DSL Syntax for Bindi

Alex Hamilton-Smith

Australian National University, Australia
tlprofservteam@gmail.com

October, 2017

I. Introduction

When Shayne first suggested we develop a Domain-Specific-Language for the Bindi problem I was sceptical. Yes, Bindi requires as a first step the translation of sometimes complex natural language requirements to computation but this did not seem to me to be beyond what we could do with established programming languages. I suspected it would only complicate the issue to introduce a new language. As usual I'd understood one thing and misunderstood two others.

The Domain-Specific-Language this document aims to define is as simple as it can be and specific to the Bindi problem. The Bindi problem is that of formally defining requirements for Courses and Degrees/Programs and collecting a user plan before automating actions in response to problems Bindi detects.

By requirements we mean formal notation for Course level requirements:

1. course x is incompatible with course y
2. course x is a prerequisite for course y
3. course x is a corequisite for course y

And Degree/Program level requirements of a similar format:

1. Degree x requires compulsory courses y
2. Degree x requires y number of units
3. Degree x requires y number of units from a list, z, of courses

II. Definitions

In order to be sufficiently specific in the following sections we shall define some terms. First, we must refer to a data structure which contains a set of potential enrolments for a single student over the course of a finite number of semesters. This is the thing we inspect for errors. In line with the current ***Bindi data model***, we shall call this, the EnrolmentOutline.

Similarly, we shall take from the data model the idea of a CourseVersion and a CourseInstance. Regarding their difference, the CourseVersion is all non-temporally-specific information. For example this includes things like course name and course code which are the same across two different semesters. The CourseInstance contains reference to a specific semester in which the course was run and includes information such as the identity of the convenor.

Finally, our last important concept is that of the SubPlan. A subplan in our data model is defined as a set of rules, defined in this DSL, which are applicable under that subplan. To be somewhat more specific, a subplan could include any of the following:

1. Degree aka Program
2. Specialisation
3. Minor
4. Major
5. International Student Visa Requirements

6. Double Degree Requirements
7. Course Requirements (Pre(co)requisites or incompatibilities)
8. etc

III. Syntax

Specifically, we can say that each course or subplan contains some set of rules set out in the following syntax:

i. $req(co, cl, no\_unit, sem)$

This loosely says that our CourseOutline *co* must contain from the list *cl*, a quantity of units equal to or greater than *no_unit* prior to semester *sem*.

Parameters:

- *co* and *cl* are the only mandatory parameters, the rest are optional and have default values.
- *cl* may be a singleton and in practice often will be
- *no_unit* will default to 6 units if not specified
- *sem* will default to infinity, i.e. 'units before semester' need not be checked

ii. $incompat(co, cl)$

This simply says that our CourseOutline *co* must not contain any items from the list *cl*.

iii. $req\_units(co, no\_unit)$

This says that our our CourseOutline *co* must contain a quantity of units equal to or greater than *no_unit*.

- Note this could have been done using $req(co, cl, no\_unit, sem)$ with an all-encompassing *cl* but is included for its more concise notation and execution.

IV. Examples

The following examples illustrate use of the syntax in practice for real Programs and Course as of October 2017.

i. Masters of Computing

Click here for Programs and Courses listing.

1. $req\_units(co, 96)$
2. $req(co, [COMP6250, COMP8260, COMP6442], 18)$
3. $req(co, [COMP8715, COMP8755], 12)$
4. $req(co, [COMP*]^1, 36)$

ii. Masters of Computing Artificial Intelligence Specialisation

1. $req(co, [COMP8420, COMP8600, COMP8620, COMP8650, COMP8670, ENGN6528, COMP6260, COMP6262, COMP6320], 24)$
2. $req(co, [COMP8420, COMP8600, COMP8620, COMP8650, COMP8670, ENGN6528], 12)$

iii. Course Example: *COMP*8260

1. $req(co, [COMP6250, COMP8701], 6, enrol\_sem^2)$

V. Summary and Discussion

The proposed DSL rule syntax for the Bindi system is simple and yet powerful enough to describe requirements for Courses and Programs.

Some additional areas for consideration may include:

1. On processing of these rules we may expect to return a boolean for pass or fail. It may be worth considering the inclusion of

[1] Where COMP* refers to any COMP1234 with any variation of digits.

[2] This is the semester in which *co* contains *COMP*8260

human-readable feedback about the type and location of any errors, bearing in mind that a single course outline may contain many detectable problems.

2. Some Programs (such as the advanced versions) contain requirements regarding ongoing GPA. If this is to be assessed the collection of grades for each completed CourseInstance must also be assessed and consideration given to how to process GPAs for incomplete enrolments[3]. It may not be advisable to collect grades from non-advanced programs as data entry could be onerous and without benefit to the user for non-advanced cases, likely to be in the majority.

3. Some consider might be given to the creation of special lists for common and large collections such as:

   (a) All COMP codes.
   (b) All COMP1000 codes.
   (c) All COMP2000 codes.
   (d) All COMP3000 codes.
   (e) etc.

   These would for obvious reasons be handled better by a function rather than a set of manually updated lists.

4. Sometimes a subplan will mandate the inclusion of some other subplan. e.g. Master of Computing requires inclusion of a Masters of Computing Specialisation. COMP8260 requires Masters of Computing.]

   We may wish either to define these requirements in DSL rules or to instead apply them at the UX layer through constrained selections. e.g. Masters of Computing cannot be selected without also selecting a Specialisation from the relevant dropdown menu. The latter option is more user-friendly. "Why did the system allow me to pick an impossible combination at the first screen?"1

[3] Do we try to estimate future grades? Can we provide useful probabilistic information?

VI. Feedback

We're more than happy to accept any feedback on this document either at the email address at the top of this document or via issues on our gitlab. If you spot any errors or omissions please let us know.