StarGazer: A Semantic Application

Alvaro Graves, James Michaelis, Joshua Shinavier and Giovanni Thenstead

Rensselaer Polytechnic Institute
Troy, NY, USA
Agenda

- Motivation
- Related Work
- Benefits of Semantic technologies
- Implementation
- Evaluation
- Conclusions
Motivation

- Scenario
  - An amateur astronomer is searching for a star in the field.
  - She forgot her star charts or they don't have the exact information she is looking for....
- But...
  - She has her cellphone/iPhone/etc.
  - She has Internet access.
Goals we want to accomplish

- Web-based application
- Interoperable
- Extensible
Use cases

- Select all the stars from certain area of the sky and show their name, apparent magnitude, distance and spectral type ordered descending by apparent magnitude.

- Export (and publish) a geolocation.

- Obtain the information related to certain star.
Related Work
Related Work

• Several related eScience works
• VSTO observatory:
  • System for integrating data from heterogeneous subdomains in astronomy (solar, space, and solar-terrestrial physics)
  • Caters to a broad range of user backgrounds, with varying degrees of training in different domains.
Related Work

- Stargazer not presently focused on domain data integration
- However, it is designed with Semantic Web driven usability in mind.
- Many of these eScience applications like VSTO could be useful in field research settings, but aren’t implemented for this purpose (e.g., no mobile endpoints)
Related Work

• In addition to eScience applications, lot of prior work on defining domain ontologies for Astronomy and its sub-disciplines.

• Differing functionality aims for these ontologies
  • Facilitating domain intuitive user queries
  • Domain data integration from heterogeneous sources.
Related Work

- In Stargazer domain ontology, one of the primary objectives is to facilitate class hierarchy based inference.
- Domain-intuitive querying not of interest to us, since functionality for direct querying from Stargazer front-end not provided.
- Since we only rely upon central database, heterogeneous data integration not of concern.
Benefits of Semantic Web
What to discuss

• Current Stargazer features
• Possible improvements
Current Functionality

• Class hierarchy based Inference
• Internal function GetAllStars
  – Runs a query to get back all stars meeting a user query.
Current Functionality

In our ontology, bunch of kinds of stars in could be returned by user query.

Inference used to show these subtypes are kinds of stars, and should be returned.
Current Functionality

- **Linked data based star descriptions**
- Uses Representational state transfer (REST) functionality to retrieve RDF-based descriptions of individual stars, via dereferenceable URIs
- Allows for very data-specific descriptions to be given for returned results, which could be more cumbersome to implement otherwise.
Future Work

• Data inference driven user querying

• Example:
  • Given: An observer’s position (in degrees latitude and longitude)
  • A section of the sky (denoted by right ascension and declination values)
  • Query: Find any visible constellations classified in the 1700s
  • Assuming stars are asserted as constellation members, and constellations given a date of classification, inference could come into play deciding which of these dates fall into the time range “1700s”.
Future Work

• **Distributed data access**
• Lots of Astronomy data on the web, but few means of integrating it.
• Rather than enter this data into central repository, could link to it via dereferenceable URIs.
• Our individual star descriptions provide an example of this functionality
Future Work

• Verifying data credibility
• Information regarding the authorship of certain entities can be asserted.
• Through this, users could assert metrics by which to trust data.
• For example, “Trust data sources created by agencies with IAU endorsement.”
Implementation
StarGazer Implementation

- ontology design
- project structure
- front end
- triple store
- Java beans
- resource API
- source code management
Ontology Design

- StarGazer ontology
- hierarchy of celestial objects
- RDFS inference
- OWL expressivity
- functional properties as documentation
- reflected in beans interfaces
- Protégé used for ontology design and data entry
Project Structure

- choice of Java programming language:
  - prior experience of contributors
  - tool availability
- IDE-agnostic
  - Eclipse and IntelliJ IDEA environments used
- built using Maven 2.x
  - uniform build system
  - eases dependency management

http://maven.apache.org/
Front-End

• use of Google Web Development Toolkit (GWT)
  • web-accessible UI
    • support for basic HTML components
  • client/server architecture bridges JavaScript/Java boundary
• integration with Maven using GWT-Maven
Triple Store

- use of OpenRDF Sesame framework
  - previous experience
  - variety of pluggable components
    - RDFS inferencer
- MemoryStore Sail implementation
  - fast and simple
  - good for small, static data sets
  - easily swapped out for persistent / more powerful triple store
- triple store is loaded with the ontology
Java Beans

- treats RDF resources as Java objects with “get” and “set” methods for property values
  - “deep integration” simplifies manipulation of RDF data in Java
- use of OpenRDF Elmo BeanPool toolkit
  - compatible with Sesame
  - permits SPARQL-style queries for beans
- Java class hierarchy corresponds to class subsumption hierarchy in StarGazer ontology
Search API

• search API allows retrieval of celestial objects subject to various constraints
  • currently: location, time, name, NGC number

• retrieve-all-and-filter
  • convenient but not scalable

• all available fields are retrieved for returned objects
Resource API

• all StarGazer resources exposed as Linked Data
  • includes individuals, classes, and properties
• alternately serve HTML or RDF/XML
  • using content negotiation
• description contains all statements in which the resource is the subject
• TODO: 303 redirection for resource URIs
HTML for Resources

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sg:hasDeclination</td>
<td>+63.87°</td>
</tr>
<tr>
<td>rdf:type</td>
<td>MainSequenceStar</td>
</tr>
<tr>
<td>sg:hasSpectralType</td>
<td>G</td>
</tr>
<tr>
<td>sg:hasName</td>
<td>Sun</td>
</tr>
<tr>
<td>sg:hasMainSequenceObject</td>
<td>coreObject, radiationZone, photoSphere, corona, chromoSphere, convectionZone</td>
</tr>
<tr>
<td>sg:hasSolarMasses</td>
<td>1.0</td>
</tr>
<tr>
<td>sg:hasMeanSolarRadius</td>
<td>1</td>
</tr>
<tr>
<td>sg:hasRightAscension</td>
<td>286.13°</td>
</tr>
</tbody>
</table>
RDF/XML for Resources
Source Code Management

• StarGazer project hosted at Google Code
• version control using Subversion over HTTPS
• check out a copy from:
  • http://stargazer.googlecode.com/
Evaluation
Evaluation: Goals

- Implicit Testing
  - Testing how well custom application interfaces with our API.

- Explicit Testing
  - Measuring against current web services for stargazing and tracking.

- Durability Testing
  - How well does the annotation system work?
  - Test and compare annotated catalog against competitors.
Evaluation: Preamble

• Test Bed:
  • IBM Thinkpad T60
  • 2.0 GHz Intel Duo Core T25000
  • 2.5 GB RAM @ 2.0GHz
  • Windows XP, Service Pack 3
  • Java JDK 6 Update 5

• Metrics (Retrieval Success and Speed):
  • Search by Location
  • Search by Name
  • Search by NGC No.
Evaluation: Data Store

- Consists of equatorial coordinates for mapping star spatial dimensions.
  - Latitudinal and Longitudinal
  - Right accession and Declination coordinates
  - Time in day, month, year, hour, minutes, and seconds.
- Limited, but useful for testing accuracy and speed benchmarks.
Evaluation: Interfacing API with Custom App
Evaluation: Extending Annotation System

• How does our API matchup against existing Sky-gazing applications?
  • Test retrieval success
  • Annotation success
Evaluation: Extension to Google™ Sky
Evaluation: Conclusions

• Much more work to be done
  • Way to successfully extend Google Sky™ and other web services.
  • Improving query response
  • Possibly offloading data to multiple stores, thus improving access.
  • Porting the API to other languages
General Conclusions
Conclusions

- StarGazer is an amateur astronomer assistant

- It take advantage of semantic technologies, such as OWL to allow interoperability

- The ontology created is extensible as well as the application
Future work

- Add user information, such as notes and comments
- Possibility of adding information about planets and other celestial objects
- Redevelop using AJAX, making a nicer interface
- Look into implementing Semantic Web based functionality extensions mentioned earlier.
Thanks for your time