A Collaborative Framework for Knowledge Acquisition and Management for Bioinformatics Applications

Keywan Hodaei Esfahani

A Thesis in the

Department of Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science

Concordia University

Montreal, Quebec, Canada

January 2008

Your file Votre référence
ISBN: 978-0-494-42531-2
Our file Notre référence
ISBN: 978-0-494-42531-2

Abstract

A Collaborative Framework for

Knowledge Acquisition and Management for Bioinformatics Applications

Keywan Hodaei Esfahani

We study Software Engineering Organizations (SEOs) in the area of Bioinformatics in the context of Knowledge Intensive Firms. From this perspective, we characterize the challenges SEOs may face in this area and show that the situation can be much improved by following proper knowledge management practices. In response to these challenges and considering the various software development activities in this area, we propose a Collaborative Knowledge Management Framework (CKMF). The framework has four components: data model, knowledge management database, constraints, and management committee. Data model has three layers and is the central knowledge repository and acts as knowledge transfer media. Knowledge management database stores and manages information about concepts and relationships captured in the data model layers. It can also support version management. Constraints encode the semantic integrity of the application domain. . Managing committee is an elected body or committee in charge of defining and enforcing constraints and version management.

Deploying the proposed framework, we can better identify, preserve, and institutionalize the knowledge possessed by the software developers. This will reduce the impact of attrition on SEOs, will ease steep learning curves for the Software Engineers new to the field or organization, will provide a history of knowledge evolution in the organization that can be used for postmortem analysis, and will greatly facilitate the flow of knowledge among the experts within and across organizational boundaries.

We develop a prototype knowledge management of the proposed framework, and demonstrate a mapping between major needs and the framework elements. We also compare our approach with existing Bioinformatics Software Engineering tools and facilities. Our attempt in this work has been to offer a means for knowledge management in SEOs that captures individual creativity and team work, and acknowledges the importance and values of collective achievements at the same time.

Acknowledgments

This research would not have been possible without the support of many people. I would like to thank my supervisor, Professor Nematollaah Shiri, who gave me his constant support, valuable advice, and generous attention during the course of my studies in Concordia University. His patience and profound knowledge were my most valuable resources. I would also like to thank Dr. Christopher Baker, who helped me a lot with biology concepts and applications. Especial thanks to my thesis examination committee for their valuable comments and feedbacks.

This research was supported in part by Natural Sciences and Engineering Research Council (NSERC) of Canada, Genome Quebec, and Concordia University.

Thanks to the faculty members and researchers in the Genome Quebec Project. My special thanks to Halina Monkiewicz and other staffs in the CSE department for their support during my studies in the master's program. .

I am extremely grateful to my wife, Bahareh Amini, who endured this long process with me, always offering support, love, and technical insight. Her encouragement and energy was the driving force that kept me going.

List of Abbreviations

Abbreviation	Phrase
SE	Software Engineer
SEO	Software Engineering Organization
KIF	Knowledge Intensive Firm
KW	Knowledge Worker
DM	Data Model
KMD	Knowledge Management Database
MC	Managing Committee
SAN	Storage Area Network
NAS	Networked Attached Storage
SMIS	Storage Management Initiative Specification
CKMF	Collaborative Knowledge Management Framework

List of Figures

List of Tables

Table of Content

Chapter 1

Introduction

This work reports my research work in the context of the Génome Québec [106] which supports major genomics and proteomics research initiatives in academic and industry domains.

As in any interdisciplinary project, the major issue faced was effective communication between biology specialists and scientists, and computer scientists and software engineers (SE) involved in bioinformatics research and development, mainly due to their different nature of expertise, background, and vocabulary. Biologists complained that they had to repeat the same training cycle every time a new SE joins the team. Also, when a SE leaves the team, the knowledge acquired by him/her would be lost since there was no systematic, convenient method or infrastructure that could preserve the knowledge. Almost every time that such project finishes, the "collective" knowledge that was acquired by SEs became unavailable since that knowledge was not institutionalized in the first place.

These problems and issues led us towards working on a framework for acquiring domain knowledge in the field of Bioinformatics and making that knowledge available for transfer and distribution.

In the rest of this chapter, we will review the current status of Software Engineering and how it is affecting the economy as a whole. We will then recall the concepts of Software

Engineering Organizations, Knowledge Intensive Firms, Knowledge Workers, and describe how they are related to our research. This helps us reemphasize the importance of "knowledge" in modern economy in general and in our context of software engineering practices in bioinformatics. We will finally close this chapter by providing a list of contributions and the organization of this report.

1.1 Current status of Economy and Software Engineering

Economy is shifting focus as knowledge based industries are becoming the major role players compared to earlier heavy manufacturing industries. The emerging areas such as Bioinformatics and reliance on information technologies to achieve its goals are testaments of this change. Software development which is an essential part of Bioinformatics is a quickly changing and knowledge intensive discipline involving many people working in different phases and activities [80]. Computer software as we know it today has been around for about more than half a century, however Software Engineering is relatively a young discipline [89] and still growing [33]. The young and evolving software industry is one of the pillars of today's economy in industrialized as well as in developing countries. The impact of Software Engineering virtues and misdeeds go far beyond software industry and affect other industries and economy as a whole. Bioinformatics is no exception.

In our research we have been dealing with a wide range of entities that could be categorized as suppliers and/or consumers of knowledge in the field of Bioinformatics.

An entity that is somehow involved in Software Engineering activities could be categorized as a Software Engineering Organization (SEO). This classification is broad and covers a wide range of entities from a company that produces commercial software systems to an IT department in an institution. The idea of this definition is to maximize the number of entities that might be able to contribute to the knowledge acquisition and distribution process.

SEO is a special type of Knowledge Intensive Firm (KIF). KIFs are the "skeleton" of knowledge based economy. They mostly consume existing knowledge and produce new knowledge in the course of their daily activities. SEOs involved in Bioinformatics demonstrate a similar behavior: they consume existing knowledge which consists of domain knowledge (biology) and Software Engineering knowledge and they produce new knowledge. Knowledge is the primary resource for KIFs. In most businesses, including bioinformatics, knowledge exists in the minds of Knowledge Workers (KW) in form of experience and know-how, and it does not exist independently. Bulk of the knowledge exists in the minds of Bioinformaticiens or it is embedded in the systems, programs, or pieces of code that have been developed by Bioinformaticiens.

The main asset of a KIF in general and a SEO in particular is its intellectual capital, as it is often in sectors such as consulting, law, investment banking, and advertising [80]. The primary concern of a KIF is preserving the knowledge created and accumulated by its KWs and making that knowledge available for reuse in future by the same or other KWs. Since Software Engineering is a knowledge intensive discipline, the main resource for SEs, who are KWs of a SEO, is their knowledge that they use to perform their duties

(which could be creating a new piece of software or maintaining an exiting software system).

1.2 Problem Statement

SEOs involved in Bioinformatics face the same challenges as other KIFs. Identifying these challenges could help us better understand the problems and needs in Bioinformatics applications. A major concern for SEOs we focus on in this research is lack of well established knowledge management practices that could help software developers take advantage of their most valuable asset, namely the knowledge. Our goal in this work is to support and facilitate software development activities in bioinformatics domain through CKMF, a framework we propose for knowledge acquisition and management, which helps identify, preserve, and institutionalize the knowledge possessed by experts. Unavailability of institutionalized knowledge increases the impact of attrition on organization's competency, increases learning time and cost and thus defers productivity, and hampers the flow of knowledge within and across organizational boundaries. In this regard, CKMF also reduces the impact of attrition, eases steep learning curves, provides a history of knowledge evolution, and facilitates the flow of knowledge within and across organizational boundaries.

1.3 Contributions of the Thesis

We present a framework for knowledge acquisition and management for Bioinformatics software development environments. The goal is to support and facilitate gathering, storing, managing, and distributing the domain knowledge which SEs gain in the course of projects as well as Software Engineering "best practices" in Bioinformatics domain. The framework serves as a collaboration platform, knowledge repository and transfer media, which supports rapid development and integration of applications in the target domain.

Looking at the challenges KIFs and SEOs face, our framework is designed to provide a basis for capturing SEs skills and know-how while not disrupting their autonomy and creativity, helping alleviate the communication problems among SEs and SE teams, and providing information that can be used as input to an objective quality assessment process. Using CKMF in a Bioinformatics software development environment shall reduce the impact of attrition, shall ease learning curves for those new to the environment, shall help avoid repeating mistakes and will facilitate the knowledge movement within and across organizational boundaries.

The main contributions of this thesis are as follows:

- Introducing an architecture of the proposed framework
- Describing the components of framework architecture
- Presenting a prototype of CKMF and illustrating its components in Bioinformatics domain
- Presenting a preliminary evaluation of the proposed framework

1.4 Thesis Outline

The rest of this document is organized as follows. In chapter two we present a background and review related work. This includes the basics of knowledge management (including a quick review of KIF and KW characteristics and behavioral patterns) and a review of related research on software reuse and knowledge preservation techniques. In chapter three, we will present our proposal, a Collaborative Knowledge Management Framework for Bioinformatics. We will discuss the requirements of the proposed framework and describe details of its components and characteristics. In chapter four, we provide technical details on the design of our framework and illustrate its applications. Chapter five introduces implementation details of a CKMF prototype system we developed. In chapter six, we evaluate CKMF using a set of known Software Engineering metrics and compare it with related existing Software Engineering tools in the field of Bioinformatics. In the last chapter, we will provide concluding remarks and discuss possible future research.

Chapter 2

Background and Related Work

This research focuses on the role of knowledge in SEOs and demonstrates importance of knowledge management systems in SEOs. In this chapter we introduce the concepts and terms used in the domain of knowledge management and elaborate on how they are related to the subject of this research. We also review research in Software Engineering domain related to our work.

2.1 Knowledge Management Basics

Knowledge management methodologies and systems create value from intangible assets of an enterprise [19]. The most valuable intangible asset of an organization is the knowledge it posses. In this section, we recall definitions of "knowledge" and "knowledge management."

2.1.1 Knowledge

Some authors, most notably in IT literature, define knowledge by differentiating among knowledge, information, and data [2]. The generally accepted understanding of the

demarcation of data, information, and knowledge is a progression of these elements from unstructured to structured [23]. The advocates of this view offer the following definitions: Data is raw strings of numerical and non-numerical characters, which when processed, becomes information and when authenticated, becomes knowledge [27, 56]. Since the definition of data is simple and clear, it is easy to recognize data in a context. The same is not true for information and knowledge. Generally, it is not feasible to distinguish between information and knowledge based on structure, accuracy, or utility of the supposed information or knowledge. The differentiating factor is the fact that knowledge exists as processed information in the mind of individuals. In other words, knowledge is personalized information related to facts, procedures, concepts, interpretations, ideas, observations, and judgments [2].

There is an alternative way of looking at the hierarchy of data, information, and knowledge. Tumoi [23] presents a model that explicates the relationship between these three terms in a new way. This model exposes the need for reconsidering the traditional hierarchy of data, information, and knowledge if it is going to be used to provide information system support for knowledge management and organizational memory. Since our study aims to support knowledge management in SEOs, we need to consider the alternative hierarchy that Tuomi introduces.

In the alternative view, the hierarchy of data, information, and knowledge is turned upside down. Data emerges last, only after knowledge and information are available. No isolated pieces of simple facts can exist unless someone has created them, with intended meanings, using his or her knowledge. Only after a meaning structure or semantics is

developed and fixed, data can emerge, because the already created meaning structure or semantics is required to represent information.

The meaning structure that underlies knowledge for an individual is articulated through cognitive effort to become focal and structured [23]. When the meaning is expressed within a linguistic and conceptual context, it can be documented in a verbal and/or textual form. At that point it is typically called information. The next step would be splitting information into atoms having no meaning that would need to be taken into account in automatic processing. These atoms are called data. In short the model explicitly emphasizes that knowledge comes first. When knowledge is articulated, verbalized, and structured, it becomes information. When information is assigned a fixed representation and standard interpretation, it becomes data. The model also implies that even the most elementary piece of data has already been influenced by the knowledge processes that led to its identification and collection.

Traditional hierarchy and reverse hierarchy share one point in the way they define knowledge and that point is critical to their definitions. The shared point is the fact that knowledge does not exist outside of an agent (a knower). The fact that a knower is required for existence of knowledge indicates that knowledge is the result of some kind of cognitive processing. In line with this view, it is possible to say that information is converted to knowledge once it is processed in the mind of individuals. Looking at the relationship between information and knowledge and following the above mentioned view, it is also perceived that knowledge becomes information once it is articulated and presented in the form of text, graphics, words, or other symbolic forms [2].

This view leads to two significant implications. First corollary is that since knowledge is always shaped by one's mind and needs as well as one's initial stock of knowledge [31], individuals need to share a certain level of background knowledge in order to arrive at the same interpretation and possibly the same understanding of data or information presented. Another important insinuation of this view is related to the public perception of systems designed to support knowledge in organizations. Since the distinction between information and knowledge might not be that obvious, the knowledge management systems may not appear radically different from other types of information systems in the organization, however knowledge related systems should be geared toward enabling users to assign meaning to information and to capture some of their knowledge in information and/or data [2].

Knowledge may be viewed from several perspectives [2]:

- A state of mind: Knowledge can be described as "a state or fact of knowing" with knowing being a condition of "understanding gained through experience or study; the sum or range of what has been perceived, discovered, or learned" [Schubert et al. 1998]. In this perspective, the focus is on enabling individuals to expand their personal knowledge and apply it to the organization's needs.
- An object: Knowledge may be viewed as an object [Carlsson et al. 1996; McQueen 1998; Zack 1998a] which can be stored and manipulated.
- A process: Knowledge can be viewed as a process of simultaneously knowing and acting [Carlsson et al. 1996; McQueen 1998; Zack 1998a]. The focus is on applying expertise [Zack 1998a].

- A condition of having access to information: This perspective could be seen as an extension of the view of knowledge as an object, with a special emphasis on accessibility of the knowledge objects. The emphasis is on how knowledge should be organized to facilitate access and retrieval of content.
- A capability: Knowledge can be viewed as a potential for influencing future action [Carlsson et al. 1996]. This view can be extended to suggest that knowledge is not so much a capability for specific action, but the capacity to use information. Knowledge can be used decide what information is necessary in decision making [Watson 1999].

Each one of above perspectives leads to a different perception of knowledge management. We will discuss the impact of these perspectives when we discuss knowledge management.

Knowledge is categorized in various ways. A well-known classification which is related to our field of research divides knowledge into two main types: explicit knowledge and tacit knowledge. Explicit knowledge, also called codified knowledge, is expressed knowledge, which can be easily documented and communicated [80]. Examples of explicit knowledge are organization's processes, templates, and policies. Tacit knowledge on the other hand is personal knowledge that is gained through experience [80]. Tacit knowledge is deeply rooted in action, commitment, and involvement in a specific context [68]. This type of knowledge is hard to express and is fundamentally influenced by beliefs, perspectives, and values of the knowledge carrier. Examples of tacit knowledge are cognitive skills and experience. Each type of knowledge can be found at an individual, group, organization, multiple organizations, or industry-wide levels. While the

two types of knowledge are not entirely disjoint, there is a noticeable distinction between them. No knowledge is entirely tacit and no knowledge is entirely explicit [5]. Another way to express this distinction is to consider them as different dimensions or aspects of knowledge which exist in a symbiotic relationship [5] and complement each other.

2.1.2 Knowledge Management

In this section, we review the literature related to Knowledge Management (KM) and its objectives. The aim of KM is to continuously improve an organization's performance by improving and sharing of organizational knowledge throughout the organization. The goal of KM is to provide the right knowledge to the right people at the right time and in the right format [19]. KM is perceived to capture and integrate crucial elements from various sources, such as groupware, databases, applications, and most important of all, experts' minds, and make them readily available to users in an organized and logical form [28]. It provides a mechanism to improve both the knowledge and the learning process by determining the organization's knowledge needs, current state of organizational knowledge, and the gaps in knowledge and barriers to organizational learning, and then develop, implement, and improve proactive knowledge management strategies to support organizational learning [49].

Knowledge goes through various phases and transformations in an organization. KM is defined as a set of proactive activities to support an organization in creating, assimilating, disseminating, and applying its knowledge [49]:

- Knowledge creation: Members of an organization create knowledge through learning, problem solving, innovation, creativity, and importing from outside

sources [80]. Knowledge creation can be defined as the improvement of or increasing the certainty of a piece of knowledge, and occurs during a learning experience [49]. A lesson learned from an experiment is an example of an output generated by knowledge creation process.

- Knowledge assimilation: After knowledge is created, members of an organization try to capture, refine, and store it. This is done through an iterative process where in each step the more effective practices receive more attention and support and the less effective practices receive less attention and support. Elements that receive more attention and support are used more often and become engraved in organizational memory and therefore become part of organizational knowledge. The elements that receive less attention and support are not engraved in organizational memory and do not become organizational knowledge and therefore are forgotten over time. When some element of the created knowledge is absorbed and becomes part of organizational knowledge or vanishes completely and does not become part of organizational knowledge, assimilation process is complete. Knowledge assimilation is collecting and refining the created knowledge and storing it alongside existing knowledge in the organization's memory [49]. The concept of organizational memory has been the focus of many researches and it is difficult to define. One reason for this difficulty is that, while some researchers have tried to apply the concept of memory in biology and psychology to organizations, it is unclear whether or not information processing ideas that are derived primarily from work on biological organisms can be extended to social and organizational phenomena [5]. A working definition of

organizational memory would be that it stores information from organizations history and experience, and can be used for making decisions. The organizational memory is contained in organizational members, files, records, culture, processes, procedures, organizational structure, and physical structure.

- Knowledge dissemination is the process of distribution of the knowledge for use in another learning experience.
- Knowledge application is the use of past knowledge to help solve the current problem. In applying the past knowledge, a decision maker must adapt the knowledge to the current situation [49].

2.2 Knowledge Intensive Firms

KIFs can be defined as organizations that offer the use of fairly sophisticated knowledge or knowledge based products [5]. According to this definition, since software is a knowledge product, we may consider SEO as a kind of KIF. All activities in an organization related to creation of a product from the inception of initial idea until it is realized as a product and is ready for the market, can be grouped into two phases: (1) research and development (R&D) and (2) manufacturing. In the R&D phase, the organization works on a problem (or a need) and tries to provide a solution. In the manufacturing phase, the solution tailored in the first phase is formulated and becomes a solid and tangible product or service. The cost of each phase is calculated as the sum of the costs of machinery, raw material, and workforce used in the process. The products or services that KIFs offer are characterized by R&D costs that outweigh manufacturing

expenditure [5]. In line with this observation, SEOs are characterized by high cost of R&D compared to low cost of manufacturing.

Acknowledging that the idea of KIFs and related concepts are difficult to substantiate and any evaluation of knowledge intensiveness is contestable [6], we propose a measure to act as an indicator of knowledge intensiveness of organizations. It is important for our discussions to have an idea of how knowledge intensive an organization is. We have observed through our experience and this research that visibility and tangibility of KIFs' characteristics are linked positively to knowledge intensiveness of the organization. In other words, the more knowledge intensiveness an organization is, its knowledge related characteristics are more visible and more tangible. The measure we propose is the proportion of R&D cost and manufacturing cost. This indictor could be close to zero for organizations that have negligible R&D costs compared to manufacturing costs. The proposed indicator can also be quite large if the manufacturing cost of the organization is negligible compared to the R&D cost. Since the value of this indicator for SEOs is very large, we can conclude that SEOs are among the most knowledge intensive organization and therefore their knowledge related characteristics are very visible and tangible. This degree of visibility and tangibility for knowledge related characteristics makes SEOs very sensitive to issues related to knowledge management.

In terms of nature of the work and how they are managed and organized, KIFs share a number of characteristics [5]. Some of these, discussed below, are related to our work in this research:

- Highly qualified individuals perform knowledge based work, using intellectual and symbolic skills. In a KIF, the emphasis is not on the knowledge implanted in

techniques, rules, or procedures, although they are important. Instead the organization heavily relies on the knowledge that exists in the cognitive skills of knowledge workers. They are responsible for analyzing the requirements and providing solutions. In this process, knowledge is the input as well as the output. Knowledge means competence for a KIF. The more knowledgeable professionals are in a KIF, more effective they can be in performing their duties, and this translates into more competencies for the organization. Any attempt for increasing the competence level of KIF should be directed at finding ways to record, preserve, and maintain intellectual and symbolic skills of the individuals.

- A fairly high degree of autonomy and downplaying of organizational hierarchy exists in KIFs. The knowledge workers face unique problems in the course of their daily work and they need to provide solutions for those problems. The knowledge used in the process of providing a solution is not related to the ranks of an organization. To create an effective solution, the knowledge workers need to get a good understanding of the specifics of the problem, while maintaining a general view of the problem and its environment. Higher ranks of the organization might have more general experience but details of a specific and particular problem are less known to them; Therefore the higher rank managers know less of what can and should be done to solve a specific problem compared to the professionals working in the field. Such environment requires knowledge worker to have considerable discretion. Looking at professional communities involved in knowledge work, a high degree of self determination and collegial relationships across hierarchical positions are visible. Any means used to collect knowledge

from such autonomous individuals or teams should be as non-intrusive and non-disruptive as possible to make sure that it has little or no impact on their performance.

- There is clearly a need for extensive communication in KIFs aimed at coordination and problem solving tasks. Team work and collaboration for creating a common and shared understanding is common in knowledge based tasks and projects. Since often a project undertaken by a KIF is somehow unique, the structure of the team and responsibilities of the members are not well known and clear in advance. Teams adjust and shape themselves as they become more involved in the project and acquire more knowledge about the specifics of the problem. Traditional planning, rules, and methodologies which go into details in describing how things should be done are not effective enough and suitable in these settings. The high degree of agility required in the team structure (and the role each team member plays) plus the need to be in sync and agreement with the client during entire span of the project, highlights the need for extensive communications among professionals and between the professionals and the client. Knowledge management systems that might be used in these environments should not add to the communication burden and preferably should help KWs with their communication needs.
- It is difficult to objectively and accurately assess the quality of knowledge work. KIFs often deal with complex and fairly specific types of problems and this makes it difficult to define a set of rules and criteria for general quality assessment, which is important for organizations. Organizations require

measuring the quality of the final products to make sure that clients are satisfied and are able to track and manage quality trends in order to improve themselves (and therefore increase competency of their organizations). Organizations also need to assess the quality of contributions of the teams and their members. Since each problem is fairly unique in KIFs' field of work, using a predefined set of metrics for quality assessment might not work. On the other hand, defining a set of metrics for each project is also problematic because it makes it too difficult to compare the quality of different projects. To assess the quality of team members and teams in a project, considering relative autonomy of the knowledge workers and their position as the best information source to note what is being done, it is obvious that the knowledge workers need to be involved in defining the set of metrics (which will be used in assessing the quality of their contributions) as well as the assessment process itself. This will reduce objectivity of the output of process with regards to the quality of contributions of the knowledge workers. A knowledge management system that keeps track of the knowledge used and created during the course of a project, could help in the quality assessment and increase the objectivity of the assessment process.

Each of the above mentioned characteristics display the complexity of the inner working of a KIF and is considered a challenge for traditional management and organizational doctrines. KIFs are looking for tools, systems, or concepts to help them cope with this level of complexity that has not been dealt with before. We have considered the above challenges in the design of our framework. The framework has the ability to capture intellectual and symbolic skills while being least intrusive and least disruptive. The

framework management structure is tailored to include and involve KWs giving them opportunity to preserve their autonomy and creativity. CKMF also facilitates communication among KW individuals and the teams, and acts as a knowledge transfer media. The framework structures keep tracks of knowledge used and created in different projects and the evolution of knowledge over time. This information can be useful as an input to an assessment process.

2.3 Knowledge Workers

KW refers to those who create knowledge or to those whose use of knowledge is a dominant aspect of their work [21]. This definition can be expanded to include people with a high degree of education or expertise involved in creation, distribution, or application of knowledge. [21]. According to these definitions, Software Engineers (SE) are considered as KWs.

These definitions characterize a KW as an elite and distinguished element of the workforce who is required to be creative and make extensive use of knowledge in her daily work. Contrary to the industrial age era when machines and capital were the driving forces of economic development and prosperity, in today's knowledge based industries, KW and the knowledge carried are the main assets. In the past, from an economic standpoint people were needed but most of them were easily replaceable because they were mainly using their body and physical strength in general to perform routine tasks. Today KW is not only needed and can not be replaced easily; organizations try to keep their KWs as long as possible in order to preserve the knowledge in the organization.

Since a KW is supposed to be creative, apply his/her knowledge to solve particular problems, and create the knowledge in the process, s/he needs a high level of autonomy and self-ruling in work places and work patterns. On the other hand, organizations need to have some level of control over their projects and their KWs involved in the projects for administrative and financial purposes. It is important for organizations to maintain a balance between control and autonomy to properly manage KW in the work environment and motivate KW to share his/her knowledge with the organization and other KWs. Through its contribution and management mechanisms, KFM helps individual KWs and teams of KWs of SEOs maintain their autonomy and have some control over knowledge management process while enabling the management of SEO to participate in the knowledge management process and have some say in there as well.

2.4 SEO as KIF

The principle mission of a SEO is producing and maintaining software systems. Each software system is created to solve a problem or fulfill a requirement in a domain. SEs need to get acquainted with the target domain to be able to work effectively towards solving problems in that domain. There are two types of knowledge involved in the work of SEs: Software Engineering knowledge and target domain knowledge. Our framework is designed to capture the domain knowledge of SEs as well as Software Engineering best practices employed by SEs. Both types of knowledge that exist within SEs' minds in the form of experience and know-how are important for a SEO and are considered the main assets and intellectual capital of the organization. Managing the intellectual capital of the

organization is emerging as a pivotal task for surviving in today's competitive marketplace [24].

Organizational knowledge which is the "sum" of KWs' experience and know-how in an organization has been recognized as a key for success and competitive advantage in all activities [61]. KIFs strive to preserve the organizational knowledge by detaching it from KWs and institutionalizing it. A variety of perspectives suggest that the ability to marshal and deploy knowledge dispersed across the organization is as an important source of organizational advantage [90]. Knowledge is the key factor for an organization's competitive advantage, and because of this the production environment and infrastructure are playing a diminishing role and intellectual capital and knowledge management a growing one [38].

Software development is a knowledge intensive and complex process. New software systems tend to get more complex over time and so does the process of developing and maintaining such systems. Organizations cannot afford to increase resources along with the increasing demands [80] and sometimes increase in resources would not be helpful or effective at all; therefore, they expect a rise in productivity and performance to manage the situation. Productivity and performance are functions of the knowledge being used or applied in the daily operations and tasks. Knowledge must be continuously maintained and improved to guarantee organization performance and productivity increase. Knowledge is improved through the organizational learning process, which creates and shares knowledge from one part of the organization to another [49]. Knowledge in Software Engineering is diverse and its proportions immense and steadily growing. Organizations have problems identifying the content, location, and use of the knowledge.

An improved use of this knowledge is the basic motivation and driver for knowledge management in Software Engineering and deserves further analysis [80]. Some of the obstacles that SEOs face with regards to the organizational learning process are listed in Appendix A.

Our work yields a knowledge management tool for SEOs. It acts as a knowledge repository and a transfer media which facilitates the flow of knowledge from one part of a SEO to another, and thus improves the organizational learning process. The framework data model and its accompanying knowledge management database are designed to help with identifying the content, location, and use of each piece of a SEO knowledge.

Among the challenges SEOs face at the present time, they have to deal with the following problems that are directly related to knowledge management [80]:

- Loss of knowledge due to attrition. When workforce attrition occurs, the knowledge which KW used to carry is also lost and should be deducted from the organization's intellectual capital. In case of a SEO, this loss is two fold: loss of Software Engineering knowledge, and loss of domain expertise. Our framework helps reduce the impact of attrition through sharing the knowledge of individual SEs and distributing it in the SEO, thus making it true organizational knowledge.
- Lack of knowledge and an overly long time to acquire it due to steep learning curves. As computer systems find their ways into more complex domains, it becomes harder for SEs to absorb the necessary domain knowledge in a reasonable time. In fact, the more complex target domains get, the learning curve becomes steeper. Our framework enhances the organizational learning process through capturing knowledge of SEs and facilitating the distribution of the

captured knowledge. This helps ease the curve of learning path and reduces the time SEs need to acquire enough knowledge to become productive.

- People repeat mistakes and perform rework because they forgot what they learned from previous projects. Postmortem analysis is not common for software projects and the reason is lack of properly recorded information about the course of the project, the decisions made, and the reasons for the decisions. CKMF makes it possible to maintain a history of decisions SEs make during the course of projects and the results of those decisions are traceable over time and their affects can be realized as well. This history could be used as an input to an analysis process and the results could help evaluate past decisions in order to recognize mistakes and avoid them in future.
- Movement of knowledge within and across organizational boundaries, in an effective and cost-efficient manner in a distributed software development environment. The general trend in software industry is moving towards more integration. Software companies design and builds systems that enable users to consolidate their investments and integrate their systems as much as possible. Responding to this trend, individual SEOs, even competitors, have to work together to make their products compatible with other software and hardware products. In recent years, SEOs have been leaning more and more towards collaboration frameworks that enable them to share knowledge. CKMF acts as a collaboration platform that enables different groups share knowledge and cooperate while maintaining their unique views and their differences. It facilitates the movement of knowledge within and across organizational boundaries. The

fruit of this collaboration over time could be a consensus and a unified understanding of the target domain, which will be valuable for all participants. Our work is inspired by our experience in an industry effort of this kind. That effort is "CIMSAN Initiative", sponsored by Storage Networking Industry Association (SNIA). This program allows participants which are software and hardware companies share knowledge and collaborate in developing and improving a standard for managing storage area networks (SAN). The goal of companies that participate in this task is development and implementation of "SNIA's Storage Management Initiative (SMI)." The main goals of CIMSAN initiative, which are indicators of today' software industry trend, are: (1) Ease the implementation of the SMIS specification in vendor products (through ongoing developer symposia and plug-fests), (2) Reduce multi-vendor integration costs, (3) Build seamless interoperability between products, and (4) Forward the development of the CIM/WBEM based SMI Specification (SMIS).

A more detailed discussion about the above mentioned challenges can be found in Appendix B.

2.5 Related Work

The demand for more complex software systems today with many functionalities and for more varieties of software systems is greater than ever before. In some cases, the time-to-market window and production cycles have shrunk from several years to few months. The increasing size of software systems, degree of their complexity, more complex

standards, and more sophisticated user demands aggravate the situation. SEOs need to improve product time-to-market, software quality, and staff productivity simultaneously to keep up with the growing demands. They also need to reduce costs of development phase as well as maintenance phase.

Many studies show that the overall cost and time of software development, and the quality of the code developed, correlate most closely with the amount of new code written [67]. Based on this finding, it is clear that less new code should be written to improve many aspects of the software development process.

The idea of systematic reuse (the planned development and widespread use of software components) was first proposed in 1968 by Doug McIlroy [67]. Since then as we go forward in time and while Software Engineering matures, the importance of reuse is realized more deeply and the efforts are focused towards applying reusability characteristics to elements that are more conceptual and more abstract, and thus less tangible. The trend we observe in the history of Software Engineering is moving from reuse of simple and tangible pieces like code towards reuse of more abstract and conceptual elements such as design and architecture. It is now well known that software evolution and reuse is more likely to receive higher payoff if high-level artifacts, such as architectures and designs, can be reused and can guide low-level component reuse [54]. The ideal reuse technology provides components that can be easily connected to make a new system. SE does not have to know the implementation details of the component. The resulting system will be efficient, easy to maintain, and reliable [45]. Active areas of research in the past years related to reuse include reuse libraries, domain engineering methods and tools, design patterns, componentry, and generative reuse [34].

Our proposal is offered in the form of a framework. Frameworks share many characteristics with reuse techniques in general [50], and object-oriented reuse techniques in particular [45]. A brief review of reuse techniques in Software Engineering can be found in Appendix C. We review the framework concept and its application in our research in this chapter.

The other topics of interest related to our research are the collaboration platforms and methodologies in Software Engineering. In today's global economy, success and survival of individual firms strongly depend on their belonging to a network of collectively interacting firms. Because of this relationship, the knowledge building process is not confined to the organization itself, but transcends to the network of organizations to which the firm belongs [81]. The members of such networks need to collaborate to eventually create knowledge about what they do. The way collaboration among members of the network is organized and carried out is an active research topic in software industry.

2.5.1 Framework Concept

In this research, we are offering our proposal in the form of a "framework". In this section, we review the concept of framework and how we are going to take advantage of this concept in our research.

Framework concept is a cornerstone of modern Software Engineering. Framework development is popular and widely used because it promotes reuse of design and source code [58]. A framework can be defined as a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact [45].

Frameworks can probably be best described as the context that defines specific "pattern" of communication and cooperation among software components [45]. This definition identifies the product of our research more closely than other definitions.

Application frameworks consist of ready-to-use and semi-finished building blocks. The overall architecture, i.e. the composition and interaction of building blocks, is predefined as well. Using the framework to produce specific applications usually requires adjusting of building blocks to specific needs [74]. Using a framework results in some form of standardization of applications for the specific domain. Since it is not possible to anticipate all aspects and features required for development of applications in a domain, an application framework must have a mechanism to allow some degree of flexibility so that it can be can easily adapted to specific needs. CKMF is designed to allow different viewpoints to be accommodated and enables stake holders to extend existing concepts and build upon them.

Frameworks are tightly coupled to patterns in the way that they could represent the instantiation of a solution to a problem in a defined context [92]. The result of instantiation of the proposed framework is a product that can be described and identified by the definitions mentioned above. We have offered a sample instantiation of our proposed framework in the context of Bioinformatics in chapter 6. We have to emphasize that our attempt is more on illustrating the ideas behind the proposed CKMF, its merits, and advantages. More formal work is required to turn CKMF into a useful practical tool. Our proposal thrives on reuse techniques in Software Engineering and offers some features that are very much required in dynamic software development environments such as collaboration and version management. The final product is meant to help with

capturing, documenting, and transferring knowledge in a software development environment.

The three major stages of framework development are domain analysis, framework design, and framework instantiation [58]. In domain analysis stage efforts are aimed at identifying the domain's requirements and possible future requirements. In our research we focused on Bioinformatics field and we tried to identify current and possible future requirements of SEs in the field. The framework design phase defines the framework's abstractions. For realization of CKMF in the field of Bioinformatics, we used the requirements that we identified in the first phase and designed the abstractions. To comply with CKMF requirements, the abstractions had to be categorized into Core and Common layers. In the instantiation phase, the framework abstractions are implemented, generating a software system. In our prototype, we implemented some of the abstractions and generated sample applications.

2.5.2 Collaboration

Over the past decade, design and implementation processes indicate moving away from individual decision makers and getting close to groups engaged in collaborative work [22]. Greater competition in the market and globalization of economy has changed the structure of many organizations and these changes are more evident in SEOs. The new structures have intensified the demands for higher levels of support in distributed and collaborative work [10].

Inter-organization and intra-organization collaborations contribute to the emergence of standards and pseudo-standards. When entities collaborate, they try to help each other

achieve their goals. To maximize the benefits of collaboration, the participants need to agree on a set of rules for their cooperation and the results of their work. The rules governing the collaboration define how participants work together, and the rules about the results of cooperation defines how results should be produced and in what form and shape and under what conditions they should be presented. All participants need to agree with these rules for collaboration to work. CKMF provides a set of rules that govern collaboration and provides an infrastructure deciding about the results of cooperation, and how and when those results should be released.

2.5.3 Common Information Model

Common Information Model (CIM) [100, 102] is a conceptual information model for describing entities in the Internet, enterprise and service provider environments. It provides a consistent definition and structure of management information using object oriented techniques. CIM is a standard developed by Distributed Management Task Force. Its formal purpose is to represent and organize the information in a managed environment.

Our work in this thesis is inspired by our experience in software industry where we were involved in a multi corporation effort (refer to appendix D for list of participating companies), aimed at developing a management standard for Storage Area Networks (SAN) and Networked Attached Storages (NAS) called Storage Management Initiative Specification (SMIS). The program in which we were involved was called "CIM-SAN". The CIM/WBEM managed SAN or CIM-SAN, is the first permanent and open multi vendor development and demonstration environment for accelerating the implementation

of CIM/WBEM technology into products [99]. Among the primary goals of CIM-SAN, the following are related to our research:

- Facilitate the implementation of the SMIS specification in vendor products through ongoing collaboration among developers participating in symposia and plug fests
- Reduce multi vendor application integration costs and time
- Build seamless interoperability among products
- Forward the development of the CIM/WBEM based SMIS software products

CIM-SAN program is based on the highly successful CIM-SAN-1 demonstration at the Storage Networking World conference in October 2002, where 17 vendors integrated 32 products creating 97 points of interoperability among those products [99]. CIM-SAN has proved to be a catalyst in accelerating the development and integration of more functional network storage management products through promoting and facilitating collaboration, reducing time and cost of application integration, and making them interoperable. All of these are achieved while various software and hardware companies have been contributing to the development and evolution of SMIS.

The ongoing success of CIM-SAN program is a clear sign that SEOs are desperately in need of collaboration platforms that allow them to cooperate and work together through sharing knowledge in order to achieve their goals and create mutual benefits. This program is creating value for all participants, and this explains why they participate in the program and contribute.

2.5.4 Version Management

Version Management is part of Software Configuration Management (SCM). SCM is defined as the process of control of the evolution of complex systems. More pragmatically, it is a discipline that enables us to keep evolving software products under control, and thus contributes to satisfying quality and delay constraints [30].

SCM can be utilized as a management support discipline. In this capacity, SCM is concerned with functionalities such as identification of components and their versions, change control (by establishing strict procedures to be followed when performing a change), status accounting (recording and reporting the status of components and change requests), and audit and review (quality assurance functions to preserve consistency) [17].

Management of change through versions is an integral part of CKMF and is a key notion for management of the data model. CKMF allows participants to contribute their versatile points of view. These view points are reviewed in the next iteration of knowledge refinement, and their commonalities are mined and consolidated into certain artifacts. These artifacts are introduced in the next version of the framework data model. The version management process keeps track of changes that happen between releases of the framework data model and allows users to recognize differences of data model versions.

Chapter 3

Our Proposal

We propose a collaborative framework for knowledge acquisition and management (CKMF) for SEOs in field of Bioinformatics, which serves as a repository and knowledge transfer media, and supports rapid development and integration of applications in the target domain. The framework is designed to facilitate collaboration among parties working in Bioinformatics with overlapping contexts and help them capture knowledge and converge their interpretations and understandings over time.

CKMF has four components: data model, knowledge management database, constraints, and managing committee. The data model has three layers: Core, Common, and Extension. The Core layer acts as a conceptual foundation for other layers of the data model and includes basic and general notions applicable to target domain at large. It supports basic vocabulary for analyzing and describing basic concepts and entities of the target domain. The Common layer builds upon the Core layer and provides a basis for development of applications. It captures concepts related to particular areas of a domain, but independent of a specific technology or implementation. The Extension layer builds upon the Common layer and models objects that are technology and/or application specific additions to the Common layer. Any contributor can define Extension schemas as needed. The purpose of knowledge management database is handling physical and logical structural information of the data model. This database helps with exploration and

management of the data model and the framework itself. Constraints are the semantic safe guards of the framework and are mostly applicable to the data model. Managing committee is responsible for maintaining the integrity of the framework and enforcing constraints. Members of managing committee may include framework contributors and users.

In this chapter, we present the goals of the framework, its architecture and characteristics. At the end, we will discuss version management in the framework.

3.1 The Goals

Our main goal in the proposed framework is developing a mechanism for gathering, storing, managing, and distributing knowledge which software engineers gather in the course of projects as well as Software Engineering's best practices in a target domain, which is Bioinformatics in our research. The framework is meant to serve as a collaboration platform, knowledge repository, and transfer media. It is aimed to support rapid development and integration of applications in the target domain.

Knowledge management methodologies and systems create value from intangible assets of an enterprise [19]. The most valuable intangible asset of an organization is the knowledge it generates. We deal with two types of knowledge in SEOs: Software Engineering knowledge and target domain knowledge. CKFM aims to capture the target domain knowledge of Software Engineers as well as Software Engineering expertise which is best practices employed by Software Engineers. SEOs highly value both types of knowledge and consider them as their main assets and intellectual capital. The sum of

SEs' knowledge constitutes the SEO's knowledge, and when properly captured and documented, it formally becomes the organizational knowledge. Since organizational knowledge has been recognized as a key for success and competitive advantage in all activities [61], SEOs attempt to preserve this knowledge by detaching it from individuals and institutionalizing it. The first goal of CKMF is to capture and preserve the knowledge that exists within the Software Engineers' minds in the form of experience and know-how.

A variety of perspectives suggest that the ability to marshal and deploy knowledge dispersed across the organization is as an important source of organizational advantage [90]. Availability of knowledge is the key factor for an organization's competitive advantage. The software development process relies heavily on the two types of knowledge previously mentioned. Software systems get more complex over time, so does the process of developing and maintaining such systems. SEOs cannot afford to increase resources in response to the increasing demands, and sometimes increase in resources would not be helpful or effective at all. Therefore, they look for a rise in productivity and performance to manage the situation. Productivity and performance are functions of the knowledge being applied in the organization's daily operations and tasks. SEOs have to continuously maintain and improve their organizational knowledge to secure an increase in their performance and productivity. For this purpose, they have to invest in organizational learning process, which creates and shares knowledge from one part of the organization to another [49].

The second goal of CKMF is serving as a knowledge repository and a knowledge transfer media to help organizational learning process function effectively so that it can provide

availability of organizational knowledge throughout the organization and facilitate flow of knowledge from one part of organization to another.

The third goal CKMF is to facilitate collaboration among individuals or teams of Software Engineers within and across organizational boundaries. Considering this goal, the knowledge acquisition process is designed to be open and iterative in order to encourage different teams and/or individuals work together and share knowledge while keeping their differences. The data model is the main vehicle for capturing and storing knowledge in the framework. The different layers of the data model support different levels of agreement among framework contributors. If contributors are allowed to maintain their unique point of view while they share their common views with others, they will be more enthusiastic to participate in the collaboration process, and this is a motivation for our framework to facilitate collaboration. Participants in collaboration usually aim to expand the extent of their agreement over time and thus share more common knowledge. The knowledge acquisition of the framework is an iterative process which facilitates convergence of different points of view over time. At every iteration, a re-factoring process will review the acquired knowledge and refine it, if necessary. The idea is to recognize the common notions, formalize them, and make them reusable. The iterations aim at converging different perspectives and views and the final product could be a basis for developing standards, which are desired in general in evolving fields.

The fourth goal of CKMF is supporting rapid development and integration of applications in the target domain. This goal is linked to our efforts to increase the level of reuse among framework participants. Practitioners in many engineering disciplines rely heavily on well understood technologies and components that have been standardized. They

create customized systems economically by reusing the existing components and building only the parts that are specific to their application. Contemporary software systems have been simple enough for massive technology reinvention to be economically feasible. However, as software system complexity increases, technology reinvention becomes unaffordable [12]. The other factor which affects the development process is time. Building a system from scratch clearly takes much more time than building a system from existing components. The more we can reuse existing knowledge and tools, the less time we need to complete the job. The resulting system will be of higher quality, and thus more reliable, durable, and maintainable. The proposed framework aims to encourage reuse of code, design, architecture and in short "knowledge". This provides a more effective development environment and process for applications of the target domain. The similarities in building blocks of these applications make them more compatible structurally, and hence easier to integrate and interact with.

3.2 The Framework Architecture

As mentioned above, CKMF has the following four main components:

- Data Model: DM is the central knowledge repository and acts as knowledge transfer media in CKMF.
- Knowledge Management Database: KMD manages information about concepts and relationships captured in the data model layers.
- Constraints: Constraints are used to guarantee the semantic integrity of the framework components.

- Managing Committee: MC is in charge of defining and enforcing constraints and version management

The framework has also a complementary component:

- Partition: Partition represents a logical grouping of related concepts in the DM.

3.2.1 Data Model (DM)

DM has three layers:

- Core Layer
- Common Layer
- Extension Layer

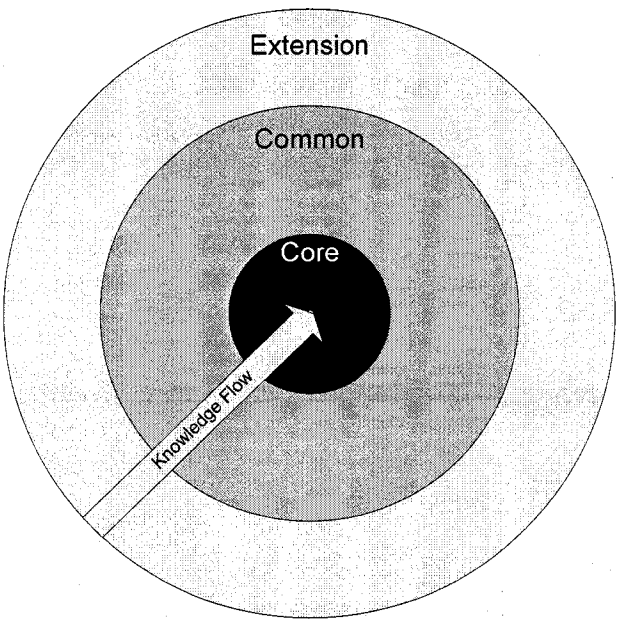

Figure 1: Framework DM Structure

The DM component has the following capabilities:

- Abstraction and classification: To reduce complexity of the problem domain, high level and fundamental concepts (the objects of the target domain) are defined. These objects are then grouped into types (classes) by identifying common characteristics and features (properties), relationships (associations) and behavior (methods).
- Inheritance: Through inheriting fundamental concepts from higher level, additional details can be provided at lower level concepts. A subclass inherits all the information (properties, methods, and associations) defined for its higher level classes. Subclasses are created to put the proper level of details and complexity at the right level in the data model. This can be visualized as a triangle, where the top of the triangle is a fundamental object, and more details and more classes are defined as one moves closer to the base.
- Depiction of dependencies: The DM expresses the semantics of the entities and their relationships and associations. Further semantics description may be provided through properties (specifying common characteristics and features) of the associations and relationships.
- Definition of methods: The ability to define standard behavior (methods) for entities is another form of abstraction. It is useful to define the standard behavior and bundle it with an entity.
- Efficiency and ease of extensibility: Because of the depth and width of the target domain, it is impractical in general to develop a model that covers every aspect of the domain. A protocol is defined for future extensions and the DM is designed in such a way that extensions are considered as part of its natural evolution. Taking

advantage of the architecture and following the protocol, the DM evolves over time and grows in depth and width.

- Verification: The taxonomical structure of the DM can be verified, for which there are different approaches. One of the promising approaches is using an inference engine or a reasoner with existing ontologies in the target domain to verify the taxonomic architecture of the DM.

As shown in Figure 1, the knowledge is created at the Extension layer, and then flows towards inner layers. It then goes through review and refinement process. The refined and consolidated knowledge becomes part of the Common layer. The process of review and refinement process is also applied to the knowledge in the Common layer. This results in refinement and consolidation of knowledge that moves to the Core layer.

3.2.1.1 Core Layer

Software systems change over time and these changes can be the cause of defects and malfunctions in the software system or its related systems. Although many of these changes cannot be avoided, every effort should be made to minimize the number and the scope of these changes. Currently, when a change is made to a software program, most of the time the entire program is reengineered [32]. The reengineering process is time consuming and costly. The concepts of "enduring business themes" and "business objects" have been introduced as a way of minimizing the impacts of future changes on existing software systems. The main idea is to identify aspects of the environment in which the software will operate that will not change and cater the software to these areas. This yields a stable core design for the software system [32].

The Core layer is an information model that captures notions that are applicable to all areas of the target domain. It provides a basic vocabulary that consists of most basic and stable notions in the domain for analyzing and describing concepts and entities in the target domain. The Core layer is a subset of DM, not specific to any application in the domain. Its purpose is providing a conceptual framework for the rest of the model. The Core layer is the main part of the model with regards to verifiability. Every effort should be made to make sure all the definitions in the Core layer are as accurate and concise as possible.

3.2.1.2 Common Layer

The Common layer captures notions that are common to particular application areas in the target domain, but are independent of a particular technology or implementation. The information model at this level is specific enough to provide a basis for the development of applications. It provides a basis for extension into application-specific areas.

3.2.1.3 Extension Layer

The Extension layer consists of extension schemas. Each extension schema represents and models objects that are technology-specific and/or application-specific additions to the Common layer. Any research group or developer group that needs specific extensions to the Common layer can define extension schemas. Extension schemas allow various participants in the framework present their point of view using some already agreed upon concepts found in the Common and Core layers.

3.2.2 Knowledge Management Database (KMD)

KMD is a relational database designed to store physical and logical structural information of the DM in the framework. This database includes information about classes, their data members, their methods, relationships between classes in a release, relationships between classes in two consecutive releases, layers, partitions, applications using the DM, and relationships between these applications. As the DM grows, the number of components in each layer increases, making it too complex to handle manually and in an ad hoc fashion. The purpose of KMD is making the framework manageable, keeping the complexity under control, and making the knowledge it contains more readily accessible and usable. Based on our experience in developing a prototype, KMD turned to be important and required even at a controlled environment of the lab. KMD facilitates the management of the DM and helps with exploring the DM and its elements. A new user can use KMD to get easy access to the structural information of the DM and look up the applications that might be similar to what s/he is about to develop and learn from those applications how to use the model.

3.2.3 Constraints

Constraints are the semantic safe guards of the framework. While they are mostly applicable to DM design and structure, it is also possible to have constraints that are more general. In the beginning, we have a set with few constraints, but as the framework

matures, constraints may be added or modified by MC. Examples of basic constraints are as follows:

- A class should take care of its own error and exception handling. In other words, a class should not propagate any error or exception outside its scope
- A class in DM can be member of one layer only
- A class in DM can only inherit from a class in an immediate inner layer or in the same layer
- No assumption implying a programming language dependency or operating system dependency is allowed

Constraints should be defined accurately so that they are not open to interpretations. They should be designed in such a way that they are either satisfied or violated. There should be no fuzzy constraints unless the nature and foundation of the target domain is fuzzy. The point is that the mathematical basis of an application should only interpret the expressions.

3.2.4 Managing Committee

MC is the governing body of CKMF and is responsible for maintaining its integrity through defining and enforcing constraints. MC is also in charge of re-factoring process, and decides about the time and properties of the next version of CKMF DM. Being in charge of re-factoring process means that MC is the final authority in knowledge refining task, which considers different view points of participants contributed to extension schemas and tries to facilitate convergence. MC reviews extension schemas, identifies

commonalities, and decides which concepts (presented in extension schemas) are solid and coherent enough to move to the Common layer in the next version. MC is also responsible to review the feedbacks received from participants, and look at the usage patterns and approve or disapprove suggested modifications. For any change in the Core or Common layers, MC is responsible to analyze the consequences of those changes and their possible impacts on the applications. Possible types of modifications are: modifying a concept or a relationship between concepts, moving a concept from the Common layer to the Core Layer or vice versa, or depreciating a concept or a relationship between concepts.

MC should include technical and administrative stakeholders of the framework. As previously mentioned, like other KWs, Software Engineers desire autonomy and avoid explicit coordination and control to some extent. MC as a means of control and coordination involves technical staff as well as administrative and managerial personnel of participating organizations. This involvement in decision making alleviates the problem of avoiding explicit coordination and control and addresses the desire of autonomy for technical staff. It also resonates with management because they get to retain some level of decision making powers. The composition of MC and its internal rules should be decided according to the specifics of the target domain and participating organizations.

3.2.5 Partition

A partition is a logical grouping of related concepts in DM. Partitions can represent groups of concepts used in specific solutions or related to certain applications. Partition is

the closest concept in our framework to the design pattern concept in Software Engineering. Partitions have four properties: name, problem description, solution description, and consequences and trade-offs. A concept can belong to more than one partition. Figure 2 illustrates examples of partitions in DM.

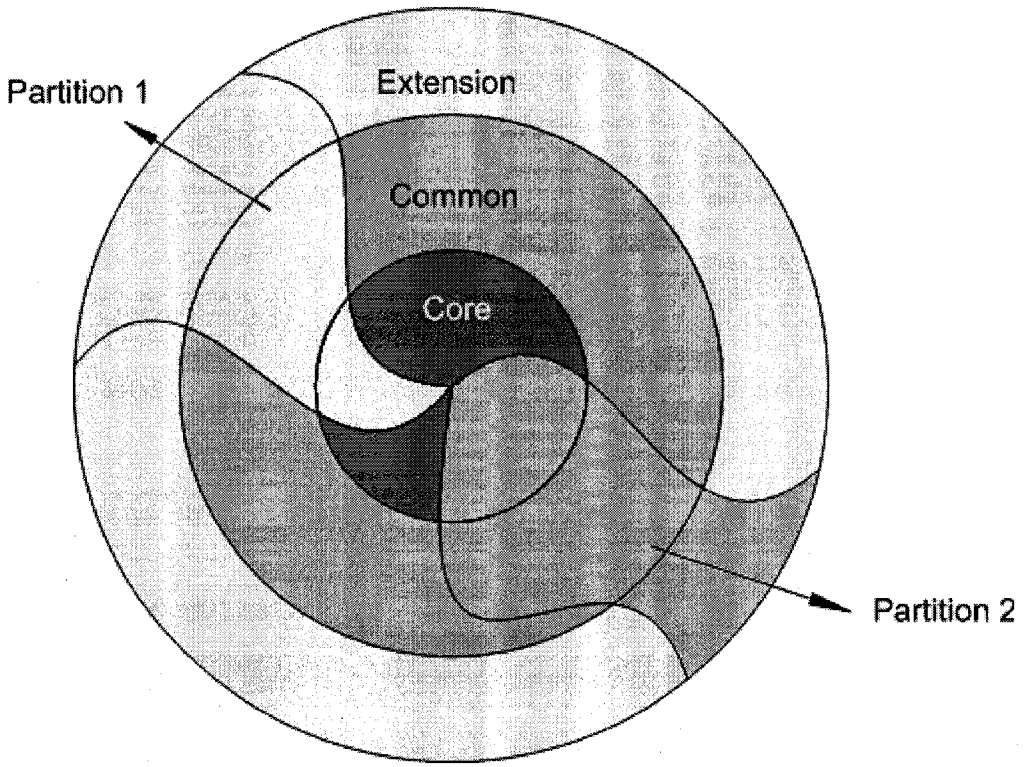

Figure 2: Partitions in DM

Partitions break the complexity of the DM and help users search for what they need and understand it faster.

3.3 Knowledge Acquisition Process and Version Management in CKMF

The knowledge acquisition process in the framework is designed to be open and iterative. DM of the framework has a layered structure to encourage higher levels of contribution from various groups in such a way that their different points of view and diversity are preserved. Although diversity and different points of view in contribution phase are allowed and welcomed to encourage maximum input from participants, the ultimate goal of CKMF is bringing different points of view together and creating a consensus. To realize this goal, we need to:

- Clearly identify components and their versions in DM
- Control change by establishing strict procedures for performing a change
- Monitor status through recording and reporting the status of components and change requests
- Audit and review quality assurance functions to preserve integrity and consistency of the framework

MC is responsible for identifying the objects that should be monitored by the version management process. MC also has to provide procedures to follow when a change is being performed. The change requests can be related to the current version or can apply to future versions. Any request to perform a change in the current version should be a response to a critical problem affecting integrity and consistency of the various components of the framework. Any request that is not critical will be considered for future versions. MC is in charge of receiving, classifying, and processing of change

requests. MC is also responsible for defining quality assurance procedures to be applied to all changes in order to ensure integrity and consistency of CKMF.

3.3.1 CKMF Initialization

CKMF initialization is a prerequisite for knowledge acquisition process in the framework. The first task of initialization is creation of MC. As previously mentioned in section 3.2.4., MC should include technical and administrative stakeholders of the framework. Technical members of MC in form of subcommittees are in charge of domain analysis which is aimed at identifying the domain's requirements and possible future requirements of SEs in the field. The products of this effort are used to define the first set of concepts in framework's DM. For each concept, a set of attributes and operations are identified and defined. Based on CKMF requirements for a layered DM structure, the concepts are to be categorized into members of Core and Common layers. The concepts that are more abstract and solid are placed in Core layer and the rest are assigned to Common layer. KMD is populated with the physical and logical structural information of these concepts. To complete the initialization of CKMF, MC needs to define details of management and operational processes that will govern the use and evolution of CKMF. After initialization, the concepts that are defined in DM constitute version 1 of DM in CKMF. This version is released to stakeholders and users of CKMF.

3.3.2 Knowledge Acquisition Process

Knowledge acquisition process starts every time version x of DM is released. SEs of participating organizations take advantage of KMD to find concepts that are of interest to them and use those concepts either "as is" or extend or customize them. Those who extend or customize concepts are encouraged to define their extended or customized concepts as part of Extension layer in DM. The physical and logical structural information of the concepts defined in Extension layer are recorded in KMD. Information about applications that use concepts from Core, Common, and Extension layers are also recorded in KMD. During this period, MC is ready to receive reports of possible problems in definition of concepts, their attributes, and/or their operations, proposals to add concepts to Core or Common layers, suggestions about Constraints and etc.

After version x of DM is used by participating organizations for a while, different view points and contributions of these organizations are mainly recorded in form of concepts that are defined in Extension layer. One of the main responsibilities of MC is to facilitate convergence of these versatile viewpoints. For achieving this goal, version x of DM should go through re-factoring process. As mentioned in section 3.2.4, MC is the entity in charge of re-factoring process. MC reviews concepts defined in Extension layer, identifies commonalities, and decides which concepts are solid and coherent enough to move to the Common layer in the next version of DM. In some cases this review might trigger a change in an already existing concept in Common or Core layers. If a change is to happen affecting an existing concept in Core or Common layers, MC analyzes the consequences of those changes and their possible impacts on the existing applications. Possible types of modifications affecting existing concepts are:

- Modifying an exiting concept (in Core or Common layers)
- Modifying a relationship between two existing concepts (in Core or Common layers)
- Moving an existing concept from Common layer to Core Layer or vice versa
- Depreciating an existing concept (in Core or Common layers)
- Eliminating a relationship between existing concepts (in Core or Common layers)

The "sum" of all these changes may result in creation and release of version x+1 of DM. This new version will be released for use in the field and will go through the same cycle. Each time this cycle happens, the concepts are refined and re-categorized. The knowledge that is captured in the design of these concepts becomes more solid and more refined each time.

Chapter 4

Technical Design

In this chapter, we describe details of the technical design of CKMF components. Among the components, the design of MC is domain specific. The Constraints are also defined and enforced by MC according to the target domain requirements. The DM will be shaped based on the structure and type of the knowledge that will be captured from the target domain. The KMD manages physical and logical structural information of the DM. We have developed an implementation model to address the details of how DM could be transformed into actual executable code in order to fully realize the benefits of knowledge captured in DM. We have used the Service Oriented Architecture (SOA) to design our implementation model. We describe our implementation model in this chapter.

4.1 DM Implementation Model

The implementation of DM is recommended to be as portable as possible and independent of operating system and programming language. We can have such an implementation of the DM model based on SOA. Simply put, SOA is an architectural approach in which functionalities and features of a software system or a group of software systems are constructed and delivered as services to either end-user applications or other services [52]. Clearly, this is a preliminary system prototype illustrating ideas.

More detailed and elaborate work is required for mapping layered conceptual architecture to a SOA based deployed architecture. This mapping in our prototype is done manually, for illustrative purposes.

Design of SOA systems involves description of atomic and composite services and orchestration of those services to form a distributed application [36]. Figure 3 demonstrates a basic SOA.

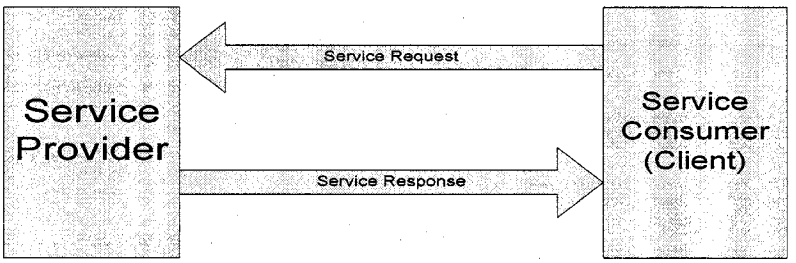

Figure 3: Basic Service Oriented Architecture

There are three main components in a basic SOA architecture: service provider, service client, and service registry. Service providers publish descriptions in service registries that are subsequently discovered by service clients. Service clients utilize these descriptions to invoke the service hosted by the service provider. Figure 4 displays the components of a basic SOA and their relationships.

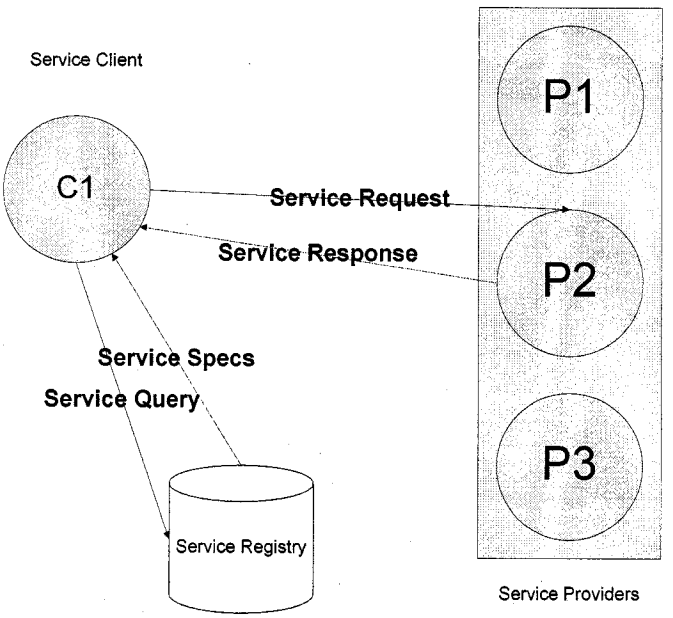

Figure 4: Components of Service Oriented Architecture

4.2 Service provider

The service provider should be capable of handling the full life cycle of classes and objects defined in the DM of the framework. The service provider will receive requests to:

- Define a Class
- Make a Class obsolete
- Instantiate an Object
- Delete an Object
- Make an Object persistent
- Access an Object data member with respect to the access levels defined in class definition

- Invoke an Object method with respect to the access levels defined in class definition

The service provider is responsible for processing these requests. All objects created by the service provider are assigned a unique ID and any subsequent call to that object is possible through this ID. Messages sent to the service provider and the responses sent back from the provider can take different shapes. The preferred format of the messages would be XML.

We have two service providers in the implementation model of the framework. The first one is the Core Service Provider, used to materialize the classes defined in the Core layer of DM and their corresponding objects. The second one is the Common Service Provider that will handle classes defined in the Common layer of DM and their corresponding objects.

4.3 Service Client

Any entity that uses the services exposed by a service provider is called a service client. An entity could be a service client as well as a service provider. Common Service Provider is also a client of Core Service Provider, and acts as a pass through for requests sent to Core Service Provider. This feature could be disabled if direct access to classes defined in the Core layer is not intended for other service clients such as the Extension layer and external applications.

4.4 Service Registry

Service registry is a data repository that manages the data about service providers and the services they offer. If a service registry has information on a service that matches a client's criteria, the registry provides a contract and an endpoint address. Service registry is a basis for service cataloging and classification and it represents a unified environment for publishing and discovering services [4].

KMD in CKMF can function as a service registry. We have designed KMD as a relational database. The design details of KMD are presented in this section.

4.4.1 High Level ERD of the KMD

Figure 5 is the high level entity-relationship diagram (ERD) of the KMD. Figures 6, 7, and 8 display more details about more important relations in the KMD.

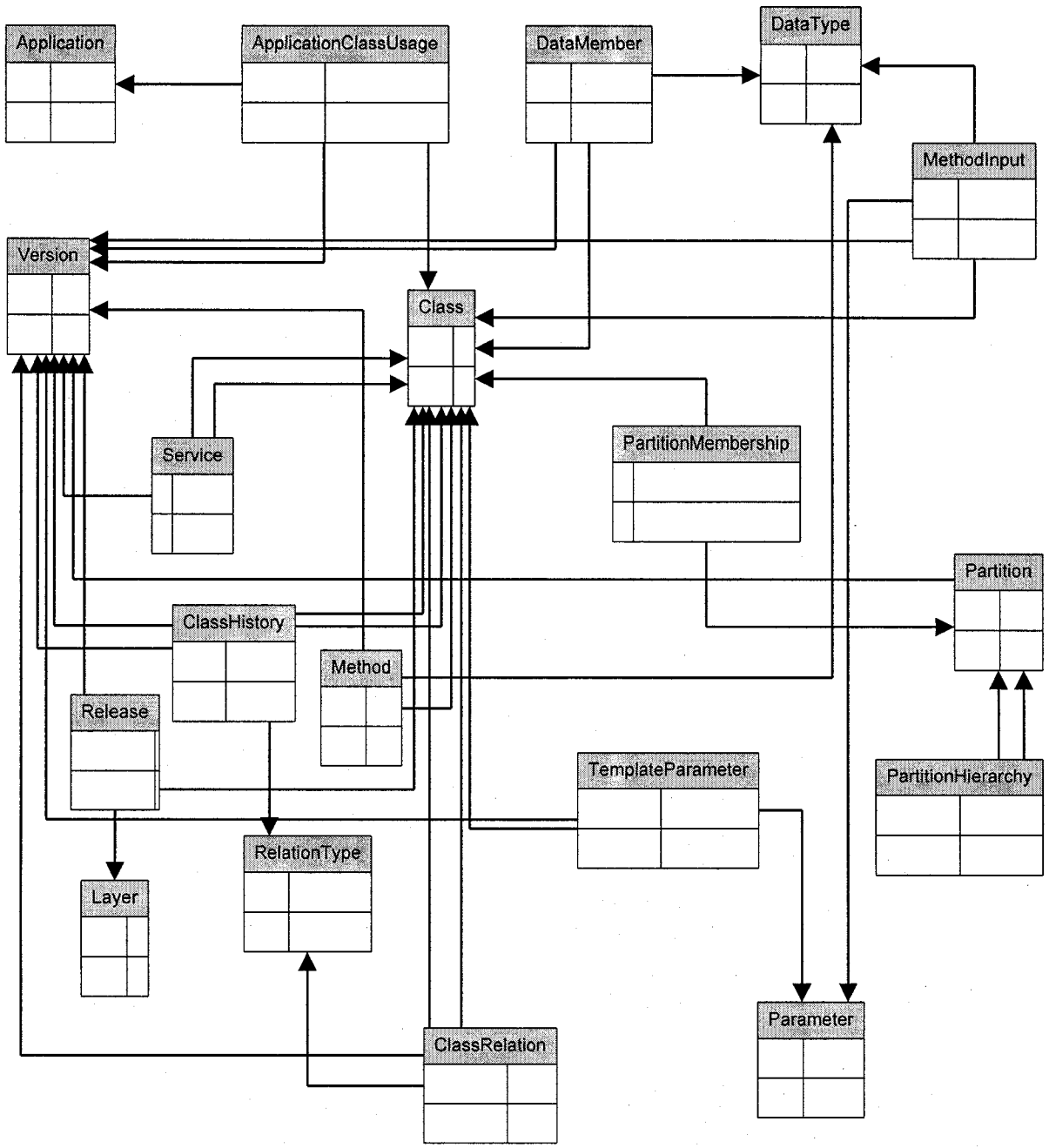

Figure 5: High Level Entity-Relationship Diagram of KMD

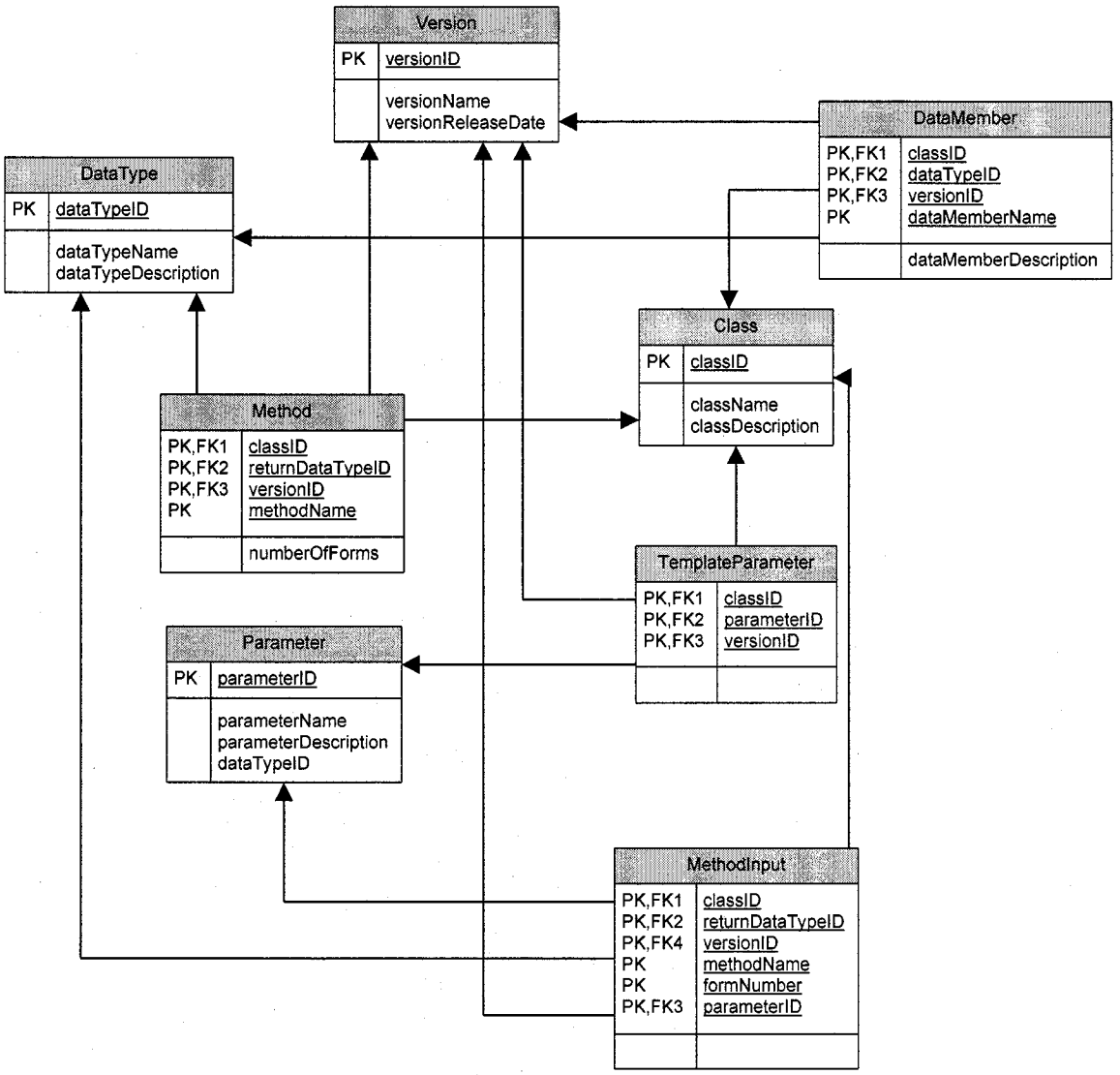

Figure 6: Relation Class and its relationships, Part 1

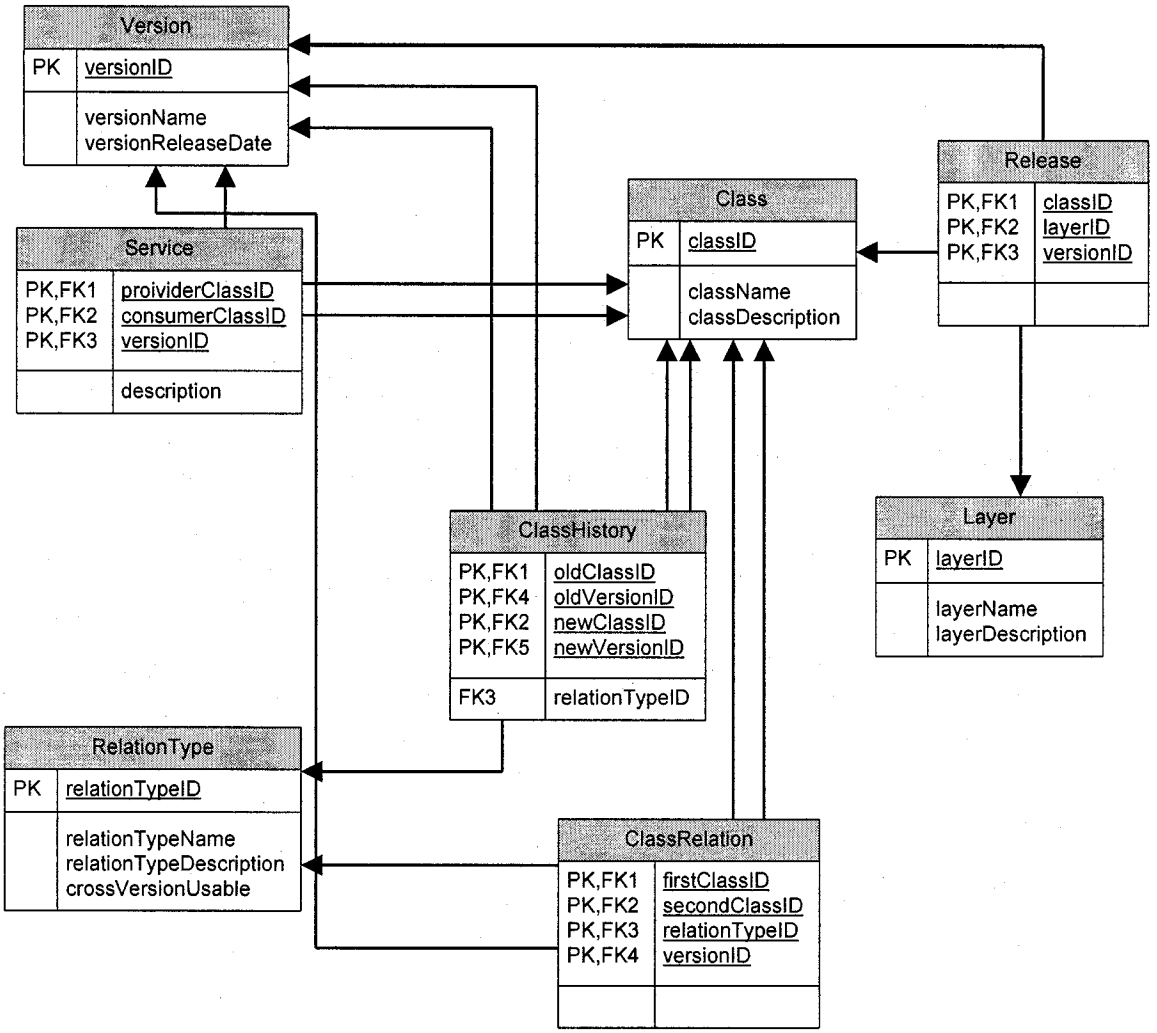

Figure 7: Relation Class and its relationships, Part 2

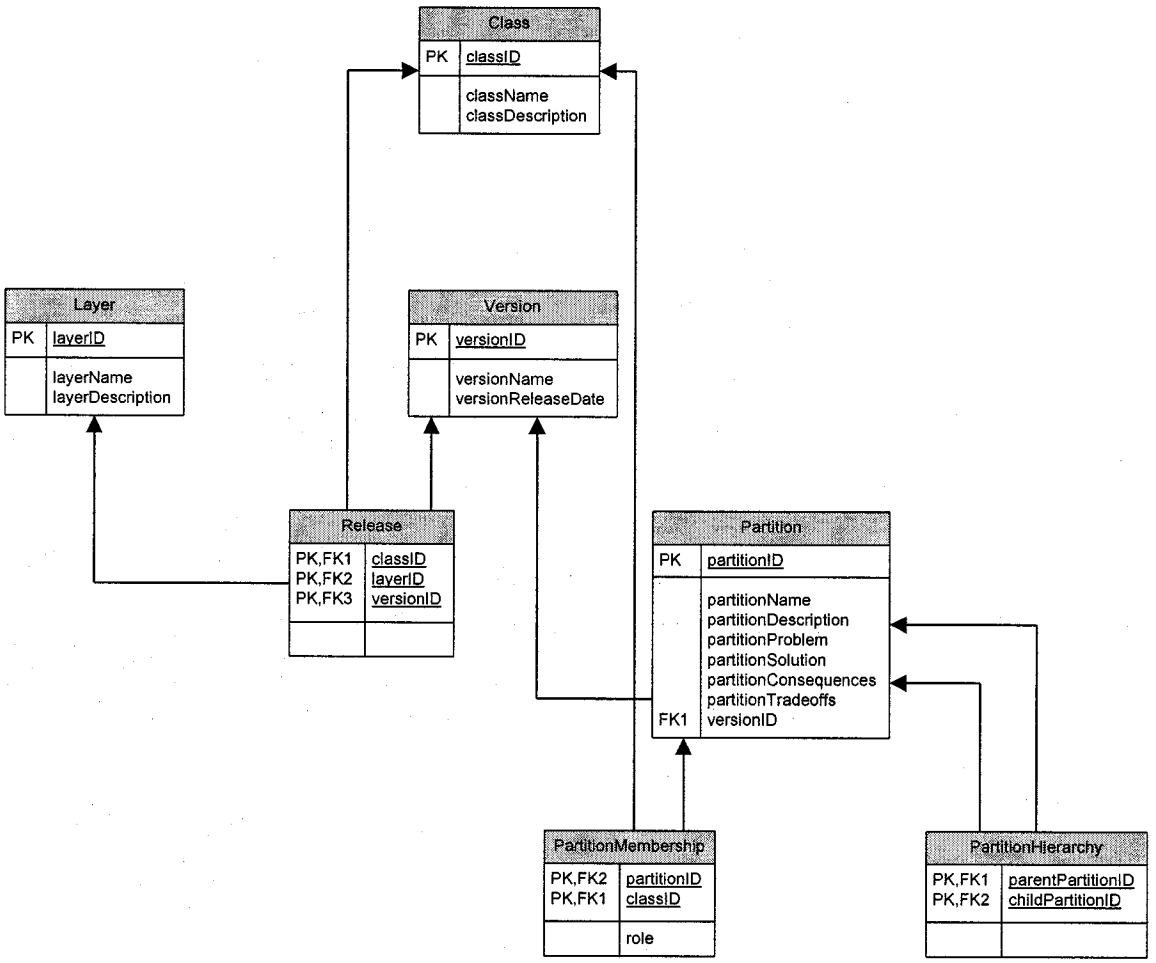

Figure 8: Relation Layer, relation Partition, and their relationships

4.4.2 Tables of the KMD

The KMD has 19 relations shown in the high level ERD (Figure 5). In this section, we describe the purpose of each relation and its structure. The relations are introduced in the order of importance of their role. The attributes that are marked with "*" are part of the primary key of the relation.

4.4.2.1 Relation Class

Class represents a class of objects in the target domain. Class is the template or blueprint for a set of objects in the target domain that share similar properties and behavior.

Attribute Name	Attribute Data Type	Attribute Size
classID*	Long	4
className	Text	255
classDescription	Memo	-

Table 1: Relation Class

4.4.2.2 Relation DataType

DataType represents a data type in the target domain. Since we do not know what kind of data in the target domain we will be dealing with, we intentionally want to avoid defining a set of data types and thus restricting ourselves. This relation enables the users of the framework to define proper data types based on the specifications of the target domain.

Attribute Name	Attribute Data Type	Attribute Size
dataTypeID*	Long	4
dataTypeName	Text	255
dataTypeDescription	Memo	-

Table 2: Relation DataType

4.4.2.3 Relation Version

Version models the concept of version required for management of DM evolution. An instance of version has a name and a release date. The MC decides when a new version is needed and what it should be named.

Attribute Name	Attribute Data Type	Attribute Size
versionID*	Long	4
versionName	Text	255
versionReleaseDate	Time	8

Table 3: Relation Version

4.4.2.4 Relation DataMember

DataMember represents data members or properties of classes in the target domain. Each class has a number of properties or data members. A data member belongs to a class in a specific version of DM and has a data type, a name, a description. In other words, classes can have different data members in different versions of DM.

Attribute Name	Attribute Data Type	Attribute Size
classID*	Long	4
dataTypeID*	Long	4
versionID*	Long	4
dataMemberName*	Text	255
dataMemberDescription	Memo	-

Table 4: Relation DataMember

Attribute Name	Reference Table
classID	Class
dataTypeID	DataType
Versioned	Version

Table 5: Foreign Keys of DataMember

4.4.2.5 Relation Method

Methods model the behavioral aspects of a class in the target domain. A Method has a name, a return data type, one or more than one form (for polymorphic methods), and belongs to a class in a specific version of DM. In other words, classes can have different methods in different versions of DM.

Attribute Name	Attribute Data Type	Attribute Size
classID*	Long	4
returnDataTypeID*	Long	4
versionID*	Long	4
methodName*	Text	255
numberOfForms	Integer	2

Table 6: Relation Method

Attribute Name	Reference Table
classID	Class

returnDataTypeID	DataType
Versioned	Version

Table 7: Foreign Keys of Method

4.4.2.6 Relation Parameter

Parameter models a place holder, which has a name, a description, and a data type. Parameters are used in any operation or transaction that there is a need for a placeholder.

Attribute Name	Attribute Data Type	Attribute Size
parameterID*	Long	4
parameterName	Text	255
parameterDescription	Memo	-
dataTypeID	Long	4

Table 8: Relation Parameter

Attribute Name	Reference Table
dataTypeID	DataType

Table 9: Foreign Key of Parameter

4.4.2.7 Relation MethodInput

A method can have more than one shape. Each shape of a method has a number of input parameters. MethodInput models input parameters of specific shape of a method.

Attribute Name	Attribute Data Type	Attribute Size

classID*	Long	4
returnDataTypeID*	Long	4
versionID*	Long	4
methodName*	Text	255
formNumber*	Integer	2
parameterID*	Long	4

Table 10: Relation MethodInput

Attribute Name	Reference Table
classID	Class
dataTypeID	DataType
versioned	Version
parameterID	Parameter

Table 11: Foreign Keys of MethodInput

4.4.2.8 Relation TemplateParameter

Some classes could be better defined using parameters. These kinds of classes are template classes. A template class needs some values for its parameters to actually become a fully functional class.

Attribute Name	Attribute Data Type	Attribute Size
classID*	Long	4
parameterID*	Long	4

versionID*	Long	4

Table 12: Relation TemplateParameter

Attribute Name	Reference Table
classID	Class
parameterID	Parameter
Versioned	Version

Table 13: Foreign Keys of TemplateParameter

4.4.2.9 Relation Layer

Layer models a layer in the DM of the framework. A layer is a grouping of classes in DM. By default, there are three layers: the Core, the Common, and the Extension in DM. A class can only belong to one layer in each version of DM.

Attribute Name	Attribute Data Type	Attribute Size
layerID*	Long	4
layerName	Text	255
layerDescription	Memo	-

Table 14: Relation Layer

4.4.2.10 Relation Release

Release represents the relationship among classes, layers, and versions. In other words, release specifies which classes have been member of which layers in a version. If a class

is not member of any layers in a version, that class is not present in the DM of that version.

Attribute Name	Attribute Data Type	Attribute Size
classID*	Long	4
layereID*	Long	4
versionID*	Long	4

Table 15: Relation Release

Attribute Name	Reference Table
classID	Class
layered	Layer
versioned	Version

Table 16: Foreign Keys of Release

4.4.2.11 Relation Service

Service models the provider/consumer relationship among classes in a version of DM. Classes rely on other classes to provide some type of service. The class that provides the service is the provider and the class that receives the service is the consumer. Keeping track of these dependencies is necessary for change and maintenance purposes.

Attribute Name	Attribute Data Type	Attribute Size
providerClassID*	Long	4
consumerClassID*	Long	4

versionID*	Long	4
Description	Memo	-

Table 17: Relation Service

Attribute Name	Reference Table
providerClassID	Class
consumerClassID	Class
Versioned	Version

Table 18: Foreign Keys of Service

4.4.2.12 Relation RelationType

RelationType lists the possible types of relations two classes can have. A relation type has a name, a description, and a switch that indicates that relation type is applicable across versions of DM.

Attribute Name	Attribute Data Type	Attribute Size
relationTypeID*	Long	4
relationTypeName	Text	255
relationTypeDescription	Memo	-
crossVersionUsable	Boolean	1

Table 19: Relation RelationType

4.4.2.13 Relation ClassRelation

ClassRelation models the relationship between classes in each version of DM. A record in ClassRelation includes the ids of two related classes, the relationship type, and the version of DM in which this relation exists. The type of the relationship should be selected from those types that are applicable to the same version of DM. In other words, the cross version relationship types are not usable in this table.

Attribute Name	Attribute Data Type	Attribute Size
firstClassID*	Long	4
secondClassID*	Long	4
relationTypeID*	Long	4
versionID*	Long	4

Table 20: Relation ClassRelation

Attribute Name	Reference Table
firstClassID	Class
secondClassID	Class
relationTypeID	RelationType
versioned	Version

Table 21: Foreign Keys of ClassRelation

4.4.2.14 Relation ClassHistory

ClassHistory models the relationship between classes in consecutive versions of DM. A record in ClassHistory includes the ids of two related classes, the relationship type, and

the versions of DM in which this relation exists. The type of the relationship should be selected from those types that are applicable to different versions of DM. In other words, only the cross version relationship types are usable in this table.

Attribute Name	Attribute Data Type	Attribute Size
oldClassID*	Long	4
oldVersionID*	Long	4
newClassID*	Long	4
newVersionID*	Long	4
relationTypeID*	Long	4

Table 22: Relation ClassHistory

Attribute Name	Reference Table
oldClassID	Class
oldVersionID	Version
newClassID	Class
newVersionID	Version
relationTypeID	RelationType

Table 23: Foreign Keys of ClassHistory

4.4.2.15 Relation Partition

Partition represents a logical grouping a concepts and their relationships in a specific version of DM. A partition has a name, a description, is related to a problem, and introduces a solution which implies some consequences and trade-offs. Partitions are

defined within confines of a version of DM. Partition is similar to the concept of design patterns in Software Engineering.

Attribute Name	Attribute Data Type	Attribute Size
partitionID*	Long	4
partitionName	Text	255
partitionDescription	Memo	-
partitionProblem	Memo	-
partitionSolution	Memo	-
partitionConsequences	Memo	-
partitionTradeoffs	Memo	-
Versioned	Long	4

Table 24: Relation Partition

Attribute Name	Reference Table
Versioned	Version

Table 25: Foreign Key of Partition

4.4.2.16 Relation PartitionHierarchy

PartitionHierarchy models the parent/child relationship that might exits among partitions. A partition could contain other partitions and this relationship is modeled in PartitionHierarchy table. Partition hierarchy is close to the concept of design pattern language in Software Engineering. The parent/child relationship can only exist between partitions that are defined in the same version of DM.

Attribute Name	Attribute Data Type	Attribute Size
parentPartitionID*	Long	4
childPartitionID*	Long	4

Table 26: Relation PartitionHierarchy

Attribute Name	Reference Table
parentPartitionID	Partition
childPartitionID	Partition

Table 27: Foreign Keys of PartitionHierarchy

4.4.2.17 Relation PartitionMembership

PartitionMembership represents which classes are part of which partitions. Each record in PartitionMemebrship has ids of the partition and the class which is a member of that partition, and a description of the role the class in the partition. Since the partitions are defined in the confines of a version of DM, the classes that are member of a partition, should also be part of that version. In other words, a class should have been released in a version of DM to be eligible for membership of partitions defined in that version.

Attribute Name	Attribute Data Type	Attribute Size
partitionID*	Long	4
classID*	Long	4
Role	Memo	-

Table 28: Relation PartitionMembership

Attribute Name	Reference Table
Partitioned	Partition
classID	Class

Table 29: Foreign Keys of PartitionMembership

4.4.2.18 Relation Application

Application represents the applications that use DM of CKMF. Each application is introduced by a name, a description, a list of authors, a deployment date, contact information, and a URL.

Attribute Name	Attribute Data Type	Attribute Size
applicationID *	Long	4
applicationName	Text	255
applicationDescription	Memo	-
applicationAuthors	Memo	-
deploymentDate	Date/Time	8
contactInfo	Memo	-
URL	Text	255

Table 30: Relation Application

4.4.2.19 Relation ApplicationClassUsage

ApplicationClassUsage shows which application is using which class in what version of DM. This table is a useful in helping new users find sample applications and also is useful for MC in assessing the impact of any changes in DM.

Attribute Name	Attribute Data Type	Attribute Size
applicationID *	Long	4
classID *	Long	4
versionID *	Long	4

Table 31: Relation ApplicationClassUsage

Attribute Name	Reference Table
applicationID	Application
classID	Class
Versioned	Version

Table 32: Foreign Keys of ApplicationClassUsage

Chapter 5

Prototype

To illustrate how CKMF works and to show how it facilitates development of applications in a target domain, we built a prototype in the domain of Bioinformatics. We started by looking at various bio-applications and tools used to develop such applications. Examples of systems and tools we looked at include: GenBank® and DNA Data Bank of Japan, both from the International Nucleotide Sequence Databases [INSD], the Basic Local Alignment Search Tool (BLAST), UniProt Knowledgebase, created by merging the data in Swiss-Prot, TrEMBL and PIR-PSD, BioPerl, and NCBI C++ Toolkit. This activity helped us select a set of concepts as initial candidates to develop Core and Common layers of the DM component in our CKMF. We should emphasize that these concepts were chosen as initial candidates for the implementation of the prototype and by no means are considered finalized or optimal or even suitable choices for an instantiation of the framework in an industrial application. The candidate concepts in our prototype were implemented as classes in standard C++ to make sure we can provide and guarantee the OS portability feature of the DM, as part of a portable CKMF. Next we developed the KMD component. Even with a limited set of concepts and very limited number of contributors, we had to deal with huge amounts of information and high levels of complexity. We were not able to manage this complexity without the KMD of CKMF.

Because of KMD's essential role, it was important for us to work with KMD efficiently and easily. To this end, we developed a user-friendly front end for KMD.

After we built DM and KMD, we developed three sample applications that used the concepts captured in the DM to perform simple tasks. Although very simple in nature, these applications were chosen in a way to represent the building blocks of more complex real life applications. We used standard C++ to develop these applications. The development of sample applications helped refine our understanding of biological concepts and make them more usable in CKMF.

In this exercise we did not have a formal MC. The MC component of CKMF will be needed in any instantiation of the framework.

5.1 Choice of Programming Language and OS

According to the implementation model, service providers in the framework implementation do not have any specific programming language or OS requirements or restrictions. Considering the framework characteristics, service providers should be implemented in a way that they are programming language and OS portable as much as possible.

To satisfy portability requirement, we considered C++ and Java programming languages for prototype implementation. Both of these languages are considered to be portable, and hence suitable for implementing the prototype considering the functionality expected of service providers.

There are many studies that compare C++ and Java programming languages from different perspectives [59, 72, 77]. In most cases, Java is considered superior, for the following reasons [72]:

- A C++ program will have three times as many bugs as a comparable Java program.
- C++ has two to three times as many bugs per hour, noting that Java and C++ are similar syntactically.
- The time it takes to fix a bug in C++ is twice it takes in Java. Combined with the (almost) triple bug rate, debugging a C++ application takes approximately six times it takes to debug a Java application.

The above factors are not the only deciding factors. There is a long and ongoing debate [98, 84, 62, 41, 82] about the need for multiple inheritance support in an object-oriented programming language. The availability of multiple inheritance is important for expressing relationships among classes that belong to complex and rich classification hierarchies. There are two situations that call for multiple inheritance specifically. They are "multiple classification" and "dynamic classification" [82]. Multiple classification corresponds to cases where an abstraction is a specialization of more than one other abstraction. Dynamic classification is needed when an abstraction participates in different specialization relationships at different phases of its lifetime.

Since C++ supports multiple inheritance, a class can inherit from more than one class. Java does not support multiple inheritance and therefore a class can inherit from one class only but can implement more than one interface. An interface within an object-oriented programming language is a type, just as a class is a type. Like a class, an interface defines

methods. Unlike a class, an interface cannot have data members and never implements its methods; instead, classes that implement an interface, implement the methods defined by the interface [36].

We chose to implement Core service provider and Common service provider in standard C++ programming language on Windows OS. The choice of C++ programming language was made because of its expressive power for supporting multiple inheritance.

5.2 Core Service Provider

In our prototype, Core service provider is implemented as a dynamic link library (DLL) on Windows platform. Figure 9 displays a simplified class diagram of the Core layer in the prototype:

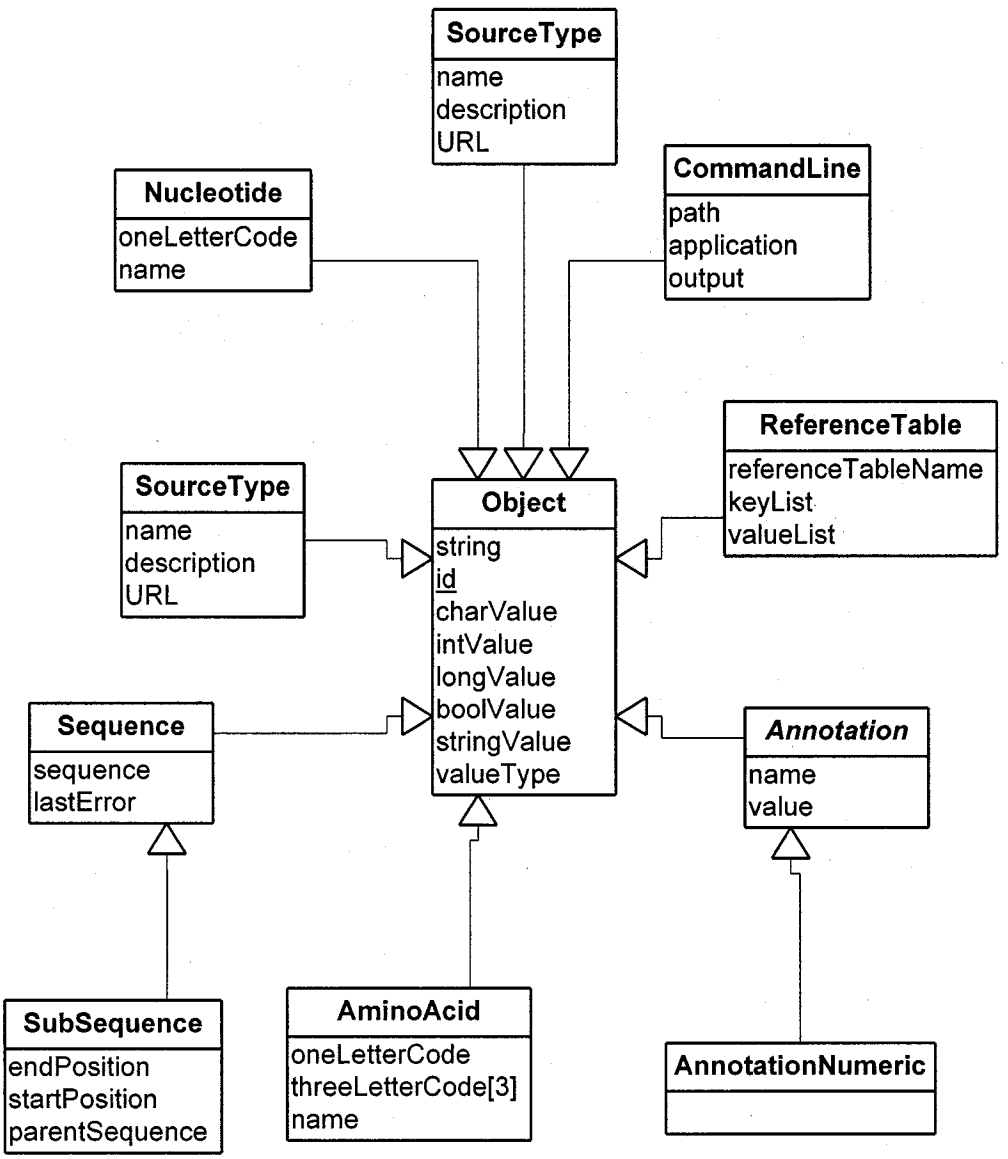

Figure 9: Simplified Class Diagram of Core Layer of the Prototype

For a more detailed diagram and definition of classes mentioned in Figure 9 please refer to Appendix E.

5.3 Common Service Provider

Common service provider is implemented as a DLL on Windows platform. Common service provider relies on Core service provider DLL and uses the services it offers to implement its functionality. Figures 10 and 11 display simplified and partial class diagrams of the Common layer of the prototype. The classes that are marked as "external", are the ones that are defined in the Core layer.

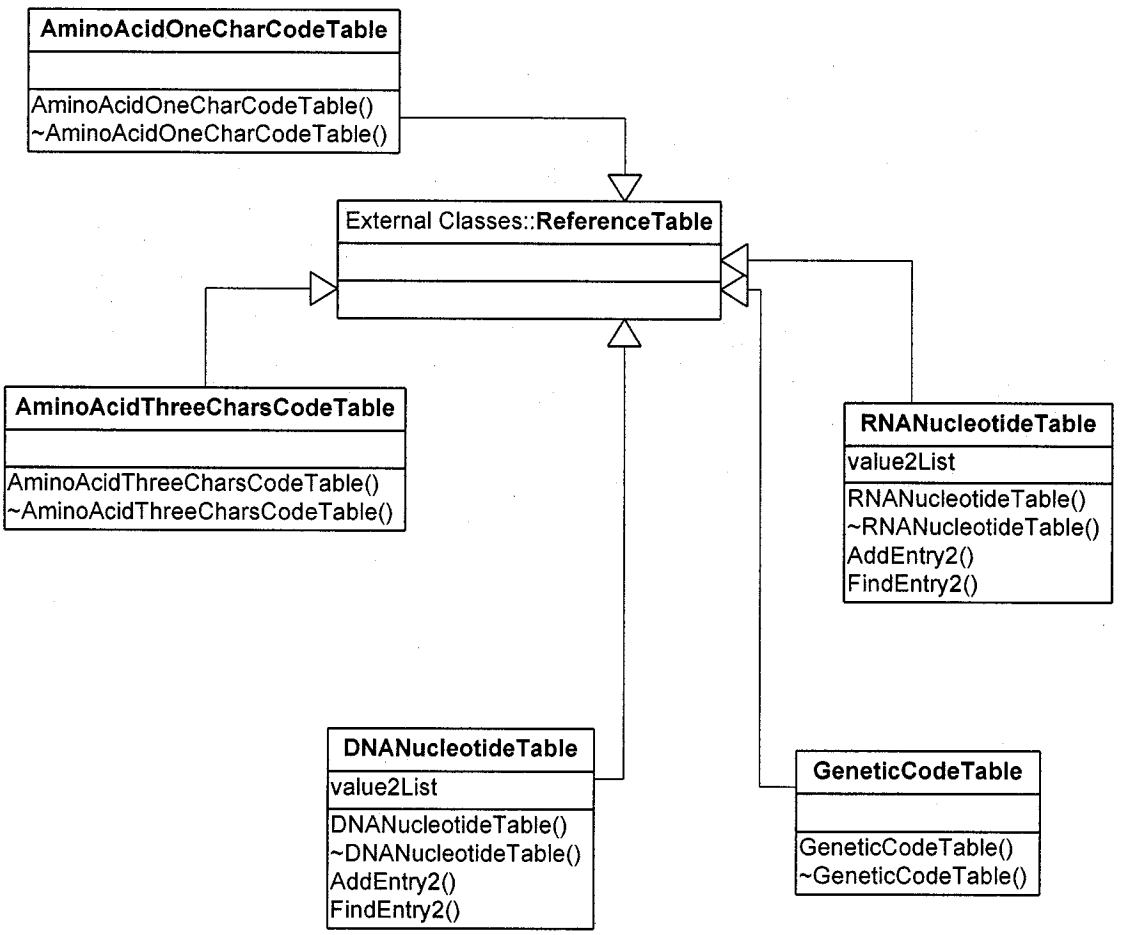

Figure 10: Simplified and Partial Class Diagram of Common Layer of the Prototype, Part 1

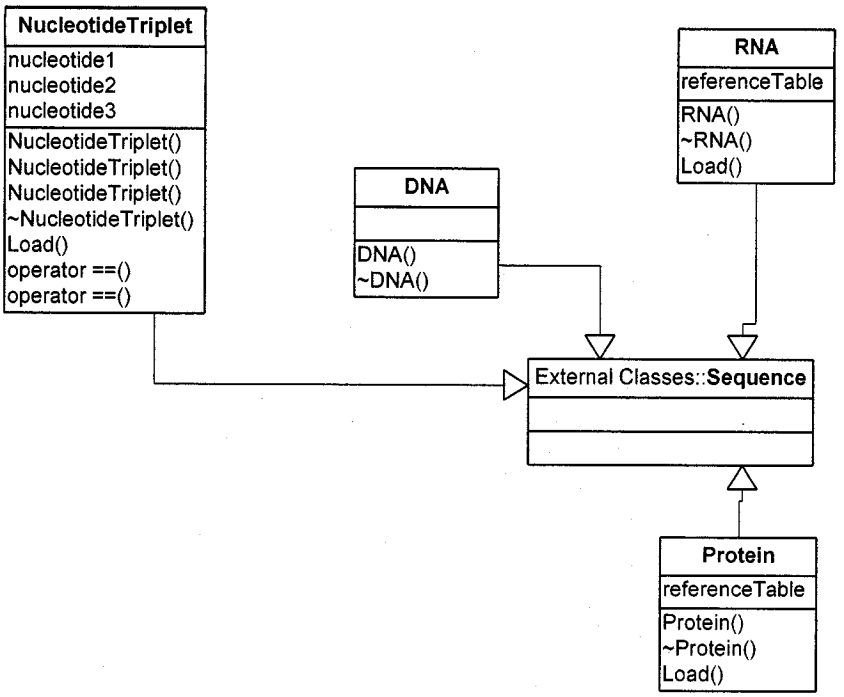

Figure 11: Simplified and Partial Class Diagram of Common Layer of the Prototype, Part 2

5.4 KMD as Service Registry

We developed the KMD component of CKMF as a service registry. Even with the limited set of concepts and limited number of contributors involved in the prototype, we had to deal with too much information and a high level of complexity. Without KMD, it was impossible for us to manage this complexity. Considering the fact that this implementation was just a prototype with just a few contributors, the role of KMD became more apparent in our experience in building the running prototype. We found that KMD should be one of main focus points of any instantiation of the framework. Because of its essential role, it was important for us to work with KMD efficiently and easily. To this end, we developed an application as a user-friendly front end for KMD. This application was developed using VB.NET in .NET environment.

5.5 Sample Applications

The first sample application purpose was loading a sequence (which could be a protein or a gene) from a file and converting it into internal representation, and then writing the internal representation to a file. The purpose of the second application was to do a simple sequence alignment and develop a match score. The third application was designed to a do a BLAST query.

Chapter 6

Assessment of the Framework

To illustrate the applicability of the proposed ideas, we consider bioinformatics as the target domain, and describe a CKMF in this context. We will then evaluate the framework and compare it with existing software tools and techniques in bioinformatics. As evaluation metrics, we consider portability, adaptability, understandability, reliability, and maintainability, to evaluate and compare CKMF. While the tools, libraries, and programming languages considered in this evaluation and comparison are not of the same type, they are important and widely used in the development of bioinformatics applications. We use them in our comparison because we believe they are suitable indicators of tools available to Software Engineers in this target domain and represent the behavior and characteristics of these tools and techniques.

Existing Software Engineering tools, libraries, and programming languages specifically designed or customized for development of bioinformatics applications we will use in our evaluation are as follows:

- BioPerl is a set of Perl modules providing a specialized bioinformatics toolkit for format conversion, data manipulation, sequence analysis, and etc. [86, 104]. This tool is an extension of Perl and is managed as an open source project.
- NCBI C++ Toolkit provides a set of free, portable, public domain libraries with no use restrictions [107]. This toolkit offers various functionalities such as

networking and interprocess communication, multithreading, sequence alignment, and BLAST query and communication.

- BioJava is an open-source project aimed at providing a Java framework for processing biological data [103]. It provides support for sequence manipulation, file parsing, and access to BioSQL and Ensembl database.
- BioPython is a set of freely available Python tools for computational molecular biology [105], which is a collaborative and distributed project to develop Python libraries for bioinformatics applications.
- OBDA/BioSQL is a module for accessing sequence data in a database. OBDA stands for Open Biological Database Access and has been implemented in BioJava, BioPerl, BioPython, and BioRuby [108].

6.1 Portability

Portability is defined as the ease with which software can be transferred from one computer system or environment to another [15]. Portability can also be defined as the ability of a software unit to be ported (to a given environment). A program is portable if the cost of porting is less than the cost of redevelopment [63]. Some factors that affect portability are: language dependency, operating system dependency, and environmental assumptions.

The life of a software unit can be affected by some initial assumptions. If a system analyst or designer is looking at a specific problem and s/he has some assumptions about the available implementation tools and environment, this might affect the way s/he looks

at the problem, analyzes it, and provides a solution. Making assumptions at the analysis phase and the design time is inevitable. The assumptions made by a system designer/developer provide the foundation of his/her understanding and affects every step s/he takes. To make sure a piece of software is portable, the number of assumptions also matters. At the analysis phase, it is important to have fewer assumptions. This leads to fewer restrictions at the design time. At the design phase, it is important to avoid any language or operating system dependencies. If the product of the design phase is independent of the language and operating system of the implementation platform, that product can be ported more easily to various environments and platforms.

CKMF has been designed with the explicit goal of language and operating system independence. We have made no assumptions that lead to a specific programming language or an operating system dependency. This is one of the principles of the framework. In other words, making such assumptions in the design and/or extension of the framework is not allowed at any time.

BioPerl provides some degree of operating system independence but its existence is based on Perl. This means any system using BioPerl will be dependent on Perl. Such a system will be bound by strengths and weaknesses of Perl. The same problem holds for BioJava, BioPython, and NCBI C++ Toolkit. ODBA/BioSQL is different. There are at least three different implementations of ODBA/BioSQL, hence providing options and some degree of freedom. The other strength for OBDA/BioSQL, from the portability point of view, is the possibility of implementing it in other languages since its specification is independent from programming languages and operating systems.

6.2 Adaptability

The rapid changes in technology, environment and user requirements make adaptability an important factor for survival and success of a software system. This is more of an issue in domains of dynamic nature and changing technology. A piece of software is adaptable to the extent that it can effectively be changed in order to meet changing requirements stemming from changes in its working environment [9]. Adaptability of a software product can be defined as ability of adapting it for different specified environments without applying actions or means other than those provided for this purpose for the software considered [101]. CKMF has been designed with adaptability as a core requirement. That is made possible through mechanisms such as layers, partitions, and controlled versioning. According to the above definition of adaptability, unlike other Software Engineering tools mentioned above, CKMF is adaptable by design because it provides actions and means to adapt its elements for different environments and applications.

6.3 Understandability

Understandability is the ability of comprehension. CBSE technology promotes software reuse and speeds up new software development. Before using an existing piece, e.g. a design or an actual code snippet, that piece should be fully understood. Software understandability or comprehension is defined as a characteristic of software quality [14], an indicator of ease of understanding of software systems. In fact, the difficulty of understanding an existing system limits its reuse [16]. The difficulty of understanding is a

direct consequence of raw complexity. Software systems tend to be among the most complex man made systems in the world. As they evolve and grow, they become more complex. There are various mechanisms to manage and control complexity. These mechanisms do not reduce complexity, but rather manage it so that those dealing with the system can cope with its complexity. If a software system does not have such mechanisms from the start, it can easily get too complex in an unmanaged way, making its understanding very difficult if not impossible.

Looking at the structures and facilities of BioPerl, BioJava, BioPython, NCBI C++ Toolkit, and ODBA/BioSQL, we note while they are powerful, they are limited in managing complexity. They are mostly flat environments or have concepts with limited capability for managing complexity of entities and the relationships among entities.

One purpose of layers in DM of CKMF is categorizing concepts so that a user does not have to face all the concepts simultaneously. The Core includes the most fundamental concepts. The Common layer is built upon the Core layer concepts and includes more application-oriented concepts. The External layer includes the most specific concepts. Another concept in CKMF is partition, which helps managing complexity and therefore increases understandability. Recall that a partition is a logical grouping of related concepts in the DM. Various partitions help users find faster what they need and comprehend it better.

6.4 Reliability

Software reliability is defined as probability of execution without failure for some specified interval, called the mission time [65]. This definition is similar to the hardware reliability definition. We are mostly concerned with reliability of the tools and models used by a user to build his/her own software. In other words, our primary concern is not the reliability of the system that builds upon the foundation we provide, but rather the reliability of the foundation. There are three principal reliability strategies: fault prevention, fault removal, and fault tolerance [64]. Fault prevention starts at system requirements analysis phase and continues to design and implementation phases. The goal of fault prevention process is to reduce the number of faults introduced in the first place. Fault removal uses design review, code inspection, and development testing to identify and remove faults in the code. Fault tolerance defines the threshold fault a system can tolerate without causing failures.

In CKMF, all three strategies are in place. Fault management is part of the system requirements document, which contains instructions and standards for fault prevention, fault removal, and fault tolerance.

BioPerl, BioJava, and BioPython offer fault tolerance capability. The user can use different options for fault removal mainly using debuggers and similar tools. NCBI C++ Toolkit and ODBA/BioSQL do not offer any specific option for fault management, however, they offer fault tolerance and fault removal options that are specific to the language used for their implementations.

6.5 Maintainability

Traditionally, software maintenance phase begins when a software system is completed and delivered to the user. This is not really the case in today's Software Engineering. Maintenance should begin as early as possible. Maintainability should be a factor in system architecture design. Software maintainability can be defined as prediction of the ease with which a system can evolve from its current state to its future desired state [1]. Software maintenance activities can be classified into four categories: corrective, adaptive, perfective, and preventive. Corrective maintenance refers to fixing problems. We discussed this subject in reliability section. Adaptive maintenance refers to modifications that adapt to changes in the environment, including but not limited to changes in user and system requirements, changes in the hardware or software components of the environment. Perfective maintenance refers to enhancements, which make the product better in some way, by attempting to prevent malfunctions or improve maintainability. We also discussed preventive maintenance earlier in the reliability section.

The dynamic environments and ever-changing requirements make maintainability a strong quality requirement for software systems. Software maintenance accounts for more than sixty percent of the costs in the software life cycle [66]. This huge cost is a reminder of the importance of maintainability. Looking at different domains, we observe environments that are more dynamic, and demand a greater degree of maintainability. In many high tech domains, almost everything is new; new and changing requirements are emerging all the time. Such domains pose a challenge for maintenance, in particular adaptive and perfective categories.

CKMF has a different approach to maintainability. It allows users to have a say in what should be done next and how to manage changes. This is part of the MC responsibilities in which every contributor has a voice. The MC and its supporting structures are representatives of the collective conscience of all stakeholders. This combination guarantees that the framework adapts to changes in the environment and to future trends in a timely fashion. This may be done in BioPerl, BioJava, BioPython, NCBI C++ Toolkit, and/or ODBA/BioSQL by users suggesting or proposing changes or reporting bugs, however there is no guarantee that the next version of the tool will incorporate the proposed change.

Another benefit of the management structure in CKMF is making sure that every version is backwards compatible as much as possible. This ensures smooth and seamless transition of applications that rely on the framework with minimum possible disruption. This is very similar to the approaches of BioPerl, BioJava, BioPython, NCBI C++ Toolkit, and ODBA/BioSQL for version management.

6.6 Summary of Evaluation

The following table summarizes our comparison of different tools using the metrics introduced. The rows in the table include the tools with which we compare CKMF, and the columns include the metrics of the comparison. Each entry in the table indicates our ranking of the particular tool in terms of the metric heading. A dash in a cell means that the metrics cannot be used to rank the item.

Tools	Portability		Adaptability	Understandability	Reliability	Maintainability
	OS	PL				
BioPerl	Good	-	Poor	Fair	Good	Fair
NCBI C++ Toolkit	Good	-	Poor	Fair	Fair	Fair
BioJava	Good	-	Poor	Fair	Good	Fair
BioPython	Good	-	Poor	Fair	Good	Fair
OBDA/BioSQL	Good	Good	-	Fair	Fair	Fair
CKMF	Good	Good	Good	Good	Good	Good

Table 33: Summary of Comparisons

Chapter 7

Conclusions and Future Work

Lack of a generally accepted knowledge management system in SEOs is source of many problems in knowledge acquisition, transfer, and application, system development, information integration, and developing data modeling and system development expertise. We have proposed a framework with four elements (layered data model, managing committee, constraints, and management database) and an incremental approach for knowledge acquisition. The layered data model and other elements make the framework flexible enough to handle variations and extensions and the incremental approach makes it possible to develop a common understanding gradually while knowledge is gathered, refined, and experts contribute. Through the development of a running prototype, we studied feasibility of the framework instantiation and illustrated its ability to acquire and refine knowledge. We also demonstrated in a limited way the effect of the framework on software development.

What remains to be done is reengineering some existing bio-applications using the framework to demonstrate its usefulness in facilitating software development process in bioinformatics. We need usability tests, which require engaging some research groups in bioinformatics to use and contribute to the framework to further enrich the knowledge contents of the framework.

7.1 Conclusions

In our work, we studied the true nature of SEOs and viewed them as knowledge intensive organizations. SEOs share certain characteristics with other KIFs, explained as follows:

- Software Engineers are highly qualified individuals that perform knowledge-based tasks using their intellectual and symbolic skills.
- Software Engineers desire a high degree of autonomy and their importance and autonomy downplays the traditional organizational hierarchy.
- There is a need for extensive communication for coordination and problem-solving in SEOs.
- It is very difficult if not impossible to perform a subjective and accurate quality assessment of the work done by Software Engineers.

In our assessment of the framework and in our experience in developing the prototype, we attempted to stress and address the challenges presented by the above mentioned characteristics of SEOs. Our goal was to capture the knowledge Software Engineers use and/or create in their routine work in a non-intrusive way and with minimal impact on the performance of Software Engineers. The process of knowledge management in the framework works in a way that gives Software Engineers a high degree of freedom and autonomy, and at the same time involves them in the management process so that they can be creative and productive while they are ready to make compromises needed in an organization. CKMF provides a basis for collaboration and communication in and across organizational boundaries and keeps tracks of knowledge used and created in projects related to specific domains, and the evolution of knowledge over time. CKMF helps different viewpoints converge over time. These viewpoints are valuable assets in an

environment that depends on team work and collaboration for creating a common and shared understanding in a domain. Since quality assessment and evaluation is part of CKMF management activities, and Software Engineers are involved in the management and decision making process, they will be engaged in the evaluation of the work done; This in turn improves the quality of assessment and evaluation.

Our work touched upon the issues (mentioned in chapter 1) that cause software development to be viewed as mostly an individual creative activity rather than an engineering discipline and a team effort:

- The framework management process alleviates the problem of SEs avoiding explicit and direct control and coordination. The management structure involves Software Engineers as well as administrative body of the organization and since Software Engineers are part of decision making process, they are less concerned about the control and coordination process.
- The knowledge acquisition process of the framework is designed to merge with and absorb in the routine daily work processes. This process captures and records the knowledge of users of CKMF while they use it in their daily work. This applies to everybody who uses (and in fact contributes to) the framework, including even those who possess key domain knowledge and may not be willing to share it with others.
- The dominant culture in SEOs rewards and appreciates individual creativity and achievements more than team efforts. This is a cause for concern because it promotes individualism rather than team work. This bias encourages individuals, teams, or individual organizations spend most of their energy and resources into

individual achievements, which hampers sharing the achievements; Helping others make progress will not be among primary concerns in such environments. This focus on individuals can seriously affect and hamper the process of collective achievement and hurt the organization in a long run. On the other hand, if the cost of promoting team work is losing individual creativity, it also hurts the organization because valuable individual contributions to the collective process will be affected. Our work in this research was an attempt to offer a balanced approach. This approach acknowledges the importance of individual creativity. It gives individuals room to think freely and improvise. In this settings, creativity flourishes and results in achievements. These achievements are then shared "properly" with others so that the organization as a whole benefits from the individual creativity. The framework enables individuals, teams of individuals, or organizations as a whole to be creative and create knowledge using shared common knowledge. The knowledge that is created by each contributor can be refined in future iterations and become a part of common knowledge. This is how framework promotes individual creativity and team work together. The team work part is divided into two phases. Phase one is encouraging the use of common knowledge that is already gathered and documented. In this phase there is no explicit push to create harmony or agreement, so individuals get to be creative and dynamic and they can improvise freely. The results of this phase are captured in the framework DM and KMD. In phase two, different viewpoints contributed by various individuals and individual teams are closely examined and common elements are recognized. MC that represents contributors will then decide what

elements are solid and mature enough to move into common layer and become part of common and shared knowledge. This process guarantees individual achievements are encouraged and those achievements are shared with others over time.

- In SEOs sometimes there exist some non-technical concerns among individual SEs and managers that might affect the willingness of SEs to share knowledge. The process of knowledge acquisition and knowledge transfer in the framework and the management structure of the framework can help in alleviating these concerns. In CKMF, SEs are involved in knowledge creation and its distribution process. They are not merely simple knowledge workers that create knowledge that have no control over their products. They are part of management mechanism and this composition helps ease possible tensions among SEs and managers.

The ultimate purpose of our framework is solving some of the problems SEOs face today in the context of knowledge management. Based on our previous experience in desiging the framework and developing of the running prototype, we can list the following benefits SEOs can expect from using the framework in their development environment:

- The amount of knowledge lost due to attrition is reduced
- The [steep] learning curves are smoothed and this results in faster learning and hence acquiring knowledge more rapidly
- The history of knowledge creation and evolution is recorded and this helps in remembering the mistakes and poor decisions, thus avoiding repeating past mistakes. All this results in less corrective actions and rework

- Movement of knowledge within and across organizational boundaries is facilitated. The knowledge moves around in an effective and cost-efficient manner

7.2 Future Work

Based on our study in this research and our experience in SEOs, we identify several possible directions of this work. The most important extension of current research could be application of CKMF in SEOs that are active in domains similar to Bioinformatics, which we refer to as "generalization of CKMF". Other future extensions might include automatic code generation for DM, creating a persistent object storage and request broker, automatic populating of KMD, and opening MC structure using a wiki. We briefly describe these future directions.

7.2.1 Generalization of CKMF

When we take a closer look at CKMF, we notice that there is nothing in the architecture or design of CKMF that is specific to Bioinformatics as an application domain. In other words, CKMF is not specific to Bioinformatics. Based on our industry experience, we see the potential of generalizing CKMF so that it can be used in SEOs active in domains that are similar to Bioinformatics. The possible candidate domains should be similar to Bioinformatics in areas such as "rapid changes in technology", "lack of widely used or established standards", and "need for integration and collaboration". We need to formalize prerequisites and preconditions of CKMF application and come up with a

suitability function to decide if CKMF is appropriate for application in a specific domain. We would also need to formalize the framework instantiation process, which is the third major stage of framework development [58].

7.2.2 Automatic Code Generation

We would like to develop a code generator that can generate code from the specification stored in KMD. For this, we need to use a formal specification language to describe the concepts and the relationships among concepts in KMD. For each concept, we should describe the properties (i.e. data members) and the behavior (i.e. methods). These descriptions stored in KMD should be processed to generate the code for concepts and their dependencies. The code generator can be designed to have three main modules. A module is responsible for interfacing with KMD and retrieving the specification of selected concepts and their relationships. The second module is in charge of translating specification of each concept to an intermediate language. The third module is responsible for translating the specification of the concepts in the intermediate language into the target programming language. This module should have the ability to accept plug-ins, which should also be provided for each target programming language. In general, a plug-in should provide transformations from intermediate language structures to the target language structures. The plug-ins are developed according to the specification. More details of functionalities of plug-in should be provided.

7.2.3 Persistent Object Storage and Request Broker

To facilitate use of the framework and support integration of applications, we can create and add a service to CKMF. This service should provide a storage service for objects of concepts in DM. Applications that use these concepts will have the option to make their objects persistent. This service greatly expands where and when CKMF can be used. It helps the applications using the framework to maintain their objects when they are not needed for a period of time or between various runs of the application. The storage service will reduce the complexity of the architecture of applications because it eliminates the need to handle storage and retrieval of objects. The second functionality that this service will provide is object request broker. Object request broker is a piece of software that handles the communication of messages from the requesting program (client) to the object as well as the values returned from the object back to the caller. When we enable programs to make their objects persistent, one can go one step further and let different applications share their objects. There are many strategies and architectures we can use for this purpose. One possible solution is using an OBR. In this solution, we will have a service that takes care of object instantiation, persistence, and sharing of objects. When an application needs an instance of a class in the framework DM, it will use the service to instantiate an object of that class. The object will be instantiated in the central repository that is managed by the service. After the object is created, the application will be able to use it through the service, i.e. the application can access the properties of the object or invoke its methods. Since objects are maintained in the repository, they can be easily shared between applications. Sharing policies and restrictions have to be clearly defined in advance.

7.2.4 Automatic Populating of KMD

At present, we populate KMD tables manually. This is feasible because we are dealing with a fairly small amount of data. When the framework is used in a development environment and the number of contributors grows, we will face two issues if we do this manually. The first issue is the volume of data that should be inserted into KMD. The more CKMF is used and the more contributors contribute, the amount of data that should be entered into KMD or updated grows. This means more time and resources for keeping the framework DM and KMD in sync. As this task gets more demanding, we get farther from one of our original goals, namely the desire that the framework processes be non-disruptive to the primary functions of contributors. Keeping KMD up-to-date is not a primary function of a contributor and if it becomes a drag on time and resources of contributors, it prohibits framework acceptance. The second issue is reliability of the manual process. When humans are involved in the process, it increases the risk of making mistakes. Human can misread a specification or interpret them improperly. It is very difficult if not impossible to guarantee correctness of all data entered or updated in the manual process. When the volume of data being processed increases, this task becomes even more difficult.

To avoid the above mentioned issues, we need to have a process in place that automatically populates KMD tables and updates them when relevant changes happen. This automatic process should also perform some checks to make sure the data that is being processed is correct and free of errors. To have such an automatic process, we need to start from specification. We define a set of tests (syntactic and semantic) to perform to

ensure a specification is done correctly and properly. Based on these tests, we select a formal specification language which is capable of satisfying the requirements, and we adapt this language for describing the DM elements. This means that all elements of the DM should be specified using this language. Contributors will also use this language to specify their Extension schemas. After specifications are expressed using the language adapted, we develop an application that reads these specifications and populates or updates KMD.

7.2.5 Opening MC Structure Using a wiki

We mentioned the importance of management process in the framework. One of the tasks of MC was managing the input from contributors about the changes. MC structure is designed to include SEs that have field experience as well as administrative staff of the organization. It is obvious we can not include every single individual in MC. What we can do instead is opening up MC structure into the contributors. If discussions of MC and process of decision making in MC are transparent and open to input from all contributors, we will be closer to our goal of involving all contributors. The more we involve contributors in MC processes, we are more confident that every change made to the framework DM reflects the needs more realistically and in the best possible way since we had the maximum input to make the decision about each change.

To provide such a degree of openness and transparency, we need to look into new technologies. One candidate that is not so new is wiki. In the past few years, wikis have become popular tools in environments that need effective collaboration and knowledge sharing. Wikis are also used in areas such as defect tracking, requirements management,

test case management, and project portals. We can adapt a wiki as our formal documentation system. All documents describing elements of the framework should be written in the wiki. MC will decide which documents may be edited and by whom. There will be a section in wiki site dedicated to discussions about current version of the DM and suggestions about changes for next version. All meetings of MC should be documented and the minutes should be available on the wiki site. Contributors should be able to discuss the notes and provide feedbacks.

References

[1] V. S. Alagar, Qiaoyun Li and O. S. Ormandjieva, "Assessment of maintainability in object-oriented software", in *Proceedings of 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems*, pp. 194-205, 2001.

[2] M. Alavi and D. E. Leidner, "Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues", *MIS Quarterly,* vol. 25, pp. 107, March 2001.

[3] C. Alexander, "A Pattern Language: Towns, Buildings, Construction", New York: Oxford University Press, 1977.

[4] E. Al-Masri and Q. H. Mahmoud, "Interoperability among Service Registry Standards", IEEE Internet Computing, vol. 11, pp. 74-77, 2007.

[5] M. Alvesson, "Knowledge Work and Knowledge-Intensive Firms", Oxford; New York: *Oxford University Press*, pp. 271, 2004.

[6] M. Alvesson, "Knowledge work: Ambiguity, image and identity", *Human Relations*, vol. 54, pp. 863, July 2001.

[7] A. Aurum, P. Parkin and K. Cox, "Knowledge management in software engineering education", in *Proceedings of IEEE International Conference on Advanced Learning Technologies*, pp. 370-374, 2004.

[8] Y. Awazu, "Knowledge management in distributed environments: Roles of informal network players", in *Proceedings of The 37th Hawaii International Conference on System Sciences*, 2004.

[9] L. Baekgaard, "Designing adaptable software--parameterization of volatile properties", in *Proceedings of the 1990 Conference on Software Maintenance*, pp. 335-342, 1990.

[10] C. Barnatt, "Challenging Reality : In Search of the Future Organization", Chichester, England ; New York: Wiley, 1997.

[11] V. R. Basili, L. C. Briand, W. M. Thomas and Maryland Univ., College Park, MD. Dept. of Computer Science., "Domain analysis for the reuse of software development experiences", in *NASA. Goddard Space Flight Center, Proceedings of the 19th Annual Software Engineering Workshop*, pp. 11-34, 1994.

[12] D. Batory and S. O'Malley, "The design and implementation of hierarchical software systems with reusable components", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 1, pp. 355-398, 1992.

[13] J. Bih, "Service oriented architecture (SOA) a new paradigm to implement dynamic e-business solutions", *Ubiquity*, vol. 7, pp. 1, Aug. 8 2006-Aug. 14. 2006.

[14] B. W. Boehm, "Characteristics of Software Quality", Amsterdam New York: North-Holland Pub. Co., American Elsevier, 1978.

[15] G. Cardino, F. Baruchelli and A. Valerio, "The evaluation of framework reusability", *ACM SIGAPP Applied Computing Review*, vol. 5, pp. 21-27, 1997.

[16] G. Cardino and V. R. Basili, "The qualification of reusable software components", in *Software Reusability* W. Schäfer, R. Prieto-Díaz and M. Matsumoto, Eds. Ellis Horwood, pp. 117-119, 1994.

[17] R. Conradi and B. Westfechtel, "Version Models for Software Configuration Management", ACM Computing Surveys, vol. 30, pp. 232-282, 1998.

[18] I. Crnkovic, "Component-based software engineering: Building systems from software components," in *Proceedings of the 26th Annual International Computer Software and Applications Conference*, pp. 816-817, 2002.

[19] A. Cuzzocrea and C. Mastroianni, "A reference architecture for knowledge management-based web systems", in *Proceedings of The Fourth International Conference on Web Information Systems Engineering*, pp. 347-351, 2003.

[20] T. H. Davenport, "Can You Boost Knowledge Work's Impact on the Bottom Line?", *Harvard Management Update,* vol. 7, pp. 10-11, 2002.

[21] T. H. Davenport and L. Prusak, "Working Knowledge: How Organizations Manage what they Know", *Harvard Business School Press*, pp. 199, 2000.

[22] J. G. Davis, E. Subrahmanian, S. Konda, H. Granger, M. Collins and A. W. Westerberg, "Creating Shared Information Spaces to Support Collaborative Design Work", *Information Systems Frontiers*, vol. 3, pp. 377, September 2001.

[23] C. Derby, "Knowledge management for engineers", in *Proceedings of International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, pp. 760-765, 2003.

[24] K. C. Desouza, "Barriers to Effective Use of Knowledge Management Systems in Software Engineering", *Communications of the ACM,* vol. 46, pp. 99-101, January 2003.

[25] K. C. Desouza, Y. Awazu and P. Baloh, "Managing Knowledge in Global Software Development Efforts: Issues and Practices", *IEEE SOFTWARE,* vol. 23, pp. 30-37, September-October 2006.

[26] K. C. Desouza and J. R. Evaristo, "Managing knowledge in distributed projects", *Communications of the ACM,* vol. 47, pp. 87-91, April 2004.

[27] F. I. Dretske, "Knowledge and the Flow of Information", Cambridge, Mass.; *The MIT Press-Bradford Books*, 1981.

[28] A. N. Dwivedi, R. K. Bali, R. N. G. Naguib and N. S. Nassar, "The knowledge management landscape: Implications for clinical knowledge management and practice", in *Proceedings of 26th Annual International Conference of the Engineering in Medicine and Biology Society,* pp. 3171-3174, 2004.

[29] E. Eide, A. Reid, J. Regehr and J. Lepreau, "Static and dynamic structure in design patterns", *in Proceedings of the 24rd International Conference on Software Engineering*, pp. 208-218, 2002.

[30] J. Estublier, "Software configuration management: A roadmap", in *Proceedings of the Conference on the Future of Software Engineering*, pp. 279-289, 2000.

[31] L. Fahey and L. Prusak, "The eleven deadliest sins of knowledge management", *California Management Review,* vol. 40, pp. 265, Spring 1998.

[32] M. E. Fayad and A. Altman, "Thinking objectively: an introduction to software stability", *Commun ACM*, vol. 44, pp. 95, 2001.

[33] G. Ford and N. Gibbs, "A mature profession of software engineering (final report)", Tech. Rep. AD-A307889; CMU/SEI-96-TR-004; ESC-TR-96-004; NIPS-96-74173, 1996.

[34] W. B. Frakes and Kyo Kang, "Software reuse research: status and future", *IEEE Transactions on Software Engineering*, vol. 31, pp. 529-536, 2005.

[35] E. Gamma, "Design Patterns : Elements of Reusable Object-Oriented Software", Reading, Mass.: Addison-Wesley, 1995.

[36] G. C. Gannod, J. E. Burge and S. D. Urban, "Issues in the design of flexible and dynamic service-oriented systems", in *ICSEW '07: Proceedings of the 29th International* Conference on Software Engineering Workshops, 2007, pp. 118.

[37] M. L. Griss, "Software reuse: From library to factory", *IBM Systems Journal*, vol. 32, pp. 548-566, 1993.

[38] B. H. Hansen and K. Kautz, "Knowledge Mapping: A Technique for Identifying Knowledge Flows in Software Organisations", *Lecture Notes in Computer Science,* vol. 3281, pp. 126-137, 2004.

[39] S. Henninger, "An evolutionary approach to constructing effective software reuse repositories", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 6, pp. 111-140, 1997.

[40] E. Horowitz, A. Kemper and B. Narasimhan, "Application generators: Ideas for programming language extensions", in *Proceedings of the 1984 Annual Conference of the ACM on the Fifth Generation Challenge*, pp. 94-101, 1984.

[41] C. Hu, "When to use an interface?", *SIGCSE Bull*, vol. 38, pp. 86-90, 2006.

[42] C. Huang and W. Y. Liang, "Explication and Sharing of Design Knowledge Through a Novel Product Design Approach", *IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews,* vol. 36, pp. 426-438, May 2006.

[43] G. P. Huber, "Organizational Learning: The Contributing Processes and the Literatures", *Organization Science,* vol. 2, pp. 115, 1991.

[44] Y. Hung and S. T. Chou, "On constructing a knowledge management pyramid model", in *Proceedings of Intl. Conf. on Information Reuse and Integration*, pp. 1-6, 2005.

[45] R. E. Johnson, "Frameworks = (components + patterns)", *Commun ACM*, vol. 40, pp. 39-42, 1997.

[46] R. E. Johnson, "Documenting frameworks using patterns", in *OOPSLA '92: Conference Proceedings on Object-Oriented Programming Systems, Languages, and Applications*, pp. 63-76, 1992.

[47] C. Jones, "Economics of Software Reuse", *Computer*, vol. 27, pp. 106-107, July 1994.

[48] J. Keys, "Software Engineering Handbook", *Auerbach*, pp. 874, 2003.

[49] T. Kotnour, C. Orr, J. Spaulding and J. Guidi, "Determining the benefit of knowledge management activities", in *Proceedings of International Conference on Systems, Man, and Cybernetics*, pp. 94-99, 1997.

[50] C. W. Krueger, "Software reuse", ACM Computing Surveys, vol. 24, pp. 131-183, 1992.

[51] C. W. Lillie, "Distributed network of reuse libraries offers the best approach to successful software reuse", in *Proceedings of Third International Conference on* Software Reuse: Advances in Software Reusability, pp. 207-208, 1994.

[52] Y. Liu and I. Traore, "Complexity measures for secure service-oriented software architectures", in *ICSEW '07: Proceedings of the 29th International Conference on Software Engineering Workshops*, 2007, pp. 78.

[53] P. Louridas, "Using wikis in software development", IEEE Software, vol. 23, pp. 88-91, 2006.

[54] C. Lung, S. Bot, K. Kalaichelvan and R. Kazman, "An approach to software architecture analysis for evolution and reusability", in *CASCON '97: Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research*, pp. 15, 1997.

[55] R. R. Macala, L. D. J. Stuckey and D. C. Gross, "Managing domain-specific, product-line development", *IEEE Software*, vol. 13, pp. 57-67, 1996.

[56] F. Machlup, "Knowledge, its Creation, Distribution, and Economic Significance", Princeton, N.J.: *Princeton University Press*, 1980.

[57] O. Marjanovic, "Sharing and reusing learning experiences – the knowledge Management Perspective", in *Proceedings of The Fifth IEEE International Conference on Advanced Learning Technologies,* 2005.

[58] M. E. Markiewicz and C. J. P. d. Lucena, "Object oriented framework development", *Crossroads*, vol. 7, pp. 3-9, 2001.

[59] J. Mayrand, J. Patenaude, E. Merlo, M. Dagenais and B. Lague, "Software assessment using metrics: A comparison across large C++ and Java systems", *Ann. Softw. Eng.*, vol. 9, pp. 117-141, 2000.

[60] R. J. McQueen, "Four views of knowledge and knowledge management", in *Proceedings of Americas Conference on information Systems*, 1998.

[61] J. Mesaric, "Knowledge management - necessity and chalenge in small and medium enterprises", in *Proceedings of 26th Int. Conf. Information Technology Interfaces*, pp. 481-485, 2004.

[62] M. Mohnen, "Interfaces with default implementations in java", in *PPPJ '02/IRE '02: Proceedings of the Inaugural Conference on the Principles and Practice of Programming, 2002 and Proceedings of the Second Workshop on Intermediate* Representation Engineering for Virtual Machines, pp. 35-40, 2002.

[63] J. D. Mooney, "Portability and reusability: Common issues and differences", in *Proceedings of the 1995 ACM 23rd Annual Conference on Computer Science*, pp. 150-156, 1995.

[64] J. D. Musa, "Introduction to software reliability engineering and testing", in *Proceedings of the Eighth International Symposium on Software Reliability Engineering - Case Studies*, pp. 3-12, 1997.

[65] J. D. Musa, A. Iannino and K. Okumoto, "Software Reliability : Measurement, Prediction, Application", New York: McGraw-Hill, 1990.

[66] S. Muthanna, K. Kontogiannis, K. Ponnambalam and B. Stacey, "A maintainability model for industrial software systems using design level metrics", in *Proceedings of the Seventh Working Conference on Reverse Engineering*, pp. 248-256, 2000.

[67] P. Naur, B. Randell and J. N. Buxton, "Software Engineering: Concepts and Techniques : Proceedings of the NATO Conferences", New York: Petrocelli-Charter, 1976.

[68] I. Nonaka, "A Dynamic Theory of Organizational Knowledge Creation", *Organization Science,* vol. 5, pp. 37, February 1994.

[69] C. E. Oancea and S. M. Watt, "Parametric polymorphism for software component architectures", in *OOPSLA '05: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages, and Applications*, pp. 147-166, 2005.

[70] Y. Ouyang and D. L. Carver, "Enhancing design reusability by clustering specifications", in *Proceedings of the 1996 ACM Symposium on Applied Computing*, pp. 493-499, 1996.

[71] T. Peachey and D. Hall, "Knowledge management and the leading IS journals: An analysis of trends and gaps in published research", in *Proceedings of The 38th Hawaii International Conference on System Sciences*, 2005.

[72] G. Phipps, "Comparing observed bug and productivity rates for Java and C++", *Software Pract Exper*, vol. 29, pp. 345-358, 1999.

[73] G. Pour, "Towards component-based software engineering," in *Proceedings of the 1998 IEEE 22nd Annual International Computer Software & Applications Conference*, pp. 599, 1998.

[74] W. Pree, "Meta patterns - A means for capturing the essentials of reusable object-oriented design", in *ECOOP '94: Proceedings of the 8th European Conference on Object-Oriented Programming*, pp. 150-162, 1994.

[75] R. Prieto-Diaz and G. Arango, "Domain Analysis and Software Systems Modeling", Los Alamitos, CA, USA: IEEE Computer Society Press, 1991.

[76] K. J. Ransom and C. D. Marlin, "Supporting software reuse within an integrated software development environment", in *Proceedings of the ACM SIGSOFT Symposium on Software Reusability*, pp. 233-237, 1995.

[77] K. Reinholtz, "Java will be faster than C++", *SIGPLAN Not.*, vol. 35, pp. 25-28, 2000.

[78] Robert S. Hanmer,Kristin F.Kocan, "Documenting architectures with patterns", *Bell Labs Technical Journal*, vol. 9, pp. 143-163, 2004.

[79] O. M. Rodríguez, A. I. Martínez, A. Vizcaíno, J. Favela and M. Piattini, "Identifying knowledge management needs in software maintenance groups: A qualitative approach", in *Proceedings of The Fifth Mexican International Conference in Computer Science*, pp. 72-79, 2004.

[80] I. Rus and M. Lindvall, "Knowledge Management in Software Engineering", *IEEE Software,* pp. 26-38, May-June 2002.

[81] W. Selen, "Learning in the new business school setting: A collaborative model", *The Learning Organization*, vol. 8, pp. 106, 2001.

[82] Y. Shan, T. Cargill, B. Cox, W. Cook, M. Loomis and A. Snyder, "Is multiple inheritance essential to OOP? (panel)", *SIGPLAN Not.*, vol. 28, pp. 360-363, 1993.

[83] K. Sherif and A. Vinze, "Domain engineering for developing software repositories: A case study", *Decision Support Systems*, vol. 33, pp. 55-69, 2002.

[84] G. B. Singh, "Single versus multiple inheritance in object oriented programming", *SIGPLAN OOPS Mess.*, vol. 6, pp. 30-39, 1995.

[85] H. Siy and A. Mockus, "Measuring domain engineering effects on software change cost", in *Proceedings of International Software Metrics Symposium*, pp. 304-312, 1999.

[86] J. Stajich and E. Birney, "The Bioperl project: motivation and usage", *SIGBIO News*, vol. 20, pp. 13-14, 2000.

[87] D. Stenmark and R. Lindgren, "Integrating knowledge management systems with everyday work: Design principles leveraging user practice", in *Proceedings of The 37th Hawaii International Conference on System Sciences*, 2004.

[88] M. E. Stropky and D. Laforme, "An automated mechanism for effectively applying domain engineering in reuse activities", *in TRI-Ada '95: Proceedings of the Conference on TRI-Ada '95*, pp. 332-340, 1995.

[89] K. Surendran and F. H. Young, "Teaching software engineering in a practical way", in *Proceedings of the 14th Annual Conference of the National Advisory Committee on Computing Qualifications*, pp. 75-80, 2000.

[90] D. J. Teece, "Capturing Value from Knowledge Assets: The New Economy, Markets for Know-How, and Intangible Assets", *California Management Review,* vol. 40, pp. 55, Spring 1998.

[91] I. Tuomi, "Data is more than knowledge: Implications of the reversed knowledge hierarchy for knowledge management and organizational memory", in *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences*, pp. 12, 1999.

[92] A. Valerio, G. Succi and M. Fenaroli, "Domain analysis and framework-based software development", *SIGAPP Appl. Comput. Rev.*, vol. 5, pp. 4-15, 1997.

[93] P. Vitharana, "Risks and challenges of component-based software development," *Communications of the ACM*, vol. 46, pp. 67-72, 2003.

[94] A. Vizcaíno, M. Piattini, M. Martínez and G. Aranda, "Evaluating collaborative applications from a knowledge management approach", in *Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, pp. 221-225, 2005.

[95] Y. Wang and M. Chen, "A collaborative knowledge production model for knowledge management in complex engineering domains", in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, pp. 5050-5055, 2004.

[96] Y. Wang, J. Wang and S. Zhang, "Collaborative knowledge management by integrating knowledge modeling and workflow modeling", in *Proceedings of Intl. Conf. on Information Reuse and Integration*, pp. 13-18, 2005.

[97] J. Ward and A. Aurum, "Knowledge management in software engineering – describing the process", in *Proceedings of The 2004 Australian Software Engineering Conference*, pp. 137-146, 2004.

[98] C. P. Willis, "Analysis of inheritance and multiple inheritance", *Software Engineering Journal*, vol. 11, pp. 215-224, 1996.

[99] CIM-SAN. Available: http://www.snia.org/smi/tech_activities/CIMSAN/, 2007.

[100] "CIM Concepts White Paper, CIM Versions 2.4+", Distributed Management Task Force, 2003.

[101] "ISO/IEC 9126-1:2001 Software engineering – Product quality – Part 1:Quality model", International Organization for Standardization and International Electrotechnical Commission , 2001.

[102] "The Common Information Model, CIM Version 2.7", Distributed Management Task Force, Technical Committee, 2003.

[103] www.biojava.org

[104] www.bioperl.org

[105] www.biopython.org

[106] www.genomequebec.com

[107] www.ncbi.nlm.nih.gov/IEB/ToolBox/CPP_DOC

[108] www.open-bio.org

Appendices

Appendix A: Organizational Learning Process Obstacles in SEOs

There are certain elements in the Software Engineering environments and SEs' work habits and patterns that make it difficult for organizational learning process to be effective and therefore disrupt the smooth flow of knowledge in a SEO. In spite of all efforts aimed at making software development an engineering discipline and a team effort, it is still mostly an individual creative activity. The main reasons are as follows [48]:

- SEs desire autonomy and escape explicit coordination and control [to some degree] as it is expected of KWs
- Often certain KWs possess key target domain knowledge that are not willing to share with fellow KWs
- The dominant culture in SEOs usually values individual achievements more than team efforts
- There are political concerns among individual KWs and managers that affect the willingness of KWs to share knowledge

Appendix B: KM Related Challenges in SEOs

Among the challenges SEOs face at the present time, they have to deal with the following problems that are directly related to knowledge management [40]:

- **Attrition Problem**

  Economies of industrialized countries as well as developing countries continue to depend more on KIFs every day. This degree of attention shift to KIFs increases the demand for KWs. When demand is high, KWs tend to become mobile and move from organization to organization. Unlike in the industrialized era economy where people were needed but replaceable, KWs cannot be easily replaced. In KIFs today, KWs are the main assets carrying the intellectual capital of the organization. When workforce attrition occurs, the knowledge which KW used to carry (which is sum of his/her experience and know-how) is also lost and should be deducted from the organization's intellectual capital. In case of a SEO, it should be noted that this loss is two fold: loss of Software Engineering experience and know-how, and loss of domain experience and know-how. Most SEOs suffer from lack of a mechanism for retaining the acquired knowledge crucial for the business and managing it. Our framework helps reduce the impact of attrition through sharing the knowledge of individual SEs and distributing it in SEO, thus making it true organizational knowledge .

- **Steep Learning Curves**

  KW accumulates knowledge over the course of his/her tenure with KIF. In a SEO, KW acquires Software Engineering experience over the course of projects s/he is involved in but s/he also needs to gain knowledge about the target domain of each

project. Comparing average lifespan of projects to average length of SE career, it is clear that to be an effective player, KWs in a SEO have to absorb a great deal of domain knowledge in a relatively short time. The amount of time KW needs to absorb this knowledge depends on many factors such as prior knowledge of the domain or similar domains, degree of acquaintance with prerequisites, pace of learning, and available training materials and resources and their quality. Among KIFs, the situation of KWs in SEOs is fairly unique. As previously mentioned, KWs are expected to face unique problems and are supposed to come up with solutions using their knowledge. SEs often face problems in domains in which they have no or little prior knowledge. SEs first have to educate themselves about the domain in order to be able to address the problem. Only after this phase, SEs can go forward and use their Software Engineering skills in combination with the knowledge they have acquired about the domain to provide a solution. Every time a new project in a new domain begins, the cycle of knowledge acquisition has to be repeated. As we go forward, computer systems find their ways into more complex domains, and this makes it harder for SEs to absorb the necessary domain knowledge in a reasonable time. In other words, the more complex target domains get, the learning curve becomes steeper. The framework enhances the organizational learning process and this helps to ease the curve of learning path. In absence of established systems for maintaining and managing the acquired knowledge, KWs who have already acquired domain knowledge become even more important and more valuable because what they have gained (in form of experience and know-how) is vital for the success and sometimes survival of the organization. SEOs become more and more dependent on individual

KWs or team of KWs. This inevitably leads to imbalance in organization structure and disrupts its ability to manage and use its intellectual capital. SEO can not afford to lose key KWs and if they do, they lose their ability to fulfill their duties in certain areas because the new KWs need time to learn what was learnt before. Steep learning curves for complex domain have lead to emergence of new hybrid disciplines which focus on training a new breed of SEs that specialize in a certain complex domain. This approach has it own shortfalls and setbacks and is not necessary at all if SEOs are able to manage their knowledge and use that.

- **Repeating Mistakes**

In every discipline it is very common to make mistakes in executing tasks. What is considered not acceptable is making the same mistakes more than once. It is not difficult to spot KWs in SEOs making the same mistakes over and over. In many cases, there could be confusion about the nature of the mistake and KWs might debate whether something is a mistake or not. SEs or teams of SEs easily forget about their mistakes in a project and are prone to make the same mistakes again in some future projects. It is also very common in a SEO to spot teams that make the same mistake in different projects. In many well-established disciplines there is a great emphasis on postmortem analysis of projects. Special attention is paid to problems and mistakes in a project, which are carefully analyzed to find out what caused the mistakes and what could have been done to avoid them. The lessons learnt from this analysis are documented so that they can be used later in future projects. This makes occurrence of the same mistakes less probable. Such a postmortem analysis is not

common for software projects for various reasons. The most important reason is lack of properly recorded information about the course of the project, the decisions that were made, and the reasons for the decisions made. KWs in SEOs do not generally document all aspects of their work, and even if they try to do that, the documentation methods do not capture many important aspects of the SEs'work. It is well known that SEs are constantly under lots of pressure to meet deadlines and produce results and this makes it harder for them to think of benefits of creating knowledge for future use. There are SEOs that have mandatory documentation standards and force their SEs to document their every move. The burden of such methods clearly affects productivity of SEs and reduces the overall output of the SEO. Every time there is a conflict between documentation standards and delivery deadlines, managers give priority to delivery at the cost of poor or no documentation. It is very likely for SEOs to make the same mistakes because they lack a generally accepted mechanism of tracking decisions made during projects, analyzing these decisions, storing the results of analysis, and finally making the results available to be used in future projects. The framework makes it possible to maintain a history of decisions SEs make during the course of projects and the results of those decisions are traceable over time and their affects can be realized as well. This history could be used as an input to an analysis process and the results could help evaluate past decisions in order to recognize mistakes and avoid them in future.

- **Knowledge Movement Within and Across Organizational Boundaries**

Software systems are not like isolated islands anymore. In a knowledge based industry, many software systems interoperate with other software systems to create tangible benefits for their users. The general trend in software industry is moving towards more integration. The whole software industry is moving towards a service-oriented architecture and moving away from a product-based architecture. The customers look at software as a service. They may need certain features from one system and certain features from another system and they like to see these two systems work together to provide the functionality they have in mind. To satisfy these requirements, individual SEOs, even competitors, have to work together to make their products compatible with other software and hardware products. The need for cooperation and compatibility forces individual SEOs to come together and share knowledge in order to reach a common understanding. The ideal outcome in this situation is all SEOs share their knowledge and come up with a standard. In reality, this does not happen that often because in most cases the target domains are like uncharted territory for SEOs and they do not have enough knowledge to come up with a standard in the beginning. At the same time SEOs can not afford to spend enough time to acquire a deep understanding of the target domain and have to create products based on the knowledge at hand or they will lose their customers and their market share. This often leads to creation of various premature de facto standards. SEOs will be divided among these competing camps and they will start a war of standards which hurts all the advocates as well as users of the competing systems. In

the past few years, SEOs have realized the disadvantages of this approach and they are leaning more and more towards collaboration frameworks that enable them to share knowledge and incrementally come to an agreement over time.

The framework is designed to act as a collaboration platform. This characteristic enables several SEOs work together and cooperate while they maintain their unique views and their differences. The fruit of this collaboration over time could be a consensus and a unified understanding of the target domain. One of the main motivators of this work is our experience in an industry effort of this kind. That effort is "CIMSAN Initiative", sponsored by Storage Networking Industry Association (SNIA). This program allows participants which are software and hardware companies share knowledge and collaborate towards developing and improving a standard for managing storage area networks (SAN). The goal of companies that participate in this effort is development and implementation of "SNIA's Storage Management Initiative (SMI)." The main goals of CIMSAN initiative, which are indicators of today' software industry trend, are:

a) Ease the implementation of the SMIS specification in vendor products (through ongoing developer symposia and plug-fests)
b) Reduce multi-vendor integration costs
c) Build seamless interoperability between products.
d) Forward the development of the CIM/WBEM based SMI Specification (SMIS).

Appendix C: Related Software Engineering Concepts

- **Reuse Libraries**

In early years, software reuse was mainly done through personal and organizational information preserving structures. Formalizing the processes of preservation and location facilitates reuse and makes it more attractive for a wider range of people and activities. The reuse library is the proper center of activity for these formalized processes and the proper access point for sharing assets.

Reuse libraries consist of repositories for storing reusable assets, a search interface that allows users to search for assets in the repository, a representation method for the assets, and facilities for change management and quality assessment [34]. Software reuse libraries offer services that include classification and cataloging, certification, storage, search, extraction, and maintenance. All these functions are required to exploit the potential of software reuse [51]

There are many types of software reuse libraries: domain specific, general purpose, private, commercial, government, nonprofit, and public domain libraries.

Libraries of randomly collected code often address a small number of typical developers' needs. Developers are reluctant to look for components in the library and prefer to write their own [37]. To increase reuse level as well as level of contribution to the library, incentives need to be focused. One common mistake is to offer a "reward" for contributing to a library but not for using the library, which can easily increase library size without increasing its reuse level.

- **Domain Engineering**

Domain Engineering (DE) is one of many approaches for implementing systematic reuse.

DE is a structured method of developing prefabricated building blocks to advance the development of software systems [75, 11, 88, 55]. DE approaches the problem by defining and facilitating the development of software product lines (or software families) rather than individual software products. This is accomplished by considering all of the products together as one set, analyzing their characteristics, and building an environment to support their production [85].

The building blocks in DE can take different shapes. The most common form of building blocks is code. In recent years, there has been an emphasis on the reuse of more abstract and more conceptual artifacts of the software development process such as design, architecture and results of requirement analysis process. However, for any of these to be reusable, they have to be designed with reuse in mind [83].

DE consists of three main phases: domain analysis, domain design and domain implementation. During the domain analysis, the scope is defined and functionality and composition of the domain are determined. In the next phase which is domain design, architecture for the domain is provided, and a family of general solutions for recurring problems in the domain is designed. During domain implementation, reusable assets including code, test cases, and documentation are developed [83].

- **Design Patterns**

Patterns as an architectural concept were introduced by Christopher Alexander in 1977. Design patterns are valuable because they highlight common structure, capture design expertise, and facilitate restructuring of software systems [29]. Alexander states that each pattern describes a problem which occurs over and over again in the environment, and then describes the core of the solution to that problem, in such a way that one can use this solution a million times over, without ever doing it the same way twice [3].

In a software development context, a design pattern is an artifact which names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design. The design pattern identifies the participating classes and instances, their roles and collaborations, and the distribution of responsibilities. It describes when it applies, whether it can be applied in view of other design constraints, and the consequences and trade-offs of its use [35]. Point of view is a crucial factor in deciding what could be considered a design pattern. In general, design patters are not about primitive entities such linked lists or hash files. They are not about complex and domain specific models that span an entire system either.

Considering the above definition, four essential elements in design patters can be identified [35]:

  - Name: Pattern name is a handle that can be used to refer to a design problem, its solutions, and consequences.

- Problem: The problem is when and where the pattern should be applied. It explains the problem and its context.
- Solution: The solution describes the essentials of the design, how they are related and how they collaborate. This element doesn't provide a detailed design or implementation of a solution. Instead, it offers an abstraction of a design problem and describes a template like solution.
- Consequences: The consequences describe the results of applying the pattern and trade-offs of this application. They also help the pattern users understand costs and benefits of applying the pattern.

- **Componentry**

One of the essential characteristics of engineering disciplines is building a product by assembling pre-made, standard components [73]. Component Based Software Engineering (CBSE) has emerged in software development industry where reusability is receiving a lot of attention. CBSE is concerned with the development of software systems from reusable parts (components), the development of components, and system maintenance and improvement by means of component replacement or customization [18]. Constructing system from components and producing reusable components for different systems require specific technologies developed for this very purpose. The great interest in CBSE in recent years has resulted in emergence of several component development, integration and deployment technologies. The most common and widely used examples are Object Management Group (OMG)'s Common Object Request Broker Architecture (CORBA) Component Model (CCM),

Sun's Enterprise JavaBeans (EJB), and Microsoft's Distributed Component Object Model (DCOM) [34].

CORBA CCM, EJB, and DCOM are similar in providing a platform for developing, integrating, and deploying distributed components. All of these technologies try to provide a degree of relief for software developers from concerns such as component location, programming language used to create the component, operating system on which component works, communication protocol used to contact component, or hardware platform component works on.

In CBSD, component developers encounter certain risks and challenges in developing components, managing component development projects, and subsequently marketing the components. Application assemblers' risks and challenges primarily concern the assembly of components in applications, the management of component based application assembly projects, and the uncertainties of the component market. Customers of component based systems face both risks and challenges in using component based applications to meet their enterprise requirements, as well as in managing their component based and legacy application systems and in achieving and sustaining strategic competitive advantage over their rivals. [93]

- **Generative Reuse**

The concept of generative reuse is tightly coupled to the domain engineering process. Generative reuse is done by encoding domain knowledge and relevant system building knowledge into a domain specific application generator [34]. To build a new system in the target domain, developers begin by specifying a very limited prototype

of the desired application using a domain specific specification language. The specification is then translated into code by the generator. In an iterative process, developers incrementally extend and modify the prototype until it meets all of the requirements [34, 40]. The processes of specification translation and code generation can be entirely automated, or may require manual intervention.

Generative reuse has some distinct characteristics compared to more conventional software development methodologies:

- No explicit coding phase: The software specification is literally turned into executable code using the domain knowledge that is already gathered. This is done by the application generator so developers are not involved in writing executable code.
- Simplified testing and maintenance: The system specification is expressed using a [formal] specification language; therefore it is at a level of abstraction much higher than a program written in a conventional programming language. Since developers are dealing with a higher level of abstraction, testing and maintenance processes are easier to manage.
- Better documentation: Documentation is aided by the fact that the product of developers, which in this case is system specification, is semantically rich and easily readable. Using formal specification in the process increases the level of understandability and may provide the option of generating some documents automatically.

- More direct programming by end users: The high level nature of the specification language may make it possible for less sophisticated end users to contribute directly to programming.

Application generators are appropriate in domains where many similar software systems are written, one software system is modified or rewritten many times during its lifetime, or many prototypes of a system are necessary to converge on a usable product [50]. An important part of making domain engineering repeatable is a clear mapping between the outputs of domain analysis and the inputs required to build application generators [34]. Using the output of domain analysis, the commonalities in the domain are identified and they are implemented once when the application generator is built and then reused each time a software system is built using the application generator.

- **Collaboration**

Over the past decade, designing and implementing processes have been witnessing a distinct move away from individual decision makers and towards groups engaged in collaborative work [22]. Greater competition in the market and globalization of economy has changed the structure of many organizations and these changes are more evident in SEOs. The new structures have intensified the demands for higher levels of support of distributed and collaborative work [10].

Inter-organization and intra-organization collaborations encourage the emergence of standards and pseudo-standards. When entities collaborate, they try to help each other achieve their goals. To maximize the benefits of collaboration, the participants need

to agree on a set of rules for their cooperation and the results of their work. The rules governing the collaboration defines how participants work together and the rules about the results of cooperation defines how results should be produced and in what form and shape and under what conditions they should be presented. All participants need to agree with these rules for collaboration to work. If this agreement is done in a formal fashion, the result can be called a standard. If the agreement among parties is not formal, the results can be looked at as a pseudo-standard. The fact that standards play an important role is well known in many industries and the same is also true for software engineering industry.

- **Version Management**

Version Management is part of Software Configuration Management (SCM). SCM is defined as the control of the evolution of complex systems. More pragmatically, it is the discipline that enables us to keep evolving software products under control, and thus contributes to satisfying quality and delay constraints [30].

SCM can be utilized as a management support discipline. In this capacity, SCM is concerned with functionalities such as identification of components and their versions, change control (by establishing strict procedures to be followed when performing a change), status accounting (recording and reporting the status of components and change requests), and audit and review (quality assurance functions to preserve consistency) [17]. Each SCM method relies on a version model. A version model defines the objects to be versioned, version identification and organization, as well as operations for retrieving existing versions and constructing new versions [17].

Objects that are monitored by version management process and their relationships constitute the product space. The version of these object make up the version space. A specific version model is characterized by the way the version space is structured, by the decision of which objects are versioned, by the relationships among version spaces, and by the way reconstruction of old and construction of new versions are supported [17].

Management of change through versions is an integral part of our framework and is a very important feature of the management of the framework data model.

Appendix D: Participants of CIMSAN Program

- Appiq
- BMC Software
- Brocade
- Computer Associates
- Crossroads
- EMC
- Hewlett-Packard
- Hitachi Data Systems
- IBM (Tivoli)
- Inrange
- InterSAN
- LSI Logic
- McData
- Network Appliance
- Prisa Networks (now EMC)
- Qlogic
- Quantum
- StorageTek
- SUN Microsystems
- Veritas

Appendix E: Core Layer of the Prototype

In this appendix, we present a more detailed version of class diagram presented in Figure 9. We also present the interfaces of the following classes of Core Layer:

- Object
- AminoAcid
- Nucleotide
- Sequence
- ReferenceTable

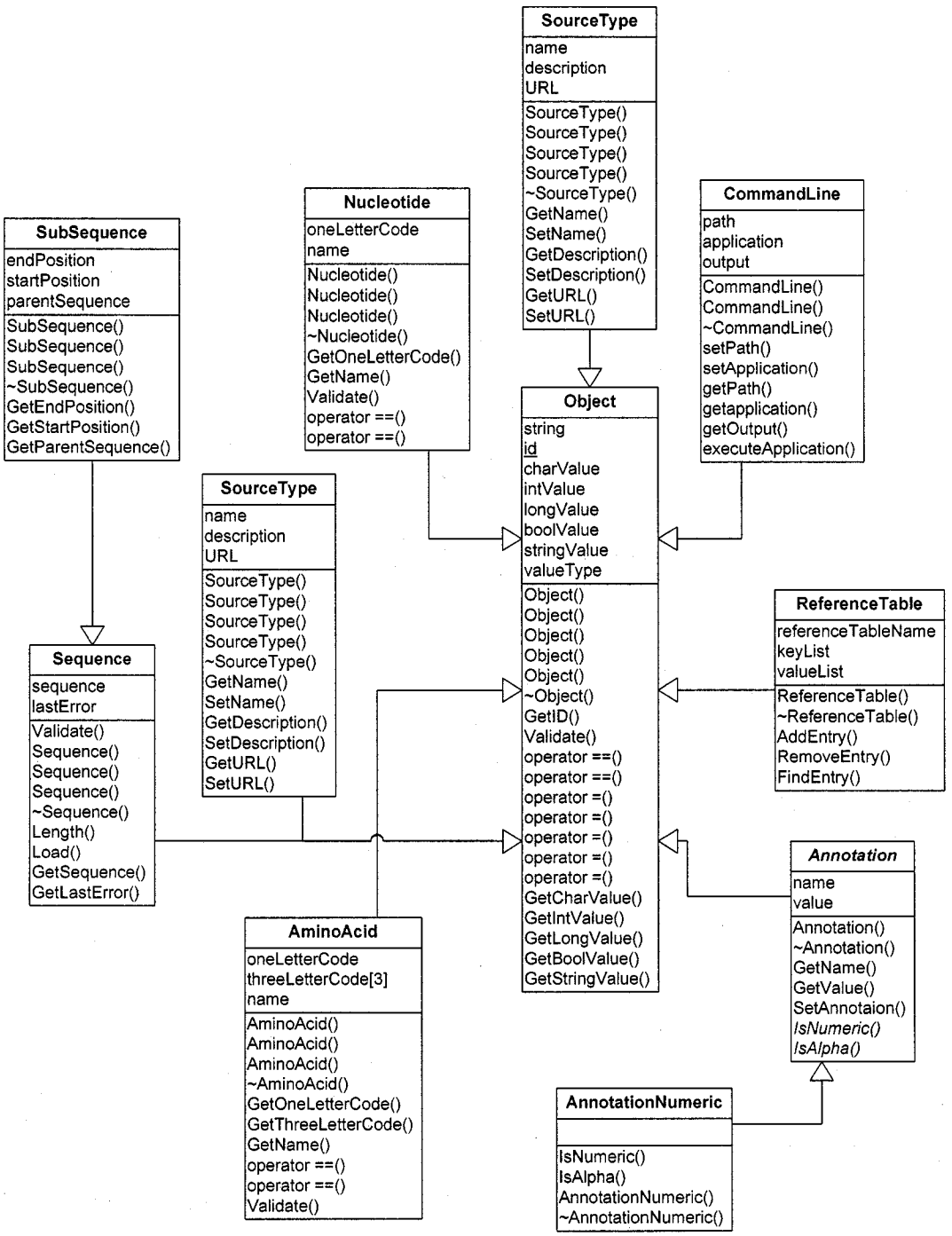

Figure 12: Simplified Class Diagram of Core Layer of the Prototype (detailed version of Figure 9)

- C++ declaration of "Object" Class

```
class Object
{
public:
```

```
        Object(void);
        Object(char);
        Object(int);
        Object(long);
        Object(bool);
        Object(string);
        virtual ~Object(void);
        long GetID() {return id;}
        virtual bool Validate() {return true;}
        virtual bool operator==(Object& a);
        virtual bool operator==(const Object& a);
        virtual char operator=(char a);
        virtual int operator=(int a);
        virtual long operator=(long a);
        virtual bool operator=(bool a);
        virtual string operator=(string a);
        virtual char GetCharValue();
        virtual int GetIntValue();
        virtual long GetLongValue();
        virtual bool GetBoolValue();
        virtual string GetStringValue();

    private:
        static long id;
        char charValue;
        int intValue;
        long longValue;
        bool boolValue;
        string stringValue;
        VALUETYPE valueType;
    };
```

- C++ declaration of "AminoAcid" Class

```
class AminoAcid: public Object
{
public:
        AminoAcid(char oneLetterCodeP, char threeLeterCodeP[],
string& nameP);
        AminoAcid(const AminoAcid& a);
        AminoAcid(char oneLetterCodeP);
        virtual ~AminoAcid();
        char  GetOneLetterCode() const;
        const char* GetThreeLetterCode() const;
        const string&     GetName() const;
        virtual bool      operator==(AminoAcid& a);
        virtual bool      operator==(const AminoAcid& a);
        virtual bool Validate();

    protected:
        char  oneLetterCode;
        char  threeLetterCode[3];
        string      name;

    };
```

- C++ declaration of "Nucleotide" Class

```
class Nucleotide: public Object
{
public:
      Nucleotide(char, string&);
      Nucleotide(char);
      Nucleotide(Nucleotide&);
      virtual ~Nucleotide();
      char  GetOneLetterCode() const;
      const string&     GetName() const;
      virtual bool Validate();
      bool operator==(Nucleotide& a) ;
      bool operator==(const Nucleotide& a);

protected:
      char  oneLetterCode;
      string      name;
};
```

- C++ declaration of "Sequence" Class

```
class Sequence: public Object
{
public:
      Sequence();
      Sequence(Sequence& sequenceP);
      Sequence(list<Object>& sequenceP);
      virtual ~Sequence();

      long Length();
      virtual bool Load(string URL, SourceType sourceType);
      list<Object>& GetSequence();
      string&     GetLastError();

      virtual bool Validate();

protected:
      list<Object> sequence;
      string lastError;
};
```

- C++ declaration of "ReferenceTable" Class

```
class ReferenceTable: public Object
{
public:
      ReferenceTable(string name);
      virtual ~ReferenceTable();

      virtual bool AddEntry(Object& key, Object& value);
      virtual bool RemoveEntry(Object& key);
      virtual Object* FindEntry(Object& key);

protected:
      string referenceTableName;
      list<Object> keyList;
      list<Object> valueList;
};
```