RDF in a .NET World

Kal Ahmed (@kal_ahmed)
About Me

• Developer, Consultant
• Co-founder of NetworkedPlanet
• Co-creator of BrightstarDB
• .NET Programmer since 2004
• BrightstarDB
• RDF Data Binding
• LINQ to SPARQL
• OData/SPARQL Interop
• Questions
• Persistent Quad Store for .NET
  – Pure C# implementation
  – Classic .NET, Mono, Portable Class Library, Android, iOS (soon)
  – Embeddable .NET API
  – REST-based server
  – SPARQL 1.1, lots of RDF syntaxes
  – Small footprint, easy to deploy
  – Open-Source, MIT licensed
  – http://brightstardb.com/
RDF DATA BINDING
Open vs Closed World

• Open World Models
  – Scale better
  – Allow for better data integration
  – Allow for easier data migration

• Closed World Models
  – Easier to reason
  – Easier to integrate
  – Almost always exist at some layer
Compromise

- Domain Model as a View
Our Approach

• Bind domain type to an rdf:Type
  – Makes it easy to find instances
  – An RDF resource may have multiple types

• Unique instance identifiers bind to RDF resource URIs
  – Use a configured URI prefix + Id
  – Applications will need to share an addressing scheme

• Class properties bind to RDF literal properties

• Class relations bind to RDF resource properties
“Contract-First” Code Generation

Decorated interface definitions + conventions

```
[Entity]
public interface IPerson {
    string Id { get; }
    string Name {get; set;}
    ICollection<IPerson> Knows {get; set;}
}
```
public interface IPerson {

    [Identifier("http://example.com/person/")]
    public string Id { get; }

    [PropertyType("foaf:name")]
    public string Name { get; set; }

    [PropertyType("foaf:knows")]
    public ICollection<IPerson> Knows {get; set;}

    [InverseProperty("Knows")]
    public ICollection<IPerson> KnownBy {get; set;}
}
[assembly:NamespaceDeclaration("foaf",
  "http://xmlns.com/foaf/0.1")]

[Entity("foaf:Person")]
public interface IPerson {

  [Identifier("http://example.com/person")]
  public string Id { get; }

  [PropertyType("foaf:name")]
  public string Name { get; set; }

  [PropertyType("foaf:knows")]
  public ICollection<IPerson> Knows { get; set; }

  [InverseProperty("Knows")]
  public ICollection<IPerson> KnownBy { get; set; }
}
[assembly: NamespaceDeclaration("foaf",
    "http://xmlns.com/foaf/0.1")]

[Entity("foaf:Person")]
public interface IPerson {

    [Identifier("http://example.com/person")]
    public string Id { get; }

    [PropertyType("foaf:name")]
    public string Name { get; set; }

    [PropertyType("foaf:knows")]
    public ICollection<IPerson> Knows { get; set; }

    [InverseProperty("Knows")]
    public ICollection<IPerson> KnownBy { get; set; }
}
public interface IPerson {

    [Identifier("http://example.com/person/")]
    public string Id { get; }

    [PropertyType("foaf:name")]
    public string Name { get; set; }

    [PropertyType("foaf:knows")]
    public ICollection<IPerson> Knows {get; set;}

    [InverseProperty("Knows")]
    public ICollection<IPerson> KnownBy {get; set;}
}
[assembly:NamespaceDeclaration("foaf",
                "http://xmlns.com/foaf/0.1")]

[Entity("foaf:Person")]
public interface IPerson {

    [Identifier("http://example.com/person")]
    public string Id { get; }

    [PropertyType("foaf:name")]
    public string Name { get; set; }

    [PropertyType("foaf:knows")]
    public ICollection<IPerson> Knows { get; set; }

    [InverseProperty("Knows")]
    public ICollection<IPerson> KnownBy { get; set; }
}
[assembly: NamespaceDeclaration("foaf",
  "http://xmlns.com/foaf/0.1/")]}

[Entity("foaf:Person")]
public interface IPerson {

  [Identifier("http://example.com/person/")] public string Id { get; }

  [PropertyType("foaf:name")]
  public string Name { get; set; }

  [PropertyType("foaf:knows")] public ICollection<IPerson> Knows {get; set;}

  [InverseProperty("Knows")]
  public ICollection<IPerson> KnownBy {get; set;}
}
Data Binding In Operation

• Data object tracks the quads loaded for an entity.
  – Lazy loading by default
  – Can be eager loaded by LINQ queries

• Data context object tracks changes
  – Collection of quads to be added to the store
  – Quad patterns to be removed from the store.
  – Patterns allow wildcard binding
Saving Changes

• BrightstarDB Transaction
  – List of guard patterns
  – List of quad patterns to delete
  – List of quad patterns to add

• SPARQL 1.1 UPDATE
  – DELETE WHERE
  – INSERT DATA
Additional Features

- Type Casting
- Optimistic Locking
- Composite Keys
- “Unique” Identifiers
RDF Data Binding

• It can be useful
• It is relatively easy to implement basic data binding
• The approaches used here should work with Java, Python etc.
• Covers basic CRUD operations, but no query...
LINQ TO SPARQL
LINQ to SPARQL

- Language INtegrated Query
  - Common language for data access in .NET
  - Supports ADO.NET, XML documents, in-memory collections and now SPARQL endpoints.
  - Uses introspection to provide Intellisense auto-completion in Visual Studio
Implementation

- Parse the LINQ query
- Walk the query tree generating SPARQL expressions
- Wrap the expressions in SELECT or CONSTRUCT as appropriate
- Execute the SPARQL query
- Data bind the results
from p in Context.Dinners select p.Id

OR

From p in Context.Dinners select p

SELECT ?p WHERE {
    ?p a nerd:Dinner .
}
from p in Context.Dinners
where p.Id.Equals("1")
select p.Rsvps;

SELECT ?v1 WHERE {
  <http://nerddinner.com/dinners/1>
    a nerd:Dinner .
  <http://nerddinner.com/dinners/1>
}
from x in Context.Dinners
from r in x.Rsvps
select r.AttendeeEmail

SELECT ?v1 WHERE {
  ?x a nerd:Dinner .
  ?r a nerd:Rsvp .
  ?x nerd:attendees ?r .
}
from x in Context.Dinners
where x.EventDate >= DateTime.UtcNow
select x.Id

SELECT ?x WHERE {
  ?x a nerd:Dinner .
  ?x nerd:eventDate ?vo .
}
Methods

• String methods:
  – String.StartsWith => STRSTARTS
  – String.EndsWith => STRENDS
  – Case-insensitive options map to REGEX

• Math operations

• Collection operations
  – Contains => IN
Eager Loading

- Avoid N+1 round-trips
- Naïve Approach:

```
CONSTRUCT {
  ?s ?p ?o
} WHERE {
  SELECT ?s WHERE {
    ...
  }
}
```
Sorting

• First problem for naïve approach
• Graphs have no sort order
• Cannot rely on serialization order
• No access to sort position in CONSTRUCT
CONSTRUCT {
  ?s bs:sortValue0 ?sv0 .
} WHERE {
  SELECT ?s ?sv0 ?sv1 WHERE {
    ... bind s and sort values here ... 
  }
}
Paging

- Handled on the server side
- If paged query is also sorted, then the server needs to apply the sorting.
CONSTRUCT { 
  ?s bs:sortValue0 ?sv0 .
} WHERE {
  SELECT ?s ?sv0 ?sv1 WHERE { 
    ...
  } ORDER BY ?sv0 DESC(?sv1) OFFSET 10 LIMIT 10
}
• When sorting, projection includes sort variables
• If one entity has multiple possible bindings for a sort value you end up with multiple “distinct” solutions.
• Solution is to use MIN and MAX to ensure that only one value binds to each projected sort variable
CONSTRUCT { 
?s bs:s orbValue0 ?sv0 
?s bs:s orbValue1 ?sv1 
} 
WHERE { 
SELECT DISTINCT ?s 
(MAX(?sv0_) AS ?sv0) 
(MIN(?sv1_) AS ?sv1) 
WHERE { 
... bind s, sv0_ and sv1_ here ... 
} 
GROUP BY ?s 
ORDER BY ASC(MAX(?sv0_)) DESC(MIN(?sv1_)) }
ODATA / SPARQL INTEROP
“OData is a standardized protocol for creating and consuming data APIs”

odata.org
Entity-Centric

- Collection of Entity Sets
- Service metadata
  - List of entity sets
  - Schema for entities
- Primary Keys
- Properties & Associations
- Results are returned as entities by default
URL-based access

http://example.org/Films

http://example.org/Films(1234)

http://example.org/Films?$filter=Runtime lt 120

http://example.org/Films(1234)/Director

http://example.org/Films(1234)?$expand=Director,Actor
• POST to entity set’s URL

• PUT, PATCH, MERGE or DELETE to the edit URL of an existing entity

• Associations can also be exposed as a collection of link resources.
Reasons to Like OData

• Schema discovery
• Client tooling support (esp for .NET)
• Easy to experiment
• Easy to use from JavaScript
• Growing set of OData consumers
  – Data browsers
  – GUI controls
  – Applications
Criticisms of OData

- Services tend to be siloes
- No shared ontologies
- Perceived as a vendor-specific protocol
Motivation

• We like the features of OData
• We like the flexibility of RDF / SPARQL

• Goals:
  – Read access to open SPARQL endpoints via OData
  – Declarative configuration
  – Automatic configuration via RDF schema / SPARQL introspection
Implementation

Annotated OData Schema

OData Request

OData Proxy

OData Result

SPARQL Request

SPARQL Result

RDF Schema

SPARQL Endpoint

http://github.com/brightstardb/odata-sparql
Annotating OData Services

- Annotations are part of the OData spec.
- We define annotations for:
  - URI namespace for entity primary keys
  - URIs for entity types
  - URIs for properties and associations
  - Direction of associations
Implementation Issues

- Similar to LINQ-SPARQL
  - DESCRIBE is not an option
  - CONSTRUCTed graphs instead
  - Server-controlled paging for OData
  - Data-bind RDF resources to OData entities
Select a Single Film

**OData**
/Films('Un_Chien_Andalou')

**SPARQL**
CONSTRUCT {
} WHERE {
}
Enumerate Films

**OData**

/Films

**SPARQL**

```sparql
CONSTRUCT {
    ?v1 ?v1_p ?v1_o.
    ?v1 ods:variable-binding "v1"
} WHERE {
    ?v1 ?v1_p ?v1_o .
    {
        SELECT { ?v1 rdf:type o:Film . } LIMIT 100
    }
}
```
Property Navigation

**OData**

/Films('Un_Chien_Andalou')/Director

**SPARQL**

CONSTRUCT {
  ?v1 ?v1_p ?v1_o .
  ?v1 ods:variable-binding "v1" .
} WHERE {
  ?v1 ?v1_p ?v1_o .
  SELECT {
    ?v1 WHERE {
    } LIMIT 100
  }
}
Filtering

OData

/Films?$filter=runtime lt 120

SPARQL

CONSTRUCT {
  ?v1 ?v1_p ?v1_o .
  ?v1 ods:variable-binding “v1” .
}
WHERE {
  ?v1 ?v1_p ?v1_o .
  SELECT {
    ?v1 WHERE {
      FILTER(?v2 < 120) .
    } LIMIT 100
  }
}
Current State

• Proof of concept
  – Built on MS libraries for OData v3

• TODO:
  – Fill in some missing pieces of support
  – Update to a v4 parser / serialization library
  – Implement what we can of v4’s new features
  – Implement proxy caching of SPARQL results to improve performance
CONCLUSION
Conclusion

• RDF doesn’t have to be hard for developers to use
• Bridging the gap between Open and Closed World really helps
• This can apply both at the language level and the protocol level
http://BrightstarDB.com/

https://github.com/BrightstarDB/BrightstarDB

http://brightstardb.readthedocs.org/

@brightstardb
@kal_ahmed