ATHABASCA UNIVERSITY

Ontology-driven Methodology for Constructing Software Systems

BY

Gordon Deline

A thesis submitted in partial fulfillment

Of the requirements for the degree of

MASTER OF SCIENCE in INFORMATION SYSTEMS

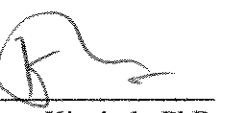
Athabasca, Alberta

October, 2006

ATHABASCA UNIVERSITY

The undersigned certify that they have read and recommend for acceptance the thesis essay ONTOLOGY-DRIVEN METHODOLOGY FOR CONSTRUCTING SOFTWARE SYSTEMS submitted by GORDON DELINE in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE in INFORMATION SYSTEMS.

Oscar Lin, PhD
Supervisor



Kinshuk, PhD
Chair

Dunwei Wen, PhD
Examiner

Date: December 2006

Abstract

Ontologies and Ontology-based systems are key building-blocks of intelligent Internet-enabled software systems. Ontology-driven Software Construction, where the software construction process itself is driven by the Ontology, is appropriate for the development of intelligent systems.

This paper explores what approach to Ontology-driven Software Construction would be suited for the development of intelligent systems. Athabasca University's eAdvisor intelligent academic advising system is used as an application example of an intelligent, Ontology-based software system. The methodology proposed in this paper synthesizes documented learnings and guidelines related to Ontology development and Ontology-driven Software Construction, to provide a method that is usable by the Development Team while being able to meet Sponsor needs by being able to fit within typical project governance frameworks.

A literature review of research related to eAdvisor, intelligent systems, software development methodologies, Ontologies, Ontology development and Ontology-driven Software Construction was used to develop a baseline Ontology-driven Software Construction methodology incorporating documented best practices and guidelines. This methodology baseline was reviewed with technical experts in focused interviews and, following qualitative analysis of the interviews, the methodology baseline was updated to incorporate input and address concerns raised during the course of the interviews.

Acknowledgements

Special thanks and appreciation are owing to the following individuals and groups.

- Dr. Fuhua Lin, my advisor and paper supervisor, for all his generous and clear advice and guidance throughout the preparation of this thesis and for access to eAdvisor and DELTA Group working papers.
- The people who agreed to take time away from their busy schedules to be interviewed by me in support of this thesis: Gail Aller-Stead, Louise Bellingham, Georgia Curtis, S. Masters, Chris Maxwell, Rob Nalecz, Dan Taylor, Liz de Villers-Seeley, Dr. Trisha Wilcox, and Irina Zaydman.
- The always helpful staff and faculty of Athabasca University's Master of Science in Information Systems program.
- My clients for their understanding and flexibility while I completed this thesis.
- My friends and family for their unwavering patience and understanding whenever I declined or rescheduled events, or made a show of attending while reading through papers.

Table of Contents

List of Figures

1 Chapter One – Study Background

1.1 Research Question

The research question is what kind of approach to Ontology-driven Software Construction might be suited for the development of systems, specifically intelligent education systems. An application example is being used to help answer this question – the eAdvisor intelligent academic advising system developed by Athabasca University's DELTA lab.

1.2 Research Goal

The objective of this thesis is to propose a methodology baseline for practical Ontology-driven construction of intelligent educational systems including analysis of the challenges, benefits, and drawbacks of the proposed methodology baseline. Methodologies, tools and techniques appropriate for Ontology-driven construction will be reviewed, and observations will be documented and analysed. Results of qualitative analysis will be incorporated to produce a methodology baseline suitable for Ontology-Driven Software Construction of intelligent academic advising systems such as eAdvisor.

1.3 Problem Statement

The problem statement involves Ontology-Driven Software Construction in general, Ontology-Driven Software Construction of Intelligent Education Systems in specific.

1.3.1 Ontology-Driven Software Construction

Ontologies, and more generally ontology-based systems, have emerged as a central issue for the development of efficient Internet-based applications. They represent a way to efficiently access a large body of Internet information spaces. Ontologies aim at capturing domain knowledge in a generic way and provide a commonly agreed understanding of a domain, which may be reused and shared across applications and groups. Ontologies provide a common vocabulary of an area and define, with different levels of formalism, the meaning of terms and the relations between them. Ontologies are typically organized in taxonomies and modeled using primitives such as classes, relations, functions, axioms and instances.

Ontology-driven software development holds much promise in general and is applicable for the development of intelligent education systems such as Athabasca University's eAdvisor but there is no widely-accepted standard methodology. Recent development of the eAdvisor system has been effectively driven by prototyping and construction of Ontologies. A methodology baseline built on best practices and qualitative analysis would allow for quantitative analysis and improvement of the software development process. Few if any companies will share information about their projects at all, and even fewer would freely discuss cancelled or failed projects. The author of this paper has personally worked on and seen a number of commercial projects using promising newly-introduced technologies or methodologies (including a project to migrate several product lines from isolated data-driven systems to an integrated meta-meta-data-driven infrastructure with many similarities to Ontology-driven systems) cancelled, with just cause, due to cost or time overruns or insufficient visibility into the process to produce

reasonable estimates or manage risk. This greatly lowers the chances of the technology being approved for use on subsequent projects and has a demoralizing impact on the team members working most directly with the new technology. In order to be widely adopted, Ontology-driven software construction needs to be acceptable to both those who work directly on the software specification and construction and those who initiate and govern projects.

For the purpose of this paper, the Development Team is composed of analysts and software development professionals, including those whose roles span both areas of expertise (systems analysts, knowledge engineers, solutions architects, etc.). For the Development Team, a methodology needs to guide them in their work and provide or embody best practices and guidelines.

For any methodology for Ontology-driven software development to be adopted, it also has to meet the needs of other major stakeholders, especially those that can start and cancel projects. This paper reduces these other stakeholders to the single role of Sponsor. The Sponsor is primarily concerned that the project is successful and predictable – that the requirements are met in the allocated time and budget.

1.3.2 Application Example: eAdvisor

The application example is an intelligent academic advising system that has been developed by Athabasca University's DELTA lab for graduate student use. This application example is appropriate as the system itself is ontology-driven, and portions of the system have been developed using what is in effect Ontology-driven software construction techniques. The intent is to trial the baseline methodology in future development work involving eAdvisor.

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 they encounter a problem.

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 member. As Carroll and Chappell point out in their proposal for an intelligent student advising system, the constraints currently faced by faculty of post-secondary educational institutions are such that little time is left for faculty to personally advise students [1].

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 tends to take much longer, courses may be taken out of sequence due to students' prior learning and experience or to facilitate immediate career objectives.

Systems have been developed to assist in this, however there are limitations with these systems. Agent-based technology is applicable to this problem domain, as is multi-Agent technology.

At Athabasca University, Lin, et al. have proposed a multi-agent service-oriented architecture (SOA) to support the development of e-Learning systems [2] using extended Petri nets to model course dependency relations [3]. Lin et al. [4] also propose a multi-agent system (MAS) that supports program planning and intelligently provides academic

advice. This system is currently in trial mode at Athabasca University and has been developed by the DELTA group of Athabasca University.

2 Chapter Two – Literature Review

This chapter summarizes and discusses research relevant to the research goal of this thesis essay. Material is organized into sections covering the application example of Athabasca University's eAdvisor system, software development methodologies, ontologies and ontology development, and finally Ontology-driven software construction. Additional background on areas not directly related to the direct focus of this paper but potentially useful for gaining an understanding of the eAdvisor Intelligent Advising System, are included in an appendix (see 10 Appendix : Further Background on Agents and Multi-Agent Systems).

*2.1 Application Example: **eAdvisor***

2.1.1 The Need for Intelligent Advising Systems

Trends toward lifetime learning, online learning, distance education, and the push for leaner operations at learning institutions have driven the demand for intelligent academic advising systems. Intelligent Academic Advising Systems fill a need both for the student and for the learning institution given that the advising process itself is becoming more complex and demanding while the time academic advisors can dedicate to each student is decreasing.

The complexity of academic advice required is increasing, both in terms of available options, preferences and constraints, and in terms of the need for replanning when the student or institution's situation changes. This is especially true in the case of distance

learning and eLearning. The many options available for lifelong, online and distance education means that the individual learners and thus the advice needed varies widely by each individual, and may change for a given person as that individual's situation changes over time. Also, programs take longer to complete, making the possibility of changes to the individual's situation as well as the various education service providers' situations more probable.

The same trends that are increasing the effort involved in academic advising are also resulting in an increased number of students to be advised. Thorpe [5] points out that given the wider audience for online and distance education, the load on advisors and tutors is much larger than in traditional learning institutions where the load can be better controlled. At the same time that the number of students to be advised is increasing, the amount of time staff can dedicate to advising students is actually decreasing as educational institutions push for leaner operations.

To some extent the pressures on staff at academic institutions can be relieved through business process reengineering such as the delegation of mentoring tasks to students as described in [6]. This type of reengineering can not be seen as the complete solution and is perhaps best suited to traditional "bricks-and-mortar" teaching institutions where students are physically co-located and the physical structures themselves place limits on the potential rate of growth of the student body.

2.1.2 Intelligent Academic Advising Systems

The use of Intelligent Academic Advising Systems to ease the burden on faculty and students alike is not a new idea and many systems have been proposed and developed. Gunadhi et al. [7] of the University of Singapore proposed an intelligent advising system

entitled PACE and discuss strategic, tactical and operational advising activities. Carroll and Chappell [1] describe a system to assist students at Sam Houston University in Texas. Two technical issues with these systems are readily apparent - they are monolithic in structure and the advising logic is tightly coupled to the system itself. More importantly perhaps, these systems are geared toward traditional on-site university learning where the students attend classes full time and complete their program in a proscribed period of time and as such do not deal with the issues often seen by the burgeoning body of distance education students.

The system described by Van Biljon et al. [8] is intended for use in an environment that shares some of the challenges seen with distance education, including longer periods of time to complete programs (such as the increased need to deal with program changes and supercession) and flexible student learning in terms of pace through courses. This system, however, is intended more as a support tool that allows the user to browse through required courses to be completed than an intelligent advisor.

2.1.3 eAdvisor Background

Lin, et al. proposed a multi-agent service-oriented architecture (SOA) to support the development of e-Learning systems [2] and the use of extended Petri nets to model course dependency relations [3]. The e-Advisor system itself is described by Lin et al. [4] as a multi-agent system (MAS) that supports program planning and intelligently provides academic advice. As discussed below, eAdvisor incorporates several Ontologies for domains relevant to academic advising, and eLearning. The system's execution is driven by these Ontologies and the development of the system itself has become effectively Ontology-driven [9].

The agent platform used by eAdvisor is JADE (Java Agent DEvelopment Framework) [10], which supports the development of FIPA compliant agent systems.

2.1.4 eAdvisor Architecture - Components

The eAdvisor architecture consists of Ontologies, Personal and Task Agents, and Web Services, as described in more detail below.

Ontology:

eAdvisor Ontologies are modelled in Protégé OWL and accessed by system components via a Java API. The main eAdvisor Ontology is the MSc IS Ontology - a domain Ontology that describes the MSc IS programme.

Personal Agent (PA):

In the context of eAdvisor, the User Interface Agent (UIA), a Personal Agent (PA), is the user's direct interface with the eAdvisor system. Personal Agents act on behalf of the user and interact with Task Agents. Users of the eAdvisor system interact with their Personal Agents via secure web pages. Personal Agents exist to support the Learner as well as Faculty users. Faculty could be Instructors, Advisors, or Administrators. To fulfill their objectives, Personal Agents interact with Task Agents (see below) and can interact with other Personal Agents in the system.

Task Agent (TA):

A Task Agents (TA) provides services to other Agents (both Personal and Task Agents). Task Agents in eAdvisor spool and prioritize requests for their services. Task Agents are deployed on an Agent Platform which supports Agent discovery, deployment and inter-Agent communication. Task Agents in eAdvisor include the Planning Agent, Notification Agent and Monitoring Agent.

Web Service (WS):

The eAdvisor architecture has been refactored to incorporate Web Services (WS) to provide interfaces to Agents and Ontologies in the eAdvisor system. Specific Web Services shown on the eAdvisor architecture diagram include the Learner Information WS, the Course Schedule WS, the MSc IS Ontology WS and the Notification WS.

2.1.5 Agent-assisted Program Planning With eAdvisor

Lin et al. [4] describe agent-assisted program planning, as supported by eAdvisor, as occurring when the student first enters a program, for each semester, and in response to changes that require re-planning. As such, planning can be categorized as strategic, tactical, and opportunistic, respectively.

Strategic Planning (entry into program):

When the student enters into the program the learner profile and preferences are first build up and an initial plan is generated as described under Tactical Planning, below.

Tactical Planning:

Tactical planning can start prior to the course schedule being finalized for a given semester, allowing administrators to base course offerings on student interests. The student's profile and preferences are updated as required and the student interacts with a Personal Agent to select courses the student would like to take over the next several semesters. The Personal Agent suggests courses based on the program requirements, learner preferences and objectives, and the student selects from these suggestions.

Opportunistic Planning (event-driven):

The eAdvisor Monitoring Agent detects a change has occurred, including the course schedule for a given semester becoming available, it notifies the Planning Agent, which

generates a new plan. The Planning Agent then notifies the learner of recommendations via the Notification Agent.

2.1.6 eAdvisor Ontology-based Knowledge Model

The eAdvisor knowledge model, as described in [4], has Ontologies as its basis. Ontologies are used to model the MSc IS program, courses, and also Concepts related to the learner (the Student Model).

MSc IS Ontology:

The MSc IS Ontology models the MSc program structure, and regulations, with Concepts such as Program, Course, and Objective. This Ontology is modeled after MSIS 2000 [11], Athabasca University's curriculum model and the ACM Computing Classification System (ACM CCS) [12] Ontology.

Course Model:

The Course concept itself is elaborated by the IEEE LOM Ontology [13] and can be considered a specialized Learning Object. Courses are indexed with metadata and are related to one or more ACM CCS-derived concepts in the MSc IS Ontology.

Student Model:

eAdvisor maintains student profiles, including basic information, preferences, objectives, relevant skills and courses. The student model was implemented as an extension of the IMP LIP (1.0.1) [14] standard to allow for interoperability of Learner Information Systems and the exchange of Learner information.

Pre-requisite Relations:

Course dependency relationships are modeled in eAdvisor using extended Petri nets. The use of Petri nets allows for potential shortening of the program duration through

concurrency by modeling of any course parallelism inherent in the curriculum. Petri nets allow for the modeling of priorities/preferences, synchronization (co-requisites and pre-requisites), and exclusion and can be verified for completeness and correctness. The extended Petri nets generated by eAdvisor model the relations between courses and can be viewed as a state machine the student traverses as course topics are successfully completed, thus meeting the pre-requisite requirements for further course topics. The student's state at any given time indicates topics that have been successfully completed and thus their knowledge level in the learning area and any valid firing sequence across the Petri nets comprises a feasible plan for program completion.

Populating the Knowledge Model:

The MSc IS Ontology was modeled based on the MSc IS Curriculum, ACM CCS Ontology and IEEE LOM Ontology. As new courses are added to the curriculum, course authors fill in natural language course descriptions and these are added to the MSc IS Ontology, tagged with metadata as per the IEEE LOM standard. These metadata tags enable the translation of descriptions into pre-requisite relations that are stored in Petri Nets Markup Language (PNML) [15] files for verification.

2.2 Software Development Methodologies

This section provides relevant background on software development and methodologies, to better frame what is reviewed in later sections.

2.2.1 Project Life Cycle and Stakeholders

The Project Management Institute defines a project as a "temporary endeavour undertaken to create a unique product, service, or result." [16]. Projects are started to fill

a need in a domain, which may be triggered by, among other things, an idea, a required upgrade, a problem with an existing system, or in answer to a strategic goal. Prior to the project starting up or in the early phases of a project and prior to the project being approved for full funding, a value assessment and cost/benefit analysis usually takes place, to justify the cost of the project in terms of the expected benefits, the output documented in a business case for the project. At the end of a successful project, a work product is deployed that fulfills the need which started the project. Projects can be divided into phases and, as projects are temporary and have a beginning and an end, these phases can be categorized as initial, intermediate, and final. When dealing with a software system, there may be many projects during the lifetime of the system itself, encompassing the system's initial development, upgrades, rollouts and even projects to close out the system. The focus of this paper is on Ontology-driven software construction and so only touches on the inputs to and outputs from that process.

The figure below illustrates the above graphically, calling out the phases where software construction takes place and thus where Ontology Driven Software Construction fits.

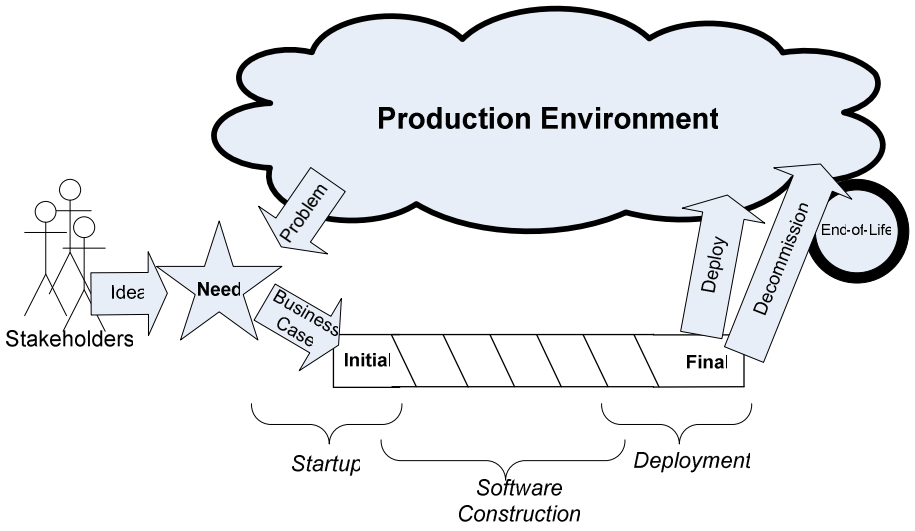


Figure 1 Software Construction in Context of the Project Life Cycle

Project Stakeholders:

Individuals and groups that work on a project, are impacted by the project, or influence a project, are referred to as the project's Stakeholders. Typical Stakeholders in a software development project include the Development Team (responsible for delivering the software work product), User (will directly use the finished work), Customer (receives work product), and Sponsor (pays the bills). One individual or group may have multiple Stakeholder roles. For example, the Sponsor for a project could also fill the User and Customer roles.

As stated in the Problem statement, this paper focuses on the outline of a methodology primarily taking into account the Development Team and Sponsor stakeholders.

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 paper 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 but beyond the scope of this paper 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.

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 which started it 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 strategic goals. This paper considers the Sponsor to be the primary stakeholder role outside of the Development Team.

2.2.2 Code and Fix

It is not possible to ignore the software development approach of not following any process at all. It is in reaction to the uncertain results of this type of "coding by the seat of the pants" approach to software development that software development methodologies have been developed.

2.2.3 The Classic Waterfall

The other extreme of the "code and fix" approach is the classic waterfall, where the project proceeds in a series of rigidly adhered-to phases that are strictly followed. In waterfall methodologies of software development, the development of the software occurs in phases that follow one after the other. Each phase's completion results in the creation of work product that leads into the next phase. The standard waterfall phases, as related by Royce [17] are:

1. Requirements
2. Analysis
3. Design
4. Implementation
5. Verification
6. Maintenance

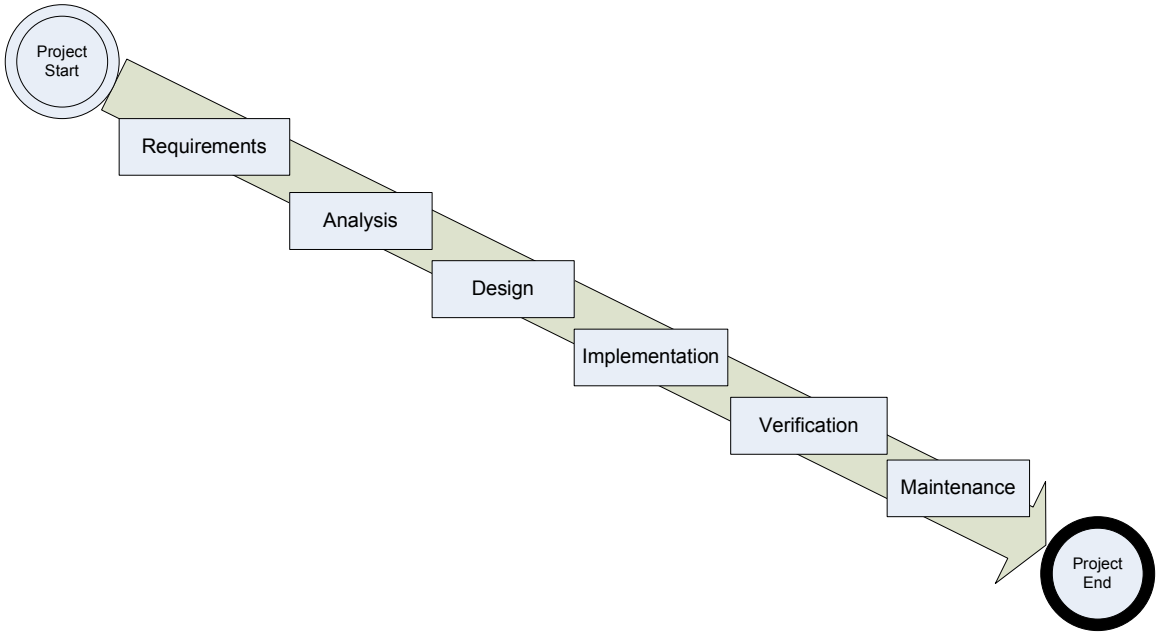


Figure 2 Classic Waterfall Software Development

This model is a very useful one as these phases are common to all software development methodologies – what differs are such things as to what extent they overlap or cycle, whether functionality is delivered in increments, or phases, and what artefacts are generated along the way. The ”pure” waterfall model, where work always flows down and once one phase starts previous phases are considered 100% complete is impractical, and Royce presented it as an idealized model, prior to presenting incremental and iterative variations that would mitigate some of the risks inherent in the ”pure” waterfall model. One issue with insisting on finishing each phase completely before the subsequent phase starts is that things can be discovered in later phases that invalidate what was done in previous phases, leading to potentially large amounts of rework.

2.2.3.1 Structured Systems Analysis and Design (SSADM)

A rigorous software development methodology that became very popular in the 1980's due to its ability to bring control and predictability to software development, Structured Systems Analysis and Design (SSADM) can be considered a classic waterfall methodology, for the analysis and design of software (SSADM does not encompass construction and testing). SSADM is mentioned here both as an example of a rigid waterfall method and because some of the SSADM phases and models have analogues or descendants in more modern methodologies. The main phases in SSADM are Feasibility Study, Requirements Analysis, Requirements Specification, Logical Design, and Physical Design. The main techniques that support this work include Logical Data Modeling Data Flow Modeling and Entity Event Modeling. The Logical Data Structure (LDS) developed during Logical Data Modeling is analogous to an Entity-Relationship Diagram (ERD). In SSADM, Data Flow Modeling produces Data Flow Diagrams, and Entity Event Modeling results in an Entity Life History for each entity.

2.2.4 Iterative / Incremental Development (Early Methodologies)

The iterative approach to software development deals with some of the weaknesses in the "pure" waterfall methodology by having the system development take place as a series of successively refined iterations. Iterative and incremental methodologies are often viewed as a recent development but in fact, as related by Laman and Biseli [18], they have their roots in work done in the 1930's on incremental quality improvement projects and were used in software development projects as early as the 1960's. Development in increments is suggested in Royce's 1970 work [17] and iterative and incremental development can

also be seen in the Spiral approach and forms the foundation of the modern Rational Unified Process and Agile development methodologies.

In a typical iterative development project, a project control list is developed to implement features that meet system requirements in a series of iterations. Iterations are intended to be relatively straightforward and to deliver an incremental amount of new functionality. The initial increment of the system meets a small yet usable subset of requirements. Use and subsequent analysis of this iteration drives redesign of subsequent iterations. This continues for subsequent iterations. This process continues until the system is completely implemented.

2.2.4.1 Spiral Software Development

The Spiral model, an iterative development methodology described by Boehm [19], delivers software in phases lasting from several months to several years. Each phase, or cycle, starts with objectives and ends with a review of what the cycle produced and the formulation of plans for the next cycle. The initial cycle may be a very rough prototype with subsequent cycles incrementally adding functionality until the system is completed.

2.2.5 Rational Unified Process (RUP)

Background:

The Rational Unified Process is an industrial-strength, productized proprietary software development methodology that is based on the spiral software development process. RUP is an iterative object-oriented development process framework initially created by Rational Software, which has since been acquired by IBM. As a framework, RUP is intended to be tailored by each development shop and leverages a number of

Rational/IBM tools to support the RUP workflows through modeling, requirements management, testing, and documentation. Visual modeling in RUP uses the UML, originally developed by Rational and now a standard maintained by the OMG.

RUP has evolved from and incorporated a number of Object-oriented methodologies and techniques, starting with the merging of the Rational Approach and Objectory Process 3.8 to form the Rational Objectory Process 4.0. The Rational Objectory Process also incorporated elements of and support for OMT Booch, UML, Requirements College, and SQA before becoming the Rational Unified Process. The Rational Unified Process added support for Configuration and Change Management, Business Engineering, Performance Testing, Data Engineering, UI Design, and newer versions of the UML.

Phases and Iterations:

Development takes place in the following phases, each of which involves one or more iterations:

- Inception
- Elaboration
- Construction
- Transition

Standard RUP workflows are intended to be tailored for and by each development shop and, in line with the iterative and incremental basis of RUP, these workflows are seen as executing in parallel, with the focus on each workflow varying throughout the software development process.

RUP Best Practices:

Some best practices of RUP, as per [20] include:

- developing software iteratively
- managing requirements
- using component-based architectures
- visually modelling software
- verifying software quality
- controlling changes to software

Tools:

The Rational Unified Process is itself a product and the use of a comprehensive family of tools is indicated by the process. As of the time of writing, the following tools were integrated with the Rational Unified Process:

- Rational Requisite®Pro (Requirements Management)
- Rational ClearQuest™ (Change Control)
- Rational Rose® (Visual Modeling)
- Rational SoDA® (Documentation)
- Rational Purify® (Run-time C/C++ Error-Checking)
- Rational Visual Quantify™, Rational PerformanceStudio™ (Profiling)
- Rational Visual PureCoverage™, Rational TeamTest (Testing)
- Rational ClearCase® (Configuration Management)

2.2.6 Agile Software Development Methodologies - Extreme Programming

Agile Software Development methodologies were developed as a reaction to "heavyweight" processes such as RUP. As can be inferred from the name, Agile

Software Development methodologies are intended to be able to adapt quickly to changing circumstances. Agile methodologies, such as XP, generally deliver software in very small increments of functionality that are timeboxed and are able to respond to changing requirements. Agile methodology best practices are intended to support rapid development and include collocation of resources and a focus on verbal communication over heavy written documentation. Key guiding principles of Agile Software Development methodologies, as documented in the Agile Manifesto [21], include:

- early and continuous delivery in small increments (seen as yielding increased customer satisfaction)
- flexibility to changing requirements, regardless of stage in delivery
- frequent delivery of working software
- collocation of business people and developers throughout the project
- empowerment of the project team members through motivation, trust, and support and use of self-organizing teams
- dissemination of information via face-to-face conversation wherever possible
- use of working software as the key metric for measurement of progress
- avoidance of 'burnout' by maintaining a consistent, reasonable workload on all

Given the focus of Agile methodologies on collocating and empowering the team members, Agile Software Development methodologies are best suited for small-to-medium size project teams made up of more seasoned, senior members.

2.2.7 Model Driven Architecture

OMG's Model Driven Architecture [**22**] can be seen as an evolution of CASE methodologies that leverages the industry-standard UML. MDA focuses on machine-

readable models of the application solution, enabling automatic generation of code and transformations between different modeling languages. In this manner, models become the main artefacts developed and maintained during the software development process. As stated in the MDA Guide [23], model-driven development "provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification." The benefits of a model-driven approach for the application solution are analogous to the benefits of Ontologies for modeling domains (see 2.3.1 Ontologies and Knowledge Bases) but differ in focus. An ontology models a business domain in a machine-readable format whereas the MDA focuses on machine-readable modeling of an application solution.

MDA Viewpoints and Models:

The three viewpoints for MDA models are the Computation Independent Viewpoint, which focuses on the system environment and requirements, the Platform Independent Viewpoint, which describes the system operation in a platform-independent manner, and the Platform Specific Viewpoint which builds on the platform independent model through the addition of platform-specific detail. The main model types that the MDA specifies include the Computation Independent Model (CIM), the Platform Independent Model (PIM), and the Platform Specific Model (PSM), which allow for modelling of the Computation Independent Viewpoint, the Platform Independent Viewpoint and the Platform Dependent Viewpoint, respectively. The MDA provides support for mapping between these models as well as transformations between models.

MDA Process:

The following diagram and associated narrative description illustrates the steps followed in Model Driven Architecture as outlined in the MDA Guide.

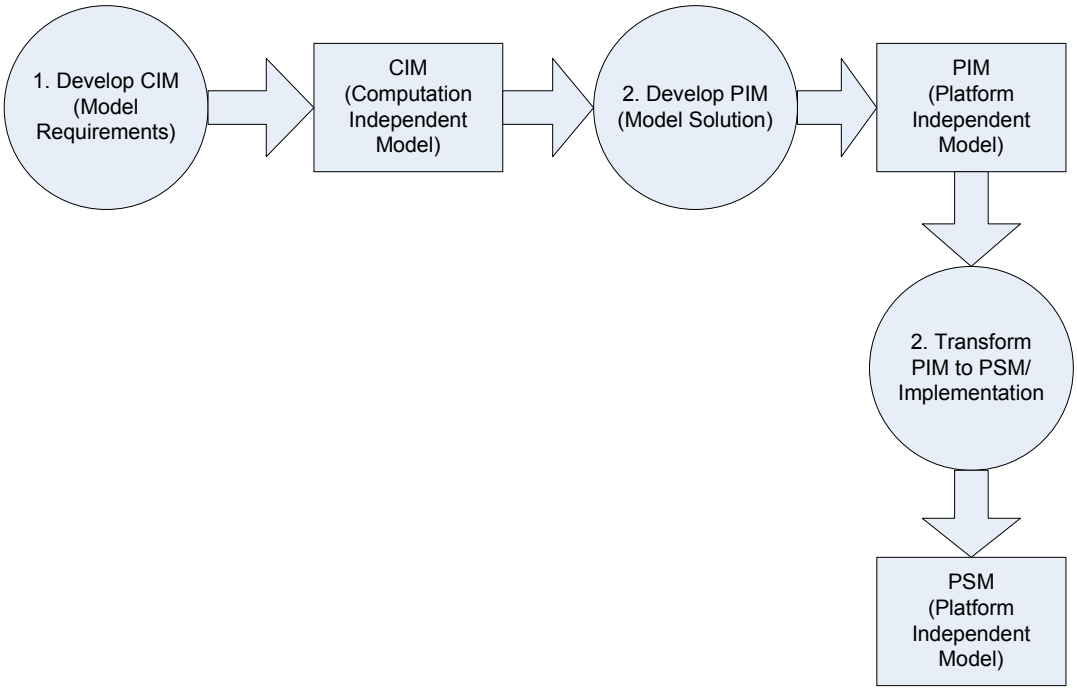


Figure 3 MDA Process Flow

1. Develop CIM (Model Requirements)

Requirements are modeled in a CIM, which is also referred to as the ”domain model” or ”business model”. It is important to note that, despite the familiar nomenclature, the CIM is not equivalent to a domain Ontology. As a CIM is an application-specific domain model it may be developed from a domain Ontology.

2. Develop PIM (Model Solution in Platform-independent manner trace to Requirements)

The system is described in a platform-independent manner via a PIM such that the PIM fulfils the requirements modeled in the CIM.

3. Transform PIM to PSM / Implementation

Finally, the PIM is transformed to one or more PSMs. A PSM could be an executable Implementation or a model that requires further transformation to become an executable Implementation. MDA proposes a number of approaches for transforming models.

One obvious advantage of the PIM to PSM transformation is that it facilitates development of platform-specific Implementations as the need to deploy to new platforms or incorporate new technologies arises. In addition, a single PIM can be transformed into multiple PSMs to support distributed and federated systems, with interoperability mappings providing support for the inter-platform connectors.

2.3 Ontologies, Description Logics, and the Semantic Web

2.3.1 Ontologies and Knowledge Bases

Ontology Definition:

Ontologies explicitly describe a domain of interest in a human-readable and machine-parsable format. In this manner Ontologies act as a repository of domain knowledge and a bridge between various groups involved in software development and also with the computer systems themselves. Ontologies document information about sets of individuals (Concepts, or Classes).

Knowledge Base Definition:

An Ontology that has been populated with data about individuals is termed a Knowledge Base[1]. The relationship between Knowledge Base and Ontology is comparable to that between a Database Instance and a Database Schema insofar as a Knowledge Base is comprised of structure and data that follows the rules of the structure.

Ontology Language:

An Ontology is a model of the world that structurally describes an abstraction of a specific domain using an Ontology Language. Ontology Languages [2]allow for the description of a domain via the major building blocks of Concepts, Relationships, and Constraints.

Concepts describe things that exist in the domain being modelled. They may also be termed ”Classes”, “Entities”, or “Things”.[3] Concepts are described via their Properties[4] common to a set of individuals. Roles or Relationships describe how Concepts in the domain are associated with each other and Constraints complete the domain model.

An Ontology models the world by describing the structure of a domain - what types of things can exist in a domain (Concepts or Classes, along with their Properties), how those things are related to each other (Relationships) as well as any applicable Constraints. The Ontology Language itself may be graphical, such as UML or Entity-Relationship diagrams (ER diagrams or ERD’s), or textual. Ontology Languages based on traditional graphical modelling techniques such as ER diagrams or UML allow for the identification of Concepts, Attributes, and Relationships while other Ontology Languages such as those

[1] This definition is included here for information purposes. This paper uses the term ‘ontology’ to refer to both populated and unpopulated ontologies.
[2] For the purpose of this paper, Ontology Language and Knowledge Representation formalism are considered synonymous.
[3] For the purpose of this paper, Concepts, Classes, and Entities are considered synonymous.
[4] For the purpose of this paper, Properties, Attributes, and Slots are considered synonymous.

based on Description Logics have richer support for Constraints and allow for a much more expressive model to be built. Constraint information is, in practice, added on to the traditional graphical models in the form of memos or annotations. UML supports the specifications of constraints in a standardized manner using the Object Constraint Language or OCL [24]. The main argument against using traditional graphical modeling languages as complete Ontology Languages are that they focus on modelling data (ER Diagramming) or an application solution (UML) as opposed to modeling knowledge about a domain.

Benefits of Ontologies

The major benefits intrinsic to Ontologies, is that they explicitly represent domain knowledge and that are machine parsable. The latter has given rise to the Semantic Web, as described in the following section.].

2.3.2 Description Logics (DL)

There are a number of formalisms for representing knowledge in Ontologies, including frame-based systems, graphs and logic-based languages (see [25], [**26**]). This paper focuses on Description Logics (DL) due to the fact that the application example's Ontologies have all been developed using OWL-DL. Modern DL systems have their foundation in First Order Logic (FOL), and balance expressivity against decidability to make reasoning feasible.

As can be seen in Moller and Haarslev's historical overview of Description Logics systems [**26**], modern DL are an evolution from early systems intended as alternatives to First Order Logic (FOL) with key DL ideas being introduced and refined by progressive DL implementations. [**26**] shows that key ideas, introduced in more detail below, such as

the inference of implicit knowledge from defined knowledge, Concepts, Roles, Subsumption, the TBox and ABox and the balance between expressiveness and decidability were added incrementally to DL by a number of DL systems over time. Sattler et al. [25] discuss in some detail other formalisms they consider related to or being contemporary to DL, specifically semantic networks, frame systems and conceptual graphs.

Basic Components of a DL System:

As presented by Baader and Nutt [27], a Knowledge Base implemented using a DL-based system consists of two logical repositories (the TBox and the ABox) which are defined using a Description Language and can be acted upon by Reasoners to make implicit knowledge explicit.

The TBox (Terminology Box) is a repository of definitions of complex Concepts and Roles based on basic atomic Concepts and Roles. The ABox (Assertion Box) is a repository of Assertions about individuals that describes the world. The Description Language is used to construct TBox and ABox statements. These statements can be seen as fragments of first-order predicate logic [27] and, as such, the Knowledge Base can contain implicit knowledge. Reasoners in a DL system use logical inferences to make the implicit knowledge in the ABox and TBox explicit.

The Open World Assumption:

A key feature of DL systems is that they work on the Open World Assumption (OWA). OWA systems assume that their knowledge of the world is incomplete (hence "open world"). If something is not known to be true then it is considered unknown as opposed to false. This is in contrast to the more familiar Closed World Assumption (CWA), as

commonly used by RDBMS and frame-based logic systems, which considers anything not explicitly known to be true to be false.

Reasoning:

Reasoners in DL systems use logical inferences – they infer information from what is explicitly stated in the TBox and ABox. There are a number of inference problems that can be considered standard to DL systems. These are covered in some detail in [**26**] and are categorized by [27] as covering Concepts, covering TBoxes alone, covering ABoxes alone, and covering TBoxes and ABoxes together.

2.3.3 The Semantic Web

The Semantic Web, as described by Berners-Lee et al. [28] is a machine-parsable overlay of the Web which supports navigation and reasoning by intelligent software (software agents) to navigate and reason. Ontologies are a key component of the Semantic Web and the Semantic Web has driven much research and development in this area. The Semantic Web is a project that is led by W3C [**29**] and has resulted in a stack of standards that provide support ranging from structuring data to Ontology definition. The main standards related to Ontologies, as of the time of writing [**30**], are described below.

RDF* [31] + *RDF Schema* [32]*: objects, classes, properties, hierarchies

The RDF (Resource Description Framework) Core Model supports the description of objects (resources) and their relationships and can be represented in XML. RDF Schema allows for classes and properties of RDF objects, and supports the definition of class hierarchies.

OWL **[**33**]*:*** Ontology definition

OWL (Web Ontology Language) supports the definition of ontologies by building on and adding to the basic support provided by RDF and RDF Schema for objects, classes, properties, and hierarchies. There are three increasingly expressive OWL sublanguages described in the OWL Web Ontology Language Guide [**34**] – OWL Lite, OWL DL and OWL Full. OWL Lite is the least expressive of the sublanguages, suitable for classification hierarchies. OWL DL, being based on Description Logics, balances expressivity with guaranteed computational completeness and decidability. OWL Full, the most expressive of the sublanguages, offers no computational guarantees. Complementary Semantic Web standards include XML [35] and XML Schema [36] for representation and SPARQL (SPARQL Protocol and RDF Query Language) [37].

2.4 Ontology Development

This section focuses on work related to Ontology development as relevant to the application example. This includes methodologies and techniques specific to building Ontologies (Ontology Engineering or Knowledge Engineering), design and structure of Ontologies (Normalization, Selection & Integration, and Segmentation or Partitioning), and documented best practices.

2.4.1 Ontology Building Methodologies

At the heart of Ontology-Driven Software Construction are Ontologies. This paper does not go into detail about the various methodologies and techniques that exist for constructing, validating and maintaining Ontologies, which may be termed Ontology Engineering, Ontology Construction, or considered part of Knowledge Engineering or

Agent-Oriented Engineering approaches. Lopez [38] gives an overview of Ontology construction methodologies that were extant at the time of writing (1999) along with a structured approach to analyse and compare these methodologies based on nine factors that describe the methodology in general and indicate the maturity and adoption of the methodology. Luck et al. [39] review Knowledge Engineering methodologies from the point of view of designing Agent-based Systems (approaches are classified as Knowledge Engineering, Agent-Oriented, and Extensions to Object-Oriented). Knublauch [40] reviews a number of methodologies, including Knowledge Engineering methodologies and Object Oriented methodologies and presents a very detailed methodology that tailors Agile methodologies to the building of Knowledge Systems.

Ontology construction can be categorized as either ”top-down” - moving from the most abstract Concept to the most concrete – or ”bottom-up”, where the process starts with concrete Concepts and the more abstract Concepts are arrived at via successive abstraction. The top-down approach could be suitable for new or existing systems that are not too complex, the bottom-up approach for reverse-engineering business knowledge from existing systems. There also exists the option of using a combination of the top-down and bottom-up approaches.

2.4.2 Ontology Normalization

Normalization of Ontologies results in cleaner hierarchies that are better-suited for modularization. Based on 15 years experience with the development of large Ontologies, Rector [**41**] describes a method for normalization of DL-based Ontologies, specifically those constructed with OWL-DL. Primitive concepts are organized in what is referred to as a “primitive skeleton”, with domain-independent abstract concepts making up the “top

level Ontology" whereas domain-specific concepts are referred to as existing in a "domain Ontology". The goal is to keep the primitive skeleton modularizable and to minimize implicit information opaque to reasoners. The Normalized form that results in the following form for the domain Ontology has the following attributes:

1. each branch forms a tree : no concept has more than one parent
2. each branch is logical and homogenous : inheritance is based on increasing specialization (subsumption as opposed to partonomy)
3. self-standing Concepts and partitioning Concepts are clearly distinguishable

- self-standing Concepts are disjoint and open
- the partitioning Concepts are organized as value types, with the children of each value type disjoint and closed (exhaustive sets)

4. classification reasoning does not result in multiple-inheritance

- axioms, range and domain constraints never imply any primitive domain concept is subsumed by more than one other primitive domain concept.

2.4.3 Ontology Discovery and Integration

Early in any software development project, a "build versus buy" decision is made. Working with Ontologies is no different and, given one of the benefits of Ontologies is that they explicitly express knowledge about a domain and the increasing number of standard Ontologies, it is even more important for Ontology-driven construction. Basically we have the following choices:

1. build an Ontology from scratch
2. re-use an existing Ontology

Existing Ontologies identified as candidates for use could be internal to the organization or external standard Ontologies. As any given Ontology is a model of a domain, which in turn is an abstraction of a specialized subset of the real world, all but the most non-trivial systems will involve multiple Ontologies.

Pinto & Martins [42] give guidelines for qualifying Ontologies that enumerate both strict and desirable requirements to aid in this process. The following options are identified for integration of an Ontology once it is selected:

1. use the selected Ontology as-is
2. refactor the selected Ontology (adapt, modify or augment)
3. specialize (define a new Ontology that extends the selected Ontology)

Barresi et al. outline a methodology for integrating Ontologies[5] [**41**] that has the following main steps:

1. identification of domains via scenarios
2. identification of candidate Ontologies for the identified domains
3. integration of the Ontologies

- As discussed in [44] this includes conversion of the Ontologies to a common format (OWL) preparatory to identification of similarities in the Ontologies, construction, testing and validation of the meta-model and model

Abels et al. [45] provide a recent (2005) overview of Ontology integration methods based on a literature review of Ontology integration methodologies, categorizing Ontology integration techniques into Ontology mapping, aligning and merging. Mapping identifies identical concepts in different Ontologies, aligning is related to this in that it makes two

[5] Barresi et al. actually use the term 'Semantic Resource' or SR, to encompass Ontologies and Ontology-like entities, such as taxonomies and dictionaries. As part of their integration process, all SRs are converted to a common OWL format.

Ontologies consistent and coherent and Ontology consists of merging multiple Ontologies. One issue with identifying candidate Ontologies for use is the actual discovery process – Concept names may not be the same, or the Ontology one is trying to source may map to a subset of an existing Ontology or to multiple existing Ontologies. Identification of candidate Ontologies using either mapping or alignment by a human is a resource-intensive task that is not scalable. Automatic mapping and alignment remains a research topic and tools that exist for these tasks can be considered informative.
In the context of the application example of eAdvisor, Wen & Lin [**46**] approach Ontology interoperation via extensions to Distributed Description Logic and E-Connections to support multi-Ontology querying and reasoning and describe a refactoring of the e-Advisor application platform to support this type of distributed Ontology reasoning performed using both local and remote Ontologies.

2.4.4 Ontology Segmentation

A topic related to both Normalization and Integration is how to deal with massive Ontologies that are too large to understand or incorporate in whole. Seidenberg & Rector discuss segmentation techniques to partition very large ontologies into manageable standalone Ontologies for performance and Ontology classification benefits [47]. Segmentation is also applicable in situations where the required Ontology is a subset of an existing Ontology.

2.4.5 Ontology Development Tools

There are a number of tools available to support Ontology development and reuse. As of the time of writing the majority of these tools were non-commercial and, of the tools

reviewed, a large number of them were not in fact still supported. Perhaps the most popular Ontology development tool is Protégé, which can be used to develop and prototype Ontologies using either frame-based specification or OWL-DL. This paper uses the term Protégé OWL in reference to Protégé being used with the OWL interface. Protégé OWL is a tool that can be used to develop and prototype Ontologies in OWL DL and is the tool that the eAdvisor Ontologies were developed using. The Protégé platform was developed by Stanford University and has, as of the time of writing, over 56,000 registered users. 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. Knublauch describes the architecture and features of Protégé OWL in a number of papers ([48][49][**50**][51]).

A review of the literature will quickly show number of other Ontology-development tools have been proposed and developed. Of these tools, Protégé enjoys the most success and the bulk of other tools, having been developed to facilitate specific research goals, are no longer maintained. The main issue with using a research-oriented tool such as Protégé OWL for Ontology-driven Software Construction is that it is not, and is not intended to be, commercial-grade. Protégé OWL is research-driven and facilitates research and prototyping admirably but to develop software outside of a research environment, the tools used need to be commercial-grade. This is not so much a matter of supported features as it is of stability and responsiveness to issues identified in the tools. At the time of writing, the support web page for Protégé [52] details the costs for consultation but there is no hard turnaround time for the resolution of issues users find in the Protégé platform (nor is such appropriate for a research platform such as Protégé). Also, a large amount of useful functionality for Protégé exists in the form of third-party plugins,

developed for prototyping and/or research. These plugins enjoy much less support than the Protégé platform itself, and are not updated as the Protégé platform itself evolves. As many of the benefits of Ontologies involve or promote reuse, tools that support Ontology maintenance and reuse are called for. Tool support is needed for storing, publishing, discovering, retrieval, and change management (including version management, and analysis of the impact of changes to interdependent Ontologies). Maedche et al. [53] state that the reuse issues seen with Ontologies, including the propagation of changes to dependent Ontologies are a variation of those seen with software systems in general and present how these challenges are addressed by KAON[6], an open-source Ontology management infrastructure. SemVersion [54] is an open source tool that supports versioning for Ontologies, including separate branches, merging, and structural and semantic diff operations. SemVersion has also been integrated with Protégé as described by Groza et al. in [55].

One notable recent development in commercial tool support for Ontology development the release of TopBraid Composer[7], due both to the published feature set and the presence of H. Knublauch, one of the developers of the OWL plugin for Protégé, as a Technical Director.

2.4.6 Ontology Best Practices

Knublauch [**56**] asserts that the Ontologies are the core of intelligent systems and thus need to be built with high quality. To ensure this, the following recommendations are made.

1. Pair Domain Experts and Systems Developers

[6] KAON has since been superseded by KAON2, available at http://kaon2.semanticweb.org/
[7] http://www.topbraidcomposer.com/

2. Incrementally develop Ontologies using OWL-DL
3. Validate Each Increment :

- As discussed in more detail below, OWL-DL allows for inconsistencies to be identified during design-time and Protégé OWL, the tool used for development of the eAdvisor Ontologies, has built-in reasoners to support this.
- Provided the tool supports, also create individuals and validate semantic restrictions (Protégé OWL supports this as well)

4. Prototype the system as developed for continual feedback

- Protégé OWL supports prototyping via customizable user interfaces for knowledge acquisition

Based on experience teaching courses, Rector et al. [57] describe commonly-seen errors and issues with people learning OWL-DL using Protégé OWL and relate some best practices and guidelines to follow during ontology development. A summary of these guidelines and best practices follows.

1. Paraphrase the description or definition before encoding in OWL.

- record this definition as a comment

2. Make all primitives disjoint

- force trees as described in the section of this paper on normalizing Ontologies

3. Use someValuesFrom as default qualifier in restrictions
4. Ensure defined classes are defined
5. Use closure restrictions where required

- recall Open World assumption

6. If the results of a classification are unexpected, check the domain and range constraints

7. Pay attention to logical and (intersectionOf) and logical or (unionOf)

- not as likely to be a problem for software development professionals

8. Watch for trivially satisfiable restrictions

9. Run the classifier early and often

10. Use only single inheritance

- as with 2, above, this is related to normalization of Ontologies as discussed in the section of the paper that discusses normalization of Ontologies

2.5 Ontology-Driven Software Construction

2.5.1 Overview

Ontology Driven Software Construction, also referred to as Ontology Driven Software Development, and Ontology Driven Architecture (ODA), makes the Ontology the driver of the software construction process. Ontology-Driven Software Construction is a convergence of traditional Software Engineering and Semantic Web practices and much of the promise implicit in Ontology-Driven approaches is the result of synergies realized from this convergence. A natural integration point with traditional Software Engineering methodologies is the OMG's Model Driven Architecture [**22**] (described in more detail in 2.2.7 **Model Driven Architecture**).

Ontology Driven Software Construction, in the context of this paper, refers to the actual process of constructing the software, and is distinct from yet could be used in conjunction with the use of Ontologies for systems specifications as described by Holten et al. [58]. Knublauch suggested the use of Agile development methodology for the development of Ontologies [40] and more recently proposed an Ontology-Driven Software Development

methodology that is built on the use of tools such as Protégé OWL [**56**] and discussed OMG's Model Driven Architecture, pointing out synergies between the two approaches and the implications of efforts that had just started at that time to bridge OWL and MOF/UML. Terrasse et al. [59] propose 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 most recent version (2006/02/11) of the W3C Semantic Web Best Practices & Deployment Working Group's working draft on "Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering" [**60**] further solidifies MDA as a vehicle to support Ontology Driven Architecture.

2.5.2 The Ontology Definition Metamodel

The OMG's final (recommended) submission for an Ontology Definition Metamodel [61] was submitted June 5, 2006. The development of this standard was triggered largely by the development of the W3C Semantic Web languages, including OWL. ODM is intended to support modeling of formal Ontologies in Description Logics or FOL and to allow an exchange of Ontology models in various formalisms. [61] explicitly states one of the potential benefits of ODM is that it can serve as a basis for marrying MDA and Semantic Web Technologies. The ODM provides for translation between various metamodels including UML and OWL via UML profiles and mappings. UML to OWL translation would be useful for "bottom-up" Ontology design, based on existing systems and OWL to UML has obvious benefits from the point of view of Ontology Driven Software development but there is a caveat in the proposal that the mappings are "strongly informative" as opposed to normative (a detailed justification for why

normative mappings were infeasible is given in an appendix). Despite this and other issues discussed in [61], ongoing work on standards and tools to translate between OWL and UML metamodels holds great promise for the viability of full Round Trip Engineering in Ontology Driven Software development.

2.5.3 Benefits of Ontology Driven Software Construction

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. The use of Description Logics allow for expressive modeling of the concrete domain in a manner that is not approached by modeling techniques currently widely used.

Driving the software construction from the Ontology itself ensures that the Ontology remains up to date and that business rules and knowledge are added to the Ontology 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 application example has been successfully been 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.

2.5.4 Challenges and Research Issues

As a summary review of the literature illustrates, Ontologies are central to a number of very interesting research areas. Of those related to Ontology-driven software construction, the following can be seen as relevant to the scope of this paper.

Enabling Ontology Reuse:

One of the key benefits of starting the software construction process with Ontologies is the potential to reuse existing Ontologies as opposed to developing a new Ontology 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
- how to control and coordinate changes to common and interdependent Ontologies

Tools and Methodologies:

Although there are a number of tools and utilities available for Ontology development, most notably Protégé, there is a noticeable lack of commercial-grade tools and proven methodologies. Although Ontology development is a very popular research topic, there is not much guidance on how to estimate the effort required to develop an Ontology. For Ontology-Driven Construction to fulfill its promise and become widely adopted it needs to both fulfill the needs of the development team and the user group in terms of functionality delivered while providing predictable, repeatable success in terms of time and effort.

Roles & Responsibilities:

In the event of a shift to an Ontology-driven Software Construction approach, the development team roles and responsibilities will be impacted. Two questions that need to be answered are:

1. who should develop new Ontologies
2. who ”controls” commonly-used ontologies

In an environment that uses Ontologies-driven Software Construction, the development of the Ontologies themselves needs more attention. As an artefact 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. This need for a combination of usually disjoint skill sets, combined with the reuse of published Ontologies, leads to the questions of who should be developing Ontologies and who should control published Ontologies once they are developed.

2.6 Proposed Solution Approach

Ontology-driven Software Construction is appropriate for the development of intelligent education systems such as the application example of eAdvisor. There is a great amount of literature on methodologies and techniques and best practices for Ontology development but these methodologies have not seen widespread acceptance. The intent of this paper is to take a first small step in synthesizing some of the large amount of work that has already been done with respect to Ontology-driven construction and propose a baseline methodology that incorporates known best practices and would be both usable by the Development Team and would also meet Sponsor needs and expectations. The application example of eAdvisor will be used to guide the development of and review of the baseline methodology and the proposed baseline methodology will be reviewed with

technical experts to identify and potentially address potential issues with the proposed methodology.

The research methodology, findings and conclusions are described in more detail in following chapters.

3 Chapter Three – Research Methodology

The purpose of this project is to identify a baseline methodology including best practices and guidelines for Ontology Driven Software Development of intelligent education systems, using the application example of Athabasca University's eAdvisor system.

3.1 Research Method

The main research methods used were a review of literature of related research (including technical and working papers of the DELTA lab related to eAdvisor), review of and experimentation with a subset of the publicly-available tools to support Ontology development, and qualitative analysis of Ontology-driven Software Construction with a group of technical experts.

Qualitative analysis was selected over quantitative analysis, despite the benefits of the latter, as there is a dearth of available quantitative data on Ontology Driven Software Construction and a study that would provide the volume of data required to perform quantitative analysis was far beyond the scope of this paper. The qualitative analysis undertaken supported development of a reasonable baseline methodology that can serve as the basis of future quantitative analysis. Goal-focused interviews served to maximize the benefit of the technical interviewees' years of experience in knowledge engineering, data architecture, and software development.

3.2 System Environment and Data-Gathering Tools

Tools used include Protégé OWL, Microsoft® Word, Microsoft® Visio and Microsoft® PowerPoint. Protégé OWL was used for reviewing and demonstrating the eAdvisor

ontologies as well as explaining ontology construction during the technical interviews. Microsoft Word was used for making and transcribing notes as well as editing and marking up ("memoing"). Visio and PowerPoint were used for graphing and diagramming. Change tracking was put in place to understand when and why changes are made to the baseline methodology. Refer to 9 Appendix : Data Management for details about what Items and Changes are tracked and in what manner.

3.3 Study Conduct

Based on the literature review research on Methodologies, Ontology-Driven Construction, and documented experience of the DELTA Group developing eAdvisor, best practices and guidelines were incorporated into a proposed baseline methodology. This methodology was reviewed via focused interviews with technical experts considered representative of the Development Team stakeholder. Qualitative Analysis of the technical interviews was performed and findings and conclusions documented. In addition to the technical interviews, a focus group session was held with a number of domain experts in the areas of eLearning, profiling and advising. The findings from this session were very interesting and valuable but outside the scope of this project and so are not included in this paper.

The study can be considered to having taken place in three major phases:

1. Review
2. Synthesis
3. Interviews & Analysis

These phases are described in more detail in the following sections and the following diagram illustrates, at a high level, the research methodology used in this project.

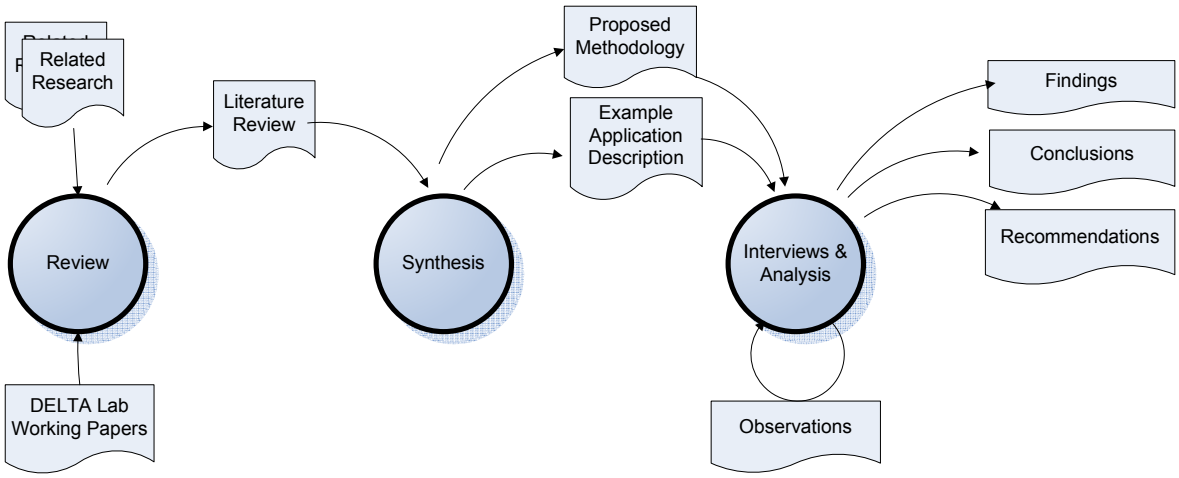


Figure 4 Study Conduct

3.3.1 Review

A review of relevant literature took place, focused by the research goal and application example to include topics related to Ontology-Driven Software Construction and the example application of Athabasca University's eAdvisor. Research related to agent technologies, intelligent systems, agent-based systems, Ontologies, Logic, and intelligent advising systems as well as working papers of the DELTA Lab were reviewed for background on eAdvisor. Literature related to software development methodologies, Ontology development, knowledge engineering, and Ontology-driven software development were reviewed for background on Ontology-driven software construction. Where applicable, publicly-available tools for Ontology development, particularly those used for development of the application example, were installed and used.

Relevant areas of the literature review are summarized in 2 Chapter Two – Literature Review with additional background information in 10 Appendix : Further Background on Agents and Multi-Agent Systems.

3.3.2 Synthesis

Building on the information gathered during the Literature Review, a proposed baseline methodology was developed, including best practices and guidelines and a non-technical presentation of the application example of Athabasca University's eAdvisor system was developed.

Guiding principles for the proposed baseline methodology were that it:

- fits within the Software Construction phases of a typical project life cycle
- fills 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:

- easy to use (for the Development Team)
- build on existing skill sets, be evolutionary, not revolutionary

Objectives specifically for the Sponsor stakeholder included:

- not be obtrusive / visible to Sponsor : seen as a variation of existing ways of doing things
- plug into existing processes and procedures, such as those for project governance

3.3.3 Interviews & Qualitative Analysis Phase

The proposed baseline implementation was reviewed with technical experts with relevant expertise in Knowledge Engineering, Ontology Development and Software Development. These technical interviews were transcribed, annotated, marked up ("memoed") and analysed, prior to inclusion in the Findings and Conclusions. Qualitative Analysis sourcebooks ([62],[63],[64]) were used to plan this part of the study and the approach was reviewed with individuals experienced in qualitative analysis.

The following diagram illustrates the process followed during the Interviews & Qualitative Analysis Phase, organized into Focus / Design / Collect / Interpret activities. Details of the Qualitative Analysis design are provided below.

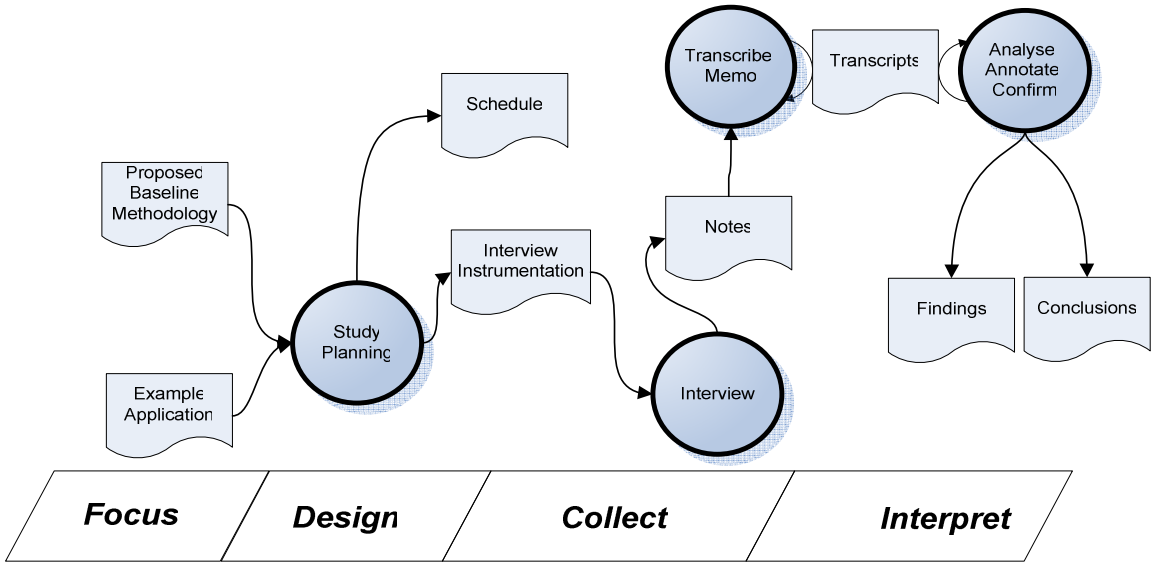


Figure 5 Reviews & Qualitative Analysis Process

Approach

The approach for the Technical stream is Goal Oriented as per the Focus and Research Questions.

Interviews with technical experts with relevant experience were held. These individuals represented the Software Developer stakeholder viewpoint. Details of these interviewees' qualifications can be found in 8 Appendix : Technical Interviewees, Focus Group .

Focus

The focus of the technical interviews was on the proposed baseline methodology.

Research Questions

The research questions were bounded by the focus of the project, as discussed above. The interviewees were asked to identify any issues or gaps they saw with the methodology as presented during the course of the interview.

Instrumentation

Instrumentation developed to support the qualitative analysis included a presentation of the application example, ontology oriented software construction, and the proposed baseline methodology. The Protégé OWL tool was used during the presentation. Notepads and paper were used to further illustrate concepts of key interest to the individual interviewees as required and appropriate.

Procedure

Interview candidates were selected and qualified based on their expertise in areas related to Ontology-Driven Software Construction and their maturity in the field of software systems development. Times for individual interviews were scheduled and arrangements for travel to the interviewees' locations were made.

The following agenda guided the individual interviews:

- Objective of the Interview
- Overview of application example

- Overview of Technical Background and Tools
- Overview of Proposed Baseline Process
- Questions
- Closing

Handwritten notes were taken during the course of the interviews and transcribed electronically preparatory to marking up (memoing) and analysis.

The average interview took just under two hours to complete.

4 Chapter Four – Research Study Results

4.1 Study Findings

This section relates findings from the literature review of areas related to the application example and Ontology-driven software construction as well as analysis of the interviews where Ontology-Driven Software construction was reviewed with technical experts.

4.1.1 Review Findings

The following findings are based on the literature review as documented in 2 Chapter Two – Literature Review as well as review of DELTA Lab's working papers related to eAdvisor. These findings are grouped according to the two research issues of Enabling Ontology Reuse and Tools and Methodologies as listed in 2.5.4 Challenges and Research Issues.

Enabling Ontology Reuse:

There are methodologies on integration of Ontologies that focus on the discovery of Ontologies by people as well as continuing research on automatic Ontology discovery, using techniques such as mapping and alignment. The eAdvisor project has approached Ontology integration both by refactoring domain Ontologies for use as eAdvisor-specific Ontologies and also with recent development work to support distributed Ontology reasoning.

The need for standards and tools to support Ontology reuse is widely seen in the literature and the active research related to these areas, such as storing, publishing, discovering, and

retrieving existing Ontologies, as well as the related area of controlling and propagating structural and semantic changes across distributed and interrelated Ontologies.

Tools and Methodologies

There are a number of tools and utilities available for Ontology development, most notably Protégé, but these are geared more toward research as opposed to commercial software development. The Semantic Web initiative has driven standards for Ontologies, namely the OWL family of languages, and tool developers are aligning with OWL support. The OMG is actively working on a mapping between Ontologies and standard software development models, as can be seen with the development of OMG ODL and this should open up the use of existing commercial tools for Ontology-driven Software Construction.

Ontology development involves iteratively finding more information about a domain, documenting it and testing the Ontology. Techniques and best practices for Ontology development, as documented in the Literature Review, are summarized in the following table.

Ontology Development Guideline	**Section**
Choosing Ontology Development Approach (top-down or bottom up) • top-down if new system or existing system is not complex • bottom-up for reverse-engineering. • Combination	2.4.1
Ontology Structure	

• Normalize Ontologies as per Rector's guidelines [**41**] • Segment overly large, unmanageable Ontologies [47]	2.4.2 2.4.4
Required domain Ontologies can be identified via scenarios, and these can drive the identification of candidate Ontologies for use (as-is, refactored, specialized, …).	2.4.3
Ontology development is suited to incremental and iterative development, including : • Pairing of domain and systems expertise • Development and validation of the Ontology and the system in small increments	2.4.6

One clear gap in Ontology development methodologies is proven Ontology effort estimation. This is shown by the lack of published research on successful estimation of Ontology development.

4.1.2 Qualitative Analysis Findings

The following findings are based on analysis of reviews of Ontology-driven Software Construction methodology with technical experts.

1. The benefits of Ontology-Driven Software Construction are clearly understood and accepted by technical experts

The benefits of Ontology-Driven Software Construction as articulated in this paper were quickly understood by the technical interviewees. The benefit of separate models for Ontology and the solution (the latter being typified by UML for a specific application solution) were also readily understood and accepted. The potential interoperability

between these two types of models offered by the OMG's Object Definition Metamodel was of particular interest to this group, despite the "informative not normative" caveat in the current version of the ODM.

2. Ontology-driven Software Construction is seen as appropriate and natural for the development of Intelligent Education Systems such as eAdvisor.

One comment expressed by the technical group was that, despite tool support for model-driven development of applications being present for many years, the comprehensive transition to true model-driven architecture has not yet been made in any large shops they had worked in. As a result, the implementations drift from the models. In the case of Intelligent Education Systems such as eAdvisor, where the Ontology is an element of the executable system and must be updated as part of the development process, Ontology-driven Software Construction is seen as being more likely to be adopted.

3. Governance Issues are important and need to be addressed or allowed for.

Although the presentation of the proposed baseline methodology reviewed governance and support for the Sponsor, the technical group actually made a number of comments and raised a number of concerns directly related to Governance. Experienced development professionals know that they often start a project with mandated constraints and that projects that are not meeting the goals of non-technical stakeholders risk being cancelled.

4. The Analyst and Developer roles can be subsumed by Development Team Member in the methodology processes.

The original presentation of the proposed baseline methodology differentiated between Analyst and Developer. Given the number of team members who fill both roles to

varying degrees, the technical interviewees commented that the two roles could be considered one for the purpose of the baseline methodology.

5. Guidelines for where to position rules and logic are seen as important.

There was considerable conversation with all technical interviewees about where logic is positioned, between the Ontology/Knowledge Base and Agents that execute against it.

6. Legacy Systems integration is a major concern.

All technical interviewees spoke of interacting with and integrating with Legacy Systems as being an issue to consider. Specific examples of where legacy systems required rework were discussed as well.

7. Several Theoretical Issues required thorough discussion.

More time was required with the technical interviewees than originally allotted to review the theoretical underpinnings of Ontology development, Distributed Logic, and Intelligent Systems including Agent technology and Multi-Agent Systems. Benefit was seen by early explanation of explicit disjoint, the Open World assumption and the ability of Individuals to change Classes.

4.2 Study Conclusions

Ontology development requires both Analysis and Development skill sets

The interviews indicated that it is not possible to cut a clear line between Analyst and Developer for Ontology development. Development of an Ontology Architect role with both Analyst and Developer skill sets is indicated. This role could be filled by an individual or a team, depending on the organization, the team, and the processes used.

Ontology-driven Construction of intelligent systems has a good chance of acceptance by the Development Team.

Based on findings from the technical interviews, the Ontology-driven software construction approach should be accepted by developers. This is due to the integral part Ontologies play in these systems, both as a runtime element, and as the starting point for software construction. As with any new technology, training will be required as Ontology-driven Software construction is adopted.

Ontology-driven software construction needs to fill the needs of the Sponsor.

Ontology-driven Software Construction needs to fit within existing project governance frameworks. This conclusion was supported by the interviews with the technical experts, where specific mention was made of the importance of adhering to pre-existing budgets, standard tools, strategic direction and enterprise models.

The adoption of Ontology-driven Software Construction would be facilitated by fitting within existing project governance frameworks. For this to occur, the time and effort required to develop Ontologies and thus Ontology-driven systems, needs to be predictable within stated ranges of uncertainty.

The project team needs to be able to estimate how much time and effort will be required, to refine these estimates and to report status against them. Accurate estimation of Ontology development effort remains a gap, and although estimation techniques specific to Ontology development may be developed, quantitative data from Ontology-driven Software Construction will need to be gathered to validate these methods.

Tools are currently immature but standards and convergence with MDA will ameliorate this.

Ontology-driven software construction is heavily reliant on tools. Tools are required to build the Ontologies and to translate from OWL to UML. Tool support is vital and immature.

Standards exist but tools need to adopt them and most notably the OMG ODL is not considered normative. Gaps in the support provided by ODM may be filled in different ways by different tools, potentially leading to divergence until ODL becomes normative and tools align to the updated standard.

Even when use of modelling tools is mandated, the tools are often not used after the initial construction is completed and so the implementations drift from the models. For Ontology-driven systems, the tool needs to be used to develop the Ontology itself so this is not a concern – the tool is already an integral and required part of the process.

Integration and interoperation of Ontologies is still progressing.

Research and tools supporting the integration and interoperation of Ontologies is still maturing. There is research related to the publishing and discovery of Ontologies and the integration and interoperation of Ontologies. Until such time as automatic methods are perfected or standards are crystallized humans will need to deal with such issues as ”same concept, different name” when searching for Ontologies.

Ontology-driven Software Construction is an iterative process that can leverage MDA.

The integration of Ontology development with MDA as an Ontology Driven Construction methodology was indicated by the Literature Review & Synthesis work and upheld by the Qualitative Analysis phase with the Technical experts.

Ontology-driven software construction is well-suited to an iterative approach and could fit into Rational Unified Process or Agile frameworks. Also, synergies between Ontology-driven software construction and Model Driven Architecture can be realized.

As the ODM is implemented in commercial tools, this will be greatly facilitated. Extensions to ODM, either in the standard or as implemented by tools, could be developed to make the transformations from Ontology to solution model and back more normative, which would enable reverse engineering and round-trip engineering.

Ontology Development Guidelines can be incorporated in Ontology-driven construction.

In terms of a process to follow for Ontology-driven construction, and considering the above conclusions regarding marrying Ontology development with MDA, existing best practices for Ontology development (as described in the Literature Review and summarized in the Findings, above) can be incorporated along with MDA guidelines for a workable Ontology-driven Software Construction process.

Legacy Systems Integration

As related in the Findings, a major concern of the technical interviewees was integration with legacy systems. The use of wrappers to mask access of legacy systems, analogous to eAdvisor's use of web services, is one approach, as is the development of front-end Ontologies.

4.3 Baseline Methodology (Recommendations)

The following sections describe a Baseline Methodology for Ontology-Driven Software Construction based on methodologies, techniques, best practices, and guidelines reviewed during the literature review and refined and adjusted based on the findings and conclusions documented above. This baseline methodology focuses on activities performed by the Development Team that are directly related to Ontology-Driven Software Construction and ensuring that the process fits in a framework that would fill the Sponsor's needs. The focus of this paper is on Ontology-Driven Software Construction in the context of Intelligent Educational Systems and, although what is proposed in this paper should work within the framework of other processes such as Requirements Management and Quality Control, these processes are not discussed in detail.

4.3.1 Overall Process Flow

The overall process flow for Ontology Driven Software construction is illustrated by the following business process model. The activities for this flow are described in more detail below.

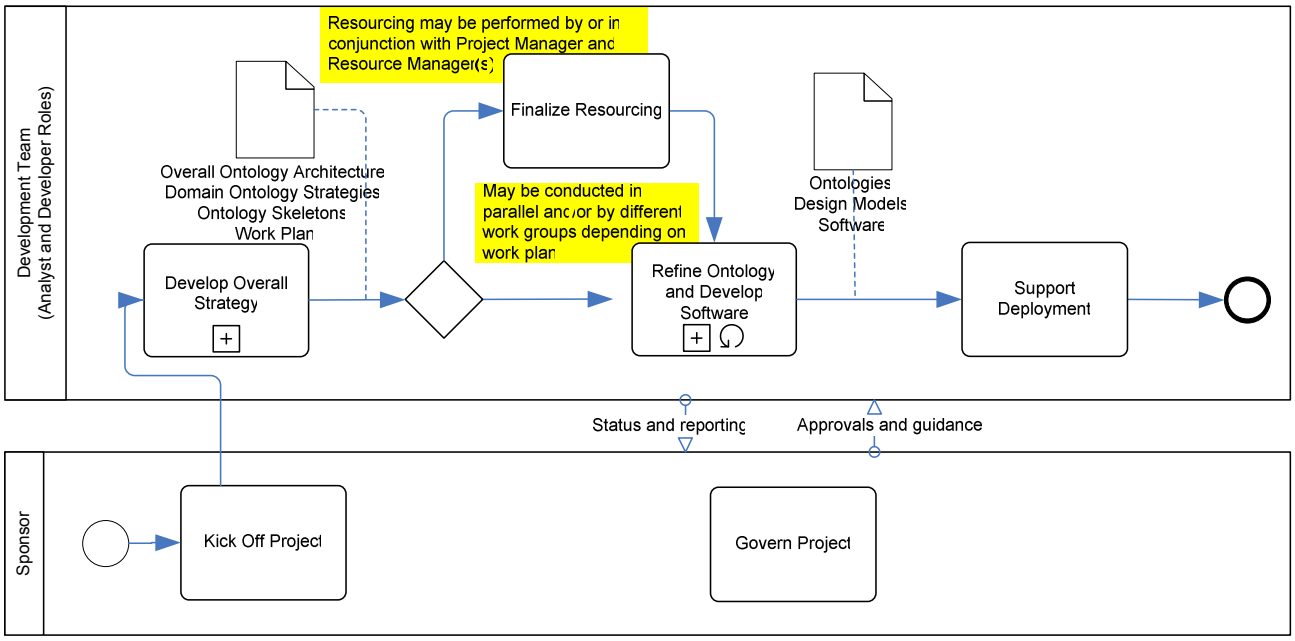


Figure 6 Baseline Methodology Overall Process Flow

1. Sponsor : Need Identified Event

A need is identified and a project is approved to address the need. Details on how the need is identified and the project approved are beyond the scope of this project and differ by organization. Generally sufficient funding is allocated for estimates to be arrived at or funding limits will be set.

2. Sponsor : Kick Off Project activity

This baseline methodology holds the Sponsor accountable for initiating the project. Details on this activity are beyond the scope of this project and will differ by organization. This initiation may include 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. Development Team : Develop Overall Strategy

Following project initiation the Development Team is accountable for developing a high-level architecture and strategy for the Ontology-Driven Software Construction work to be done. This activity is described in more detail below.

Output: Overall Ontology Architecture, Domain Ontology Strategies, Domain Ontology Skeletons, Work Plan.

4. Development Team : Finalize Resourcing

Depending on the organization and how the teams are put together, resourcing may need to be finalized. This is especially true in organizations where people are working on multiple projects.

Based on this activity, if it occurs, the timelines or resources assigned to the Work Plan may change. The activities that take place during resourcing vary depending on the organization's structure and dynamics.

Output : updated Work Plan

5. Development Team : Refine Ontology and Develop Software

The execution of the Ontology-driven Software Construction occurs primarily during the Refine Ontology and Develop Software activity. Based on the input from the Strategy, and working against the Work Plan, Ontology-driven Software Construction occurs. This activity is described in more detail below.

Output : Ontologies, Design Models, Software

6. Development Team : Support Deployment

Once the construction is complete, the system is deployed. This is often performed by a group separate from the Development Team and, in any event, is outside the scope of this paper. Regardless, the Development Team will need to support deployment via knowledge transfer and assistance resolving unforeseen issues

7. Sponsor : Govern Project

Project Governance is how projects are brought into alignment with strategic goals. For example, based on the progress of the project and 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 paper - each organization will have a different model for governance, with different funding and go/no-go gates. The purpose of this methodology process flow is to provide a structure that would support the overlay of governance, to allow for the Ontology Driven Software Construction processes to fit within typical models of project governance. There are many books such as [45] which deal with Governance and how business strategy is supported by projects via governance.

4.3.2 Development Team : Develop Overall Strategy

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

A Conceptual Architecture is developed, which encompasses the domain Ontologies, as well as an integration strategy as applicable. Following this, plans for the development / acquisition of each domain Ontology identified in the Conceptual Architecture are

developed. Finally, using the domain-specific strategy work and the overall architecture, a work plan is developed. This work plan drives the Ontology Driven Software Construction (see 4.3.3 Development Team: Refine Ontology and Develop Software) and serves as a very important hook into an organization's governance framework.

Each subprocess in the Develop Overall Strategy process flow is described in more detail below.

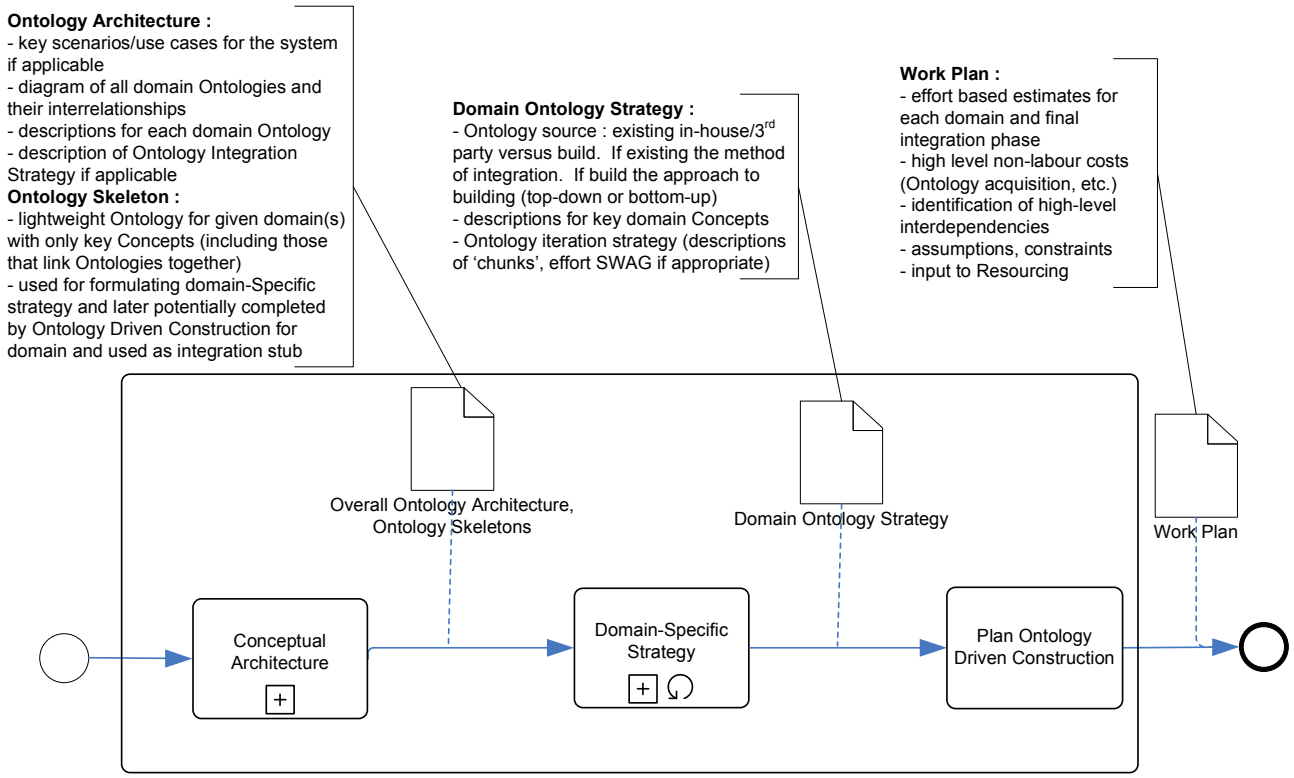


Figure 7 Overall Strategy Process

1. Conceptual Architecture

During the Conceptual Architecture task, key scenarios for the system to be developed are documented and the main domains are identified (see Barresi et al. [44]). For each domain the key concepts to support the scenarios are identified to

form a ”skeleton Ontology” for that domain. The Conceptual Architecture includes a diagram showing all the domain Ontologies with their key Concepts and, at a high level, how they are integrated (the domain Ontology skeletons and the Ontology integration strategy). For the eAdvisor domain the strategy proposed by Wen & Lin [**46**] is to be followed or at the least, not precluded. Note that strategic direction or policy may influence the output of this activity. This activity is described below in more detail (see 4.3.2.1 Develop Overall Strategy: Conceptual Architecture).

Output: Ontology Architecture, Ontology Skeleton (one for each domain, or identification of known or mandated Ontology to be integrated with).

2. Domain-Specific Strategy

During the Domain Specific Strategy activity, the construction activities related to a specific domain are examined and decisions that affect the construction and costing are made. Note that strategic direction or policy may influence this strategy.
This activity is described in more detail below (see 4.3.2.2 Develop Overall Strategy: Domain-Specific Strategy).

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

3. Plan Ontology Driven Construction

During the Plan Ontology Driven Construction a work plan is developed to drive the actual construction.

Output: Work Plan

4.3.2.1 Develop Overall Strategy: Conceptual Architecture

The Conceptual Architecture process is where the conceptual architecture is developed. This architecture illustrates the main domains and key scenarios for the system to be developed. In addition, for each domain where the Ontology has not been pre-decided, an Ontology skeleton is developed to assist in sourcing the Ontology and aid construction activities.

The tasks that make up this process are described in more detail below.

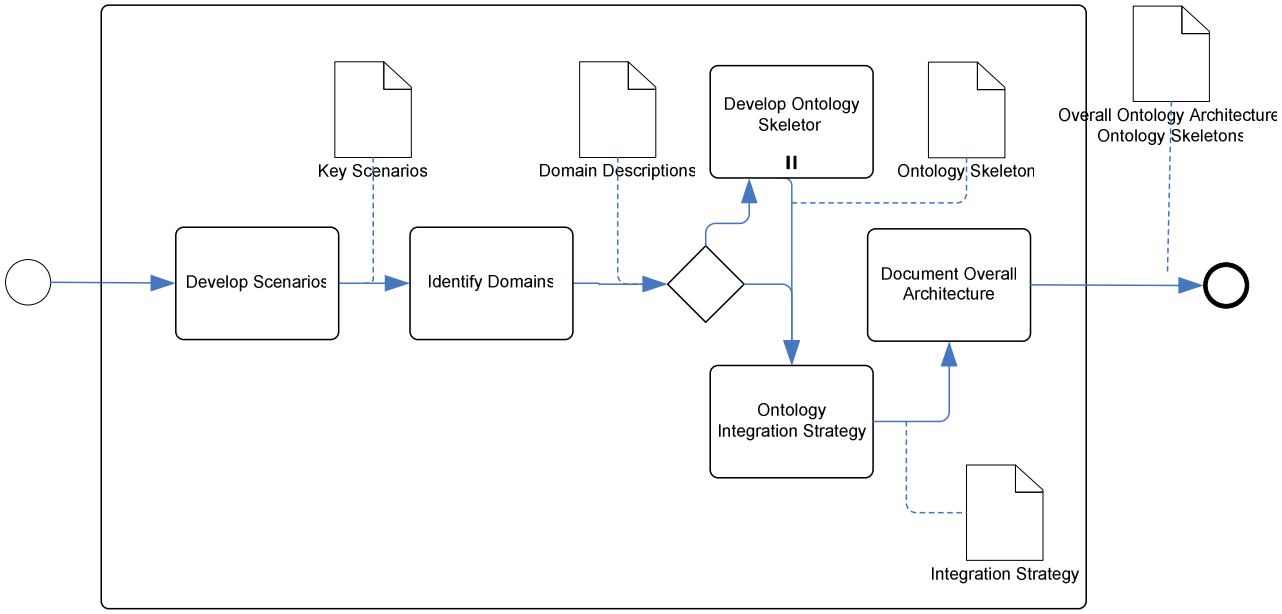


Figure 8 Overall Strategy - Conceptual Architecture Process

1. Develop Scenarios

The Develop Scenarios and Identify Domains activities, taken together, are analogous to "Domain Scoping /Scenario Characterization" as described in [44]. The technical interviews confirmed that this type of activity is familiar to software developers and data architects.

Output: Key Scenarios to be handled by the system

2. Identify Domains

This task follows on the Develop Scenarios activity and involves identifying domains that would be required to support the Key Scenarios identified by Develop Scenarios.

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

3. Develop Ontology Skeleton (optional)

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

Output: Ontology Skeleton

4. Ontology Integration Strategy

The Ontology Integration Strategy illustrates how the domain Ontologies are interrelated and relates the strategy for integrating the Ontologies.

Output: Integration Strategy

5. Document Overall Architecture

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

Output: Overall Ontology Architecture

4.3.2.2 Develop Overall Strategy: Domain-Specific Strategy

The Domain Specific Strategy process is executed for each domain that has been identified in the overall Ontology Architecture. During this process, the domain's Ontology source is identified (re-use or refactor an existing Ontology, build from scratch), the approach for integrating with the existing Ontology or building the new Ontology is decided upon and, based on that, cost/effort estimates are developed. Tasks in this subprocess are described in more detail below.

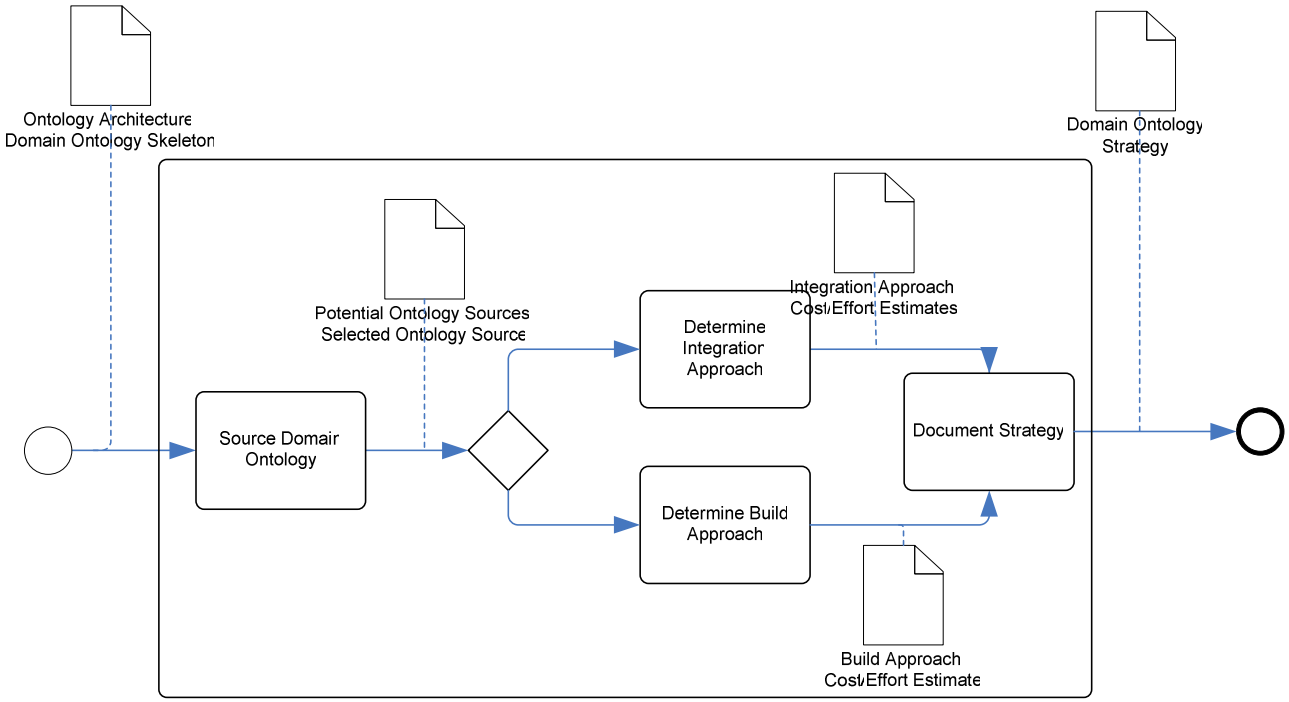


Figure 9 Overall Strategy - Domain Specific Strategy Process

1. Source Ontology

In some situations it is possible that the Ontology to be used for a given domain has been pre-decided but in all other cases this task is necessary. The Domain Ontology Skeleton can be used as a template for identifying potential Ontology sources. This is analogous to the ”Candidate SRs Identification” in [44]. Sources of candidate Ontologies include sources internal to the organization and external to the organization. In the application example only heterogeneous implementation languages are considered: OWL-DL Ontologies developed using Protégé OWL.

Output: list of potential Ontology sources for the domain, the selected Ontology Source

2. Determine Integration Approach

If an existing Ontology has been identified for the domain, how that existing 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. As mentioned before, two main approaches for Ontology development include top-down and bottom-up. An approach suggested by the technical interviews is to use top-down to obtain a clear structure and then incrementally add reverse-engineered Concepts to this structure, classifying and normalizing as this is done.

Output: Build Approach, Cost/Effort Estimates

4. Document Domain Ontology Strategy

Based on the results of preceding tasks, document the strategy for developing this domain Ontology. In effect the Domain Ontology Strategy will consist of the following:

- Ontology Source : potential sources, selected source and justification
- Approach : Integration Approach OR Build Approach
- Cost / Effort Estimate

Output: Domain Ontology Strategy

4.3.3 Development Team: Refine Ontology and Develop Software

The Refine Ontology and Develop Software 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 domains and their interdependencies. The baseline methodology suggests an integration of incremental Ontology development and downstream Model Driven Architecture for development of the systems themselves. Mature tools for integrating OWL and UML will facilitate this and OMG's ODM [61] holds great promise for facilitating this. Once Ontologies are developed to a satisfactory state, applications can be developed to run against them. The use of incremental development and applying Agile techniques to Ontology development is suggested by Knublauch [**56**]. These techniques are open to validation and refinement as the methodology is used on successive projects.

Ontology Development:

1. develop the Ontology in increments, validating each increment with Protégé's built-in reasoners. Create individuals to validate semantic restrictions (as per Knublauch [**56**]).
2. prototype as appropriate
3. maintain tree structure based on increasing specialization (untangle as required), clearly differentiate partitioning Concepts (used to untangle) from self-standing Concepts (as per Rector [**41**])

Application Development:

1. develop application models (using tools or translation tools to base on OWL as closely as possible)
2. develop applications

5 Chapter Five – Future Research

Areas for future research related to Ontology-driven software construction are numerous and include the following, already active research areas.

- Ontology integration
- Ontology change management
- Reverse-engineering of Ontologies
- Tool development
- Ontology development estimation

6 Chapter Six – Conclusions

Despite a lack of maturity in tools and processes, Ontology-driven Software Construction of Intelligent Systems holds great promise. Standard methodologies and commercial-grade tools and techniques to support Ontology-driven software can be expected to emerge as a result of the ongoing convergence between existing and emerging standards from W3C and OMG. Research into automatic Ontology discovery and integration needs to progress as well.

To ensure the chances of successful adoption by both the Development Team and the Sponsor, what is needed is a methodology that is evolutionary rather than revolutionary. Proven methods, procedures and practices need to be built upon in an incremental manner, allowing for easier acceptance. Having the proposed construction methodology fit within typical existing product and project management frameworks will ensure as little change as possible in the Sponsor's interfaces and lessen resistance from that Stakeholder group.

The methodology proposed in this thesis essay for Ontology-driven Software Construction has been driven by the application example of eAdvisor and is a suitable vehicle for validating best practices and metrics about Ontology-Driven Software Construction. It also supports the overlay of a governance process by allowing for standard checkpoints and gates.

To allow for process improvement and validation, as well as assist in estimation, project-end reviews should review metrics, produced deliverables, and update the process, guidelines and best practices as appropriate. Data to be tracked includes actual effort

expended, along with information about the environment and people doing the work. Ontologies developed as a result of these projects should be stored to allow for structural metrics to be pulled, such as the number and complexity of Concepts, relations, and rules. These metrics, along with actual values for time and effort expended, can assist in understanding and improving estimation techniques.

7 References

1 Carroll, J. & Chappell, G. (1996). *An Intelligent Universal Advisor* in Proceedings of the 1996 ACM symposium on Applied Computing, pp. 105 – 109

2 Lin, F., P. Holt, S. Leung, and Q. Li. (2006). *A multiagent and service-oriented architecture for developing adaptive e-learning systems*. In *Int. J. Cont. Engineering Education and Lifelong Learning, Vol. 16, Nos. 1/2,* pp.70-91.

3 Lin, F., Li, Q. & Wen, D. (2005). *Knowledge Modelling for Developing Program Planning Agents*. In *Proceedings of ICEB, Hong Kong*, Dec. 6-9, 2005, pp.366-371

4 Lin, F., Wen, D., Leung, S., Li, Q., and Zhang, F. (2006). *A MAS for Academic Advising and Program Planning*. (Draft paper)

5 Thorpe, M. (2001). *Rethinking Learner Support: the challenge of collaborative online learning*. Presented at SCROLLA Symposium on Informing Practice in Networked Learning, Glasgow, 14 November 2001.

6 Binkerd, C. & Fernandez, J. D. (2004). New approaches to advising and mentoring in science and technology. In Journal of Computing Sciences in Colleges, Vol 19, Issue 4 (April 2004), pp. 218 – 224, Consortium for Computing Sciences in Colleges (2004)

7 Gunadhi, H., Lim, K. & Yeong, W. (1995). *PACE: A Planning Advisor on Curriculum and Enrollment*. In Proceedings of the 28th Annual Hawaii International Conference on System Sciences, 1995, pp. 23 – 31

8 Van Biljon, J., De Kock, E., & Renaud, K. (2002). *Zazu — Investigating the*

Differences Between Experts and Novices in Using an Advisory Support Tool. In Proceedings of SAICSIT 2002, pp. 30 – 43

9 Lin, F., Holt, P., Leung, S. & Li, Q. (2006). *A multiagent and service-oriented architecture for developing adaptive e-learning systems*. In *Int. J. Continuing Engineering Education and Lifelong Learning*, Vol. 16, Nos. 1/2, pp.77–91

10 JADE : http://jade.tilab.com/

11 Gorgone, J. (2000). *A new IS graduate curriculum model — after eighteen years.* In ACM SIGCSE Bulletin, Volume 32 Issue 2 (June 2000)

12 ACM CCS : http://www.acm.org/class/1998/

13 IEEE LOM Working Group : http://ltsc.ieee.org/wg12/

14 IMS LIP 1.0.1 : http://www.imsglobal.org/profiles/index.html

15 PNML : http://www2.informatik.hu-berlin.de/top/pnml

16 A guide to the project management body of knowledge : PMBoK© Guide, 3rd Ed.

17 Royce, W. (1970), *Managing the Development of Large Software Systems*. In Proceedings of IEEE WESCON, vol. 26, no. August, p. 1-9.

18 Larman, C. & Basili, V. (2003). *Iterative and Incremental Development:A Brief History*. IEEE Computer, June 2003

19 Boehm, B. (1988). *A Spiral Model of Software Development and Enhancement,* Computer, vol. 21, no. 5, May 1988, pp. 61--72.

20 *Rational Unified Process, Best Practices for Software Development Teams*. Accessed 2006/10/02 from http://www-128.ibm.com/developerworks/rational/library/253.html

21 *Principles behind the Agile Manifesto*. Accessed 2006/10/02 from

http://www.agilemanifesto.org/principles.html

22 OMG MDA : http://www.omg.org/mda

23 *MDA Guide Version 1.0.1.* Available at http://www.omg.org/docs/omg/03-06-01.pdf

24 OCL : http://www.omg.org/technology/documents/formal/ocl.htm

25 Sattler, U., Calvanese, D. & Molitor, R. (2002). *Relationships with other Formalisms*. In Baader, F,, Calvanese, D., McGuinness, D., Nardi, D. & Patel-Schneider, P.F (Eds), *The Description Logic Handbook.* (pp. 142 – 183 chapter 4). Cambridge University Press

26 Moller, R., & Haarslev, V. (2002). *Description Logics Systems*. In Baader, F,, Calvanese, D., McGuinness, D., Nardi, D. & Patel-Schneider, P.F (Eds), *The Description Logic Handbook.* (pp. 289 – 312 chapter 8). Cambridge University Press

27 Baader, F. & Nutt, W. (2002). *Basic Description Logics.* In Baader, F,, Calvanese, D., McGuinness, D., Nardi, D. & Patel-Schneider, P.F (Eds), *The Description Logic Handbook.*. (pp. 47 – 100 chapter 2). Cambridge University Press

28 Berners-Lee, T., Hendler, J. & and Lassila, O. *The Semantic Web*. In Scientific American, May 2001

29 W3C Semantic Web : http://www.w3.org/2001/sw/

30 W3C Semantic Web Activity Statement : http://www.w3.org/2001/sw/Activity

31 RDF : http://www.w3.org/RDF/

32 RDF Schema : http://www.w3.org/TR/rdf-schema/

33 OWL : http://www.w3.org/2004/OWL/

34 *OWL Web Ontology Language Guide*. Accessed 2006/10/03 from http://www.w3.org/TR/owl-guide/

35 XML : http://www.w3.org/XML/

36 XML Schema : http://www.w3.org/XML/Schema

37 SPARQL : http://www.w3.org/TR/rdf-sparql-protocol/

38 Lopez, F. (1999). *Overview of Methodologies for Building Ontologies.* In Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends. CEUR Publications, 1999.

39 Luck, M., Ashri R., & d'Inverno, M. (2004), *Agent-Based Software Development*. Artech House

40 Knublauch, H. (2002). *An Agile Development Methodology for Knowledge-Based Systems Including a Java Framework for Knowledge Modeling and Appropriate Tool Support* (doctoral dissertation). University of Ulm

41 Rector, A. (2003) *Modularisation of domain ontologies implemented in description logics and related formalisms including OWL.* In the Proceedings of the 2nd international conference on Knowledge capture, pp 121 – 128, ACM (2003)

42 Pinto, H. & Martins, J. (2003). *A methodology for ontology integration*. In the Proceedings of the 1st international conference on Knowledge capture, pp 131 – 138, ACM 2001

43 Morris, P. and Jamieson, A. *Translating Corporate Strategy into Project Strategy: Realizing Corporate Strategy Through Project Management.* Project

Management Institute (2004)

44 Barresi, S., Rezgui, Y., Lima, C., Meziane, F., (2005). *Architecture to support semantic resources interoperability*. In the Proceedings of the first international workshop on Interoperability of heterogeneous information systems, pp 79 - 82, ACM 2005

45 Abels, S., Haak, L. & Hahn, A. (2005). *Identification of common methods used for ontology integration tasks*. In the Proceedings of the first international workshop on Interoperability of heterogeneous information systems IHIS '05 Publisher: ACM Press

46 Wen, D. and Lin, F. (2006). *Querying and Reasoning for Multiple Ontologies through Extending Description Logic*. (Working Paper)

47 Seidenberg, J. & Rector, A. (2006). *Web ontology segmentation: analysis, classification and use*. In the Proceedings of the 15th international conference on World Wide Web, pp 13 - 22, ACM 2006

48 Knublauch, H. (2003). *An AI tool for the real world - Knowledge modeling with Protégé*. In Java World, June 20, 2003

49 Knublauch, H., Fergerson, R. W., Noy N. F., & Musen, M. A. (2004) *The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications*. Presented at the Third International Semantic Web Conference, Hiroshima, Japan (2004)

50 Knublauch, H, Musen, M. A., & Rector, A. L. (2004). *Editing Description Logic Ontologies with the Protégé OWL Plugin*. Presented at the International Workshop on Description Logics, Whistler, BC, Canada (2004)

51 Knublauch, H., Olivier Dameron, O., & Musen, M. (2004). *Weaving the Biomedical Semantic Web with the Protégé OWL Plugin***.** Preseneted at the First International Workshop on Formal Biomedical Knowledge Representation, Whistler, BC, Canada (2004)

52 Protégé Support Page : http://protege.stanford.edu/support/

53 Maedche, A., Motik, B. & Stojanovic, L. (2003). *Managing multiple and distributed ontologies on the Semantic Web.* In The VLDB Journal — The International Journal on Very Large Data Bases, Volume 12 Issue 4 Publisher: Springer-Verlag New York, Inc.

54 SemVersion : http://ontoware.org/projects/semversion

55 Groza, T., Völkel, M., & Handschuh, S. (2006). *Semantic Versioning Manager: Integrating SemVersion in Protégé.* In Proceedings of the 9th International Protégé Conference (2006)

56 Knublauch, H. (2006). *Ontology-Driven Software Development in the Context of the Semantic Web: An Example Scenario with Protégé/OWL.* Presented at the International Workshop on the Model-Driven Semantic Web, Monterey, CA (2004)

57 Rector, A., Drummond, D., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., & Wroe, C. (2004). *OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns.* Presented at the 14th International Conference on Knowledge Engineering and Knowledge Management (EKAW), Whittlebury Hall, Northamptonshire, UK (2004)

58 Holten, R., Dreiling, A., and Becker, J. (2005). *Ontology-driven Method Engineering for Information Systems Development*. In Green, P. and Rosemann, M. (Eds). *Business Systems Analysis with Ontologies*. (chapter 7). Idea Group Publishing.

59 Terrasse, M., Savonnet, M., Leclercq, E., Grison, T. & Becker, T.. (2006). *Do we need metamodels AND ontologies for engineering platforms?*. In the Proceedings of the 2006 international workshop on Global integrated model management, pp 21 – 28, ACM 2006

60 *Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering* W3C Working Draft (060211 Version). Accessed from http://www.w3.org/2001/sw/BestPractices/SE/ODA/ 2006/09/28

61 *Ontology Definition Metamodel Sixth Revised Submission to OMG*. (RFP, ad/2003-03-40). Accessed 2006/09/10 from http://www.omg.org/ontology/ontology_info.htm#RFIs,RFPs

62 Miles, M. & Huberman, A., *Qualitative Data Analysis 2nd Ed.* (1994). Sage Publications, London

63 Feldman, M., *Qualitative Research Methods Vol 33, Strategies for Interpreting Qualitative Data*. (1995). Sage Publications, Inc., London

64 Wolcott, H. *Qualitative Research Methods Vol 20, Writing Up Qualitative Research*. (1990). Sage Publications, Inc., London

65 Jennings, N., Sycara, K., & Wooldridge, M. (1998). *A Roadmap of Agent Research and Development.* In *Journal of Autonomous Agents and Multi-Agent Systems*. 1(1), pages 7-36. July 1998.

66 Franklin, S., & Graesser, A. (1996). *Is It an Agent, or Just a Program? A Taxonomy for Autonomous Agents*. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, *Lecture Notes in Computer Science; Vol. 1193*. (pp. 21-35). London, Springer-Verlag.

67 Zambonelli, F., Jennings, N., Wooldridge, N., (2003). *Developing multiagent systems: The Gaia methodology*, In ACM Transactions on Software Engineering and Methodology (TOSEM), July 2003, Volume 12 Issue 3, ACM Press 2003

68 Brooks, R. A. (1985). *A Robust Layered Control System for a Mobile Robot.* IEEE Journal of Robotics and Automation, Vol. 2, No. 1, March 1986, pp. 14–23 (also MIT AI Memo 864, September 1985)

69 Sycara, K. (1998). *Multiagent Systems.* AI Magazine 19(2): Summer 1998, pp. 79-92.

70 http://java.sun.com/javaee/index.jsp

71 http://java.sun.com/webservices/index.jsp

72 http://java.sun.com/j2ee/white/connector.html

73 http://java.sun.com/integration/

8 Appendix : Technical Interviewees, Focus Group

ID	Type	Experience	Education/Qualifications	Notes
1	Development Team – Analyst	20+ years, Data Architect	Graduate studies	Worked on knowledge modeling.
2	Development Team – Analyst, Developer	25 years, Solutions Architecture, Distributed Systems, OO Software, Persistence, Performance	M.Sc, Prof Qual	Worked on knowledge modeling.
3	Development Team – Analyst, Developer	17 years, Solutions Architecture, Distributed Systems, OO Software, Persistence, Performance	B.Eng, Prof Qual	Worked on meta-data-driven construction projects
4	Development Team – Analyst, Developer	17 years, Distributed Systems, OO Software	B. Eng	Worked on meta-data-driven construction projects.
5	Functional	25 years, Learning, Preferences, Advising	M.A. (communications), Prof Qual	
6	Functional	30 years, Strategic Business, People Issues, Learning Support, eLearning, Advising	M.Admin (O.D), Prof Qual	
7	Functional	20 years, eLearning,	B.A. (HR, Training), Prof Qual	
8	Functional	30 years, Business, Communications	Social Psychology and Exec MBA, Prof Qual	
9	Functional	20 years, Career Coaching	M. Psych, Prof Qual	
10	Functional	30 years, Social Psych, Assessment	PhD Soc Psych, Prof Qual	

9 Appendix : Data Management

9.1 Configuration and Change Tracking

The following diagram illustrates how information is tracked. Items tracked are listed in the following section.

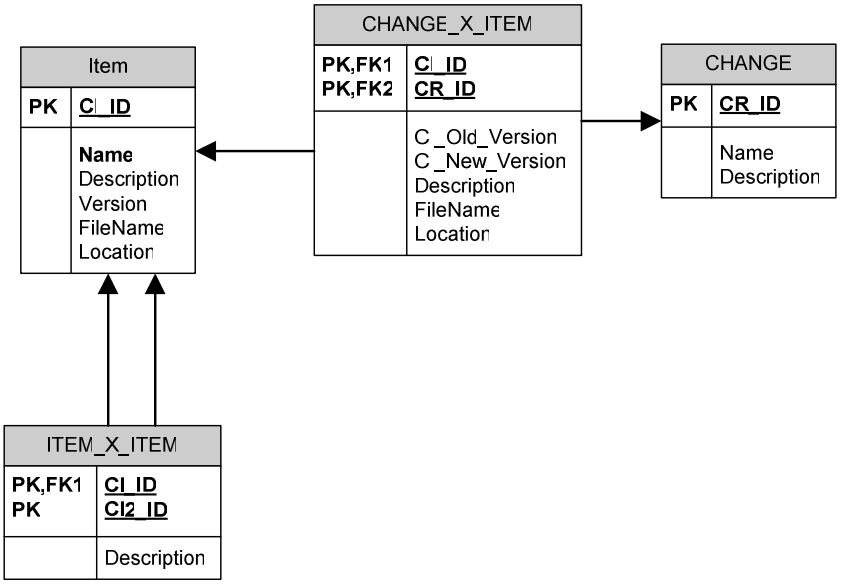


Figure 10 Configuration and Change Tracking

This configuration and change tracking information is stored in a simple database, exported to an Excel spreadsheet for reference purposes.

10 Appendix : Further Background on Agents and Multi-Agent Systems

This appendix relates information about technology and research as uncovered during the Literature Review but not considered to be directly related to the focus of this thesis. It is included here for informational and background purposes.

10.1 Software Agents

10.1.1 Software Agent Definition and Overview

Definition of Software Agents : Situated, Flexible, and Autonomous

There are many definitions of Software Agents. One of the most concise is that given by Wooldridge and Jennings [**65**] as "a computer system, **situated** in some environment, that is capable of **flexible autonomous** action in order to meet its design objectives", emphasis theirs.

Situatedness is defined as being aware of and able to change the environment. Autonomy is defined as being able to act without intervention of a human or other agent. Flexibility in turn is defined as being responsive (responding in a timely manner to changes in the environment), pro-active (capable of taking the initiative, of opportunistic and goal-oriented behaviour) and social (able to interact with humans and other agents).

Agents are commonly shown as interacting with their environment via Sensors and Effectors. The following diagram gives a high-level pictorial representation of a Software Agent to illustrate flexibility, autonomy, and situatedness. Note the sense -> reason/plan -> act feedback loop as illustrated by the dotted-line arrows running through Environment, Sensor, Agent, Effector.

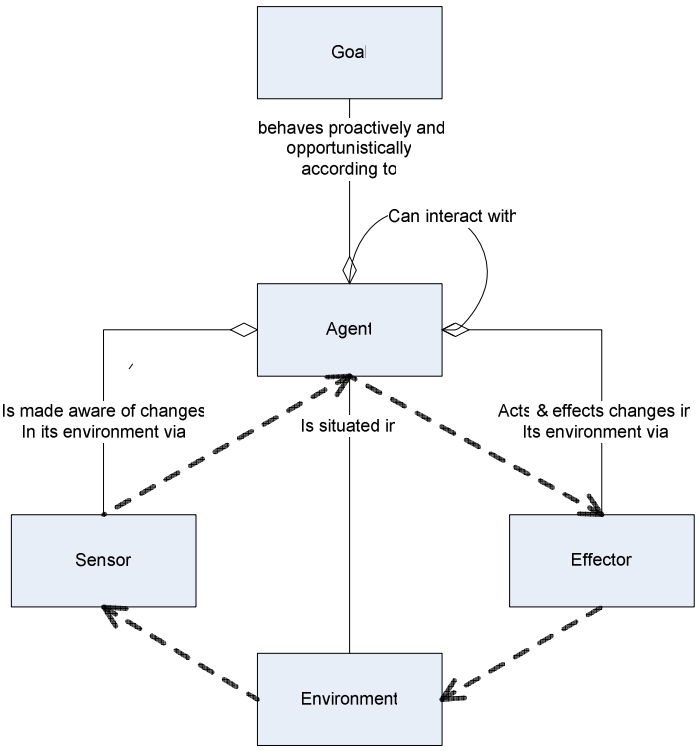


Figure 11 Basic Software Agent

Mobility and Adaptability

Some Agent taxonomies consider mobility and adaptability as mandatory attributes of Agents. For the purposes of this paper, mobility and adaptability will be considered specializations and not required attributes of all Software Agents.

Agents compared to Objects

The definition of Agents in terms of differences from Objects is quite natural, especially considering that, Object-based systems and Object Oriented programming, along with Artificial Intelligence and Human Computer Interface Design, can be seen as one of the

fields from which Software Agents has emerged as a distinct, yet related field of research [**65**].

Agents can be implemented as Objects and many Agent development toolkits have taken just this approach. The boundaries between objects and agents can become blurred as Agents can be implemented as Objects, and Agent-like behaviour is becoming more prevalent in Object-based (including Object-based and Object-Oriented) systems.

One way to view the difference between Object and Agent is that, in general, what differentiates an Agent from an Object is the level of abstraction between the two :

- Objects encapsulate functionality and data
- Agents go one step further with the addition of situatedness, flexibility, and autonomy
- Mobility and Adaptability further specialize the Agent abstraction

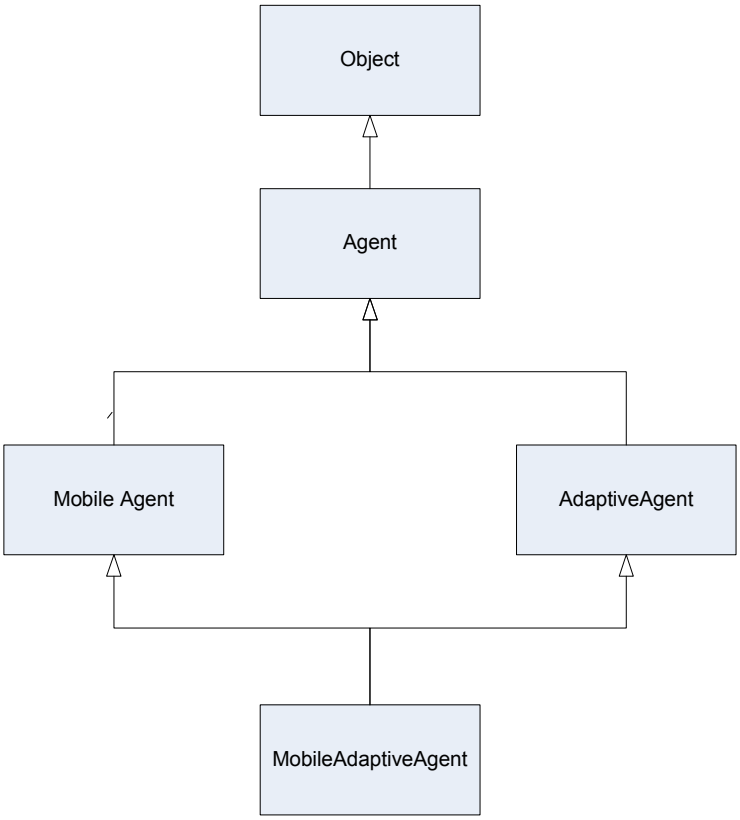


Figure 12 Object / Agent Abstraction as UML Inheritance

10.1.2 Categories of Agents

As with any technology, a common vocabulary and understanding of concepts is vital for understanding Software Agents and much work has been done on identifying and defining taxonomies such as [66]. In the face of so many ways to classify and group Agents, for the purpose of this paper, we will categorize Agents based on individual Agent architecture and functionality of the Agent.

Agent Architecture: Deliberative or Reactive

At the lowest level, Agents can be categorized as Deliberative or Reactive based on how they determine what actions to take. Deliberative Agents work from first-principles and reason about macro-level consequences to decide on an execution plan. Reactive Agents

are architected to react quickly and simply to changes in their local environment without extensive planning and reasoning about non-localized effects.

Agent Function:

Categorizing agents by function is useful to understanding and we can do so based on common roles for Agents as seen in existing systems and research, as well as for domain-specific roles. Agent functions could include infrastructure, task, domain-specific, and application-specific roles.

10.1.3 Agent Coordination

Agents can coordinate their activities in either a cooperative or a competitive manner. Cooperative interaction is characterized by planning whereas competitive interaction can be characterized by negotiation.

10.2 Multi-Agent Systems

10.2.1 MAS Definition and Overview

MAS Definition : Software Agents working together to meet their goals

Multi-Agent Systems (MAS) are composed of software agents that work together to meet their goals, using high-level protocols and languages. Wooldridge & Jennings [**65**] list the following characteristics common to Multi-Agent Systems :

1. individual agents having a limited viewpoint, with insufficient information or capabilities to solve the problem without assistance
2. lack of global system control
3. decentralized data
4. asynchronous computation

MASs compared to Distributed Object Systems

Note that many of these characteristics are common to all complex distributed systems and the comparison of MASs to distributed object-based systems leads to similar confusion as the comparison of Software Agents to Objects. There is indeed much overlap in problem areas and research between distributed systems in general and MASs. The difference between distributed systems in general and MASs stem from the fact that the components of an MAS are Software Agents (and thus situated, flexible, and autonomous).

MAS Functional Objectives

Zambonelli et al. [67] categorized Multi Agent Systems by their objective as follows:

1. distributed problem solving
2. intelligent integration of legacy systems
3. systems that are naturally modelled as a society
4. efficient use of spatially distributed information sources
5. solutions where expertise is distributed
6. performance enhancement by {computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility, reuse}

With the exception of 3 and 5 (systems modelled as a society, distributed expertise), and potentially 2 intelligent integration, these objectives are common to many non-MSA distributed systems as well. This is not unusual as MASs can be seen as specializations of distributed systems.

The application example of the eAdvisor MAS is a system naturally modelled as a society (Student, Advisor, etc.), and will allow for integration with legacy systems. Other attributes that may be seen include efficient use of distributed information sources and

distributed expertise. The Research Goal is to understand how to meet these objectives without compromising important quality, performance and maintainability metrics such as those itemized under 6. performance, above.

MAS Membership: 'Closed' versus 'Open'

MASs can be Closed or Open. Closed MASs are distributed problem solving systems that most resemble analogous to traditional distributed systems in that all agents are designed to work together to meet a specific objective. Open MASs are quite different from most traditional distributed systems in that the agents in an open MAS have not been designed to work together and may even have been developed by different organizations for different purposes. Agents may also join and leave an Open MAS, leading the MAS to dynamically change over time.

10.2.2 MAS Architectures

MAS Architectures can range from sophisticated to reactive. In the most sophisticated models, agents are able to reason about the consequences of local actions. In the most reactive models, agents are purely reactive and behave in a simplistic stimulus response behaviour. As with most things, an actual MAS architecture is not likely to be one extreme or the other but rather to combine elements of the two.

10.2.2.1 Deliberative Architectures

Deliberative agents are most commonly modelled as Belief-Desire-Intention (BDI) agents as their behavior is based on :

- **beliefs :** what the agent knows about its environment

Beliefs can be a priori or contain inference rules with forward chaining to new beliefs

- **desires :** the agent's goals

  An individual Agent's goals should be consistent, although the goals of one Agent may conflict with those of another.

- **intentions :** actions that the Agent has decided to take

An example of a sophisticated Multi-Agent architecture is RETSINA. The system is made up of "deliberative" agents called BDI Agents due to their having Beliefs, Desires and Intentions (BDI).

10.2.2.2 Reactive Architecture

Deliberative Agents that reason about all potential options before deciding on an action may be usable in planning and simulation scenarios but Agents with time-critical objectives may not react in a timely enough manner to changes in the environment to achieve their objectives. To deal with such time-critical objectives, Brooks [68] initially proposed using Reactive agents organized in what he called a Subsumption architecture to control an autonomous mobile robot. Given the type of application being addressed by Brooks the time-criticality of the Agents' reactions are apparent.

Brooks basically took the typical horizontal pipeline of functionality that processed sensor input and determined the action to be performed and replaced it with a vertical stack of simpler task-oriented behaviours, where higher-level behaviours are both more abstract and can subsume the lower-level behaviours. The overall action that is determined is a function of the behaviours' interaction.

The following two figures are based on Brook's initial application of the subsumptive architecture [68], and illustrate how Brooks took the pipeline reasoning and replaced it with a subsumptive layered architecture of behaviours.

Many systems are built in a layered architecture today to realize the same benefits Brooks gives for the subsumption architecture : decomposition into behavior/specialization layers, allowing incremental build/test and focus.

10.2.2.3 Hybrid Architectures

Subsumption architectures are very well suited to some applications but in general are too limited for many applications [**65**]. As a result, Hybrid Agent Architectures were proposed that incorporated aspects of both the Reactive and Deliberative approach. These hybrid architectures take advantage of the layered approach of the Subsumption architecture by using layers of software, with increasing layers of abstraction. The lowest layer(s), modelled after the subsumption architecture, is called the Reactive layer which performs actions based on sensor input. The next-highest layer(s) add abstraction and are referred to as the Knowledge layer(s). The topmost layer works with the MAS as a society and is called the Social Layer.

The layered architecture put forth by [**69**] is a hybrid containing elements of both reactive and sophisticated architectures. There is a reactive layer close to the environment, where decisions are made based solely on raw sensor input. This reactive layer interfaces with a knowledge-based layer of increased abstraction above it, which in turn interfaces with a social layer where inter-Agent coordination takes place

10.2.3 MAS Organization

MAS Seen as Analogous a Human Organization : A Society

Human organisation metaphor put forth by Zambonelli et al. [67].

- set of situated Agents (which may belong to multiple organisations)
- agents have roles and responsibilities/subgoals
- agents may be altruistic or opportunistic
- interactions are driven by the goals of the agents

Just as a well-defined and publicly-available Roles & Responsibilities document facilitates a project team in meeting their shared objective as efficiently as possible, each MAS, being an organisation, has a Structure. Three main categories of organization structures are

Hierarchical Structure

The hierarchical structure is the most familiar and easily understood model. Control moves from the top of the hierarchy down in the superior / subordinate path.

Community of Experts Structure

The Community of Experts organizes the MAS as a group of specialized Agents working together in a flat organization based on “rules of order”

Market Structure

The Market organizes the Agents in the MAS as Managers and Contractors, where Managers bid for services and Contractors offer services.

10.3 Systems Integration

Agents often need to integrate with other applications. This integration can leverage proven technology and development with J2EE [70] is a good choice given its ability to

be used with Protégé and JADE and the rich set of tools to support integration with other systems.

Specific areas related to J2EE to be further reviewed include Web Service support [71], and Enterprise Information System integration [72]. Sun is working toward bringing together existing integration APIs to provide robust scalable integration services [73] and this may be considered for future directions.

10.4 Complementary Technologies

The benefits of Agent based systems are being actively incorporated into technologies. Examples include Web Services and Jini at lower levels and J2EE, .NET, which provide robust frameworks that support systems providing much of what an MAS could provide. Some complementary technologies include Service Oriented Architecture, Web Services, JINI, J2EE, .NET, Ubiquitous Computing, Grid Computing, and Distributed Computing.