A Case Study of Ontology-Driven Development of Intelligent Educational Systems

Gordon Deline, Athabasca University, Canada
Fuhua Lin, Athabasca University, Canada
Dunwei Wen, Athabasca University, Canada
Dragan Gašević, Athabasca University, Canada
Kinshuk, Athabasca University, Canada

ABSTRACT

This article presents a case study of ontology-driven development of intelligent educational systems. Following a review of literature related to ontology development, ontology-driven software development, and traditional software engineering, we developed an ontology-driven software development methodology appropriate for intelligent ontology-driven systems which have ontologies as key execution components, such as e-Advisor, and which is biased toward an integration of incremental and iterative ontology development and downstream Model Driven Architecture for development of software components.

Keywords: Please insert keywords

INTRODUCTION

Recent research indicates that ontologies will play a greater role in the development and maintenance of intelligent and adaptive educational systems.

The benefits of ontologies for intelligent educational systems are that (1) intelligent agents can use the developed ontologies as the basis for their knowledge base construction, reasoning and interface design; (2) in a distributed education environment, the ontologies developed for different entities can serve as standardized and open interfaces for interoperability and communication.

Following review of the e-Advisor system and its development, we looked at how to formalize ontology-driven software construction. Areas of focus included ontology discovery, evaluation, reuse, integration, tie-ins with tra-
ditional software engineering methodologies and tools, and integration with governance frameworks.

Using e-Advisor development as a case study, specifically how ontologies are developed and maintained, we have developed an ontology-driven software development methodology that would be appropriate for intelligent ontology-driven systems such as e-Advisor.

This article presents a case study of ontology-driven software development methodology of intelligent educational systems. The organization of the rest of this article is as follows: Section II is a literature review; Section III describes e-Advisor and its ontologies that are key components of the e-Advisor architecture and their development and maintenance; Section IV and Section V review our ontology-driven development methodology as related to e-Advisor; and Section VI relates our conclusions and future work.

LITERATURE REVIEW

As related by Chandrasekaran et. al (1999), ontologies are a key component that can enable effective communication between agents in a multiple agent system, and the building of agents’ knowledge. The advantages of ontologies in learner/user models are put forth by Kay (1999) and Chen & Mizoguchi (1999). Razmerita, et al. (2003) put forward an architecture for ontology-based user modeling. Mizoguchi and Bourdeau (2000) discussed the use of ontologies to overcome problems with the use of AI in education.

The W3C’s Web Ontology Language (OWL) (Bechhofer et al., 2004) supports the definition of ontologies by building on and adding to the basic support provided by Resource Definition Framework (RDF) and RDF Schema for objects, classes, properties, and hierarchies. OWL supports publishing and sharing of ontologies, knowledge management, and software agents.

Protégé OWL is a tool that is used by domain experts to create and maintain reusable ontologies for knowledge-based systems and is the tool that the e-Advisor ontologies were developed using. As an open platform with a strong user base, a number of useful plug-ins can be found for Protégé to support such functionality as visualization and code generation, as well as the capability to build custom plug-ins. The architecture and features of Protégé-OWL are described by Knublauch (2003); Knublauch, Feargerson, Noy & Musen (2004); and Knublauch, Musen, & Rector (2004). Rector (2003) describes a method for normalization of Description Logics (DL)-based ontologies, specifically those constructed with OWL-DL. Seidenberg & Rector (2006) discuss ontology segmentation techniques and benefits.


A natural integration point between ontologies and traditional Software Engineering methodologies is the OMG’s Model Driven Architecture (MDA) (OMG MDA, 2008), with benefits that can be seen as analogous to the benefits of using ontologies to model domains.

Knublauch (2002) suggested the use of Agile development methodology for the development of ontologies and proposed an ontology-driven software development (ODD) methodology built on the use of tools such as Protégé OWL (2006), with discussion of synergies with OMG MDA and efforts to bridge OWL and MOF/UML. Terrasse, Savonnet, Leclercq, Grison, & Becker (2006) proposed the use of
ontologies as the first phase of an approach that is driven by meta-models which further refine the domain modeled by the ontologies, linking ontology concepts with UML concepts. The W3C Semantic Web Best Practices & Deployment Working Group’s working draft on Ontology Driven Architectures (W3C ODA, 2006) further supports MDA as a vehicle to support Ontology Driven Architecture (ODA).

E-ADVISOR AND ITS ONTOLOGIES

This section introduces e-Advisor and reviews the key ontologies in e-Advisor and how knowledge is acquired and maintained.

Students in degree programs have to balance personal and career objectives, preferences, and financial and time constraints against degree requirements, course availability and course interrelationships. If students go about planning their programs entirely on their own they can make non-optimal program choices that may not be discovered until a potentially non-recoverable problem arises.

In traditional learning settings where the student is collocated with the academic and administrative staff, face-to-face meetings can facilitate the program planning process but this requires a non-trivial amount of time on the part of the advising staff members. The constraints and demands faced by faculty of post-secondary educational institutions are such that little time is left for faculty to personally advise students.

The trends toward cost-effective lifelong learning have increased the demand for distance education and, especially in distance-education scenarios, both students and advisors face more problems: face-to-face interviews are infeasible; students tend to have more severe work and family commitments that studies must be balanced against; completion of a degree or program tends to take much longer; and courses may be taken out of sequence due to students' prior learning and experience or to facilitate immediate career objectives.

Complicating matters further, course delivery schedules and prerequisite relationships are dynamic and change over time as a given program curriculum is developed and maintained via course revisions, additions, and closures. The longer period of time it takes to complete a course of study in a lifelong learning and distance education environment exacerbates the impact of these constant changes on academic advising.

To assist students and advisors and address these issues in a more comprehensive manner than possible using more basic tools, many Expert Systems and Decision Support Systems (DSS) have been proposed and developed since the 1980s. These tools are able to provide more comprehensive support for academic planning, advising, and evaluation by leveraging advances in computer technology such as artificial intelligence (AI) techniques. Caroll & Chappel (1996); Murray & Le Blanc (1995); and Turban, Fisher, and Altman (1988) reviewed various approaches to the use of DSS in academic administration.

Since 2003, a number of the authors of this article have been developing e-Advisor, an intelligent system that facilitates academic advising for students and administrators in the program of Master of Science in Information Systems (MSc IS) of Athabasca University\(^1\). Since academic advising involves intensive multi-participant cooperation, multiple intelligent agents were used in the architecture of e-Advisor. The system consists of four types of user agents: Student Agent, Administrator Agent, Instructor Agent, and Advisor Agent; and two types of resource agents, Ontology Agent and Database & Knowledge Base (DB&KB) Agent. Figure 1 depicts the architecture of e-Advisor.

In the e-Advisor architecture, the ontologies formally define domain entities and the relations among them. Based on the ontologies, the structural part of the knowledge base is modeled. The Student Agent acts on behalf of the individual student and as such must identify and understand learners’ preferences and requirements to generate and maintain a
learner model for the student incorporating things such as job objectives, career track, learning style, and target date of completion. Using this knowledge and co-operating with Advisor Agents in the system, the User Agent develops and recommends personalized and correct schedules for the student. The Advisor Agent acts on behalf of an academic advisor to generate schedules for students being advised by the advisor, negotiating with Student Agents to assist in development of personalized and correct schedules for these individual students and evaluates the status of these students. The Instructor Agent assists the course instructor when defining and maintaining pre-requisites for courses the instructor is in charge of. The Administrator Agent maintains the curriculum model and conducts course curriculum projection. DB & KB Agents handle data/knowledge requests from other Agents in the system and notify interested Agents of changes in the system’s databases and knowledge bases so they can act on them. Ontology Agents support querying and knowledge inference and maintenance across the system’s ontologies on behalf of other agents in the system.

The ontologies discussed here relate to the program and courses and the student being advised. The program and courses are modeled via the Topic Ontology, Course Ontology, and Program Ontology. The student is modeled using the Student Ontology and Preference Ontology.

e-Advisor selects the ‘best’ possible Plan from a set of feasible Plans by relating the student’s preferences to each plan. This is done using associations between Program Topic (see Topic Ontology, below) and Program Preference (see Preference Ontology, below). A Program Preference, such as a Specialization, may be associated with a set of Program Topics with a varying degree (real number in [0,1]) for each association.

1. Topic Ontology

Program Topics are the lowest level concepts related to programs and courses, and describe topics that can be found in a given domain at the program level (may be taught in a course in a given program). In the Information System domain Program Topics include concepts such as Data and Object Structures, Data Management, and Data Mining.

The Topic Ontology also models relationships between Program Topics. Prerequisite dependencies are modeled using the “is prerequisite of” relation. This prerequisite relation is used by Instructor Agent to determine course prerequisites using a matching algorithm.

The initial set of Topics in the Topic Ontology and their pre-requisite relations were derived from analysis of the MSc IS program structure and curriculum course descriptions.

2. Course Ontology

The Course Ontology is made up of classes, subclasses, and properties related to courses in general, as well as courses in MSc IS program.

Figure 1. E-Advisor high level architecture
As discussed above (see Topic Ontology), a Course is made up of a set of Topics. The Course class has basic properties such as the course title, description, language, and Topics covered (see Topic Ontology, above). Course is subclassed by CoursesInMScIS, which adds properties to model elements such as prerequisites and syllabus. Two more Course subclasses, CoursesInPlans and CourseInTranscript, are further specializations.

3. Program Ontology

The Program Ontology is used to build the logical structure of an academic program, including rules and regulations. The Program class has properties that relate to the maximum duration, minimum number of credits, number of semesters per year, minimum number of courses to be taken in an academic year, etc. The MScIS class is a subclass of Program and contains properties specific to the MSc IS program. For example, since the MSc IS program has course requirements specific to its four building blocks of Foundation, Core, Elective, and Integration, the MScIS class has properties specific to these building blocks.

4. Student Ontology

The Student Ontology includes: basic identifying information (e.g., address, phone number, email, occupation, and job title), course history (e.g., courses taken and courses passed), student preferences, planning parameters, and the currently selected plan.

Students in the MScIS program can maintain their profiles by interacting with the Student Agent using a Web-based interface. Student is a specialization of User and MScIS Student is a specialization of Student with additional properties relevant to MSc IS students.

5. Preference Ontology

In order to enable the selection of an optimal plan from a given set of feasible plans it is necessary to be able to understand and relate a given student’s preferences to course selection and program planning. The Preference Ontology was modeled to assist in this by capturing the student preferences, which can then be related to program topics.

There are five program planning preferences, each with a set of subclasses. The program planning preferences are: Job Objective, Career Track, Assessment Style, Specialization, and Thesis Type.

ONTOLOGY-DRIVEN DEVELOPMENT IN CONTEXT OF E-ADVISOR

As a result of review of the experience developing e-Advisor, a review of documented best practices, and qualitative analysis with subject-matter experts, we have also developed a methodology that supports Ontology-driven Development (ODD) suitable for development of intelligent systems such as e-Advisor.

Areas reviewed and covered include documented methodologies, tools, and best practices for ontology building, ontology refactoring (normalization, segmentation), ontology discovery and integration, Model-Driven Architecture, iterative and incremental software development, and integration with project governance frameworks. Some findings from the development activity and especially the qualitative analysis are related in Deline, Lin, Wen, Gasevic, & Kinshuk (2007). In this section, we discuss why ODD was chosen as being well-suited to intelligent systems such as e-Advisor, stakeholder roles, objectives and review areas, and an overview of the methodology.

A. Why Ontology-driven Development?

We looked at ontology-driven development for our methodology from early on given the benefits of ODD and how well ODD integrated with e-Advisor development practices. We can see that agents use the ontologies as the basis
for their reasoning in plan generation, prerequisite determination, and course schedule negotiation.

The use of ontologies can be seen as an evolution of the accepted best practice of separating business knowledge from technical implementation (the ‘what’ from the ‘how’). The use of ontologies to model the domain makes business knowledge explicit, and facilitates communications between the main stakeholders of the software development cycle.

Driving the software construction from the ontology itself ensures that the ontology model remains up to date and that business rules and knowledge are added to the ontology model as opposed to becoming encoded in the implementation. This benefit echoes and has synergies with MDA, where the model becomes the repository of the system design.

Software construction of the e-Advisor itself has been effectively driven by the ontology development and this approach lends itself to other intelligent educational systems as well as systems that incorporate ontologies into their execution in general.

We are developing an Internet-based collaborative advisory system in which the ontologies serve as standardized and open interfaces for the interoperability of distributed advisory systems.

B. Main Stakeholders

There are two main stakeholder roles in the methodology we developed: Sponsor and Development Team. The Sponsor funds, authorizes, guides, and, when necessary, stops a project. The Development Team refers to both those with Analysis and Development skill sets, including knowledge workers and software developers.

The Development Team is most immediately impacted by any new technology, tool, or methodology. A software Development Team has many people involved, and typical roles/titles include Business Analyst, Systems Analyst, Technical Architect, Solutions Architect, Systems Architect, Developer (many types), Tester, Project Lead, and Project Manager. This article focuses on the Analyst and Software Developer roles, and considers them to subsume all other Development Team roles key for the software construction phases with the exception of Project Lead/Project Manager. The Project Lead/Project Manager role is important and can be considered implicit in the communications between the Development Team and Sponsor and in administration of the processes followed by the Development team but it is not addressed explicitly in this article.

Projects require resources – people to work on them, materials that are used or consumed in the process of execution. These resources are provided or funded by a Sponsor. From the Sponsor’s point of view, a project is successful if it meets the need that started it within required timelines and does not exceed the budgeted resources. A Sponsor may cancel a project if it appears probable it will overrun budgeted resources or in response to a change in strategic goals. This article presents the Sponsor as the primary stakeholder role outside of the Development Team.

C. Objectives and Review Areas

The process of developing a methodology suitable for development of ontology-driven applications was framed by the example of e-Advisor and focused by a number of objectives and review areas.

Guiding principles for our methodology were that it fit within the Software Construction phases of a typical project life cycle, and fill the needs of both Sponsor and Development Team stakeholders (proactively deal with potential resistance or concerns from the former, support adoption by the latter). Objectives specifically for the Development Team stakeholders included that the methodology be easy to use (for the Development Team), and build on existing skill sets (be evolutionary, not revolutionary). Objectives specifically for the Sponsor stakeholder included that the methodology not introduce significant changes...
to the way the Sponsor normally interacts with delivery projects.

Focus areas were also identified including ontology reuse and roles on the development team. In terms of ontology reuse, one of the benefits of starting the software construction process with ontologies is the potential to reuse existing ontologies as opposed to developing new ontologies for each application. There are a number of issues related to enabling ontology reuse, including: how to encourage ontology reuse; how to catalogue/publish, discover, and evaluate available ontologies for use; ontology integration; and ontology interaction.

Roles and responsibilities were also focal points. In an environment that uses ODD, the development of the ontologies themselves needs more attention. As an artifact that documents the domain, drives the development of and serves as an important runtime element of the system, both domain and technical expertise are required to develop an ontology.

D. Methodology Overview

Excluding project initiation and deployment of the finished product, our methodology can be viewed as being comprised of three major activities: plan what needs to be done, prepare to do what was planned, and execute against the plan to build what is needed. These high level processes are shown in Figure 2, as ‘Plan’, ‘Finalize Resourcing’, and ‘Build’. Following is an overview of these processes with a focus on the ontology-related artifacts that link these processes together.

1. Plan the work (‘Plan’ process)

‘Plan’ encompasses both what is going to be done (top-down from architecture) and how it will be done (bottom-up work plan).

First, what is going to be done is planned in a top-down manner starting with key scenarios that illustrate system requirements. The main ontology-related artifacts of interest that are produced are the Ontology Architecture and Ontology Skeletons. The Ontology Architecture shows all needed ontologies and how they are interrelated. For each ontology an Ontology Skeleton is produced, which is essentially a lightweight ontology definition showing the important classes that directly support the scenarios as well as those that are required for integration.

Figure 2. Overall flow

1. need identified event

Copyright © 2009, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
The Ontology Skeletons are used to perform bottom-up planning and assist in development of the Work Plan. For each ontology, build/buy decisions are made, a build/integration approach is determined and cost/effort estimates are developed. The aggregate Work Plan takes into account the output of this planning to determine task dependencies and sequencing and resource requirements.

2. Prepare to do the work (‘Finalize Resources’ process)

Once the planning is done, and before construction can commence, resourcing needs to be confirmed or adjusted and the Work Plan may require adjustments due to resource availability.

3. Build (‘Build’ process)

Iterative and incremental development of the ontologies and ODD of related software takes place according to the Work Plan and following best practices and guidelines validated with knowledge and software development specialists. In the case of ontology build/refactor, for each increment, the ontology is iteratively and progressively modeled, using prototypes where applicable. Once the ontology model is considered stable enough, the ontology itself is used to drive associated software development. For each increment’s application development work, driven by the same increment’s ontology development, the main steps are: develop application models based on ontology/ontologies; develop the application increment and unit test; and integration test.

Section V relates these main processes in our methodology, with a focus on ontology-related activities and artifacts. One of the assumptions of this methodology is that it is used for systems where ontologies not only drive the development but are also key run-time elements, as is the case with e-Advisor.

**PROCESS FLOWS**

The following process flows give an overview of our methodology for ontology-driven development of intelligent systems, focusing on activities performed by the Development Team that are directly related to software construction. This article discusses at a high level how these processes fit in a framework that fills the Sponsor’s needs. While the processes described in this article work with processes such as project management, requirements management, and quality control, this is not discussed in detail in this article.

**A. Top Level Flow**

The overall process flow is illustrated by Figure 2 and the processes are listed below, along with their outputs.

A need is identified and it is decided to investigate starting a project to address the need. Generally sufficient funding is allocated for estimates to be arrived at or funding limits will be set.

2. Initiate Project

This methodology holds the Sponsor accountable for initiating the project. Project initiation has ties into strategic investment decisions and, although capital planning and related topics may seem unrelated to the subject of systems development, we felt it important that our methodology not ignore the importance of this type of activity considering it is a prerequisite for development to occur. This initiation may include gathering very high level cost/effort/time estimates from the groups involved in the project, or detailed cost/effort/time estimates for arriving at higher-confidence estimates.

3. Govern Project

Project Governance is how projects are brought into alignment with strategic goals. For ex-
ample, based on the progress of the project or alignment with strategic objectives of the organization, a project can be stopped or put on hold at any time. Specification of this and other governance activities is outside the scope of this article - each organization will have a different model for governance, with different funding and go/no-go gates. The intent is to provide a structure that supports the overlay of governance, allowing for the ontology modeling and ODD processes to fit within existing models of project governance. For an overview of project governance and how business strategy is supported by projects via governance see Morris & Jamieson (2004).

4. Plan process

Following project initiation the Development Team is accountable for developing a high-level architecture and strategy for the ontology modeling and ontology-driven software construction work to be done. The Plan process is described in more detail below in ‘B. Plan process flow’, ‘C. Plan – Architecture’, and ‘D. Plan – Ontology-Specific Strategy’.


5. Finalize Resourcing

In some situations, resourcing may need to be finalized. This is especially true where people are working on multiple projects.

Based on this activity, the timelines, activity sequencing, and resources assigned to the Work Plan may change.

Output: Work Plan (updated)

6. Build

Based on the output from Plan, and working against the Work Plan, ontology modeling and ontology-driven software construction occurs. This activity is discussed further in ‘E. Build’, below.

Output: Ontologies, Design Models, Software

7. Support Deployment

Once the construction is complete, the system is deployed. The deployment of an application into an existing production environment can be considered a parallel, related project to the development activities this article focuses on and incorporates necessary processes related to maintaining the integrity of the production system (e.g. configuration management) and ensuring appropriate adoption of the changes in production (e.g. people/organizational change management). Although these activities are outside the scope of this article, the Development Team will need to support such activities via knowledge transfer and assistance in resolving unforeseen issues during the actual deployment.

B. Plan Process Flow

The Plan process sets the stage for all further development and allows for the Sponsor to have visibility into schedule and cost baselines.

At the end of this process flow, there will be more certainty around timelines and effort. This is important for both the Delivery Team and the Sponsor. The Delivery Team needs a consistent vision of where they are going in order to work together and execute effectively and the Sponsor needs to know what is going to be delivered when and confirm that the criteria used to approve the project are not compromised (for the purposes of this article we can read this in terms of forecasted spend against delivery timelines and scope).

Important ontology-related deliverables from the Architecture subprocess include an Ontology Architecture, Ontology Skeleton for each ontology, and an integration strategy as applicable. Following this, during the Ontology-specific Strategy subprocess, plans for the development/acquisition of each ontology identified in the Architecture are developed. Finally, using the ontology-specific strategy work and the architecture, the Work Plan is developed. This Work Plan is used to drive the ontology modeling and ontology-driven software con-
struction and serves as a very important hook into the governance framework.

Figure 3 shows the Plan process flow and the subprocesses and their outputs are described below and in ‘C. Plan – Architecture’ and ‘D. Plan – Ontology-Specific Strategy’.

1. Architecture

The first activity involves development of an architecture. Of interest to this article are the ontology-related artifacts within this architecture, specifically the Ontology Skeletons and Ontology Architecture. The Ontology Architecture illustrates the main ontologies that are involved in the system and how they are related. An Ontology Skeleton illustrates, for a given ontology in the Ontology Architecture, the key classes including any classes that are involved in relationships to other ontologies. The skeletons can be seen as templates that can assist in identification of candidate ontologies for reuse/integration or be used as integration stubs. During the Architecture task, key scenarios for the product/functionality to be developed are documented and required ontologies are identified. For each ontology that is identified the concepts to support the scenarios and integration are identified to form a Skeleton Ontology. This assists in sourcing ontologies for integration/reuse and assists in the decoupling of related construction by providing an integration stub. The Ontology Architecture relates all the ontologies with their key concepts and, at a high level, how they are integrated. Note that strategic direction or policy may influence the output of this activity by mandating that certain integration methods are followed or certain ontologies are used. This activity is described below in more detail below in ‘C. Plan – Architecture’.

Output: Ontology Architecture, Ontology Skeleton (or identification of known or mandated ontology to be integrated with).

2. Ontology-Specific Strategy

During the Ontology-Specific Strategy activity, the construction activities related to a specific ontology are examined and decisions that impact
or involve the construction and cost are made. These decisions, which include things such as build/buy/integrate/reuse and mandated or preferred ontologies for reuse or integration, are documented in an Ontology Strategy. Note that organizational direction or policy may influence this strategy.

This activity is described in more detail in ‘D. Plan – Ontology-Specific Strategy’, below.

Output: Ontology Strategy, including potential ontology sources, selected ontology source, Integration/Build approach, and cost/effort estimates.

3. Develop Work Plan

During the Develop Work Plan activity a work plan is developed to drive the actual construction. The Work Plan at this point in time is based on effort estimates, assumed resourcing, and decisions that have been made during Architecture and Ontology-Specific Strategy. This Work Plan will not be baselined until after resourcing is finalized.

Output: Work Plan

C. Plan - Architecture

The architecture includes the main ontologies to support the key scenarios to be supported by the system/functionality to be developed. In addition, for each case where the ontology has not been decided, an ontology skeleton is developed to assist in sourcing the ontology and aid construction activities.

This process flow is illustrated in Figure 4, and the activities and their outputs are listed.

1. Identify Scenarios

The first activity is to identify the key scenarios to be supported by the new system/functionality. This is analogous to documenting system requirements via use cases or scenarios. Our intent is to both document required system functionality and assist in the identification of required ontologies.

Output: Key Scenarios to be handled by the system

2. Identify Ontologies

This task follows on the Develop Scenarios activity and involves identifying ontologies
that are required to support the Key Scenarios identified earlier.

Output: descriptions of ontologies required to support the Key Scenarios identified by the Develop Scenarios activity.

3. Develop Ontology Skeleton

In the event that the source of a given ontology is not mandated, a skeleton ontology is developed, including the Concepts required to support related Key Scenarios. This skeleton serves as a template for sourcing the ontology, and as an input to the construction activities.

Output: Ontology Skeleton

4. Ontology Integration Strategy

The Ontology Integration Strategy determines how the ontologies are interrelated and relates the strategy for integrating the ontologies. It is finalized after all ontologies are identified and their skeletons defined.

Output: Integration Strategy

5. Document Architecture

This task involves packaging the output from all previous tasks into an overall architecture document.

Output: Ontology Architecture

D. Plan – Ontology Specific Strategy

The Ontology-Specific Strategy process is executed for each ontology in the Ontology Architecture. The ontology’s source is identified (re-use or refactor an existing ontology, build from scratch), the approach for integrating with or refactoring an existing ontology or building a new ontology is decided upon and, based on that, cost/effort estimates are developed. The flow is illustrated in Figure 5 and activities and their outputs are listed below.

1. Identify Ontology Source

The Ontology Skeleton can be used as a template for identifying potential ontology sources. This is analogous to “Candidate SRs Identification” in (Barresi, Rezgui, Lima & Meziane, 2005). Sources of candidate ontologies include sources

Figure 5. Plan – ontology-specific strategy flow
internal to the organization and external to the organization.

Output: potential ontology sources for the ontology, selected Ontology Source

2. Determine Integration Approach

If an existing ontology has been identified, how that ontology is going to be re-used is determined, along with associated costs.

Output: Integration Approach, Cost/Effort Estimates

3. Determine Build Approach

If the decision is made to build a new ontology, how that ontology is going to be built is decided, and thus the activities and cost/effort estimates determined. One approach is top-down modeling to obtain a clear structure and then incremental addition of concepts to this structure, classifying and normalizing as this is done.

Output: Build Approach, Cost/Effort Estimates

4. Document Strategy

Based on the results of preceding tasks, the strategy is documented.

Output: Ontology Strategy

E. Build

The Build process is where the ontologies and software are implemented and is driven by the project-specific Work Plan. Depending on the Work Plan, there may be a number of parallel and interrelated instances of this process involved in one project. This is primarily a function of the number of ontologies and their interdependencies as well as resource availability and capacity. Our methodology is biased toward an integration of incremental ontology development and downstream Model Driven Architecture for development of software components.

This use of incremental development and application of Agile techniques to ontology development is in line with suggestions made by Knublauch (2006) and was validated and confirmed in focused interviews with software development and knowledge experts.

Each increment focuses on ontology development to a point where the model can drive the application design and/or the knowledge base can support application execution, followed by ontology-driven development of related software.

While modeling is both a science and an art, certain guidelines, suggested by the literature and confirmed by our review of e-Advisor development and qualitative analysis, are documented in our methodology. For ontology development, the main steps are: develop the ontology in increments, validating each increment with Protégé’s built-in reasoners, and creating individuals to validate semantic restrictions (cf. Knublauch (2006)); prototype as appropriate; maintain tree structure based on increasing specialization (untangle as required), clearly differentiate partitioning Concepts (used to untangle) from self-standing Concepts (cf. Rector (2003)).

For each increment’s application development work, which is driven by the same increment’s ontology development, the main steps are: develop application models based on ontology/ontologies; develop the application increment and unit test; integration test.

CONCLUSION AND FUTURE WORK

This article has reviewed e-Advisor as an intelligent web-based intelligent advising application and a methodology for Ontology-Driven Development that is suitable for intelligent systems such as e-Advisor, whose execution is driven by ontologies. Our methodology, developed using the e-Advisor system and development as a case study, and reviewed with knowledge and software development experts, meets the needs of the project Sponsor as well as the Development Team, leverages traditional software engineering techniques to support
incremental, iterative ontology modeling and ontology-driven development of intelligent systems.

Future work and areas to review include incorporation of ontology change management, reverse-engineering of ontologies, and ontology development estimation.

REFERENCES


ENDNOTE

Gordon Deline is a senior project manager, project and risk management, at CBC/Radio-Canada. Gordon has over 20 years of experience in software systems research and development in North America, Asia, and Europe, designing and implementing virtual reality, telerobotic/supervisory control, object database, supply chain, reverse logistics, and business intelligence systems. Current research interests include systems development in the context of alignment and integration with strategy and business knowledge, and distributed heterogeneous implementation teams with client/vendor/competitor membership.

Fuhua Lin is professor of school of computing and information systems of Athabasca University, Canada. He is conducting research in intelligent systems, multi-agent systems, virtual reality, and their applications. Lin has more than sixty publications, including edited books, journal papers, book chapters, conference papers, and reviews. Lin’s current main research interest is in the design and the development of reasoning models and mechanism of intelligent agents that facilitate adaptive learning. Lin has taught various undergraduate as well as postgraduate courses in mathematics, computer science, and information systems. Furthermore, he has led and actively participated in numerous national and international research projects since 1986.

Dunwei Wen is an associate professor in the school of computing and information systems at Athabasca University, Canada. His research interests include natural language processing, information retrieval, machine learning, intelligent agents, knowledge representation, reasoning and discovery, and software architecture for business and e-learning. Wen’s current research aims to advance theories and techniques that facilitate accessing, sharing and reusing different forms of information and knowledge in learning communities through intelligent services such as automated question-answering, knowledge querying and text mining.

ENDNOTE

http://www.athabascau.ca
Dragan Gašević is an assistant professor in the school of computing and information systems at Athabasca University and an adjunct professor at Simon Fraser University. He is a recipient of Alberta Ingenuity’s 2008 New Faculty Award. His research interests include semantic technologies, software language engineering, and learning technologies. He has (co-)authored around 190 research papers published. He has been serving on editorial boards of three international journals and has edited special issues in journals such as IET Software and IEEE TSE. He has been the organizer, chair, and member of program committees of many international conferences.

Kinshuk is professor and director of school of computing and information systems at Athabasca University, Canada. He also holds iCORE/Xerox/Markin Industrial Research Chair in adaptivity and personalization in informatics. He has been involved in large-scale research projects for exploration based adaptive educational environments and has published over 200 research papers in international refereed journals, conferences and book chapters. He is founding chair of IEEE Technical Committee on Learning Technology and editor of the SSCI indexed Journal of Educational Technology and Society.