to the following information: the constraints specified for each event, the possible evening plans that were rejected, and the final evening plan that the user accepted. The EveningAgent informs the ProfileAgent of each of these every time a user visits GraniteNights. Note that there is no information available about why a user rejects a plan, although some information can be extracted by comparing the rejected plan(s) with the accepted version.

For example, a user specifies that (s)he wants a single event for the evening, which should be a pub offering live entertainment. GraniteNights' first candidate plan suggests going to the pub *Estaminet* at 18.00. The user was in that pub recently, and would like to try something else, so (s)he asks for the next solution. GraniteNights suggests *The Blue Lamp* at 18.00 and the user accepts this plan. The ProfileAgent has now been informed of the user's constraint that the pub must have live entertainment, and also that (s)he accepted the second plan and

rejected the first. By comparing the twothe ProfileAgent could determine that it was the choice of venue that was incorrect, and not the time.

The architecture of GraniteNights was designed to abstract away the implementation details of the ProfileAgent, allowing us to experiment with different profiling techniques by simply writing new pluggable ProfileAgent modules. The current implementation of the ProfileAgent is basic, as it ignores all information about rejected or accepted plans, and simply caches user specified constraints for each available information event. Using this information, the most frequently cited constraint becomes the user's preference. For example, as shown in Figure 12, user *Pete* has used the system on three occasions, asking for pubs serving *Hobgoblin Ale* twice and *Flowers Ale* once, this means that Pete's current preference is for pubs serving *Hobgoblin*. The ProfileAgent stores the user profiles using RDF; an example profile is shown in Figure 12.

6 Discussion & Related Work

In this paper we have shown that RDF is a good alternative to SL as a content language for agent interaction. Agents communicating through RDF are not

```
<ep:User>
  <ep:name>pedwards</ep:name>
  <ep:pword>gnes</ep:pword>
  <ep:preferences>
    <pub:EnglishPub pub:serves="beer#hobgoblin" />
  </ep:preferences>
  <ep:interactions>
    <rdf:Seq>
      <rdf:li>
        <ep:Interaction ep:date="20030315T164433" >
          <ep:pref>
            <pub:EnglishPub pub:serves="beer#hobgoblin" />
          </ep:pref>
        </ep:Interaction>
      </rdf:li>
      <rdf:li>
        <ep:Interaction ep:date="2003037T153710" >
          <ep:pref>
            <pub:EnglishPub pub:serves="beer#flowersale" />
          </ep:pref>
        </ep:Interaction>
      </rdf:li>
      <rdf:li>
        <ep:Interaction ep:date="2003032T121567" >
          <ep:pref>
            <pub:EnglishPub pub:serves="beer#hobgoblin" />
          </ep:pref>
        </ep:Interaction>
      </rdf:li>
    </rdf:Seq>
  </ep:interactions>
</ep:User>
```

Fig. 12. Example RDF Profile Instance.

new, and are indeed central to the Semantic Web vision [17]. However, in the Agentcities project there has been a commitment to FIPA standards, and thus to FIPA-SL. Although FIPA-SL is more expressive than RDF (for example, it allows quantified variables), we feel that the much larger user-base of RDF, the variety of tools available, and the fact that it is based on XML, make it a better choice as content language for agent-based Web services. In addition, we feel that the current expressiveness of RDF is more than sufficient for a large range of applications. There are several RDF based solutions already being used within Agentcities, mainly DAML+OIL and OWL for describing ontologies and DAML-S for service description, but there are also projects committed to using RDF for content, for example in the Travel Agent Game in Agentcities (TAGA) [19].

Through developing GraniteNights we have also shown how re-use of components and ontologies facilitates construction of advanced composite services, such as the process of putting together an evening plan. In creating GraniteNights we have also attempted to supply the Agentcities network with a set of new reusable components, such as the information agents and their ontologies, in the hope that others will make use of these and combine them with other services in a new and interesting manner. This component re-use model is very much in the spirit of Agentcities and many projects are deploying simple component services suitable for integration into larger applications, for example Whitestein Technologies' CAMBIA service for currency exchange[12], and the agent-based medical services developed by GruSMA [16].

We have also demonstrated how multi-agent applications can be given a user-friendly Web interface which still remains a loosely coupled component of the system by using the gateway agent in BlueJADE. Many other Agentcities projects are also using BlueJADE, and there is currently significant effort going into improving integration between agent platforms and Enterprise Application Servers [3].

GraniteNights is also a first step towards user profiling using RDF for profile acquisition and representation: creating profiles which are meaningful to the user, and to a range of systems outside the originating application. Learning symbolic user profiles is a well researched field, for example in [15] where symbolic rules were learned for a user's meeting preferences. However, the interest in such profiling in a Semantic Web environment is only now slowly gaining momentum. For example, the Internet Content Rating Association has recently started a project to investigate "Customization and Personalization through RDF"[13]. Some work has also been done in representing user interest profiles with respect to a specified ontology and, by using RDF, deploying and keeping these profiles up to date in a different applications. [13].

[12] http://zurich.agentcities.whitestein.ch/Services/CambiaService.doc
[13] http://www.icra.org/cprdf/

7 Future Work

GraniteNights came together as a joint project between different members of the Aberdeen research community, each person bringing different technologies and different perspectives. We are all excited to see our work integrated into one application like GraniteNights, and we are all keen to improve the current implementations of each module.

In the current information layer of GraniteNights the information about public houses and restaurants is all hand generated by the project members, and while this has worked well, there are several short-comings in the current data. The PubAgent does not have information about drinks served beyond beers, and the RestaurantAgent does not know about individual dishes. We are currently exploring links with local goverment and other service providers with a view to accessing a number of available data-sources to replace some of the current information agents, and also for generation of new agents, such as an HotelAgent, CastleAgent and WhiskeyDistilleryAgent.

As mentioned above, the current ProfileAgent implementation is very simple. We plan to improve this module with a more sophisticated solution in several steps: Firstly we intend to explore the use of RDF inference to generalise better when generating user preferences. As shown in Figure 12, user *Pete* currently has the preference *a pub should serve Hobgoblin*, because that is the most frequent constraint he has specified. However, closer inspection of the ontology shows that *Hobgoblin* and *Pete's* other preferred beer, *Flowers*, are in fact both sub-classes of the class *Real Ale*. A more intelligent ProfileAgent should be able exploit this relation, and generate the preference *Pete likes pubs serving Real Ales*. Secondly, we will look into the use of knowledge intensive Machine Learning algorithms for generating the profiles, in contrast to the current statistical approach. We have performed some preliminary work using the Inductive Logic Programming system Progol for learning from data marked up using RDF [8], and plan to explore the use of Case-Based Reasoning [1] and Explanation Based Generalisation [18], as these are also capable of learning from symbolic data. Thirdly we will explore further the advantages of expressing the user model as RDF, the re-usability of such a model will enable us to make a ProfileAgent that is not only usable within GraniteNights, but also within other projects, such as the upcoming Information Exchange[14] project in Aberdeen.

GraniteNights' current user-interaction model is based on the user sitting at home, having access to a computer, and planning their evening before they go out. While this is useful, a more common scenario is a group of people having been to the cinema, and only when the film has finished agreeing to extend their evening with a drink in a pub. Having GraniteNights accessible on their PDA just then would be very useful. Implementing this would also raise several interesting issues surrounding localised information delivery, for generating output such as "200m down Union St, on your right hand side, there is a pub serving *Director's Bitter*. The current modularised architecture makes writing another

[14] http://www.csd.abdn.ac.uk/research/iexchange/

client-interface to work with mobile devices very easy, and we are current exploring mobile/wireless access to GraniteNights through a local wireless service provider.

References

1. A. Agnar and P. Enric. Case-based reasoning : Foundational issues, methodological variations,and system approaches. *AI Communications*, 7:39–59, 1994.
2. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.
3. S. Brantschen and T. Haas. Agents in a j2ee world. Technical report, Whitestein Technologies AG, 2002.
4. M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In *Proc. Programming Languages: Implementations, Logics, and Programs*, 1997.
5. M. Champion, C. Ferris, E. Newcomer, and D. Orchard. Web services architecture. W3c working draft, World Wide Web Consortium, 2002.
6. Dick Cowan, Martin Griss, and Bernard Burg. Bluejade - a service for managing software agents. Technical report, Hewlett-Packard Labs, 2001.
7. J. Dale and L. Ceccaroni. Pizza and a movie: A case study in advanced web-services, 2002.
8. P. Edwards, G. A. Grimnes, and A. Preece. An empirical investigation of learning from the semantic web. In *ECML/PKDD, Semantic Web Mining Workshop*, 2002.
9. P. Gray, K. Hui, and A. Preece. An expressive constraint language for semantic web applications. In *E-Business and the Intelligent Web: Papers from the IJCAI-01 Workshop*, pages 46–53. AAAI Press, 2001.
10. J. Hendler and D. L. McGuinness. The darpa agent markup language. *IEEE Internet Computing*, 2000.
11. Ora Lassila and Ralph R. Swick. Resource description framework (rdf) model and syntax specification. W3c recommendation, World Wide Web Consortium, 1999.
12. D. L. McGuinness and F. van Harmelen. Web ontology language (owl): Overview. W3c working draft, World Wide Web Consortium, 2003.
13. S. Middleton, H. Alani, N. Shadbolt, and D. De Roure. Exploiting synergy between ontologies and recommender systems. In *11th International WWW Conference*, 2002.
14. L. Miller, A. Seaborne, and A. Reggiori. Three implementations of squishql, a simple rdf query language. Technical report, Hewlett-Packard Labs, 2002.
15. Tom M. Mitchell, Rich Caruana, Dayne Freitag, John McDermott, and David Zabowski. Experience with a learning personal assistant. *Communications of the ACM*, 37(7):80–91, 1994.
16. A. Moreno and D. Isem. Offering agent-based medical services within the agentcities project. In *Proceedings of Agents Applied in Health Care at 15th European Conference on Artificial Intelligence*, 2002.
17. T. R. Payne, R. Singh, and K. Sycara. Browsing schedules - an agent-based approach to navigating the semantic web. In *Proceedings of The First International Semantic Web Conference (ISWC)*, 2002.
18. F. van Harmelen and A. Bundy. Explanation based generalisation = partial evaluation. *AI*, 36:401–412, 1988.
19. M. P. Wellman and P. R. Wurman. A trading agent competition for the research community. In *IJCAI Workshop on Agent-Mediated Electronic Commerce*, 1999.