BAB 4
ANALISIS

Bab ini berisi tentang analisis yang dilakukan penulis terhadap struktur objek pembelajaran untuk kebutuhan personalisasi, model ontologi yang diterapkan pada objek pembelajaran, serta *tools* yang sesuai untuk pengembangan prototipe *semantic portal* sebagai implementasi dari penerapan ontologi pada aplikasi berbasis *web*.

4.1 Analisis Objek Pembelajaran

Sebelumnya telah dipaparkan dalam Bab Konsep Pembelajaran, bahwa personalisasi *e-learning* dapat diterapkan pada aspek materi pembelajaran, baik bentuk maupun cara representasinya. Materi pembelajaran perlu dirancang sedemikian rupa sehingga mampu mendukung konsep personalisasi, dimana setiap peserta didik bisa memperoleh materi yang berbeda sesuai preferensi dan kebutuhannya pada suatu mata kuliah yang sama. Oleh karena itu, bentuk dan struktur materi pembelajaran menjadi wacana utama yang perlu dianalisis, sehingga dapat ditemukan format yang paling sesuai untuk digunakan dalam metode *e-learning* yang mendukung personalisasi.

Materi ajar yang digunakan dalam lingkungan pembelajaran *online* membutuhkan *metadata* dan sekumpulan aturan standar yang diikuti agar dapat memiliki interoperabilitas yang baik terhadap berbagai sistem *e-learning*. [22]. Penggunaan *metadata* dimaksudkan untuk memberi informasi tambahan sehingga objek pembelajaran dapat dengan mudah teridentifikasi dalam suatu proses *indexing* atau *searching*. *Metadata* ini juga yang akan digunakan untuk mendukung personalisasi objek pembelajaran.

Untuk membangun objek pembelajaran yang terstruktur, *metadata* yang digunakan harus mengikuti suatu standardisasi yang telah umum digunakan, misalnya LOM [12], Dengan kriteria ini, *learning resources* akan lebih mudah

untuk diatur oleh sistem yang terstandardisasi dan ditranslasi ke format *metadata* lain jika diperlukan.

Dalam tugas akhir ini, penulis mengikuti standar SCORM seperti yang telah dilakukan pada pengembangan sistem SHECAR [46]. Standar SCORM digunakan untuk menjaga kompatibilitas *content e-learning* terhadap berbagai *Learning Management System* (LMS), mengingat kebanyakan LMS saat ini juga telah mengikuti standar SCORM. Maka, struktur objek pembelajaran yang dirancang pun mengikuti standar yang didefinisikan dalam SCORM, termasuk *metadata* yang digunakan. Berikut penjelasan mengenai struktur beserta *metadata* yang digunakan untuk setiap unit objek pembelajaran.

a. Asset

Asset merupakan komponen terkecil dari objek pembelajaran yang mengandung isi materi ajar sebenarnya, baik dalam bentuk teks, gambar, audio, dan sebagainya. *Metadata* yang digunakan sebagai informasi tambahan pada *Asset* yaitu *name, keyword, contributor, creation date, last modified*.

b. Shareable Content Object (SCO)

SCO merepresentasikan unit seperti paragraf, *body*, referensi, tabel, kesimpulan, *section, list*, dan unit lainnya yang tersusun dari sekumpulan *asset* dan menjadi bagian dari suatu unit dokumen yang lebih besar. *Metadata* yang digunakan pada unit ini antara lain *name, description, keyword, contributor*.

c. Lesson

Unit ini didefinisikan sebagai dokumen yang menjadi bagian suatu mata kuliah. Terdiri dari dua jenis tipe dokumen yaitu *course material* dan *assessment material*. *Course material* merepresentasikan materi ajar berupa bab atau subbab dalam suatu mata kuliah. Sedangkan *assessment material* merepresentasikan dokumen untuk melakukan evaluasi terhadap hasil belajar peserta didik berupa tugas, *quiz*, atau ujian. *Lesson* merupakan wadah *(container)* yang terdiri dari satu atau lebih SCO. *Metadata* yang didefinisikan untuk *Lesson* antara lain *name, title, description, keyword, interactivity type, interactivity level, difficulty*.

d. Course

Course didefinisikan sebagai suatu mata kuliah yang dapat disusun oleh beberapa *Lesson*. *Course* dapat memiliki keterkaitan dengan *course* lain, misalnya hubungan prasyarat atau memiliki kesamaan bagian atau unit penyusunnya seperti pada *lesson, sco,* bahkan *asset*. Metadata yang digunakan diantaranya *name, description, contributor, difficulty, interactivity level, interactivity type*.

Tabel 4.1 merupakan ringkasan daftar *metadata* yang akan diimplementasikan dalam penelitian ini. Pemilihan *metadata* ini berdasarkan pendefinisian yang telah dilakukan pada penelitian sebelumnya (SHECAR).

Tabel 4.1 *Metadata* Objek Pembelajaran

Unit Objek Pembelajaran	Metadata yang digunakan
Course	*name, description, contributor, keyword, status, version, difficulty, interactivity level, interactivity type, creation date, last modified*
Lesson	*name, title, description, keyword, creation date, last modified, contributor, interactivity type, interactivity level, difficulty, status, version*
Shareable Content Object	*name, description, keyword, last modified, creation date, contributor, status, version*
Asset	*name, keyword, contributor, creation date, last modified*

4.2 Analisis Ontologi

Pada penelitian ini penulis menggunakan pendekatan ontologi pada struktur objek pembelajaran yang telah didefinisikan. Setiap unit pada objek pembelajaran akan direpresentasikan sebagai entitas *(class)* pada ontologi. *Metadata* yang telah didefinisikan juga akan tetap digunakan sebagai *properties* dari tiap *class* tersebut. Beberapa penambahan *property* selain *metadata* juga mungkin dilakukan untuk menyesuaikan kebutuhan pemodelan ontologi.

Penulis memilih untuk memodifikasi ontologi objek pembelajaran yang sudah pernah dibuat oleh para pakar di bidang *e-learning* dan tidak membuat dari awal agar lebih menghemat waktu, mengingat proses pembuatan ontologi bisa memakan waktu yang cukup lama. Lagipula, karena *ontology reuse* merupakan *reuse of domain knowledge*, maka penggunaan ontologi yang sudah ada akan sangat bermanfaat apabila nantinya dibutuhkan interaksi antar sistem yang dikembangkan pada domain yang sama [38]. Disamping itu, pemanfaatan ulang ontologi dan improvisasi yang dilakukan terus menerus oleh pengguna menjadi hal krusial dalam realisasi *semantic web* yang sedang berkembang.

4.2.1 *Ontology Mapping*

Berdasarkan penelitian yang dilakukan oleh Verbert dan Duval terhadap empat buah model objek pembelajaran yang umum digunakan (Learnativity, SCORM, CISCO[1], NETg[2]), dihasilkan sebuah pemetaan yang menggambarkan perbandingan struktur unit yang menyusun tiap model tersebut. Penelitian ini juga menghasilkan sebuah model baru yang mampu mengakomodasi perbedaan-perbedaan yang terjadi di antara keempat model tersebut, yaitu ALOCoM *(Abstract Learning Object Content Model)*. Hasil pemetaan pada penelitian tersebut dapat dilihat pada Tabel 4.2 berikut.

Tabel 4.2 Pemetaan Model Unit Objek Pembelajaran [29]

<table>
<tr><td>ALOCoM</td><td>Content Fragment</td><td>Content Object</td><td colspan="4">Learning Object</td></tr>
<tr><td rowspan="3">Learnativity</td><td rowspan="3">Raw Media Element</td><td rowspan="3">Information Object</td><td colspan="4">Application Specific Object</td></tr>
<tr><td colspan="4">Aggregate Assembly</td></tr>
<tr><td colspan="4">Collection</td></tr>
<tr><td>SCORM</td><td>Asset</td><td>Sharable Content Object</td><td colspan="4">Content Aggregation</td></tr>
<tr><td rowspan="3">CISCO</td><td rowspan="3">–</td><td>Content Item</td><td colspan="2" rowspan="3">Reusable Information Object</td><td colspan="2" rowspan="3">Reusable Learning Object</td></tr>
<tr><td>Practice Item</td></tr>
<tr><td>Assessment Item</td></tr>
<tr><td>NETg</td><td>–</td><td>–</td><td>Topic</td><td>Unit</td><td>Lesson</td><td>Course</td></tr>
</table>

[1] http://www.cisco.com
[2] http://www.netg.com/research/whitepapers/index.asp

Pada penelitian berikutnya, model ALOCoM yang telah dihasilkan tersebut direpresentasikan dalam bentuk ontologi. Ontologi ALOCoM kemudian cukup banyak diterapkan di berbagai pengembangan aplikasi *e-learning* berbasis *semantic web*. Pembahasan lebih detil mengenai struktur ALOCoM dan ontologinya akan dipaparkan pada subbab selanjutnya.

Dari hasil analisis penulis, ontologi ALOCoM merupakan pilihan yang paling baik untuk diadopsi dalam penelitian ini. Selain karena strukturnya yang *compatible* dengan model SCORM yang digunakan pada penelitian, *source code* untuk ontologi tersebut bisa dengan mudah didapatkan melalui situs resmi ALOCoM[3]. Dengan demikian, penulis dapat mempelajari ontologi dengan mudah dan memodifikasinya sesuai kebutuhan *portal* yang ingin dibangun.

4.2.2 ALOCoM *Ontology*

Sebelum membahas lebih dalam mengenai ontologi ALOCoM, penulis akan menjelaskan secara ringkas tentang struktur ALOCoM. Model ini memiliki tiga komponen objek pembelajaran, yaitu:

- *Content Fragment (CF)*

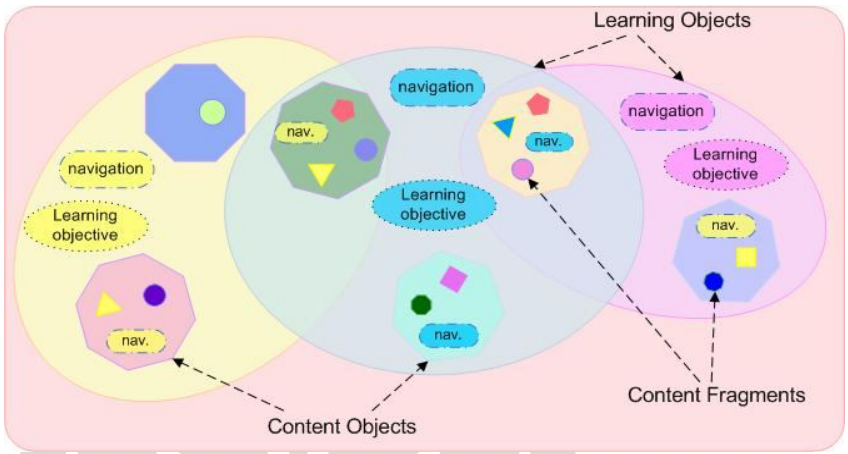
  CF adalah unit objek pembelajaran yang memiliki bentuk paling dasar, seperti teks, audio, atau video. Pada dasarnya, CFs merupakan *raw digital resources.* CF dispesialisasikan ke dalam dua bentuk, yaitu elemen *discrete* (teks, grafik, gambar) dan *continuous* (audio, video, simulasi, animasi).
- *Content Object (CO)*

  CO merupakan gabungan dari beberapa CFs dan memiliki navigasi. Elemen navigasi mengatur struktur atau urutan CFs dalam CO. Selain CF, CO juga bisa disusun dari CO lain.
- *Learning Object (LO)*

  LO merupakan unit terbesar dalam struktur objek pembelajaran. LO didefinisikan sebagai kumpulan dari beberapa COs dengan *learning objective* yang saling berhubungan.

[3] http://jelenajovanovic.net/ontologies/loco/alocom-core.rdf

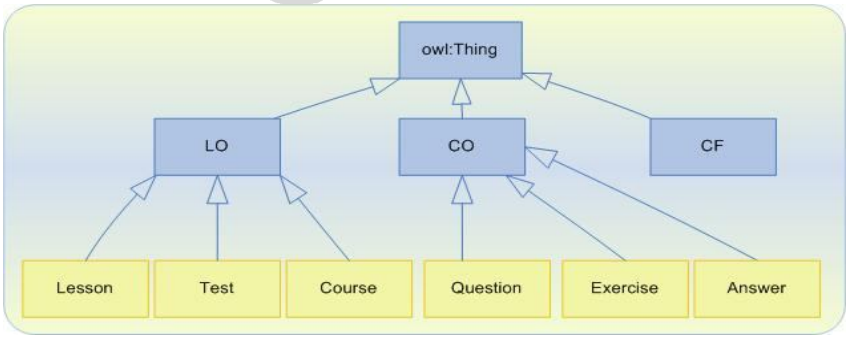
Gambar 4.1 memberikan ilustrasi mengenai struktur dan hubungan antar unit objek pembelajaran pada model ALOCoM.



Gambar 4.1 Struktur ALOCoM [30]

Dari definisi yang sudah diberikan, penulis dapat menyimpulkan bahwa *Content Fragment* bisa direpresentasikan sebagai *Asset*, *Content Object* sebagai *Shareable Content Object*, dan *Learning Object* sebagai *Content Aggregation* yang kemudian terbagi dalam dua bentuk yaitu *Lesson* dan *Course*.

Ontologi ALOCoM mendefinisikan tiga *class* utama yaitu `ContentFragment`, `ContentObject`, dan `LearningObject`. Ketiga *class* ini merupakan *subclass* dari `ContentUnit`. Gambar 4.2 menggambarkan konsep utama dari ontologi ALOCoM (tidak semua *class* dan *property* ditampilkan).



Gambar 4.2 Konsep Utama Ontologi ALOCoM [30]

Secara ringkas, ontologi ALOCoM memiliki 36 *classes*, 4 *object properties*, dan 7 *datatype properties*. Bahasa yang digunakan adalah OWL *(Web Ontology Language)*.

4.3 Analisis *Tools*

Dalam penelitian ini dibutuhkan setidaknya dua *tools* utama yang digunakan dalam proses pengembangan *semantic portal*. Pertama, *tool* untuk merancang atau membangun ontologi sebagai basis dari *portal*. Kedua, *tool* untuk mengembangkan *portal* itu sendiri.

4.3.1 *Tool* untuk Pengembangan Ontologi

Dari hasil studi literatur yang telah dilakukan, penulis menemukan tiga buah alternatif *ontology engineering tool* yang umum digunakan untuk membangun sebuah ontologi. Ketiga *tools* tersebut yaitu Protégé, Altova *Semantic Work*, dan SWOOP.

Protégé[4] adalah *open-source software* yang dikembangkan oleh sebuah organisasi yang bernaung di bawah Standford. Aplikasi ini digunakan untuk membuat sebuah domain ontologi, menyesuaikan form untuk entri data, dan memasukan data. Protégé menyediakan berbagai format penyimpanan seperti OWL, RDF, XML, dan HTML. Kemudahan *plug and play* yang disediakan dalam *tool* ini membuatnya fleksibel untuk pengembangan prototipe yang terus berkembang. Protégé menggunakan bahasa pemrogaman Java dan menyediakan fitur yang dapat dengan mudah digunakan melalui *Graphical User Interface* (GUI).

Altova *Semantic Work*[5] dibuat oleh perusahaan pembuat *sofware* Altova. Pengembangan ontologi dengan Altova *Semantic Work* menggunakan konsep visualisasi berupa gambar-gambar. Beberapa fungsi yang disediakan *tool* ini diantaranya pembuatan dan perubahan ontologi dalam format RDF, RDFS, dan OWL secara visual, pemeriksaan sintaksis untuk menyesuaikan kemampuan dengan spesifikasi RDF/XML, *auto generated* RDF/XML dan format N-triples,

[4] http://protege.stanford.edu
[5] http://www.altova.com/products_semanticworks.html

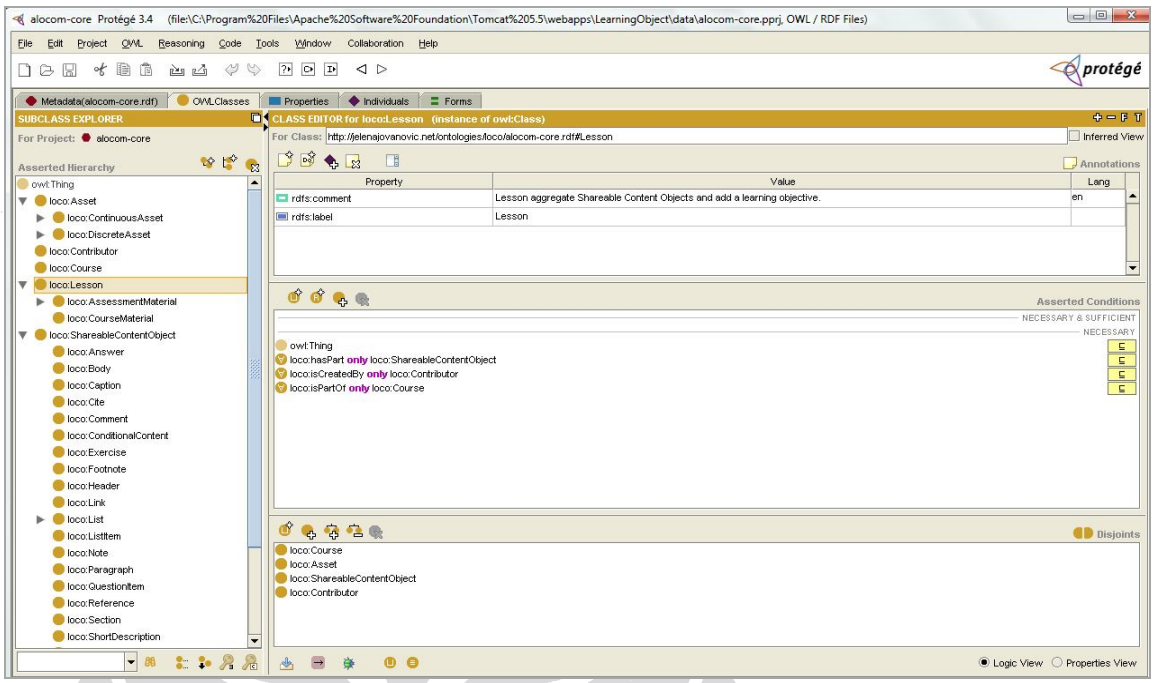
serta mencetak desain RDF/OWL berbentuk gambar untuk membuat dokumentasi *semantic web*.

Tool ketiga, SWOOP[6], berasal dari Mindswap yang merupakan organisasi yang bergeral di bidang *semantic web*. SWOOP dibuat dengan menggunakan bahasa pemrograman Java yang berbasis *Windows Base Application*. Tampilan *browser* untuk ontologi pada *tool* ini menyerupai *web browser*. Perubahan ontologi dilakukan dengan metode *inline,* yaitu semua perubahan yang dilakukan akan diikuti *class-class* yang mengikutinya. SWOOP dirancang untuk mengakomodasi kebutuhan OWL, termasuk RDF dan N3.

Berdasarkan pengujian yang telah dilakukan terhadap ketiga *tools* tersebut, Protégé merupakan *tool* yang paling direkomendasikan untuk pengembangan ontologi [53]. Beberapa faktor yang menjadi parameter dalam pengujian diantaranya kemudahan instalasi, kelengkapan dokumentasi, kemudahan penggunaan, kelengkapan fasilitas atau fitur yang ditawarkan, serta lisensi. Dari keseluruhan faktor tersebut, Protégé menempati urutan teratas yang hampir memenuhi semua parameter. Altova berada di urutan kedua dan SWOOP di urutan terakhir. Hal ini yang kemudian menjadi landasan bagi penulis untuk memilih Protégé sebagai ontologi editor dalam tugas akhir ini.

Protégé yang digunakan dalam tugas akhir ini adalah versi 3.4. Penggunaan Protégé dalam penelitian ini yaitu untuk melakukan modifikasi pada ontologi objek pembelajaran ALOCoM agar sesuai dengan struktur yang berstandar SCORM yang telah ditetapkan sebelumnya. Fitur-fitur yang dimanfaatkan pada penelitian ini diantaranya pembuatan *class*, *property, restriction,* serta *ontology test* untuk mengecek konsistensi ontologi. Untuk pembuatan *instances* data tidak menggunakan *tool* ini melainkan aplikasi lain, yang selanjutnya akan dibahas pada Bab Disain Ontologi dan Implementasi Portal. Gambar 4.3 memperlihatkan *screenshot* layar Protégé versi 3.4.

[6] http://www.mindswap.org/2004/SWOOP



Gambar 4.3 *Screenshot* Protégé 3.4

Protégé terdiri dari beberapa *tabs* untuk menampilkan *metadata* dari ontologi, OWL *Classes*, *Properties*, *Individuals*, dan *Forms*. Gambar 4.3 menunjukkan tampilan pada saat *tab* OWL *Classes* dipilih. *Window* bagian kiri merupakan *explorer* yang menampilkan daftar *Class* yang terdapat pada ontologi. Sedangkan *window* bagian kanan merupakan *editor* untuk melakukan perubahan pada *class*, berupa penambahan *property* atau *restriction.*

4.3.2 *Tool* untuk Pengembangan *Semantic Portal*

Pada tugas akhir ini, *semantic portal* dikembangkan menggunakan portalCore, sebuah *open-source software* dari proyek SWAD-E[7] yang mengembangkan SWED. Alasan pemilihan portalCore sebagai *tool* dalam penelitian ini antara lain [5]:

a. portalCore merupakan *family tool* dari Jena, sebuah *framework* berupa API *(Application Programming Interface)* untuk *semantic web* berbasis Java dan telah banyak digunakan dalam pengembangan aplikasi berbasis ontologi.
b. Efisiensi, karena penggunaan portalCore merupakan *software reuse* sehingga penulis tidak perlu mengembangkan *portal* dari awal. *Source code* portalCore

[7] portalCore diunduh dari http://www.swed.org.uk/swed/swed_technical_resources.htm

yang disediakan juga memudahkan penulis dalam mempelajari penggunaan aplikasi ini.

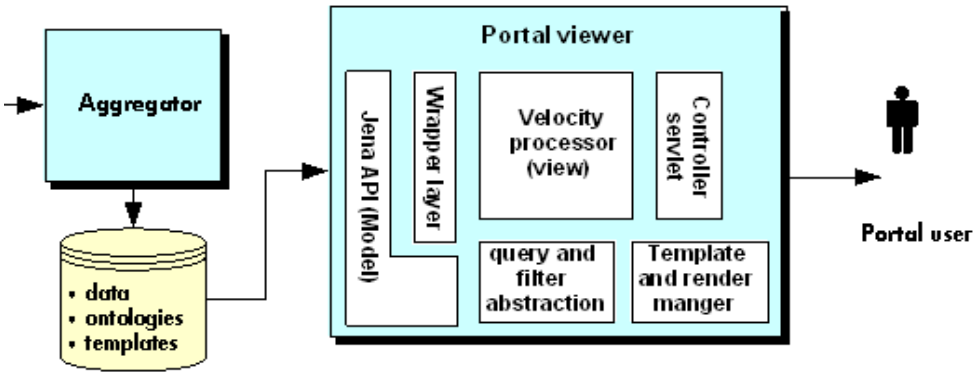
Selain portalCore terdapat beberapa alternatif lain yang dapat digunakan untuk mengembangkan *portal*. Alternatif pertama yaitu menggunakan Jena[8]; sebuah *framework* yang telah banyak digunakan dalam pengembangan aplikasi berbasis *semantic web*. Jena merupakan Java *Application Programming Interface* yang dirancang khusus untuk aplikasi berbasis *semantic web*. Jena dapat digunakan untuk membuat dan memanipulasi *RDF graphs.* Jena memiliki beberapa komponen utama, yaitu *resource interface* untuk merepresentasikan *resources, property interface* untuk merepresentasikan *properties, literal interface* untuk menggambarkan *literals,* dan *model interface* untuk menggambarkan *graph* RDF. Namun, penulis menemui kesulitan dalam menentukan metode penerapan Jena yang sesuai untuk mengembangkan sebuah *portal*. Awalnya, penulis ingin menggunakan Eclipse[9] sebagai *tool* untuk mengimplementasikan Jena dalam sebuah *Java project*. Namun metode ini ternyata bisa memakan waktu yang cukup lama karena harus melakukan pemrograman Java secara manual. Selain itu konfigurasi yang harus dilakukan untuk bisa mengintegrasikan antara fungsi dan tampilan *portal* juga tergolong kompleks.

Alternatif lain adalah menggunakan Protégé *Web Browser*[10]. Aplikasi ini menampilkan ontologi pada *knowledge base* melalui *browser* sehingga dapat diakses melalui jaringan. Namun, penulis menemukan beberapa kendala dalam penggunaan *tool* ini. Pertama, penyajian informasi pada aplikasi ini sepertinya lebih ditujukan untuk pengguna yang sudah memahami ontologi karena pemakaian kata *classes, instances,* dan *prefix* pada konsep yang dapat membingungkan pengguna. Kedua, informasi yang ditampilkan tersimpan pada *project files* Protégé sehingga formatnya masih sangat terbatas. Akhirnya penulis menyimpulkan bahwa aplikasi ini belum mendekati konsep *semantic portal* yang diharapkan dan membutuhkan waktu yang lebih lama untuk penyesuaian.

[8] Jena dapat diunduh di http://jena.sourceforge.net/downloads.html
[9] http://www.eclipse.org
[10] http://protege.cim3.net/cgi-bin/wiki.pl?ProtegeWebBrowser

Struktur portalCore

portalCore menggunakan pendekatan MVC (*Model-View-Controller*). Gambar 4.4 menunjukkan struktur portalCore yang memiliki komponen utama berupa portal viewer, yang menerima input berupa ontologi, data RDF, dan *templates*.



Gambar 4.4 Struktur portalCore [48]

Model disajikan oleh sejumlah Java *classes* yang memiliki keterkaitan, yang menggunakan *library* Jena untuk membungkus data (ontologi dan *instance* data). Data dapat berasal dari *multiples files* ataupun *database*. *View* menggunakan *Velocity template engine* untuk menghasilkan halaman *portal*. *Controller* berupa Java *servlet* dengan sejumlah *built in actions* dan memiliki kemampuan untuk memanggil *Velocity template* yang bersifat dinamis.

Komponen portalCore

Terdapat empat grup komponen *model* pada pendekatan MVC [47], yaitu:

a. Filters and facets

Facets ialah kelompok atribut yang digunakan dalam pencarian informasi pada *portal*. *Facet Java Interface* digunakan untuk merepresentasikan definisi dari sebuah *facet*. *FacetState interface* mendefinisikan batasan pencarian untuk sebuah *facet*. Sedangkan sebuah *FilterState* merupakan kumpulan dari *FacetStates* (satu untuk setiap *facet* yang diatur untuk *portal)*.

b. Datasources and stores

Objek *DataSource* mengenkapsulasi semua informasi yang dibutuhkan untuk konfigurasi portal dan menyediakan akses terhadap ontologi dan *instance* data melalui sebuah abstraksi *DataStore*. *DataStore* menyimpan data dengan menggunakan sebuah implementasi yang ditawarkan oleh Jena *MultiModel interface*. Ada dua jenis implementasi *DataStore* yang disediakan, menyimpan data pada sebuah *database* atau pada *files* yang tersimpan dalam memori.

c. RDF (resource) wrappers

portalCore dibangun di atas Jena *framework* yang menyediakan berbagai *interface* untuk memanipulasi data RDF dan OWL. RDF data dapat diakses melalui *set of wrapper objects* yang membuatnya lebih mudah untuk dimodifikasi.

d. Rendering support

Bagian ini merupakan sebuah objek tunggal, the VMRendermanager, yang menyediakan sekumpulan fungsi yang beraneka ragam untuk membantu proses *generate* halaman tampilan dari data.

portalCore adalah *tool* untuk membangun *faceted-browsing interfaces portal* yang berasal dari kumpulan data RDF [40]. *Interface* yang ditampilkan menggunakan *multiple-dimension* (*facets*) untuk mengakses informasi pada *portal*. Hanya satu tipe objek yang ditampilkan untuk satu kumpulan *facets.* Selain melalui *browse,* akses informasi pada *portal* juga bisa dilakukan melalui *text search* yang menggunakan *Lucene engine* untuk mengindeks data RDF berdasarkan nilai *property*. *Query* pada portalCore menggunakan RDQL (RDF *Query Language*), *predecessor* SPARQL yang sudah direkomendasikan oleh W3C. *Query* RDF sudah diabstraksi dalam portalCore sehingga tidak perlu membuat *query* secara langsung untuk mendapatkan data.

Selain komponen *portal viewer,* masih ada komponen *aggregator* pada portalCore yang menyediakan layanan *scanning* atau sebagai *harvester* data RDF secara periodik agar tetap *update.* Namun, dalam penelitian ini komponen tersebut belum dimanfaatkan mengingat pengembangan *portal* yang dilakukan masih berupa prototipe.

BAB 5
DISAIN ONTOLOGI DAN IMPLEMENTASI PORTAL

Pada bab ini dijelaskan secara spesifik langkah-langkah yang dilakukan penulis dalam membangun prototipe *semantic portal*, mulai dari tahap perancangan model ontologi, persiapan data, hingga implementasi model ontologi pada *semantic portal* untuk mendemonstrasikan penggunaan ontologi tersebut secara langsung dalam sebuah aplikasi berbasis *semantic web*.

5.1 Kerangka Pengembangan

Alur pengembangan prototipe *semantic portal* dalam penelitian ini terbagi menjadi tiga bagian utama, yaitu:

a. Input data *portal*

   Bagian ini merupakan tahap inisiasi yang harus dilakukan sebelum *semantic portal* benar-benar mulai dibangun. Tahap ini terdiri atas persiapan data dan pendefinisian *rules*.

b. Konfigurasi *portal*

   Tahap ini terdiri atas tiga bagian yaitu pendefinisian *datasources*, *facets*, dan *templates*.

c. Tampilan *portal*

   Tahap akhir ini merupakan proses untuk memvisualisasi tampilan antarmuka *semantic portal*. Tahap ini meliputi pembuatan *templates* untuk menampilkan data *resources* dan pengaturan *look-and-feel*.

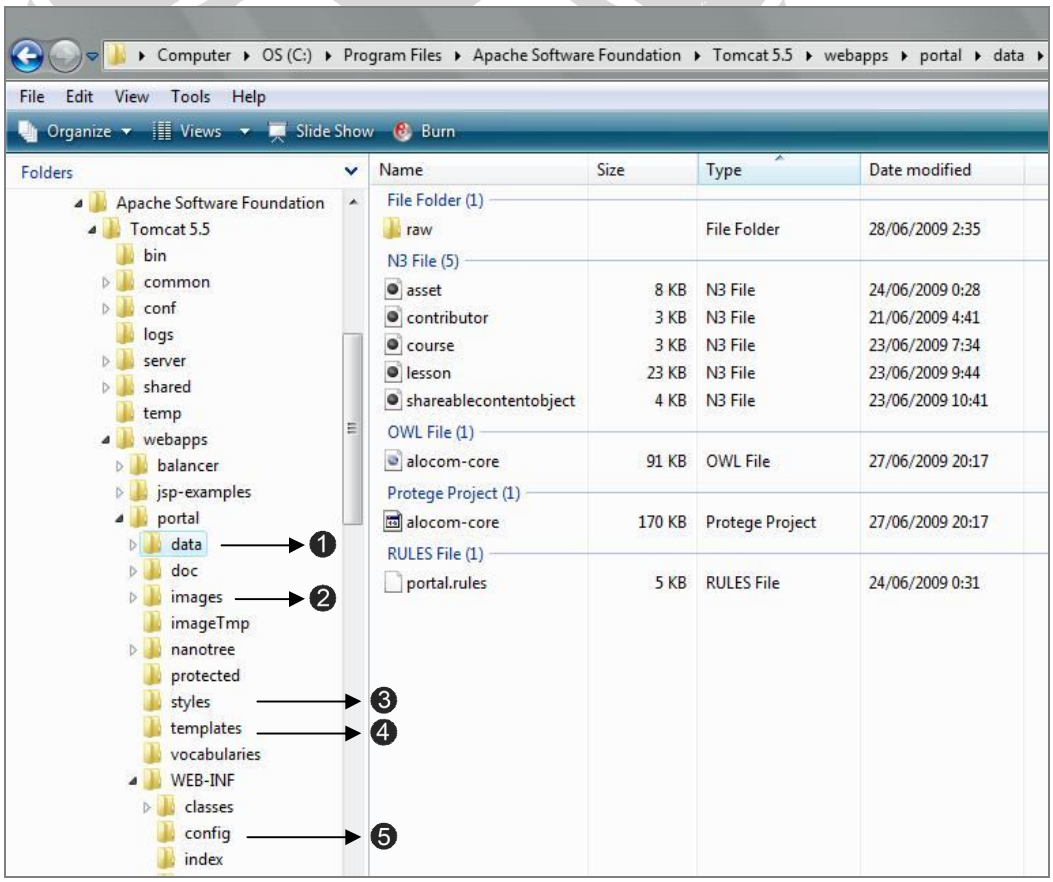
Bila diilustrasikan, alur pengembangan prototipe *semantic portal* dalam penelitian ini tampak seperti pada Gambar 5.1 berikut.

Gambar 5.1 Alur Pengembangan *Semantic Portal*

Langkah-langkah yang dilakukan pada alur pengembangan tersebut sebenarnya bukan merupakan garis lurus melainkan dapat terjadi iterasi atau pengulangan ke tahap sebelumnya. Waktu untuk mengeksekusi sebuah tahap bisa jadi berbeda dengan tahap lainnya. Pun kompleksitas kerja yang dilakukan di tiap tahap bisa bervariasi. Penjelasan tiap tahap yang dilakukan penulis selama penelitian ini akan dipaparkan pada subbab-subbab selanjutnya.

5.2 Struktur Direktori

Sebelumnya telah dipaparkan dalam bab analisis bahwa implementasi prototipe *semantic portal* pada portalCore bersifat *reuse*, bukan *build-from-scratch*. Oleh karena itu, sebelum membahas lebih jauh mengenai langkah-langkah detil yang dilakukan pada tahap perancangan dan implementasi, penulis akan menunjukkan terlebih dahulu komponen apa saja pada portalCore yang mengalami modifikasi dalam proses pengembangan *semantic portal* pada penelitian ini.



Gambar 5.2 Struktur Direktori *Portal*

Bagian ini memperlihatkan struktur direktori dari *portal* dan lokasi *file* yang mengalami modifikasi oleh penulis. Ilustrasi yang ditunjukkan pada Gambar 5.2 merupakan struktur direktori dimana semua *file* yang dibutuhkan untuk menjalankan *semantic portal* disimpan.

Pada penelitian ini terdapat lima buah direktori yang berhubungan dengan *file* yang diproses pada setiap tahapan pengembangan *portal*, yaitu:

1. \data

Direktori ini merupakan lokasi *files* yang dikerjakan pada tahapan input *portal*, yaitu file ontologi (alocom-core.owl), lima buah file *intances* data (contributor.n3, course.n3, lesson.n3, shareablecontentobject.n3, asset.n3), dan file *rules* (portal.rules). Terdapat pula *file* .csv dan *map graph* yang digunakan dalam proses *generate* RDF untuk menghasilkan data *.n3 yang disimpan dalam \data\raw.

2. \images

Direktori tempat menyimpan *image* atau gambar yang digunakan dalam *semantic portal*, baik untuk keperluan tampilan antarmuka maupun *resources* yang berbentuk *image* maupun grafis.

3. \styles

Lokasi dari *file cascading style sheet* yaitu site.css yang berisi konfigurasi untuk mengatur tampilan antarmuka dari *semantic portal*.

4. \template

Direktori dimana *files* untuk melakukan pengaturan *template* (*.vm) disimpan.

5. \config

Pada tahap konfigurasi *portal* yang meliputi pendefinisian *datasource*, *facets*, dan *templates*, *file* sources.n3 yang mengalami modifikasi disimpan pada direktori ini.

5.3 Input *Portal*

Tahap pertama pada proses pengembangan prototipe *semantic portal* dalam penelitian ini yaitu mempersiapkan berbagai data yang diperlukan sebagai input sistem. Data yang menjadi input diantaranya model ontologi itu sendiri, *instances* data, serta *rules* untuk mendefinisikan aturan-aturan pada *resources*.

5.3.1 Perancangan Model Ontologi

Bagian ini berisi penjelasan mengenai proses modifikasi yang telah dilakukan oleh penulis terhadap ontologi objek pembelajaran *ALOCoM ontology* berdasarkan kebutuhan struktur objek pembelajaran yang telah didefinisikan sebelumnya pada Bab Analisis.

Modifikasi yang dilakukan diantaranya perubahan istilah *class*, perubahan definisi *class*, penambahan *class*, pengurangan *class*, serta penambahan *property*. Tabel 5.1 menunjukkan daftar modifikasi yang telah dilakukan penulis dalam penelitian ini.

Tabel 5.1 Modifikasi Ontologi

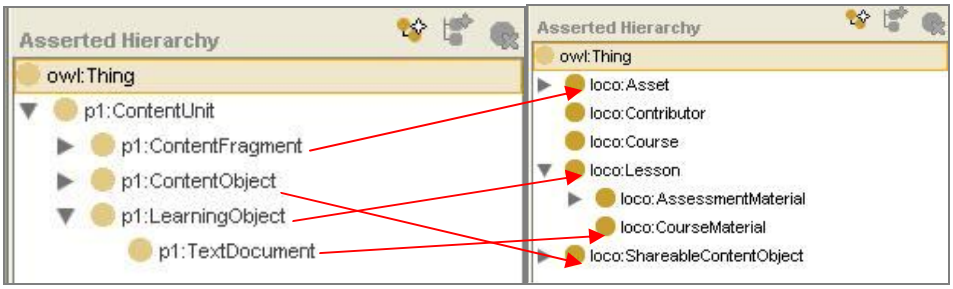
Jenis Modifikasi	Keterangan
Mengubah istilah *class*	mengubah nama *class* `ContentFragment` menjadi `Asset`
	mengubah nama *class* `ContentObject` menjadi `ShareableContentObject`
	mengubah *class* `LearningObject` menjadi `Lesson`
	mengubah *class* `TextDocument` menjadi `CourseMaterial`
Mengubah definisi *class*	memperluas definisi *class* `Lesson`, tidak hanya sebagai materi ajar namun termasuk juga unit penilaian atau evaluasi belajar peserta didik
Menambah *class*	menambah *class* `Contributor` untuk menyatakan siapa yang berkontribusi dalam pembuatan *learning material*
	menambah *class* `Reference` sebagai salah satu jenis `ShareableContentObject`
	menambah *class* `Answer` dan `QuestionItem` untuk mendefinisikan pertanyaan dan jawaban pada *class* `AssessmentMaterial`
	menambah *class* `Exercise` sebagai bagian dari `ShareableContentObject`
Mengurangi *class*	menghilangkan *class* `ContentUnit`
Menambah *property*	*property* `isCreatedBy` dan `hasCreate` sebagai *inverse property*-nya
	`hasQuestion` untuk menyatakan relasi antara *class* `AssessmentMaterial` dan `Exercise` dengan `QuestionItem`
	`hasCandidateAnswer` menyatakan relasi antara *class* `QuestionItem` dengan `Answer`
	`isAbout` mendefinisikan `CourseMaterial` yang terkait

	dengan suatu `AssessmentMaterial`
	`prerequisite` untuk menyatakan `course` yang menjadi prasyarat dari `course` lain
	`name, address, phone, photo, role, email` sebagai data profil `Contributor`
	`asset_created, lesson_created, sco_created, course_created` untuk mendapatkan data kontribusi dari tiap `contributor` berdasarkan tipe objek pembelajaran
	`description, keywords, order, status, version, date_created, last_modified` sebagai metadata objek pembelajaran untuk keperluan personalisasi
	`difficulty` menyatakan tingkat kesulitan dari sebuah objek pembelajaran
	`interactivityLevel` untuk menyatakan tingkat interaktifitas suatu objek pembelajaran
	`interactivityType` untuk menyatakan tipe interaktifitas dari suatu objek pembelajaran
	property `part` untuk mengklasifikasikan tiap *chapter* materi ke dalam bagian yang lebih umum (berdasarkan kesamaan topik)
	property `content` untuk menggabungkan *resource* dari properti `hasPart` dan `hasQuestion` pada `AssessmentMaterial`
	property `label` untuk menampilkan nama resources pada portal

Berikut penjelasan lebih mendetil dari perubahan atau modifikasi yang dilakukan terhadapa ontologi objek pembelajaran ALOCoM.

a. Perubahan istilah beberapa *class*

Karena ALOCoM menggunakan beberapa istilah yang berbeda untuk mendefinisikan unit objek pembelajaran seperti yang digunakan dalam SCORM, maka penulis perlu melakukan beberapa perubahan yang sesuai dengan pemetaan antara kedua model berdasarkan persamaan definisi. Gambar 5.3 memperlihatkan perubahan istilah *class* yang dilakukan, diikuti dengan penjelasan untuk setiap perubahan.



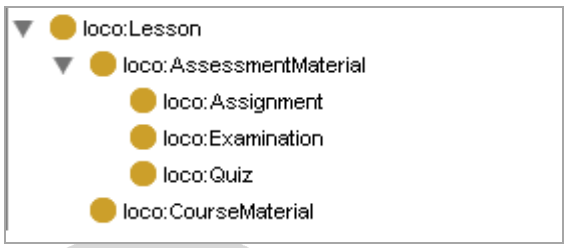
Gambar 5.3 Perubahan Istilah *Class*

- Mengganti istilah `ContentFragment` dengan `Asset`. Keduanya memiliki definisi yang sama, yaitu sebagai unit terkecil dalam sebuah struktur objek pembelajaran.
- Mengganti istilah `ContentObject` dengan `ShareableContentObject`. Kedua *class* ini juga memiliki definisi yang sama sebagai unit yang terdiri dari beberapa *asset* atau *content fragment* dan bersifat *shareable* atau bisa digunakan di beberapa *learning material* yang berbeda.
- Mengganti istilah `LearningObject` dengan `Lesson` karena definisi kedua *class* yang sama yaitu sebagai wadah *(container)* yang tersusun dari beberapa `ShareableContentObject`.
- Mengubah *class* `TextDocument` menjadi `CourseMaterial`. Yang dimaksud dengan `TextDocument` adalah unit bahan ajar yang berisi materi pembelajaran untuk peserta didik yang berbentuk dokumen teks. Pada pengembangan ALOCoM selanjutnya, selain dokumen teks, bahan ajar juga bisa didefinisikan dalam bentuk *slide* presentasi, laporan, ataupun bahan *assessment.* Sedangkan dalam penelitian ini, materi ajar yang dimaksud nantinya hanya akan disajikan dalam satu bentuk yaitu *web page* berbasis XML/HTML. Oleh karena itu, penulis memilih istilah yang lebih umum yaitu `CourseMaterial`.

b. Perubahan definisi *class* `Lesson`

Pada ontologi ALOCoM, *class* `LearningObject` yang ekuivalen dengan *class* `Lesson` hanya memiliki satu jenis tipe dokumen yaitu `TextDocument`. Pada penelitian ini penulis memperluas definisi `Lesson` dengan membagi kelas tersebut ke dalam dua kategori tipe dokumen pembelajaran, yaitu `CourseMaterial` dan

`AssessmentMaterial`, seperti yang terlihat pada Gambar 5.4. `CourseMaterial` adalah dokumen yang berisi materi yang disampaikan kepada peserta didik untuk dipelajari, sedangkan `AssessmentMaterial` merupakan dokumen yang berisi pertanyaan-pertanyaan untuk mengevaluasi hasil belajar para peserta didik.



Gambar 5.4 Definisi *Class* Lesson

Class `AssessmentMaterial` ditambahkan sebagai *subclass* dari `Lesson` yang *disjoint* dengan *class* `CourseMaterial`. Dokumen `AssessmentMaterial` kemudian dibagi lagi ke dalam tiga kategori yaitu `Assignment`, `Quiz`, dan `Examination`. Hal ini disesuaikan dengan bentuk komponen penilaian yang biasanya terdapat dalam suatu perkuliahan yaitu pemberian tugas, kuis, dan terakhir ujian.

c. Penambahan *class*

Terdapat beberapa *class* yang ditambahkan pada ontologi untuk menyesuaikan dengan kebutuhan sistem *e-learning* yang ingin dibangun.

- Menambah *class* `Contributor`. Dalam konsep sistem yang ingin dibangun, seorang pengajar dapat membuat lebih dari satu unit objek pembelajaran yang berbeda. Suatu mata kuliah juga dapat disusun bersama-sama dari kontribusi beberapa pengajar dalam membuat unit penyusun yang lebih kecil misalnya bab, *section*, atau bahkan unit terkecil sekalipun seperti gambar atau teks pada suatu materi ajar. *Class* ini berfungsi menyimpan data mengenai profil pembuat unit objek pembelajaran tersebut, tidak lain ditujukan untuk keperluan otorisasi materi ajar.
- Menambah *class* `Reference`, `Exercise`, `Answer`, dan `QuestionItem` sebagai *subclass* dari *class* `ShareableContentObject`. Tujuan penambahan ini untuk

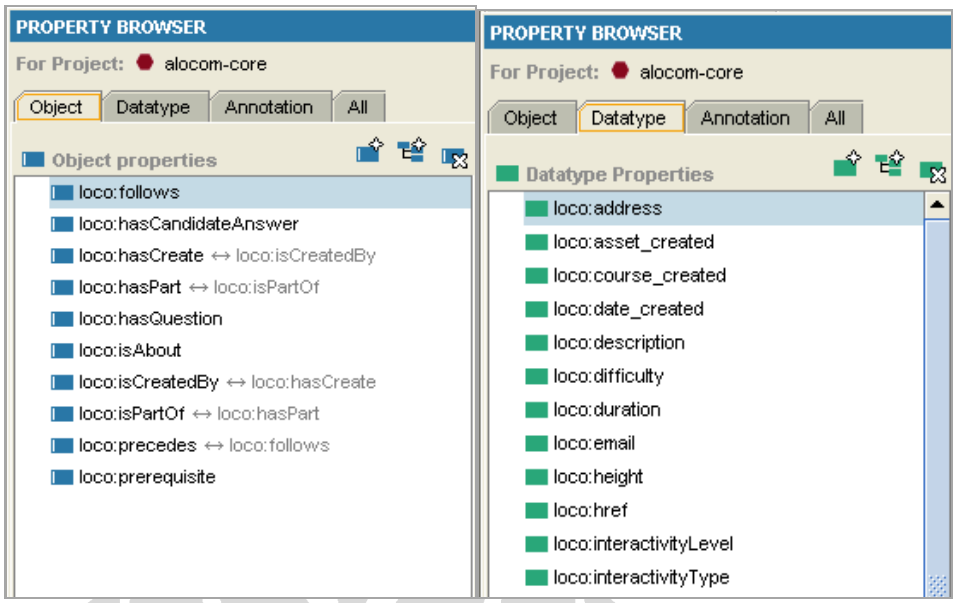
melengkapi jenis-jenis SCO yang mungkin terdapat pada suatu bab materi pembelajaran, yang belum didefinisikan pada ontologi ALOCoM. `Reference` mendefinisikan referensi yang dipakai dalam suatu materi ajar. `Exercise` berupa latihan di akhir bab untuk melatih atau me*review* materi yang baru saja diajarkan, `QuestionItem` mendefinisikan *item* pertanyaan dalam suatu `Exercise` ataupun `AssessmentMaterial`. Sedangkan `Answer` berisi teks yang menjadi kandidat jawaban dari `QuestionItem`.

d. Penghilangan *class* `ContentUnit`

Pada ALOCoM, ketiga jenis unit objek pembelajaran `ContentFragment`, `ContentObject`, dan `LearningObject` dikelompokkan ke dalam satu kelas besar yaitu `ContentUnit`. Hal ini dilakukan untuk kebutuhan integrasi ontologi ALOCoM dengan ontologi lain yang juga terkait dengan *e-learning* seperti *learning design* dan *student model ontology*. Namun, dalam penelitian ini yang dibutuhkan hanya ontologi untuk menggambarkan struktur materi dari objek pembelajaran sehingga penulis merasa tidak perlu menggunakan `ContentUnit` sebagai *container*.

e. Penambahan *property*

Penulis menambahkan sejumlah *property* pada ontologi ALOCoM untuk menambahkan informasi yang berfungsi sebagai *metadata* dari tiap unit objek pembelajaran, yang telah disesuaikan dengan standar SCORM. Gambar 5.5 memperlihatkan daftar beberapa *object* dan *datatype properties* yang didefinisikan pada ontologi ini.

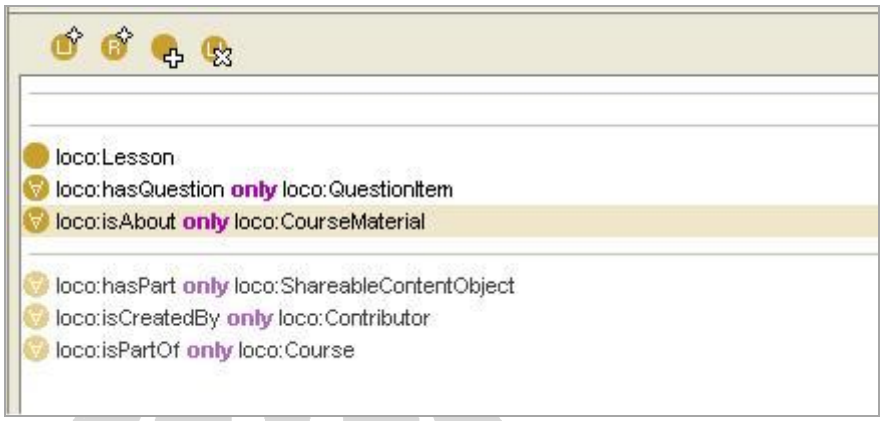


Gambar 5.5 *Datatype* dan *Object Properties*

Berikut penjelasan dari tiap *property* yang ditambahkan.

- *Object property* `isCreatedBy` dan `hasCreate` *(inverse property)* untuk menyatakan relasi antara class `Contributor` dengan tiap unit objek pembelajaran: `Course`, `Lesson`, `ShareableContentObject`, dan `Asset`. Properti ini menghubungkan *instance* dari class `Contributor` dengan *instance* dari tiap unit untuk memberikan informasi siapa yang telah membuat *instance* dari masing-masing unit objek pembelajaran tersebut.
- *Property* `hasQuestion` untuk menghubungkan antara *instance* dari *class* `AssessmentMaterial` dan *class* `Exercise` dengan *instance class* `QuestionItem` yang berisi *item* pertanyaan.
- *Property* `hasCandidateAnswer` mendefinisikan hubungan antara *class* `QuestionItem` dengan *class* `Answer`. Relasi ini menghubungkan antara pertanyaan-pertanyaan pada class `QuestionItem` dengan *instance* dari *class* `Answer` berupa teks, gambar, ataupun kombinasi keduanya, yang bisa menjadi kandidat jawaban dari tiap pertanyaan tersebut.
- `isAbout` menyatakan relasi antara `AssessmentMaterial` dengan `CourseMaterial`. Relasi ini menginformasikan kepada peserta didik mengenai materi ajar atau bab mana saja yang terkait dengan suatu tugas, kuis, maupun ujian. Pada *property* ini diberikan *restriction* bahwa *range* dari

property ini hanya berasal dari *class* `CourseMaterial`. Definisi *property* `isAbout` dapat dilihat pada Gambar 5.6 berikut.



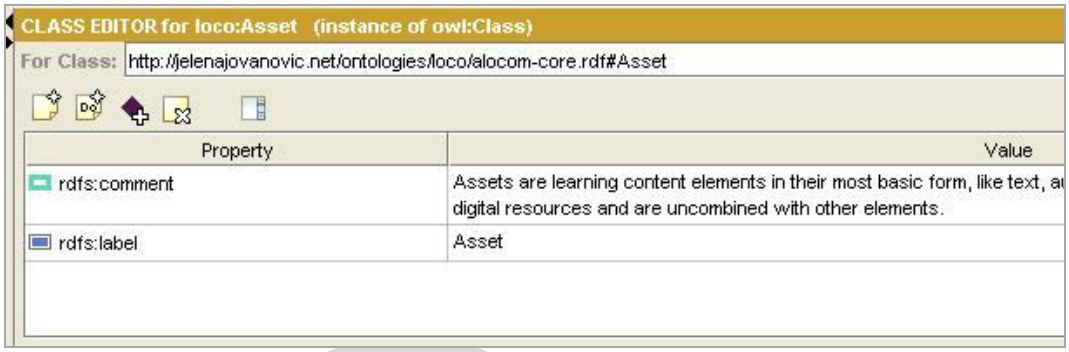
Gambar 5.6 *Property Restriction* `loco:isAbout`

- `prerequisite` merupakan relasi yang terjadi antar `Course`. Dalam suatu disiplin ilmu, bisa jadi terdapat keterkaitan antar cabang-cabang ilmu di dalamnya. Suatu mata kuliah bisa saling berkaitan dengan mata kuliah lain atau menjadi prasyarat yang harus dipenuhi sebelum mengikuti mata kuliah lain. Properti `prerequisite` ini menginformasikan bahwa untuk mempelajari suatu `Course`, peserta didik diharuskan untuk mempelajari `Course` yang menjadi prasyarat mata kuliah tersebut terlebih dahulu sebagai pondasi pengetahuan agar proses belajar menjadi lebih efektif.
- *Property* `name`, `address`, `phone`, `photo`, `role`, `email` ditambahkan sebagai data profil dari seorang pengguna sistem yang menjadi `Contributor` untuk suatu unit objek pembelajaran.
- Seorang `Contributor` bisa berkontribusi pada unit manapun, baik `Course`, `Lesson`, `SCO`, ataupun `Asset`. Properti `hasCreate`, yang merupakan inverse dari `isCreatedBy`, tidak membedakan tipe dari masing-masing unit yang dibuat oleh seorang `Contributor`. Untuk memudahkan *retrieval* kontribusi berdasarkan tipe unit objek pembelajaran, maka penulis membuat *property* `asset_created`, `lesson_created`, `sco_created`, `course_created`.
- Ontologi ALOCoM tidak menyertakan *metadata* yang mencukupi untuk kebutuhan personalisasi pada objek pembelajaran, oleh karena itu penulis

menambahkan beberapa *datatype property* seperti `description, title, keywords, order, status, version, date_created, last_modified`.

- Kategori objek pembelajaran pada level unit `Course` dan `Lesson` bisa dibedakan berdasarkan tingkat kesulitannya, yang nantinya akan digunakan untuk kebutuhan personalisasi. Oleh karena itu dibuat property `difficulty` untuk mengkategorikan suatu objek pembelajaran tergolong *difficult*, *medium*, atau *easy*.
- Selain itu, objek pembelajaran juga dapat dibedakan berdasarkan tingkat interaktifitasnya, apakah memiliki interaktifitas yang tinggi *(high),* sedang *(medium)*, atau rendah *(low)*. Karena itu, ditambahkan *property* `interactivityLevel` pada unit `Course` dan `Lesson`.
- `interactivityType` merupakan *property* yang menyatakan tipe interaktifitas yang dimiliki suatu unit pembelajaran, apakah tergolong *active*, *expositive*, atau *mixed* (campuran keduanya). *Property* ini juga digunakan untuk mengkategorisasi `Course` dan `Lesson` untuk kebutuhan personalisasi nantinya.
- `part` merupakan *datatype property* yang digunakan untuk mengelompokkan `CourseMaterial` berdasarkan topik materi yang lebih umum. Beberapa *instances* dari `CourseMaterial` yang memiliki keterkaitan atau kesamaan topik akan dikategorisasi ke dalam bagian *(part)* yang sama.
- Sebuah `AssessmentMaterial` bisa terdiri dari sekumpulan pertanyaan dan juga beberapa baris kalimat penjelasan. Sebelumnya, sekumpulan pertanyaan didefinisikan dengan *property* `hasQuestion`, sedangkan untuk isi materi selain itu didefinisikan dengan `hasPart`. Untuk menampilkan keduanya sebagai bagian tak terpisah dalam sebuah `AssessmentMaterial`, maka dibuat sebuah *property* yang menggabungkan keduanya (melalui *rules*) yaitu *property* `content`.
- *Property* `rdfs:label` ditambahkan untuk menampilkan label atau nama dengan bahasa natural yang mudah dimengerti, bukan berupa ID yang digunakan pada *resource*, sehingga ketika *resource* ditampilkan pada *portal*, tidak menggunakan *prefix* dan ID seperti `loco:Course`, melainkan sesuai

dengan label yang diberikan. Gambar 5.7 merupakan contoh penggunaan *property* `rdfs:label` untuk memberi nama pada *class* `Asset`.



Gambar 5.7 *Property* `rdfs:label`

5.3.2 Persiapan Data *Instances*

Tahap selanjutnya adalah menyiapkan data yang akan ditampilkan pada *semantic portal*, yaitu berupa *instances* pada ontologi objek pembelajaran. portalCore menggunakan data dalam format RDF, khususnya sintaks N3. Oleh karena itu, diperlukan langkah-langkah khusus untuk menghasilkan data dalam format RDF.

Instances data yang perlu disiapkan sesuai dengan *class* utama pada ontologi objek pembelajaran yang telah dimodifikasi, yaitu lima buah entitas: *Contributor, Course, Lesson, Shareable Content Object,* dan *Asset*.

Pada penelitian ini, data objek pembelajaran beserta seluruh unitnya dirancang sendiri oleh penulis dengan bersumber pada buku *"Fundamentals of Database Systems"* edisi kedua karya El Masri dan Navathe. Karena *portal* ini masih berupa prototipe, maka data yang disajikan pun hanya berupa data *dummy,* belum sepenuhnya lengkap. Terdapat tiga alternatif untuk menghasilkan data tersebut dalam format RDF, yaitu:

a. Mengetik data secara manual dengan sintaks RDF-N3 menggunakan *text editor*.
b. Menggunakan *tool* Protégé 3.4 sebagai ontologi *editor* dan memasukkan *instance* satu per satu melalui *individual form* berdasarkan *class* dan *properties*.

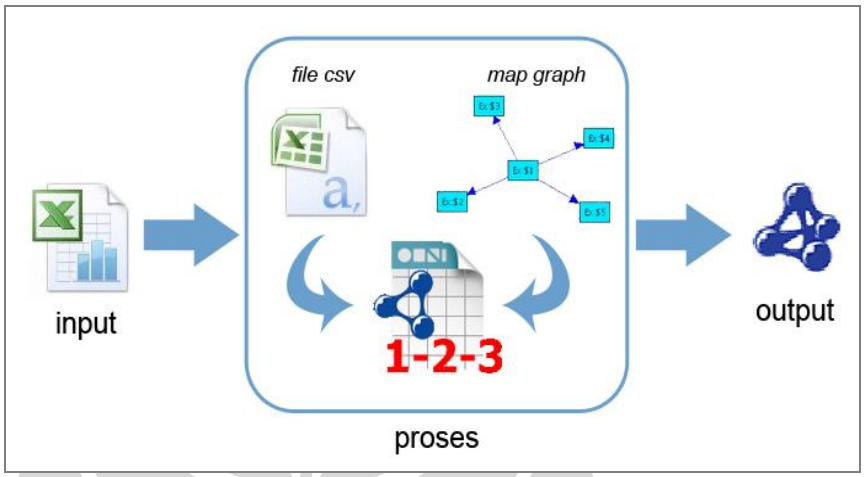
c. Menggunakan *tool converter* yang membaca *file spreadsheet* dan mengonversinya ke dalam bentuk RDF-N3

Selama penelitian berlangsung, penulis telah mencoba menerapkan ketiga metode tersebut. Cara pertama yang dilakukan yaitu mencoba memasukkan data satu per satu melalui Protégé. Namun cara ini sangat tidak efektif dan memakan waktu yang cukup lama, apalagi untuk data yang jumlahnya cukup banyak. Selain itu, karena *instance* tiap *class* dimasukkan secara terpisah, penulis mengalami kesulitan untuk melihat stuktur dan relasi data antara suatu *instance* pada suatu *class* dengan *instance* pada *class* lainnya. Sedangkan jika harus mengetik manual dari awal untuk data yang jumlahnya cukup besar seperti pada alternatif a juga sangat tidak efisien. Maka, penulis memutuskan untuk menggunakan metode ketiga yaitu dengan *tool converter* RDF123, yang mengubah *file* .xls ke dalam format RDF-N3. Walaupun penulis harus membuat data dalam dua langkah, dengan format *spreadsheet* terlebih dahulu baru mengonversinya, namun cara ini tetap lebih cepat dan efisien dibanding cara pertama. Dan penyajian awal dengan format *spreadsheet* juga lebih memudahkan penulis untuk melihat secara utuh struktur data dan keterkaitan antar *instance*. Pada pelaksanaannya, sesekali penulis juga menggunakan cara manual untuk melakukan *editing* pada file .n3 yang dihasilkan oleh RDF123 menggunakan *text editor* SciTE.

Berikut penjelasan lebih detil mengenai proses konversi yang dilakukan dengan RDF123.

1) Menyiapkan input data berupa *file spreadsheet* dan menyimpannya dalam format .csv *(comma separated values)*.
2) Membuat *map graph* (*.xgmml) menggunakan RDF123. *Map graph* dibuat untuk memetakan *instances* pada *properties* dalam suatu *class*.
3) Menyimpan keluaran *(output)* yang dihasilkan RDF123 sebagai *file* .n3.

Gambar 5.8 mengilustrasikan alur pembuatan data berformat RDF-N3 menggunakan RDF123.



Gambar 5.8 Alur Konversi Data

Langkah pertama yang harus dilakukan adalah menyiapkan input data berupa *file spreadsheet* yang kemudian disimpan dalam format .csv *(comma separated values)*. Terdapat lima entitas yang diidentifikasi dari data mengenai struktur objek pembelajaran yaitu *course, lesson, shareable content object, asset,* dan *contributor*. Masing-masing entitas dipisah datanya dalam satu *file* .csv sehingga akan terdapat lima *file* .csv. *Header* tabel dari tiap entitas disesuaikan dengan *property* dari masing-masing entitas seperti ditunjukkan pada Gambar 5.9.

Tiap kolom dari tabel berisi data yang akan dipetakan ke atribut atau *property* yang terdapat pada ontologi objek pembelajaran. Tiap baris dari tabel akan menghasilkan satu *instance*.

Course

id	name	description	isCreatedBy	date_created	difficulty	interactivityType	interactivityLevel	keywords	status	version	hasPart	prerequisite

is_deleted	last_modified

Lesson

id	type	name	title	description	part	isCreatedBy	date_created	difficulty	interactivityType	interactivityLevel	keywords

status	version	hasPart	follows	precedes	is_deleted	last_modified

Shareable Content Object

id	type	name	description	isCreatedBy	date_created	keywords	status	version	order	hasPart	follows	precedes	is_deleted	last_modified

Asset

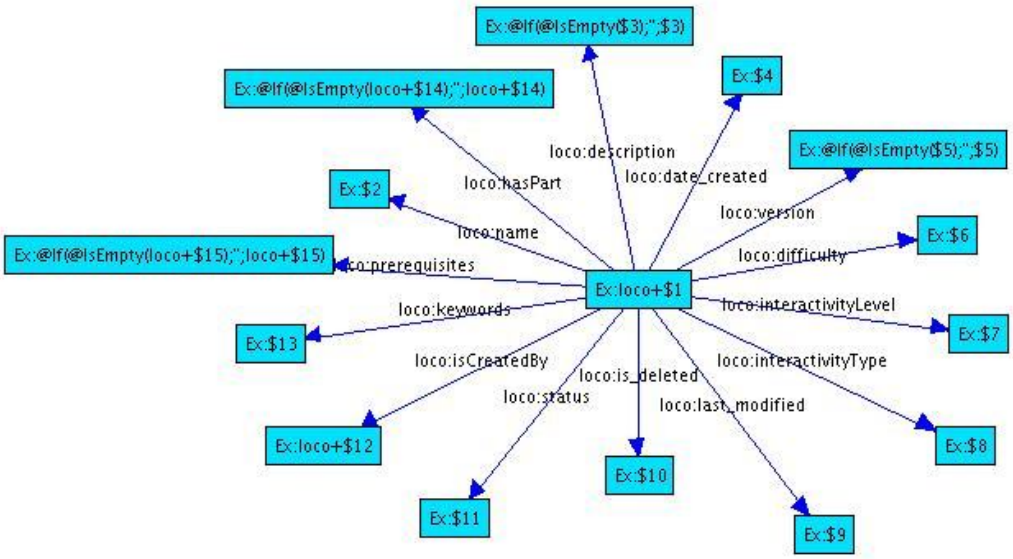
id	type	name	isCreatedBy	date_created	keywords	follows	precedes	is_deleted	last_modified	value

Contributor

id	role	name	address	phone	email	photo

Gambar 5.9 *Header* Tabel *Spreadsheet*

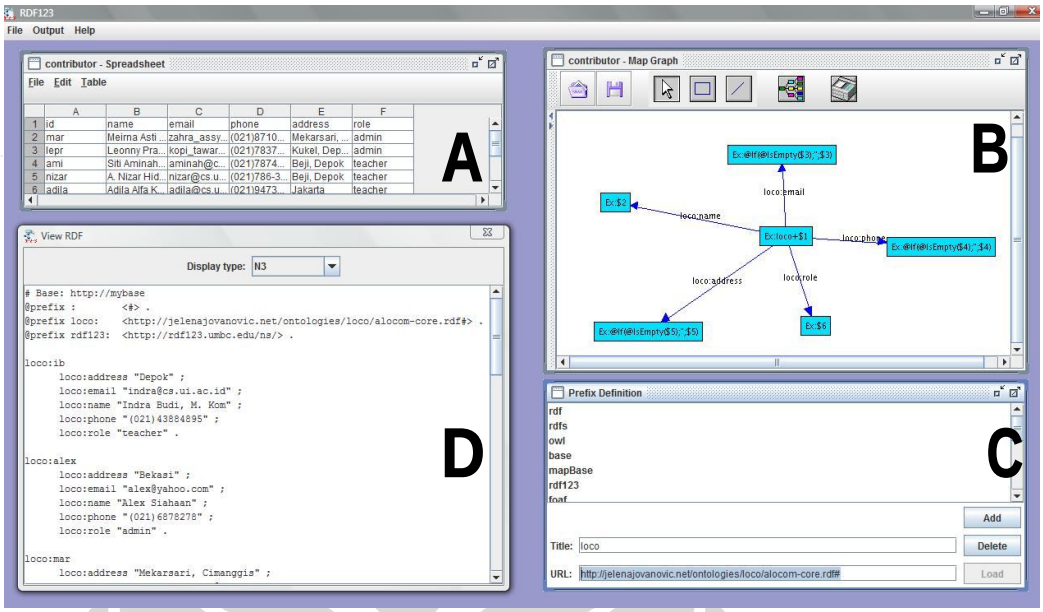
Langkah selanjutnya adalah membuat *map graph* dengan RDF123 seperti pada Gambar 5.10. Kemudian *file* .csv satu per satu akan dipetakan secara otomatis sesuai dengan *graph* yang telah dibuat untuk setiap entitas *(class)* utama yang terdapat pada ontologi objek pembelajaran. *Output* yang dihasilkan berupa data dalam format RDF dengan sintaks RDF/XML atau N3. Yang akan digunakan dalam penelitian ini adalah tipe N3 sebagai input portalCore.



Gambar 5.10 ***Map Graph Course***

Pemetaan dilakukan dengan *map graph*, suatu *template* dalam bentuk *graph* RDF dengan format yang telah didefinisikan oleh RDF123. Format yang dimaksud yaitu penggunaan *namespace* Ex:, $n, *logical expression* @If(B;E;E) dan @IsEmpty(E). B untuk *Boolean expression* dan E untuk *Expression*. *Map graph* disimpan dengan tipe *file* .xgmml dan dapat dikonversi pula menjadi format RDF. Setiap satu *file* .csv dibuat *map graph*-nya masing-masing sehingga akan terbentuk lima buah *file* .xgmml juga. *Map graph* untuk masing-masing entitas tersebut dapat dilihat pada Lampiran B.

Untuk lebih menggambarkan proses kerja program RDF123 yang digunakan pada tahap persiapan data ini, penulis menyertakan *screenshot* area kerja RDF123 pada Gambar 5.11, beserta penjelasan fungsi tiap bagiannya.



Gambar 5.11 Program RDF123

Program RDF123 terdiri dari tiga *internal frames*, yaitu bagian A, B, dan C. Bagian A merupakan *Spreadsheet Editor* yang berfungsi untuk memasukkan input berupa *file* .csv yang akan dikonversi menjadi RDF/N3. Bagian B merupakan *Graph Editor*, yaitu area untuk membuat *map graph* dari entitas yang ditampilkan pada *file* .csv di bagian A. Sedangkan bagian C merupakan *Prefix Definition* dimana seorang pengguna harus memasukkan atau menambahkan *prefix* dari ontologinya ketika ingin melakukan proses konversi. Dan terakhir, bagian D adalah *window* yang menunjukkan *output* hasil konversi data ke dalam format RDF/XML ataupun N3.

Dalam penelitian ini, total jumlah *instances* yang dibuat sebagai data untuk ditampilkan pada prototipe *semantic portal* yaitu *Course* sebanyak tiga buah, *Lesson* 30 buah, *Shareable Content Object* 20 buah, *Asset* 30 buah, dan *Contributor* sebanyak 10 buah.

5.3.3 Pendefinisian *Rules*

Rules dibuat untuk melakukan proses *inference*, yaitu memperoleh data baru dari data yang sudah ada. *Set of rules* atau kumpulan *rules* yang akan digunakan pada

semantic portal dalam penelitian ini disimpan bersamaan dengan ontologi dan *instances* data di folder yang sama sebagai *input portal*, namun dalam *file* yang terpisah yaitu portal.rules. Proses *inference* yang dilakukan pada *portal* menggunakan *GenericRuleReasoner* Jena sehingga struktur dan sintaks *rules* mengikuti *rule engine* tersebut.

Dalam portalCore terdapat beberapa RDFS *closure rules* yang telah didefinisikan antara lain:

```
[rdfs2: (?x ?p ?y), (?p rdfs:domain ?c) -> (?x rdf:type ?c)]
[rdfs3: (?x ?p ?y), (?p rdfs:range ?c) -> (?y rdf:type ?c)]
[rdfs7: (?a rdf:type rdfs:Class) -> (?a rdfs:subClassOf ?a)]
[rdfs8: (?a rdfs:subClassOf ?b),(?b rdfs:subClassOf ?c)->(?a rdfs:subClassOf ?c)]
[rdfs9: (?x rdfs:subClassOf ?y), (?a rdf:type ?x) -> (?a rdf:type ?y)]
```

Rule rdfs2 menyatakan jika *instance* x memiliki relasi p dengan *instance* y, dan domain *property* p adalah *class* c, maka x juga merupakan *instance* dari *class* c. *Rule* rdfs3 menyatakan jika *instance* x memiliki relasi p dengan *instance* y, dan *range* dari *property* p adalah *class* c, maka y juga merupakan *instance* dari *class* c. *Rule* rdfs7 menyatakan suatu *class* merupakan *subclass* dari dirinya sendiri. *Rule* rdfs8 menyatakan jika a *subclass* b, dan b *subclass* c, maka a juga subclass c. *Rule* rdfs9 menyatakan jika x *subclass* y, a *instance* x, maka a juga *instance* y.

Pendefinisian *rules* pada penelitian ini dibedakan menjadi dua jenis berdasarkan tujuan penggunaannya, yaitu untuk kebutuhan perolehan data pada suatu entitas dan untuk visualisasi pada *portal*. Berikut adalah *rules* yang didefinisikan.

a. Contributor

Pada kategori *Contributor* terdapat lima jenis *rules*, yaitu:

- Untuk mendapatkan semua jenis objek pembelajaran yang telah dibuat oleh seseorang.

  Jika A dibuat oleh B, maka B telah membuat A *(inverse).*

```
(?A loco:isCreatedBy?B) -> (?B loco:hasCreate?A) .
```

- Untuk mendapatkan (mata kuliah) *course* yang telah dibuat oleh seseorang.

  Jika A adalah sebuah mata kuliah dan A dibuat oleh B, maka B telah membuat mata kuliah A *(inverse).*

```
(?A rdf:type loco:Course),(?A loco:isCreatedBy ?B)->(?B loco:course_created ?A) .
```

- Untuk mendapatkan *lesson (course atau assessment material)* yang telah dibuat oleh seseorang.

  Jika A adalah sebuah *lesson* dan A dibuat oleh B, maka B telah membuat *lesson* A *(inverse).*

```
(?A rdf:type loco:Lesson),(?A loco:isCreatedBy ?B)->(?B loco:lesson_created ?A) .
```

- Untuk mendapatkan *shareable content object* yang dibuat oleh seseorang.

  Jika A adalah sebuah *shareable content object* dan A dibuat oleh B, maka B telah membuat *shareable content object* A *(inverse).*

```
(?A rdf:type loco:ShareableContentObject),(?A loco:isCreatedBy ?B)->(?B
loco:sco_created ?A) .
```

- Untuk mendapatkan *asset* yang dibuat oleh seseorang.

  Jika A adalah sebuah *asset* dan A dibuat oleh B, maka B telah membuat *asset* A *(inverse).*

```
(?A rdf:type loco:Asset),(?A loco:isCreatedBy ?B)->(?B loco:asset_created ?A) .
```

b. *Course, Lesson, Shareable Content Object,* dan *Asset*

Rules berikut berlaku untuk keempat kategori objek pembelajaran.

- Untuk mendapatkan rangkaian unit objek pembelajaran berdasarkan susunan atau urutannya.

  Jika A mengikuti B, maka B mendahului A *(inverse).*

```
(?A loco:follows ?B) -> (?B loco:precedes ?A) .
```

- Untuk mendapatkan data mengenai unit pembelajaran yang menjadi bagian atau yang menyusun unit lainnya.

  Jika A mempunyai bagian B, maka B adalah bagian dari A *(inverse).*

```
(?A loco:hasPart ?B) -> (?B loco:isPartOf ?A) .
```

c. Kebutuhan visualisasi

Rules berikut didefinisikan untuk membantu proses visualisasi pada *portal*.

```
# to display Contributor, Course, Lesson, SCO, and Asset name at portal viewer
(?A rdf:type loco:Contributor), (?A loco:name ?B) -> (?A rdfs:label ?B) .
(?A rdf:type loco:Course), (?A loco:name ?B) -> (?A rdfs:label ?B) .
(?A rdf:type loco:Lesson), (?A loco:name ?B) -> (?A rdfs:label ?B) .
(?A rdf:type loco:ShareableContentObject), (?A loco:name ?B)-> (?A rdfs:label ?B).
(?A rdf:type loco:Asset), (?A loco:name ?B) -> (?A rdfs:label ?B) .

#to unite related content of Assessment Material, either questions or statement
(?A rdf:type loco:AssessmentMaterial),(?A loco:hasPart ?B)->(?A loco:content ?B) .
(?A rdf:type loco:AssessmentMaterial),(?A loco:hasQuestion ?B) -> (?A loco:content
?B) .
```

5.4 Konfigurasi *Portal*

Berdasarkan penjelasan yang penulis peroleh dari situs SWAD-E mengenai *portal customization* [46], proses konfigurasi terhadap *portal* dilakukan melalui sebuah *file* sources.n3 yang mendefinisikan *DataSources, facets,* dan *templates*. *File* RDF tersebut menggunakan *vocabulary* pada portal dengan *prefix* `pcv:` dan *namespace* `http://jena.hpl.hp.com/2003/04/portal-config-vocab#`.

Berikut adalah penjelasan lebih detil mengenai langkah-langkah yang dilakukan penulis di masing-masing tahap konfigurasi.

5.4.1 Pendefinisian *DataSource*

DataSource bisa dikatakan sebagai komponen utama dari proses konfigurasi *portal* yang akan menentukan jalannya fungsi dan tampilan pada *portal*. *DataSource* menentukan ontologi, *instances* data, dan *set of rules* yang akan digunakan, *facets* dan *templates* yang akan ditampilkan kepada pengguna, serta *stylesheet* dan beberapa *property* lainnya yang digunakan dalam pengaturan *portal*.

Yang dimaksud dengan pendefinisian *DataSources* adalah pembuatan *instance* `pcv:DataSource` beserta pengaturan *property* yang dimiliki. Dalam penelitian ini,

terdapat empat buah *DataSources* yang didefinisikan berdasarkan empat entitas utama pada ontologi objek pembelajaran yang akan ditampilkan melalui *facets*, yaitu *Course, Lesson, Shareable Content Object,* dan *Asset*. Selain itu dibuat juga *datasource* tambahan untuk menampilkan semua entitas. Berikut ini contoh pendefinisian *DataSource* untuk menampilkan *Course*.

```
[]  rdf:type pcv:DataSource ;
    rdfs:label "Course" ;
    pcv:encoding "p2";
    pcv:order "20"^^xsd:integer ;
    dc:description "Prototype Learning Object Portal" ;

    pcv:sourceURL <portal://data/course.n3> ;
    pcv:sourceURL <portal://data/lesson.n3> ;
    pcv:sourceURL <portal://data/contributor.n3> ;
    pcv:sourceURL <portal://data/shareablecontentobject.n3> ;
    pcv:sourceURL <portal://data/asset.n3> ;

    pcv:ontologySourceURL <portal://data/alocom-core.owl> ;
    pcv:closureRulesURL   <portal://data/portal.rules> ;
    pcv:styleSheet "site.css" ;

    pcv:filterOnType loco:Course;
    ...
```

`pcv:sourceURL` digunakan untuk mendefinisikan *instances* data yang digunakan dalam *semantic portal* ini. Sedangkan `pcv:ontologySourceURL` untuk menentukan ontologi yang menjadi basis *portal*. `pcv:closureRulesURL` mendefinisikan *file* yang menyimpan *set of rules. Property* `pcv:styleSheet` merujuk pada *file* CSS yang mengatur tampilan *portal* dan `pcv:filterOnType` untuk menentukan tipe entitas yang akan ditampilkan melalui *facets*. Setiap *DataSource* juga membutuhkan sebuah *encoding string* yang akan digunakan untuk identifikasi jika ada *http request* pada *portal*. Selain itu, sebuah integer "*order number*" juga dibutuhkan untuk mengurutkan *DataSource*.

5.4.2 Pendefinisian *Facets*

Pada portalCore *facet* didefinisikan dalam *Datasource* melalui *property* `pcv:facet`. Langkah selanjutnya yaitu membuat *instance* dari *facet* tersebut yang mendefinisikan `pcv:linkProp` untuk menampilkan objek yang dicari berdasarkan *property*. Berikut ini contoh definisi *facet* untuk entitas *Lesson*.

```
    pcv:facet loco:p3nameFacet;
    pcv:facet loco:p3difficultyFacet;
    pcv:facet loco:p3typeFacet;
    pcv:facet loco:p3intLevelFacet;
    pcv:facet loco:p3partFacet;

...
loco:p3partFacet a pcv:Facet;
    rdfs:label "Part" ;
    pcv:linkProp loco:part;
    pcv:order "5"^^xsd:integer;
```

Pada contoh di atas, yang dimaksud dengan *instance* dari *facet* yaitu `loco:p3nameFacet`, `loco:p3difficultyFacet`, `loco:p3typeFacet`, `p3intLevelFacet`, dan `loco:p3partFacet`. Untuk *facet* `loco:p3partFacet` misalnya, akan menampilkan daftar topik materi kuliah beserta jumlah *lesson* yang termasuk dalam tiap topik tersebut. Tampilan *facet* ini akan tampak seperti pada Gambar 5.12 berikut.

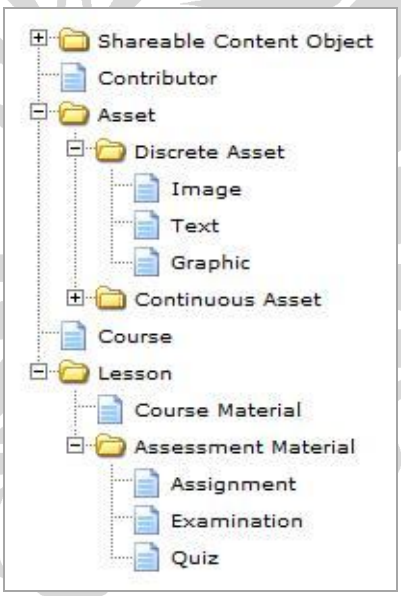
Gambar 5.12 *Facet Topic* untuk Objek *Lesson*

Jenis *facet* yang ditampilkan pada contoh di atas merupakan tipe *flat*, yang menampilkan *instance* yang dihubungkan oleh sebuah *property*. Masih ada dua tipe lain dari *facet* yang bisa ditampilkan pada *portal*, yaitu *AlphaRange* dan

Hierarchical facet. Tipe *AlphaRange* menampilkan huruf pertama dari *instance* yang diperoleh dan mengurutkannya sesuai abjad. Gambar 5.13 menunjukkan contoh *AlphaRange facet* pada *property* `name` untuk objek *Lesson*.

Gambar 5.13 *AlphaRange Facet*

Sedangkan *Hierarchical facet* memberi tampilan *classes* berdasarkan hierarkinya pada ontologi yang dibangun. Contoh dari *Hierarchical facet* dapat dilihat pada Gambar 5.14 berikut ini.



Gambar 5.14 *Hierarchical Facet*

Dalam pengembangan *semantic portal* ini, terdapat empat kelompok *facets* berdasarkan objek yang akan ditampilkan. Jumlah *facet* seluruhnya yaitu 18 buah, dengan rincian *facet* untuk tiap objek seperti pada Tabel 5.2. Selain itu, terdapat satu buah *Hierarchical facet* sebagai tambahan, yang menampilkan seluruh entitas.

Tabel 5.2 Daftar *Facet*

Objek	*Facets*
Course	Name, Difficulty, Contributor, Level of Interactivity, Type of Interactivity
Lesson	Name, Type, Difficulty, Contributor, Level of Interactivity, Type of Interactivity, Topic
Shareable Content Object	Name, Type, Contributor
Asset	Name, Type, Contributor

5.4.3 Pendefinisian *Templates*

Selain *datasource* dan *facet, template* yang digunakan untuk menampilkan data pada *portal* juga harus didefinisikan pada *file* konfigurasi yang sama (sources.n3). Jumlah *template* yang didefinisikan sesuai dengan jumlah seluruh objek atau entitas yang ingin ditampilkan, tidak hanya entitas utama yang ditampilkan melalui *facet* saja. Maka, dalam penelitian ini ada lima entitas yang akan dibuat *template*-nya, yaitu *Course*, *Lesson*, *Shareable Content Object*, *Asset*, dan *Contributor*.

Bagian ini hanya membahas mengenai pendefinisian *templates* pada *file* konfigurasi. Sedangkan langkah-langkah dalam pembuatan *templates* untuk tiap objek akan dibahas pada subbab 5.5. Berikut contoh sintaks untuk mendefinisikan *templates* objek *Shareable Content Object* dan *Asset*.

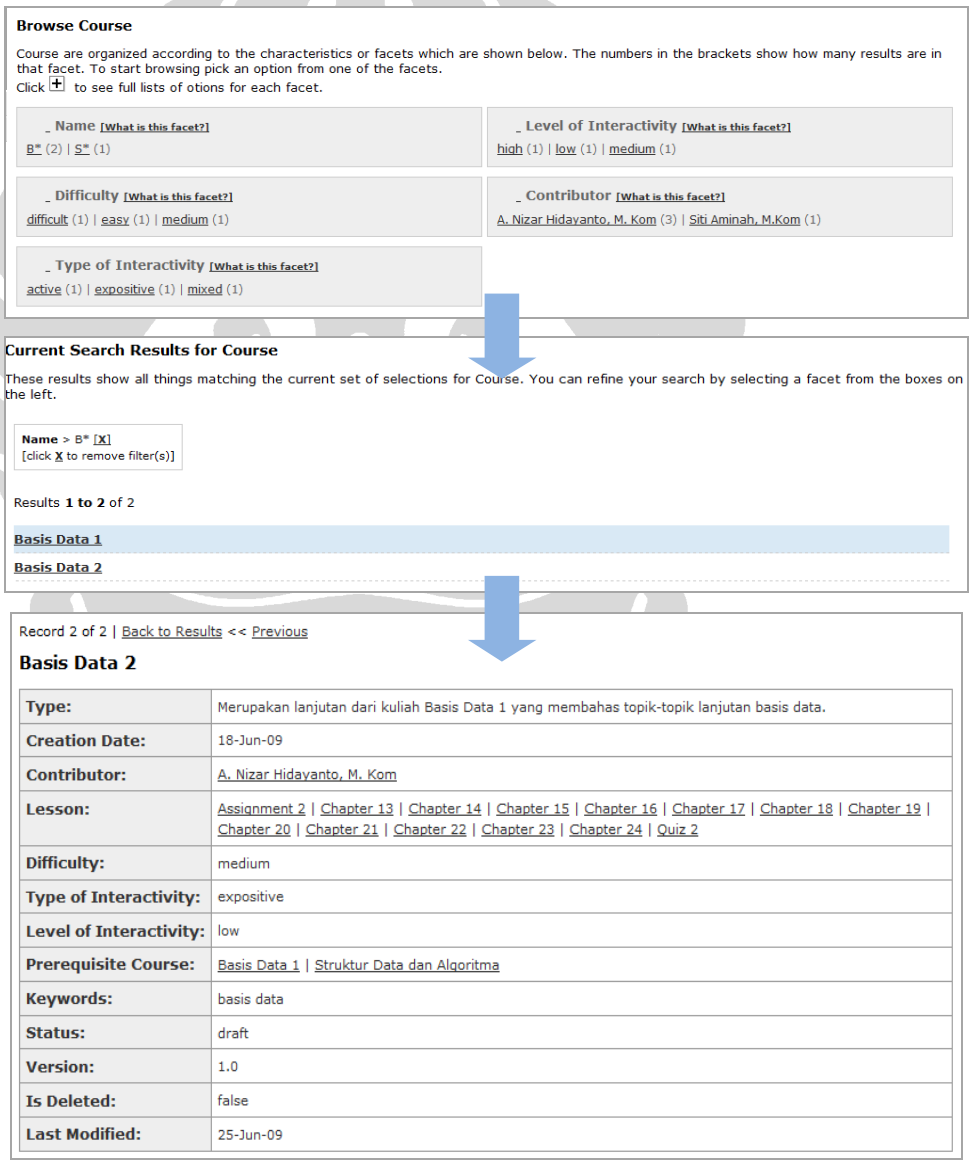
```
...
    pcv:template [a pcv:Template;
        pcv:templateContext "page" ;
        pcv:templatePath    <portal://templates/pageContent.vm>;
        pcv:templateClass   loco:ShareableContentObject;
    ];

     pcv:template [a pcv:Template;
        pcv:templateContext "page" ;
        pcv:templatePath    <portal://templates/pageAsset.vm>;
        pcv:templateClass   loco:Asset;
    ];
 ...
```

Pendefinisian *template* pada *DataSource* dilakukan melalui *property* `pcv:template` yang akan merelasikan sebuah *datasource* dengan *instance* dari `pcv:Template`. *Instance* `pcv:Template` dibuat untuk masing-masing objek yang ingin ditampilkan.

5.5 Tampilan *Portal*

Dalam penelitian ini, terdapat tiga kategori *page* yang menjadi tampilan *portal*, yaitu *browse page*, *results page*, dan *resources page*. Berikut ilustrasi yang menggambarkan halaman pada *portal* sesuai alur atau urutan tampilannya.



Gambar 5.15 Jenis Halaman *Portal*

Browse page merupakan tampilan dengan *facets* untuk suatu tipe objek. *Results page* merupakan tampilan yang berisi hasil *browse* maupun *search*. Sedangkan *resource page* merupakan tampilan yang memberikan deskripsi suatu *resource.*

Template yang didefinisikan pada tahap ini merupakan *resource page.* Seperti yang telah disebutkan pada bagian pendefinisian *templates,* ada lima buah *file templates* yang dibuat untuk menampilkan lima entitas data *(resources)*. Modifikasi tampilan juga dilakukan pada *header page, leaf page,* dan tentunya *stylesheet* untuk mengatur *look-and-feel* antarmuka *portal*.

Tampilan yang dihasilkan dari *velocity template* pada portal bersifat dinamis. Artinya, data yang ditampilkan pada halaman sesuai dengan *properties* dan *instances* dari *resource* yang diambil. Bahkan nilai dari *resource* yang berupa *instance* dari entitas lain (yang berasal dari relasi *object property*) dapat dijadikan *link* menuju *resource page* lain. Berikut contoh kode program untuk menampilkan *resource* mata kuliah yang menjadi prasyarat suatu mata kuliah *(Course)*. Relasi `loco:prerequisite` menghubungkan *instances* dari *class* `Course`.

```
1  #if($resource.hasProperty("loco:prerequisite"))
2      <tr> <th valign="top" nowrap>Prerequisite Course: </th>
3      <td>
4      #set($count=0)
5      #foreach($p in $resource.findProperties("loco:prerequisite"))
6              #foreach ($v in $p.values)
7                      #if($count>0) |
8                      #end
9                      $v.render("leaf", $request)
10                     #set($count=$count+1)
11             #end
12     #end
13     </td>
14     </tr>
15 #end
```

Baris 1 merupakan *conditional statement* yang menyatakan jika *resource Course* memiliki *property* `prerequisite`, maka akan menambahkan satu baris pada tabel *Course* yang berisi daftar mata kuliah lain yang menjadi prasyarat mata kuliah

tersebut. Baris 2, 3, 13, dan 14 adalah kode HTML untuk membuat baris dan kolom baru pada tabel, dengan label *header* baris yaitu "Prerequisite Course". Baris 4 – 12 merupakan perintah rekursif untuk mendapatkan *value* dari *resource* yang dihubungkan dengan *property* `prerequisite`, dan membuat *link* menuju halaman *resource* tersebut.

Tabel 5.3 berikut merupakan daftar *properties* yang ditampilkan di tiap *template page* yang dibuat.

Tabel 5.3 Daftar *Templates* dan *Properties*

No	*File Templates*	*Property* yang ditampilkan
1	pageContributor.vm	***Datatype Property*** `name, role, email, phone, photo, address`
		Object Property Course(`loco:course_created`) Lesson(`loco:lesson_created`) SCO (`loco:sco_created`) Asset(`loco:asset_created`)
2	pageCourse.vm	***Datatype Property*** `name, description, date_created, last_modified, difficulty, interactivityType, interactivityLevel, keywords, status, version`
		Object Property Lesson (`loco:hasPart`) Contributor (`locoLisCreatedBy`)
3	pageLesson.vm	***Datatype Property*** `name, title, description, part, date_created, difficulty, interactivityLevel, interactivityType, keywords, status, last_modified`

		Object Property
		`Course (loco:isPartOf)`
		`Contributor (loco:isCreatedBy)`
		`Previous Chapter (loco:follows)`
		`Next Chapter (loco:precedes)`
		`Content (loco:hasPart) atau (loco:content)`
		`Dealt With (loco:isAbout)`

4	pageContent.vm	***Datatype Property*** `name, description, date_created, keywords, status, version, last_modified`

		Object Property
		`Lesson (loco:isPartOf)`
		`Contributor (loco:isCreatedBy)`
		`Include (loco:hasPart)`

5	pageAsset.vm	***Datatype Property*** `Name, date_created, keywords, value, last_modified`

		Object Property
		`Content Of (loco:isPartOf)`
		`Contributor (loco:isCreatedBy)`
		`Previous Assets (loco:follows)`
		`Next Assets (loco:precedes)`

Datatype property berisi data statis yang diambil dari data RDF, sedangkan *object property* memiliki *value* berupa data dinamis yang dapat menghubungkan suatu *resource page* dengan *page* lain. Tampilan masing-masing *page* ini dapat dilihat pada Lampiran D.