Project List for SASSY

April 2022

1 Introduction

This is a list of all the projects that **might** be part of SASSY. It includes third party products such as PostgreSQL that are being evaluated, infrastructure projects that will form the backbone of SASSY, and a suite of RDF projects. There is then a set of projects that support the software architecture process, and finally a set of projects for designing SASSY itself.

The Infrastructure section lists the projects that provide the components upon which the remainder of the projects depend. These projects could form the basis of a useful set of RDF tools separate from SASSY.

The Report Generation section lists the projects that provide the ability to get nice readble reports from the RDF knowledge database. This is the most speculative part of the project, but without it SASSY fails in its basic purpose.

The Data Entry section lists projects that enable the user to enter the details of their design into SASSY. These should provide an engaging and efficient user interface that will encourage the use of SASSY.

The Software Architecture Support section is the heart of SASSY, its reason for existence.

The final section, SASSY Development, is where we build SASSY.

1.1 Development Overview

Some of the projects are quite speculative, and some involve unfamiliar third party programs and libraries that need to be trailed. Hence development will start with a feasibility study of these projects. See the *Feasibility Plan* for details.

The feasibility study will be followed by a prototype. A rough, simplified version of SASSY will be developed and then used to design the final version. This will be developed starting from the output and working backwards. This will give each step a well defined target to work towards.

The last stage is to use the prototype version to create the design and then develop an initial version. This initial version will then be steadily enhanced for the remainder of the SASSY's lifetime.

2 Infrastructure

The infrastructure projects provide a suite of RDF based tools that support the rest of the system.

2.1 CFI

Common Facilities Infrastructure
Common Development Infrastructure

A pair of libraries of useful odds and ends for large C++ projects. The CFI library is used by the other modules that require support for exceptions, strings, XML, configuration, child processes, plug-ins, logging and file descriptor streams.

The CDI library provides support for testing and tracing. Combined with the logging support tracing is available for multiple threads and multiple processes. In addition a program, trc2dot, can convert the trace logs into collaboration diagrams, which is very useful for documenting program operation.

Role: Underpins all programs in SASSY providing basic infrastructure and a framework for testing.

Status: Mostly complete. Needs a little more work on tracing and the creation of collaboration diagrams.

2.2 PostgreSQL

This will be the main storage mechanism for the RDF data. (Reference data, schemas and rules will be supplied as RDF/XML files.)

Status: Installed and working.

2.3 RDF

A library that provides a C++ interface for RDF. In addition it includes a catalogue component for managing a collection of RDF resources.

The RDF library, which is a wrapper around an existing C library called Redland, has been extended to include support for sub-models. This allows us to divide our RDF collection into small models that can then be combined as required.

Role: Provides the main data format for the information stored in the project and communicated between modules.

Status: Working version suitable for prototyping the rest of SASSY.

To Do: Full test harness and complete documentation.

2.4 RDFGUI

A GUI interface for testing RDF. Used as a preliminary version of graphical interfaces for RDF based libraries. It includes a basic interface for viewing RDF in a graph form, and an editor for schemas. A dynamic forms interface provides a way to enter RDF data, though it can be a bit tedious.

The Schema tab allows the full editing of the RDF schemas that SASSY will be using. These include some OWL extensions that allow reasoners to make inferences.

Role: Provides a platform for development and experimentation of GUI components of SASSY.

Status: Usable.

To Do: Lots. Please refer to the TODO List.

2.5 GUI Test Harness

This is a test harness for testing GUI programs. It was developed for testing VICI but can be moved to SASSY where it might become a general purpose test program for Qt based programs.

Status: The VICI version works but is incomplete.

To Do: Move to SASSY and add support for more widgets.

2.6 Reasoner

A program for making inferences on RDF data. It uses the FaCT++ library to perform the inference processing. It should be noted that FaCT++ relies on the OWL extensions to RDF that SASSY adds.

Role: Verify that a model has no contradictions, and infer additional statements into the database.

Status: Mostly done.

To Do: Needs to be added to SourceForge version control.

2.7 Rule Engines

A program for running a set of rules on an RDF model to generate a new model. The rules are stored as RDF which is converted into executable rules when the rules are run. Programs are also included that allow an ordinary text editor to be used for rule editing.

The rule engine supports user code for filters and functions that can be incorporated into the rules. It also supports recursive operation where rules can be run within rules.

Role: This is the basic processing mechanism for SASSY. Almost all processing will be performed by rule engines of one form or another.

Status: Under development.

To Do

2.8 SCAL

A C++ Actor library that might be useful for components of a server process.

Status: Working version.

2.9 TOIL

A small Forth like language that can be adapted to be the interface between a declarative language and the RDF models.

Status: Mostly written.

2.10 ZINC

A pair of libraries that provides a means for servers to report their status without being impacted by the clients.

Role: Monitoring the activity of servers.

Status: Working version.

2.11 VICI Interpreter

The interpreter library from the VICI project can be repurposed to run the rule engines.

Status: Working version, but needs to be pulled into the SASSY project and converted to use RDF instead of XML for its input.

3 Report Generation

This section lists the projects that involve converting RDF into documents describing the design of system.

3.1 Pandoc

A program for converting document formats.

Role: Converting MarkDown into the final document format which will normally be PDF but could be almost anything.

Status: Installed and working.

3.2 NLG

A library for creating simple English language sentences from a gramatical representation. This is based on the Java program SimpleNLG but has been redesigned internally [as well as being converted to C++].

The program starts by constructing a grammar tree, which is then updated to the correct English syntax. A morphology step then adjusts the words, or their endings, and finally an orthographic step adjusts the punctuation.

Role: Converting small fragments of RDF data into English for incorporation into an output document.

Status: Usable. Needs a bit of reworking and a proper set of tests.

To Do:

- Refactor as a rule for the rule engine.
- Modify to use an RDF lexicon.
- Generate MarkDown output.
- Handle text styles such as bold, parentheses and hyperlinks.

3.3 Lexicon

An RDF database of words. Probably based on WordNet and enWiktionary. The lexicon will be used to determine the part of speech (PoS) of words and for finding synonyms.

Since projects are likely to have their own vocabularies the lexicon should include a mechanism for adding any words necessary, and perhaps adjusting the defaults for PoS and synonyms.

Role: Used by NLG, and perhaps other modules, for the form of words.

Status: Seems like a good idea.

To Do

- Grab a copy of Wiktionary.
- Process into RDF
- Augment with data from WordNet

3.4 Content Planner

This is responsible for selecting the data that is to appear in a document. In effect it implements the "view point" for the document.

This module will use information supplied by the user to select data from the knowledge database for use in the document. The exact mechanism is not known at this stage but might include something like finding some specific RDF statements and then finding the path between those statements and then broadening it a bit.

3.5 Discourse Planner

This is responsible for organising the selected data into a logical structure. It will set up the sections, determine when to use tables, lists and diagrams, and allocate the RDF data to the paragraphs and other objects.

The RDF data will be assigned to classes from Rhetorical Structure Theory that are related structurally.

3.6 Microplanner

This uses a rule engine to convert normal RDF statements into a grammatical model that can be passed to the RDF interface of NLG. It works at the level of paragraphs and list clauses.

3.6.1 Reification

Each RDF statement that is selected as part of the input instance text will be converted to its reified form. This will introduce rdf:Statement nodes into the model which are more natural objects to be manipulating.

[Note that this task might have already been performed by the discourse planner, but it will be required if the microplanner is developed independently.]

3.6.2 Concepts

The terms used in the RDF of the instance text input need to be mapped onto terms that have a conceptual meaning that can be used to guide the text generation process.

This will involve using the labels provided by the user where available. Otherwise the URI will need to be interpreted. Links into the lexicon should be established so that the part of speech can be determined.

3.6.3 Discourse Structuring

This is responsible for determining the structure of a paragraph. This will use the ideas from Rhetorical Structure Theory. The text plan will be constructed from the data so that our reports do not omit anything that might be important.

3.6.4 Sentence Content Deliniation

This is responsible for aggregating the RDF statements into sentences.

3.6.5 Internal Sentence Organisation

This is responsible for things like setting the subject of a clause, and the relative subordination of the clauses.

3.6.6 Reference Choice

This is responsible for determining how objects are referenced. For example it should determine what can be replaced by pronouns.

3.6.7 Lexical Choice

Responsible for converting the concept terms into suitable words.

3.6.8 Grammar

This is responsible for setting all the grammar flags that the NLG will need.

3.7 Diagram Planner

This is responsible for converting a small RDF model into a structure representing a diagram.

3.8 Table Planner

This is responsible for creating tables of data from the RDF model.

3.9 Diagrammer

This is responsible for converting an RDF representation of a diagram into an image, probably via SVG.

3.10 Document Planner

This is responsible for setting up the basic format of the generated documents with things like ToC, index, front page, etc. and inserting the data ready for output.

3.11 Docgen

A program that converts an RDF description of a document into a Markdown description of a document.

Role: A core processing step in the creation of reports and other documents for SASSY.

Status: Mostly done.

To Do: Handle more Markdown features, such as hyperlinks. Also needs to be put under SourceForge version control.

3.12 Declarative Language

A language is required that will allow the user to define what they want in a document, the type of document, etc.

4 Data Entry

This section lists projects involved in entering the design into the SASSY RDF knowledge database.

4.1 Mind Map

A program for gathering ideas and information during the early stages of a project.

Role: Initial gathering of project information.

Status: Installed View Your Mind. It uses XML and I am not entirely convinced that reality is made of trees.

To Do

- Use it on a subproject to see how well the idea works.
- Perhaps modify it to use RDF rather than XML.

4.2 Data Entry Forms

This is a merging of the rule engine with the forms interface from RDFGUI. It might also be able to use the Discourse Planner and something like the Microplanner to create complex data entry forms.

Role: Allow the user to enter data for their particular project.

Status: Seems like a good idea.

4.3 Data Capture

This is the Data Entry program extended with the ability to easily add schema data.

The program should be efficient to use. For example it should be mostly usable from the keyboard with a minimum of mouse usage during text entry.

5 Software Architecture Support

5.1 Analysis Report

Create document generation rules for the analysis documents.

5.2 Requirements

This will involve creating a schema for a project's requirements and a dedicated version of Data Entry for entering and maintaining the requirements for a project.

5.3 Quality Attributes

This is a database of quality attributes that a project might need. It will be a reference data model.

5.4 Data Dictionary

Create a schema for organising all the terms used by a project. RDF makes it easy to provide cross references and more complex strucures for this document.

5.5 Project Tools

Create a schema for managing all the tools used by a project.

5.6 Requirements Report

Create document generation rules for the requirements, the data dictionary and the tools.

5.7 Tactics Reference

Create a reference database of well known software architecture tactics. This should include their strengths and weaknesses and the responsibilities that they entail.

5.8 Tactics Modelling

Create a schema for organising the tactics for a project. Include rules for discovering requirements that are not covered.

5.9 Module Modelling

Create a schema for organising the responsibilities and their implementing modules and tasks.

5.10 Interface Modelling

Create a schema for capturing the details of the interfaces between the modules.

5.11 Architecture View Reports

Create the document generation rules for creating various views of the architecture, such as the database view, the network view, the module hiearchy, and so on.

6 SASSY Development

6.1 SASSY Requirements

Use the prototype programs to construct a set of requirements, data dictionary, and use cases for SASSY.

6.2 SASSY Architecture Design

Use the prototype versions of the programs to create an architectural design for SASSY.

The following apply to each program

6.3 SASSY Server

One or more server processes will coordinate the processing of data and the generation of documents.

6.4 Interface Stubs

For each interface create facade classes for each module. Add stub code that allows each interface to be called and return default values. Define the abstract interface classes and stub interface classes for any callback style interfaces.

Create test harnesses that call these interfaces. These will evolve into test harnesses for the modules. The aim is to be able to develop each module in isolation from the other modules.

6.5 Test Planning

For every outcome of each use case, describe the test cases, this includes the number of tests, how the pre-condition state will be achieved, and how the post-condition state will be validated. Break this into sections for acceptance testing, system testing, integration testing and unit testing.

6.6 SASSY Detailed Design

Reverse engineer all the prototype programs to create proper design documents. Use tracing in the prototypes to generate collaboration diagrams showing the flow of control.

6.7 Tests

Create test cases for each program. Design the testing support that needs to be built into the programs.

6.8 Persistent Storage Design.

Design the RDBMS structure and complete all the RDF Schemas.

6.9 Coding

Review the prototype code looking for design patterns. Rewrite the programs while using the unit tests.