MEng Group Project Deliverable 1: Requirements Specification and Background Research

Task Summary

Developing or extending a validator for Resource Description Framework (RDF) data, using a data description schema in the Shape Expressions (ShEx) language.
The RDF data itself may be in one of several serialization formats, but the focus should be implementing the most widely used formats.
A tool will be developed to validate data following the HCLS dataset descriptor, but should be built in such a way as to be useful to any project working with RDF data, with a management interface to add and update the ShEx schemas available for users to validate against.

Background research

RDF

With work starting in 1997, a specification published as a recommendation by the W3C in 1999 and an updated version in 2004, RDF is now a mature, standard model for data interchange on the Web.
RDF is based upon the idea of making statements about resources (in particular web resources) in the form of subject–predicate–object expressions. These expressions are known as *triples*.

RDF Serialization Formats

- Turtle (Terse RDF Triple Language), a compact, human-friendly format. Standardised by the W3C, http://www.w3.org/TR/turtle/
- N-Triples, a very simple, easy-to-parse, line-based format that is not as compact as Turtle. http://www.w3.org/TR/n-triples/
- N-Quads, a superset of N-Triples, for serializing multiple RDF graphs. Adds an optional fourth element to statements for *context*, labeling what RDF graph in a dataset a particular triple belongs to. http://www.w3.org/TR/n-quads/
- TriG, like Turtle but extended with the ability to group triples into multiple graphs and name those graphs. http://www.w3.org/TR/trig/
- JSON-LD, a JSON-based serialization. http://www.w3.org/TR/json-ld/
- RDF/XML, an XML-based syntax that was the first standard format for serializing RDF, and therefore has widespread references. http://www.w3.org/TR/rdf-syntax-grammar/
- N3 or Notation3, a non-standard serialization that is very similar to Turtle, but has some additional features, such as the ability to define inference rules. Not yet standardized by the W3C. http://www.w3.org/TeamSubmission/n3/

Shape Expressions (ShEx)

ShEx is a language for expressing constraints on RDF graphs, to communicate expected graph patterns and allow validation of RDF documents. http://www.w3.org/2001/sw/wiki/ShEx
As it has not yet been standardized, there is still open discussion about how to solve various problems, so implementation is still a somewhat moving target. Some such discussions can be found here: http://www.w3.org/2001/sw/wiki/ShEx/CurrentDiscussion
There are actually two syntaxes of ShEx, SHEXc (SHEX compact format) and SHEX/RDF, but as SHEX/RDF is still more of a work in progress we are focusing on SHEXc.
This project aims to take the most recent work in progress version of ShExc and implement a validator for RDF data in various serializations.

Semantic Web Health Care and Life Sciences (HCLS) Dataset Description

The HCLS Interest Group's mission is to develop, advocate for, and support the use of Semantic Web technologies across healthcare, life sciences, clinical research and translational medicine. http://www.w3.org/blog/hcls/
As such, a primary goal is standardizing a dataset description format using RDF, to make projects which use data from multiple sources possible.

One such project which would benefit from a standardized dataset descriptor format is the Open PHACTS Discovery Platform: http://www.openphacts.org, who currently have their own guidelines for RDF dataset descriptors:
http://www.openphacts.org/specs/2013/WD-rdfguide-20131007/

The current draft of the HCLS dataset description format is hosted by the W3C here: http://www.w3.org/2001/sw/hcls/notes/hcls-dataset/, and we will be building an up-to-date ShEx schema to validate data according to the currently defined specification.

Current Solutions

W3C Shape Expressions Demo

A variety of demonstrations and examples of ShEx are hosted by the W3C:
http://www.w3.org/2013/ShEx/

One of these demos is a functional ShEx validator - dubbed *FancyShExDemo* - allowing a user to paste in a ShEx schema and RDF data serialized in Turtle format, and immediately see validation results in the browser: http://www.w3.org/2013/ShEx/FancyShExDemo.html

This project is actively developed and maintained by W3C employee Eric Prud'hommeaux on his personal GitHub account: https://github.com/ericprud/ShExDemo

The tool is implemented entirely in javascript and uses PEG.js to generate the turtle and ShEx parsers. Given the implementation is written in JavaScript, and the similarities with what we are trying to achieve, it is an ideal starting point for our project. As such, we will fork this project and may re-use portions of code from it where suitable.

Issues with this solution:

- The UI does not make validation errors immediately obvious
- The validation errors are not clear to non-technical users
- The only supported RDF serialization format is Turtle

ShExcala

The ShExcala project: http://labra.github.io/ShExcala/ is a Scala based Shape Expression implementation that can be used to validate RDF using a ShEx Schema. The implementation supports several serialization formats; Turtle, N-Triples, RDF/XML and JSON-LD.
Written by Jose Emilio Labra Gayo, the project is open source:
https://github.com/labra/ShExcala and is the basis for the online tool RDFShape.

Given the solution is based on a Scala backend it would make it unsuitable to be hosted on a W3C web server, as they do not support any form of server backends. However it could possibly be converted to client-side Javascript using Scala.js.

Issues with this solution:

- Command line only tool based on Scala, makes it unusable from the restricted W3C web environment with no server-side scripting language available

RDFShape

RDFShape is basically a web frontend to ShExcala, also written by Jose Emilio Labra Gayo. Hosted privately here: http://156.35.82.103:9000, it has all the features of ShExcala with additional options for sourcing data from URIs, SPARQL etc.

Issues with this solution:

- Requires server-side component written in Scala, makes it unusable from the restricted W3C web environment with no server-side scripting language available
- The validation errors are not clear to non-technical users

HCLSValidator

HCLSValidator: https://github.com/AlasdairGray/HCLSValidator/tree/closed-shape is a project by Alasdair Gray which modifies Fancy ShEx Demo to have a hard coded schema to verify against rather than letting the user supply one. It is based off a fairly old version of Fancy ShEx

Demo. It includes a ShEx schema for HCLS validation which is out of date. The project does however provide a good ShEx schema to use as a starting point.

Issues with this solution:

- Hard-coded to use HCLS schema, no management interface
- ShExc syntax used is out of date
- The only supported RDF serialization format is Turtle

Requirements

Functional requirements

We have decided to extend work already performed by Eric Prud'hommeaux in his ShExDemo project. As such, we have forked his repository and we intend to build upon the work already performed in parsing ShEx schemas and validating from a technical perspective. We will add numerous features and improve interface design, with greater focus on user experience.

1. Parse data descriptions in the following formats
    a. **Must** support
        i. Turtle
        ii. TriG
        iii. N-Triples / N3
        iv. N-Quads
    b. **Could** support
        i. RDF/XML
        ii. JSON-LD
2. Validation
    a. **Must** validate parsed RDF data using a ShEx schema description
    b. **Should** have varying levels of strictness.
3. User Interface
    a. **Must** have validation result responses.
    b. The validation results **should** be either success, warning or error messages.
    c. **Should** be considered usable by test group. See *Evaluation*.
    d. **Could** extend or redesign existing user interface from FancyShExDemo.html
    e. **Could** highlight the location of the warning/error directly in the code. The user **should** have the ability to filter through varying levels of detail (verbosity).
    f. The user **must** have the ability to supply an input file. This **should** be done in at least or more of the following interactions:
        i. Pasting the content of data file directly into a textarea.
        ii. Selecting a data file to upload from the user's local machine. This **could** also be done by dragging a file over a textarea.
        iii. Entering a URL to download a data file from.

4. The web application **should** have a client-side management interface to generate server-side configuration files defining available schemas
5. The results of this project **could** supply a command line interface to users who wish to use the tool locally on their machine. This **should** allow the application to be used to validate data files as part of a tool chain.

Non-functional requirements

1. As agreed with our supervisor, our software **should** support the latest versions of the major browsers (Chrome, Firefox, Internet Explorer, Safari).
2. The web application **must** fully function on the W3C website. As a result the application **must** only utilise client-side technologies such as HTML, CSS and Javascript. It **won't** use server-side technologies for any tasks such as parsing or validation, so the application can be deployed to any web hosting environment with no server-side scripting language available.

Implementation

General Technologies

- Node.JS will be used to simplify pulling in existing code using NPM modules, e.g. N3.js
- Browserify will allow us to deploy our full Node application to client-side only code running in web browsers, allowing us to build a powerful application which fits in with W3C constraints.
- jQuery will be used to smooth over gaps between browsers and avoid reinventing the wheel, as Eric's existing code and many other tools use it extensively.

Parser Libraries

We will primarily implement our RDF parsing using N3.js, which already supports parsing Turtle, TriG, N-Triples and N-Quads. It does not support full Notation3 as the specification has not yet been standardized.

The following other libraries could possibly be used to support a greater range of formats:

- jsonld.js could be used to support data in JSON-LD format
- rdflib.js could be used to support data in RDF/XML format

Testing & Evaluation strategy

Throughout the project, we will perform Unit testing on the parsing and validation components and have decided to use Jasmine as our testing suite.
Integration tests will also be performed to check that all parser output is compatible with the validator and that every feature works together as expected.

In order to evaluate the finished product, we will conduct a usability test whereby participants will perform a series of tasks using our software and complete a questionnaire afterwards to help us evaluate their experience. This will go hand-in-hand with the requirements specification for the project, which we will use to compare with the final implementation. This comparison will allow us to determine how well we have accomplished our project goals.

Risk Analysis

	Helpful	Harmful
Internal	**Strength** ● Strong programmers ● Team familiarity ● Previous web experience ● Varied Skillsets ● Democratic team	**Weaknesses** ● Not sole focus of team members ● Little experience of project subject area ● Some team members with little/no JS experience
External	**Opportunities** ● Lots of relevant open source projects ● Workflow tools such as github and google docs ● Growth in RDF tooling and documentation	**Threats** ● Reliance of developing open source projects ● Technical limitations from W3 ● Project Deadlines ● Availability of supervisors

The greatest risks are likely to come about from the combination of project deadlines and the conflicts with team members' other commitments. We will try and address this with regular meetings and shared workspaces. There is also a potential for difficulties sharing the workload because of the cross dependencies in the implementation and group members different experiences. This should also be mitigated by close team cooperation and by creating clean interfaces between the UI and the validation/parsing parts of the project.

Management Plan

Project management will be handled in a direct democratic manner, with no appointed leader. All decisions will be made jointly, with all team members discussing the pros and cons of each decision and voting for a decision or action to be performed.

Communication

The group will communicate using a variety of methods, including a private group on Facebook, a GitHub group and group emails. The Facebook group will be used for organisational communication, such as coordinating meeting times and also for non-technical discussion. To coordinate allocation of programming tasks amongst the group, we will use Github's issues feature which will also facilitate technical discussion. Github's wiki pages will be used to store research collected by everyone in the group. Furthermore the we will have at least one weekly meeting internally, and one weekly meeting with the project supervisor.

Version Control

Collaboration on the project will use a Git repository hosted publicly on GitHub, where we have forked eric's ShExDemo repository. GitHub's Issues system will be used to record any tasks which need to be formed, or problems which need to be solved. These tasks can then be assigned to a group member to work on, and commentary/discussion relating to the task can written as comments on the issue thread until the issue is solved and marked as closed.
All team members will commit their changes to the central Git repository whenever a feature or significant development has been completed, with a useful commit message summarising their work to the team.

Time Tracking

Team members will log the time spent on tasks in a shared spreadsheet. At the weekly meeting, group members should agree how much time we individually should spend on the project, and what task each group member will focus on. The time spent by each team member the previous week will be shared with the group and passed on to the project supervisor.