Using RDF/OWL in Android

One of the major arguments in service oriented computing is that we need deeper representations of meaning to enable interoperation among services. The W3C recommended data model of RDF and its extension to the Web Ontology Language (OWL) help capture the semantics or the meaning of the data. In this assignment, you will learn how to use this data model for representation and reasoning on Android devices. **You don't need a physical device for this project.** The entire application will function within a device emulator that is packaged with the android SDK.

UPDATES

- **02-15-15:** Updated PO_new.owl file can be downloaded here (corrected an error resulting in two versions of Shipping).
- **02-21-15:** Grading rubric has been added.

Tools

- You will use Androjena for navigating the RDF document. Androjena is a porting of Jena semantic web framework to the Google Android platform, which provides an API to extract data from and write to RDF graphs. Please see Jena documentation for usage.
- (Optional) You can use Protege 4.1 (or an older version) for viewing the OWL ontologies. This tool provides a graphical means by which to view the OWL ontology that is needed for the assignment.

Tutorials and Useful Materials

- Jena RDF API
- Jena OWL Inference Reasoner
- If you have problems using Androjena, please check this forum http://code.google.com/p/androjena/issues/list

Project Summary

In this project, you will design an OWL ontology for online purchase orders. Then, you are expected to create a set of instances based on your ontology. Further, you need to check whether your data conform to the ontology you create.

Part 1

In this part, you are given an RDF file demoData.rdf, containing a few statements that assert some facts about Colin. Your task in this part is to load this RDF document into Android Jena model and learn to navigate through the model. In particular, you should learn how to iterate through the statements in the model. You will need this in part 2.

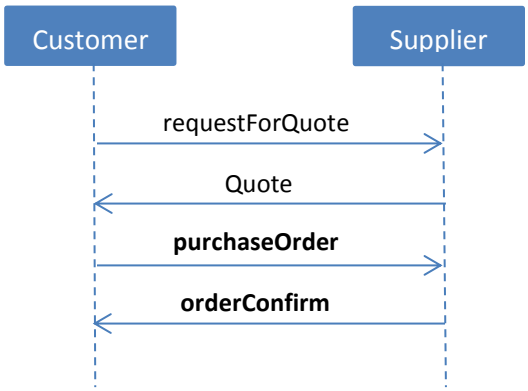
- Create a new Android project with the following values...

- Project Name: [unity_id]_soc_project2_part1
- Package Name: edu.ncsu.soc.project2.part1
- Activity Name: MainActivity
- Application Name: [unity_id] _part1
- Target: API level 16

- Download the latest binary release of Androjena and follow the instructions inside the README file to set up Androjena inside your Android project.
- Download the RDF document demoData.rdf. If you are using the Android emulator to test your application, you need to add the RDF document to the **assets** directory of the project.
- Please see this document to learn how to work with the Android file system to read and write files with the File APIs.
- Your job is to iterate through the statements in the model and print out the subject, predicate, and object of each statement.
- Add your code to the MainActivity.java file and upon success the output looks like this.

Part 2

In this part, you will design an OWL ontology for an online purchase order. Below is a typical online order processing scenario.



A customer agent sends a requestForQuote and receives a quote from a supplier agent. If the quote is accepted, the customer agent will sent a purchase order with specified information. The supplier agent will send back an order confirmation message after receiving and processing customer's purchase order. The figure above shows the interaction between one customer and one supplier. In fact, the customer may interact with different sellers, which incurs some challenges. For example, agents' information systems may not agree on the message content due to the heterogeneity in the messages. E.g., Europe suppliers use Euro values whereas US companies use inch-pound units. In this project, we will learn how Semantic Web can be used to resolve such heterogeneity issues.

Your task in this part is to design an OWL ontology for online purchase orders. Each time when the supplier agent receives a purchase order (which can be viewed as an instance of the ontology you created), the supplier agent will validate the purchase order according to the ontology created. If there is any invalidation, the supplier agent will send back a rejection message; otherwise, the supplier agent will send back a confirming message.

- **Requirements**

Your ontology must meet the following requirements.

- You will use an (incomplete) ontology which is available here as a starting point.
- Each purchase order should have an order number which is an integer. Please use **OrderNumber** as the property name.
- Each purchase order should have an order date which is of type date. Please use **OrderDate** as the property name.
- Each purchase order contains **only one** item.
- Each item has a quantity (an integer with non-negative value) associated with it specifying the number of items ordered. Please use **Quantity** as the property name.

- Use Protege to open the ontology and familiarize yourself with the classes and the properties in the ontology.
- You should edit the above ontology. Specifically,
  - don't add additional classes
  - add only properties mentioned in the requirements.
  - add additional attributes required to the existing properties.
- Please don't discuss what attributes to add for each property. If you make an assumption, write it in the README file.
- Now, you can begin coding the remainder of your assignment …

  - Create a new Android project with the following values...
    - Project Name: [unity_id]_soc_project2_part2
    - Package Name: edu.ncsu.soc.project2.part2
    - Activity Name: Checker
    - Application Name: [unity_id] _part2
    - Target: API level 16
  - Set up Androjena inside your Android project.
  - Read the OWL schema and the RDF files into your project. Here the RDF file refers to an OWL data document (i.e., an actual purchase order).
  - Bind the schema and create the inference model for validating.
  - If there are any violations, you should display a message saying "Invalid order" and a list of specific error messages such as "ERROR: Undefined Type - Instance instance_id member of class class_id."
  - If there is no violation, you should display a message like "Order confirmed." Further, you need to print out the following order information: OrderNumber, OrderDate, customer's first name and last name, item name, quantity, total price, and shipping address. Here, the total price can be calculated using Unit Price*Quantity.

Deliverables

- A README.txt file containing the project member's names, unity id's, section numbers, and any special instructions for your submission.
- A zip file containing the project and the APK file (in Eclipse, right click on the project and export as archive file). Note your zip file should also contain the owl file and the owl data document you generate based on your ontology.

Testing and grading

Your application will be tested for correct functionality. Roughly, the following will be the rubric.

- Your application should launch successfully. If it force closes or throws an ANR error, it will not be graded.
- (20 points) Syntax validation. We will do syntax validation of your OWL and RDF files.
- (40 points) Your application should be able to report the following violations:
    - Datatype incompatibility: e.g., assigning a string value to <OrderNumber>
    - Too many values: e.g., two OrderNumbers with different values
    - Cardinality restriction related: e.g., two <hasItem> properties with (explicitly stated) different item ids
- (40 points) Your application should be able to output the required order information when there are no violations as mentioned above in the data file.