

Knowledge Management for Computational Intelligence Systems*

Rosina Weber, Duanqing Wu

College of Information Science and Technology, Drexel University Philadelphia, USA

Rosina.Weber@drexel.edu

Abstract

Computer systems do not learn from previous experiences unless they are designed for this purpose. Computational intelligence systems (CIS) are inherently capable of dealing with imprecise contexts, creating a new solution in each new execution. Therefore, every execution of a CIS is valuable to be learned. We describe an architecture for designing CIS that includes a knowledge management (KM) framework, allowing the system to learn from its own experiences, and those learned in external contexts. This framework makes the system flexible and adaptable so it evolves, guaranteeing high levels of reliability when performing in a dynamic world. This KM framework is being incorporated into the computational intelligence tool for software testing at National Institute for Systems Test and Productivity. This paper introduces the framework describing the two underlying methodologies it uses, i.e. case-based reasoning and monitored distribution; it also details the motivation and requirements for incorporating the framework into CIS.

I. Introduction

Knowledge management (KM) comprises a large umbrella of initiatives that focus on the rational allocation of knowledge assets from the perspective of humans, organizations and computer systems. KM is typically implemented through the performance of knowledge tasks, e.g. create, distribute, reuse. Computational intelligence systems (CIS) use a variety of techniques, e.g. evolutionary computing, to derive solutions to real world problems. They make good candidates for a KM approach because they build new solutions at every execution. In this paper, we introduce a KM framework that performs knowledge tasks to streamline the performance of CIS.

Intelligent systems in general only learn from experience when they are designed with this specific purpose. Some learning systems are designed to learn from inputs but not from their own executions. Computer systems that deliver tasks interfacing with a dynamic environment can only be considered reliable if they are prepared to learn, adapt, and evolve. The KM framework

we present allows CIS to learn, adapt, and evolve; potentially resulting in continuous improvement and increased reliability because it is designed to enhance a system's capabilities. Managing knowledge in CIS means giving these systems the ability to learn from their own executions. The KM framework represents an additional effort to guarantee a system performs as required; therefore, reaching the core of high assurance [1]. In addition, systems engineering pursues high assurance in systems interfacing with a dynamic world where task environments evolve. Consequently, enabling systems to respond to dynamic environments and behave in conformity with the context's changes is beneficial to high assurance systems engineering.

The system that incorporates the proposed KM framework evolves because it observes its executions and uses metrics to evaluate its performance. For example, in a CIS that trains an artificial neural network (ANN), its accuracy can be used as a measure of its performance. We describe an example (Section V B) where these observations can help improve the quality and efficiency of the system. The resulting system can be configured to submit every new thing it learns to be validated by humans, so it will not act in unexpected ways. The architecture keeps the KM framework independent from the core system.

The KM framework manages knowledge bases of different scopes. In addition to storing entire executions, it also includes a module to handle lessons-learned (LL), i.e. pieces of knowledge that teach a strategy to improve individual tasks. Lessons can be learned in any context and the system will be able to incorporate them.

The KM framework is currently being implemented for the first time and hence it has not yet been validated. However, the framework uses two validated methods: case-based reasoning (CBR) and monitored distribution (MD). Both these methods have demonstrated their effectiveness and the wide range of techniques developed for CBR substantiates the application of the KM framework. For example, maintenance methods studied for CBR provide many alternatives for case base maintenance (Section VI).

In Section II, we present the motivation for this work, showing why CIS are amenable to KM. Section III describes the underlying methodologies used in the KM framework. We describe the framework in Section IV and

* Eighth IEEE International Symposium on High Assurance Systems Engineering (HASE 2004), 116-125. IEEE Computer Society: Los Alamitos, CA.

in Section V detail requirements for its integration and an example. Section VI discusses some issues related to managing expertise and covers related work particularly with respect to maintenance. Section VII presents a summary and future work.

II. Motivation

Knowledge management (KM) refers to the rational allocation of knowledge assets by means of effective and efficient organizing, planning, leading, controlling, and coordination. Typical KM solutions are described in terms of a knowledge cycle that entails knowledge tasks such as capture, distribution, and reuse [2]. KM goals are also described in terms of knowledge, such as knowledge sharing and leveraging. In fact, KM goes far beyond knowledge. It refers to a number of human abilities (henceforth referred to as KM abilities) that allow them to interface with a dynamic world, learn, evolve, adapt, and keep performing tasks they are intended to deliver.

Recent interest in this field has shown that although humans are equipped with a series of KM abilities that allow them to adjust to the world's changing conditions; they lose these abilities when organized in systems. Such a fact represents one of the KM's biggest challenges, i.e. transferring individual KM abilities to organizational contexts. Not surprisingly, few strategies have resulted in success (e.g. Senge [3]). The problem is that systemic KM outside humans has to be artificially conceived, implemented and managed to succeed. One of the difficulties is in trying to incorporate KM processes into existing systems i.e. that were conceived without it. Better results can be obtained when KM processes are part of the original and integral design and development of systems. Although challenging to conquer, KM abilities allow systems to learn, evolve, adapt, and successfully perform in the context of a dynamic world.

Similar challenges are faced by computer systems designed to deliver tasks in the context of the same dynamic world. Therefore, it is reasonable to assume that computer systems can also benefit from KM strategies. The needs and respective benefits are directly proportional to the complexity of the system's task and to the assurance levels a problem context requires. A reliable computer system should be able to learn, evolve, and adapt in order to guarantee its successful performance in the context of a dynamic world.

The simplest form of KM in a computer system occurs when it is maintained. Reasons for maintenance may originate from flaws or changing conditions. When a computer system monitors its own performance and is able to learn from it, it can guarantee longer periods of response without the need for maintenance. This self-monitoring also gives the system the ability to recognize when it fails and cannot learn, flagging its need for maintenance. Fast adaptation to changing conditions has the potential to

increase assurance levels, justifying the incorporation of KM strategies into high assurance systems.

A. Why CIS need Knowledge Management?

Computational intelligence (CI) is an emerging paradigm of information processing aimed at the design of highly intelligent systems [4]. Although there is no consensus on the definition of CI [5], there is a widely accepted view on what components CI should have [4][6], namely evolutionary computing, fuzzy computing, and neurocomputing. Consequently, CIS are characterized by a variety of methods, e.g. data-intensive learning methods, evolutionary methods, knowledge-based methods, that perform complex tasks. Contrasting traditional, logic-based, top-down artificial intelligence methods, CI methods are generally bottom-up, exploiting tolerance for imprecision, uncertainty, robustness, partial truth to achieve tractability, and better rapport with reality [6]. In CI methods, order and structure emerge from an unstructured beginning. These features and the ability to perform intelligent tasks make CIS even more suitable for KM strategies because experience can be directly reused in the performance of complex tasks.

Some evidence revealing why CIS need KM resides on the parameter configuration of CI methods, e.g. genetic algorithms (GA), artificial neural networks (ANNs). As inputs vary, the parameter configuration should also vary to conform to each input. However, there are no means available to find the optimum parameterization of operators for GAs [7] or no standard way of finding the optimal architecture for ANNs [8]. Hybrid algorithms have been designed to find the optimal design for ANNs in [9]. The benefit of learning from experience is that previous experiences can help in finding a more appropriate parameter configuration, potentially decreasing the chance of failure.

Table 1. Impact for different mutation rates

Mutation Rate	BestX	%OffOptimal	BestY	%OffOptimal
0.10	1.85	0	2.85	0
0.20	1.85	0	2.85	0
0.30	1.85	0	2.85	0
0.40	1.849	-0.05	2.849	-0.04
0.50	1.852	0.11	2.848	-0.07
0.60	1.857	0.38	2.807	-1.51
0.70	1.86	0.54	2.766	-2.95
0.80	1.63	11.9	2.325	-18.4
0.90	1.236	33.2	2.127	-25.4

Table 1 shows an example of the benefit that experience can bring to the definition of parameters in CI methods. We used a genetic algorithm to search for the global maximum of a given graph (<http://ai.bpa.arizona.edu/~mramsey/ga.html>). The required parameters to run the algorithm are number of genes, number of agents, crossover rate, and mutation rate.

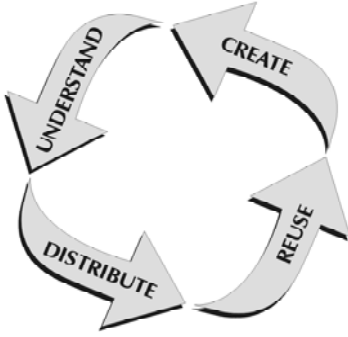


Figure 1. Conceptual KM process [2]

The results obtained represent values for the two axes x and y and the percentage of their difference from the optimal values. The results in Table 1 demonstrate how the different values for mutation rate impact the final

results. This parameter specifies what percentage of genes in the population will be replaced each generation. A value that is too high may affect the accuracy of the results; on the other hand, a value that is too low may not keep the population diverse enough and lead to premature results. In order to select an appropriate value for mutation rate, we collected the search results by setting different mutation rates. As the results show in this example, appropriate values should be between .2 and .4. As this example suggests, the use of different experiences can help finding proper parameter configurations. This indicates that GA methods have a knowledge need, and therefore can benefit from a KM approach. Given the difficulty of determining parameters for ANN, the benefit also applies to ANN methods.

Information systems (IS) can also benefit from incorporating a KM strategy into their design. The idea of learning its own executions to improve efficiency in databases has already been studied in [10]. Although traditional IS are meant to produce information rather than tasks, they can also benefit from KM. One important element missing today in IS of high consequence (e.g. hospital IS) is a sound security strategy. Developing and incorporating a security strategy into IS implies complex tasks that can benefit from useful knowledge.

In the essence of high assurance systems is the ability to guarantee that a given system will behave the way it is expected [1]. Hence, computational intelligence techniques can increase assurance to systems engineering, given their ability to handle knowledge and data in a coherent manner [4]. Along these lines, a KM approach that oversees the intensive flow of solutions derived in CI systems, represents an additional instrument to guarantee coherence, resulting in higher quality software.

The same caveat from human-based systems applies to computational intelligent systems: *the KM strategy has to be incorporated into system design*. Attempts to integrate a KM framework after a system is developed endanger the final intended result. The only exception is to systems whose architecture is designed to grow, such as research-oriented architectures. An example of such architecture is given in Section V, Subsection B.

III. Background

The KM framework we introduce in this paper is currently under development and therefore is yet to be validated. Nevertheless, this KM framework combines well validated elements of two methodologies: case-based reasoning [10] and monitored distribution [12][13]. This section describes the underlying concepts and methods used to implement the KM framework.

A. Conceptual Knowledge Management Process

KM solutions are typically presented through KM processes that detail knowledge tasks. An analysis of different KM processes described in the context of technological KM solutions resulted in the conceptual cycle [2] presented in Figure 1. It consists of the tasks *create*, *understand*, *distribute* and *reuse*.

The *create* task refers to applying different methods to collect or generate knowledge (and information) within the application's context. The *understand* task is responsible for performing all necessary steps (e.g. validate, represent, store) to make collected knowledge ready to be distributed. The *distribute* task matches stored knowledge to the knowledge needs of its proper recipients. The *reuse* task oversees that knowledge is properly reapplied back into the application's context.

This conceptual process has many tasks in common with the CBR cycle. Next, we describe the CBR methodology and then discuss its use to support KM solutions. CBR is the underlying methodology for knowledge storage (i.e. understand). MD supports distribution and reuse.

B. Case-based reasoning

Case-based reasoning [10][14][15][16] is a reasoning methodology [17] inspired by the human process of remembering a previous similar episode to solve a new problem. The act of being reminded of a previous episode is modeled in case-based reasoners by comparing a new problem with a collection of stored cases (the *case base*), often based on indexes describing the contents of the stored cases. The most similar cases are then retrieved, and can be used as references to classify the new case or the solutions from the retrieved similar case(s) can be adapted to fit the new problem. If the adaptation results successful, a new case has been created and is retained in the case base. However, adaptation is one way of acquiring cases. Other case bases consist exclusively of real experiences, where adapted cases are not learned. Cases can also describe prototypical situations or be artificially authored.

There are four basic steps in the CBR process [16]: retrieve, reuse, revise and retain. Applications that include iterative learning will also include a review step [18], as illustrated in Figure 2.

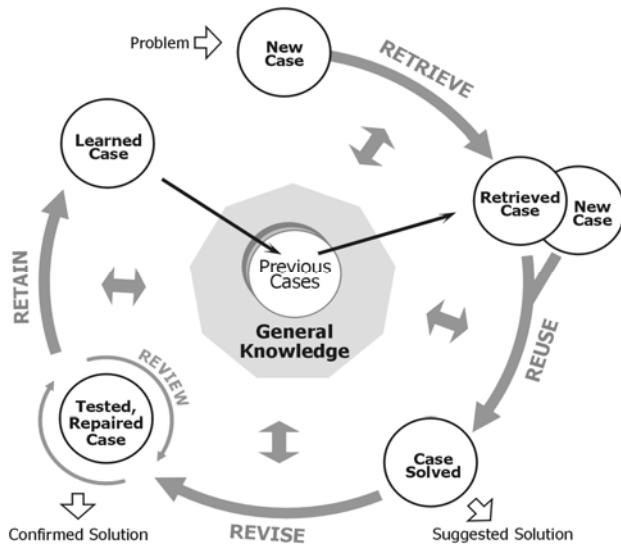


Figure 2. The CBR cycle

Indexing is the essence of case-based reasoning as it guides similarity assessment and retrieval. The collection of indexes model the answer for the question, “*What makes a case similar to another, such that the solution of one can be reused in the other?*”, representing the relevancy of the cases. Indexing determines what has to be compared between cases to assess their similarity for retrieval. Retrieval allows the reasoning task: retrieving cases with usefulness to solve or interpret the new case.

Another way to look at CBR systems is by examining knowledge representation methods involved in building a case-based reasoner. Richter [19] defined CBR systems as a reasoning method that uses four knowledge containers (KC). The four KC, sketched in Figure 3, are vocabulary, similarity measure, case base, and solution transformation.

The vocabulary includes attributes and predicates used to describe cases in different levels of abstraction. The similarity measure includes indexing, similarity functions, aggregation of similarities, and selection; also including methods to learn similarity measures, feature weights, and indexing. The case base includes case base maintenance, case acquisition, and the organizational structure of cases, which can use a flat scheme, hierarchies or networks. The solution transformation refers to knowledge used in the reuse and revise steps of the CBR cycle. The automatic adaptation of cases entails the previous determination of incoming problems and the expected contrasts between stored cases and new problems. The contrasts can be covered by new knowledge added to the system or by modifications to the knowledge.

Learning in Case-Based Reasoning. Case-based reasoners can learn by acquiring new knowledge in all of its knowledge containers. However, the most frequent targets of learning are the cases and case bases. Learning

in cases occurs as an expression of learning from experience. The outcome part of a case is designated with this goal. The case keeps records of its performance when used to attempt to solve new situations. Thus, both successes and failures can be added, enhancing the knowledge and the lessons embedded in a case. The record of the outcome resulting from reusing a given case can warn the user of possible consequences in its reuse. This process is worthwhile because it prevents the system from reusing less indicated suggestions.

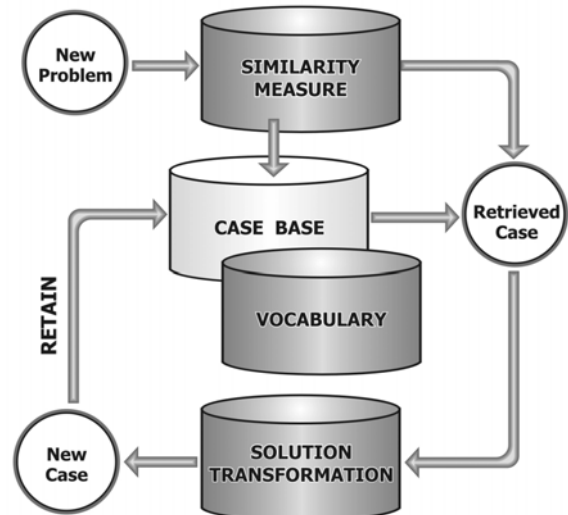


Figure 3. CBR knowledge containers

Case bases grow through incremental learning if the task and design of the system allows. From a limited set of seed cases [15], the case base can be augmented with new cases. The generation of these new cases stems from new inputs from users or from external sources. In problem-solving systems, new cases may undergo adaptation before they are added to the case base. In interpretive systems, new cases are added after proper indexing.

Since the first case-based reasoner ever developed [10], CBR has become a well established reasoning methodology [14][15][16][18]. Today, there are several significant deployed applications using CBR and a great variety of techniques have been developed. Conversational case-based reasoning (CCBR) [20] is the underlying methodology of one the most successful tools for call centers in the market today: *eGain Knowledge Agent* (www.egain.com). A number of sophisticated CBR systems have been developed and deployed at GE Transportation Systems, GE Medical Systems and GE Aircraft Engines [21].

Case-Based Reasoning and Knowledge Management. CBR is often recommended for KM tasks [22]. Among some of the possible reasons are the commonalities between the CBR cycle and the KM process [23] and the fact that CBR uses different techniques to manage a set of

knowledge containers [19]. In addition, CBR is a methodology [17], and poses less engineering requirements than techniques that represent knowledge explicitly and exhaustively (e.g. rule-based) [24][25]. And yet, the real evidence is in the number of KM solutions that use CBR [2][27]. Recently, several members of the CBR community have selected two research publications as the most representative illustrations of the importance of CBR to KM [27][28].

Table 2. Example of a lesson-learned

<i>Indexing Elements</i>	<i>Reuse Elements</i>
<u>Applicable task</u> : training ANN;	<u>Lesson suggestion</u> : check range of values for each part of field to verify if date format in data from foreign country is the same so results are interpreted the same way;
<u>Preconditions</u> : 1) input uses data from foreign country; 2) data contain field format date.	<u>Rationale</u> : data format varies from MM/DD/YYYY to DD/MM/YYYY <u>Outcome</u> : This lesson was used on 12/03/2003 at 10h04min on input case register 83245. The format date did not need to be changed.

C. Monitored Distribution

Monitored Distribution [12][13] is an approach for the proactive distribution of knowledge artifacts [29]. Examples of knowledge artifacts are best practices, lessons-learned, and alerts. The knowledge artifacts that MD distributes are lessons-learned (LL).

LL teach a strategy to perform a task, resulting in a positive impact on that task [30][28]. It may indicate how to avoid a problem or simply how to proceed in a way that may be safer, cheaper, or faster. It is a requirement that LL positively impact the task they are applied to; this forces a strong association between the lesson and its applicable task. This association demands that lessons are only retrieved in the context of the task that they apply to. LL also include preconditions for applicability. MD monitors when LL should be distributed by matching the lesson to the retrieval's context. MD's distribution module relies on CBR methodology.

A representation for LL is exemplified in Table 2, which combines indexing elements (i.e. applicable task, preconditions) and reuse elements (i.e. lesson suggestion, rationale, and outcome). This structure is responsible for facilitating the process of creating, understanding, distributing, and reusing LL. The indexing elements are used by MD to assess the similarity of a lesson with a context in order to guarantee a retrieval that is oriented to its applicability. LL are only retrieved if the context's current task and state match the applicable task and the preconditions. Lesson suggestion provides the decision or action recommended by the lesson, which replaces or modifies the original and expected decision that would be taken had the lesson not been distributed. The rationale describes reasons to store and reuse a lesson. The outcome

records the usage of the lesson, giving it another opportunity for the system to learn.

Empirical Validation. The impact of the MD approach was demonstrated in an empirical experiment [12][13] that planned military operations with and without the reuse of LL. The result of the experiment was that plans generated with the reuse of lesson-learned have reduced the number of friendly casualties by 30%, reduced the number of casualties among civilians being rescued by 24%, decreased the total time of the operation by 18%, and increased the number of casualties among enemies by 2%.

IV. KM Framework

This section describes the KM framework incorporated into the architecture of CIS. The framework can potentially be beneficial to other kinds of computer systems, but here the description is limited to CIS.

The framework consists of two modules interconnected to the target CIS (Figure 4). Both modules consist of a case base container and four KM processes. The top module is designated to describe experiences of executing the CIS in the main case base (MCB). The bottom is a lessons-learned module that describes lessons applicable to the CIS in the LL base. The top module manages experiences that reflect entire executions of the CIS. The bottom module manages experiences that are useful to individual tasks throughout the CIS.

The KM framework encircles the CIS with two modules having the same basic structure: a case base container and four processes: *creator*, *distributor*, *reuser*, and *understander*. The base containers can include several case bases, depending on the system's scope. For example, if the CIS produces outputs of distinct natures, two different MCBs may be used. If, for the same output, the CIS offers choice of methods, then additional individual case bases (ICB) may be added to store different variations of each method. Analogously, the use of two separate LL bases may be more efficient for two distinct task environments.

A. Case Bases

The MCB stores entire executions of the CIS. One MCB is needed for each main type of output the CIS produces. For example, if the CIS uses artificial neural networks (ANN) for both diagnosis and routing, the nature of the inputs and the task is sufficiently distinct to require a separate MCB. In addition to one or more main case bases, the use of additional ICBs may be necessary. For example, suppose the training for classification can be executed using two variations of the ANN architecture. In this case, an additional ICB will store data about additional executions using different methods. The execution stored in the MCB can be determined by metrics or by customization. The purpose of separating main and

individual case bases is to define parameters for similarity assessment.

B. Processes

The four processes *creator*, *understander*, *distributor*, and *reuser* are in the same context as the case bases where they exchange data. These processes are described next.

Creator: This process is responsible for acquiring data to populate the case bases. The creator collects data for both case base containers. For the MCBs and the ICBs, it collects data from the CIS about its inputs, parameters used and produced during its execution, and its outputs. Creator also collects lessons-learned for the LL base. It communicates with knowledge engineers (KE) so they can verify what has been collected.

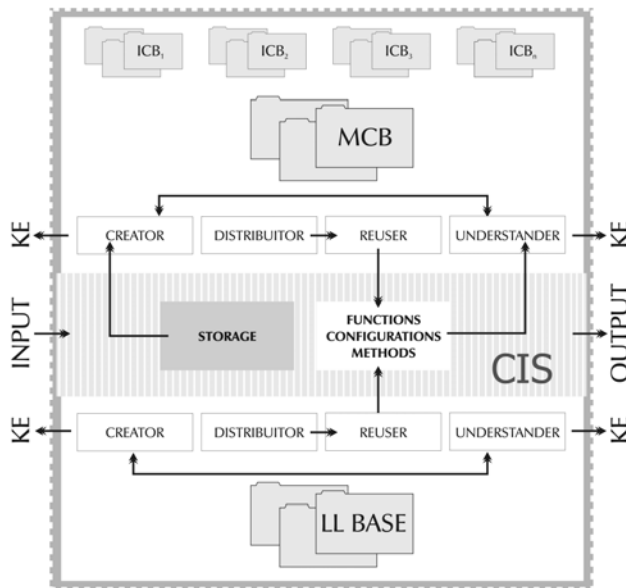


Figure 4. Architecture: CIS + KM framework

Understander: This process refers to all methods designed to prepare the data acquired by the creator process. The understander process has to prepare the data and verify its status as knowledge, i.e. its usefulness to solve problems. Understander prepares the data to be used as knowledge by assigning similarity functions and weights for each case feature. Understander is also responsible for computing maintenance metrics that determine the stage of life cycle in which a case base belongs. The life cycle stage determines maintenance requirements. The understander converses with knowledge engineers and additional CI methods. It is a design decision to determine the extent to which it is worth keeping these additional algorithms within the architecture

or let the KE compute them and enter results directly to the understander.

Distributor. This process accounts for retrieving relevant and applicable knowledge to all knowledge needs of the CIS. The distributor uses the similarity parameters defined by the understander to match cases from all case bases (i.e. MCB, ICB, LLB) to processes that correspond to knowledge needs. The identification of knowledge needs is discussed in Section V A. For example, before training an ANN, the CIS has to define a configuration of the ANN's parameters. This step represents a knowledge need because there may be knowledge available that allows the system to define the most appropriate parameters for a given input. If this knowledge is in fact available, it is stored in a case and will be retrieved when the CIS is performing this step and the input characteristics match the ones described in the case. Then, distributor calls the reuser process.

Reuser. This process triggers commands stated by knowledge retrieved by the distributor process. In the example given above, the reuser process takes the parameters and assigns them to the ANN. The reuser will also assess the need for adaptation, triggering proper methods accordingly.

C. Interfaces

The inclusion of the framework also requires additional interfaces so KE can verify data processed by the different processes, e.g. creator, understander. Another interface is needed for entering LL directly into the LLB.

Internally, the KM framework interfaces with the CIS through its processes. Figure 4 shows arrows representing the flow of data, information, and knowledge between CIS and the framework's processes.

The interference of KE characterizes the first two of a three-phase implementation of the framework. The implementation phases are described in the next section.

V. Integrating the KM framework

The integration of the KM framework into a CIS design becomes part of its development life cycle. The KM framework's development life-cycle has three stages: childhood, adolescence, and maturity. The first two phases, childhood and adolescence are characterized by full control from the knowledge engineers. During childhood, the methods and processes are implemented and tested and the first experiences are collected. In adolescence, the system is making decisions trying to find assurance of the way it learns. This is when similarity and maintenance methods are defined. In maturity, minimum human interference is needed because if the system needs maintenance, proper maintenance methods will determine the need and the system will flag it.

A. Requirements for Integration

First of all, it is necessary to assess the potential benefits of integrating a KM framework into any system. The most important question to answer is whether the system performs tasks that can be improved. For example, when task environments change, requiring small variations in the way tasks are performed. If a system includes tasks that are amenable to improvement, then the system is suitable for a KM approach.

The most important requirement to integrate the KM approach is to identify where and when these tasks are performed. The moment of performing a task that is amenable to KM represents a knowledge need. All knowledge needs of the system must be identified a priori in order to integrate the KM framework that will focus on these knowledge needs. The next subsection describes an example where this aspect can be better understood.

Finally, it is essential that the KM framework be integrated into the design of CIS. The only exception to this requirement is the case of flexible architectures that are designed to be constantly changing and therefore are flexible enough to incorporate the KM processes without jeopardizing its effectiveness. The CI-Tool is an example of such flexible architecture.

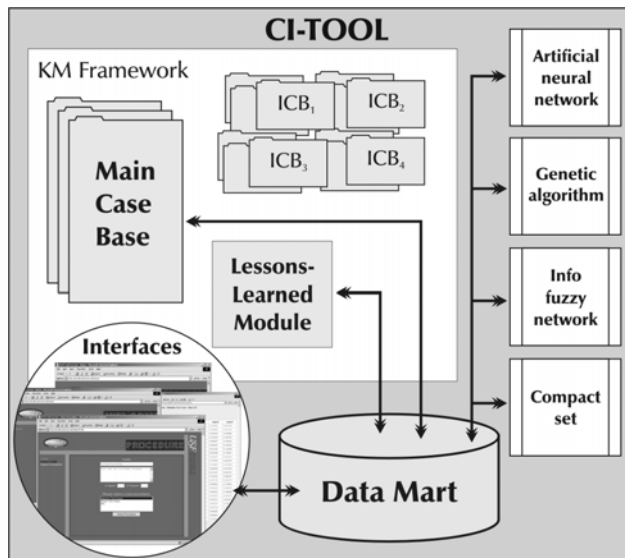


Figure 5. CI-Tool design including KM framework

B. KM Framework in the CI-Tool

The KM framework is being integrated into a computational intelligence tool for software testing (CI-Tool) currently under development at the National Institute for System Test and Productivity (NISTP) at the University of South Florida. CI-Tool uses different CI methods to support testing of software programs. Figure 5

shows the resulting architecture of the CI-Tool contemplating the KM framework.

CI-Tool meets the most important requirements to use knowledge management: it performs tasks in changing environments, uses multiple CI methods to perform them, and exhibits flexible design architecture. The changing task environments in the CI-Tool originate from the different nature of software to be tested. Some of the CI methods used by the CI-Tool are genetic algorithms and artificial neural networks. Its design architecture characterizes the development of a tool constantly ready to incorporate the latest research findings.

The CI-Tool has knowledge needs while performing its main task and also within the CI methods it employs. For example, when calling a CI method, one specific parameter configuration may have demonstrated more effective in previous executions; or after generating test cases, lessons may be applicable to include an additional test case for a specific input.

The KM framework in the CI-Tool has one main case base (MCB), four individual case bases (ICB), and one lessons-learned base (LLB). It has one of each (i.e. MCB and LLB) because the current integration targets a version of the CI-Tool designed to perform one task, i.e. generation of test cases. The flexibility of the architecture of the CI-Tool remains with the incorporation of the KM framework. As the CI-Tool adds new tasks, new MCBs, ICBs and LLBs can be included.

The four ICBs displayed in the current implementation are respective to the four methods represented in Figure 5 by four process symbols. Although the detailing of these methods is outside the scope of this paper, they are currently used in the CI-Tool to generate test cases for software testing. The KM framework stores multiple experiences describing each of the methods solving different inputs. What distinguishes the ICB from the MCB is that the MCB stores experiences for each input with one and up to four methods. The value of the MCB is to preserve the experience of each input in order to unravel variable performance of the methods for each input. Each ICB stores experiences of one method only.

VI. Discussion and Related Work

Computational intelligence (CI) methods are designed to perform a great variety of complex tasks. Some of these tasks are associated with human thinking [31], and hence they embed a component of uncertainty that may result in two features: one is that their performance can improve with experience, and other is that there may be a number of small variations to the way they can be performed. Our assumption is that CI methods are amenable to knowledge management because of these features.

The CI-Tool is an example of a CIS for software testing (ST). ST is an example of domain expertise where experienced experts are more likely to design effective

testing strategies. Experience also helps human experts to adapt to changing task environments because it is through experience that humans learn, adapt, and evolve. The proposed KM strategy aims at these same goals so that CIS can deliver high levels of assurance.

The topic of human expertise replicated in computer systems has been extensively studied mostly in the context of expert system (ES) methodology. Expert systems [32] and, particularly, rule-based expert systems, have dealt with this challenge offering results that are somehow limited. The proposed KM framework offers a new perspective to replicating human expertise, one that originates from human KM abilities and how they improve the quality of tasks. Although we do not intend to compare the two methodologies here, it is interesting to highlight the issues that have limited the performance of rule-based expert systems and how they are addressed in this KM framework, namely knowledge acquisition and maintenance.

Knowledge acquisition in expert systems is conducted either as an initial step of development or as a continuous effort. An acquisition component is sometimes included in the basic description of the ES methodology, and knowledge is either elicited directly from experts or induced with machine learning methods. The worst problem in rule-based ES is that if new rules are acquired after system's development, these new rules will not have been included in the analysis of the domain and the influence these new rules may have over existing rules in the system is not captured. Such a characteristic means that typical ES do not meet one of the grounds for computer understanding, which says that computer programs should be able to learn new knowledge and integrate it to the overall knowledge of the system [32].

The KM framework introduced here relies substantially on case-based reasoning, and therefore knowledge is acquired in the same fashion as in case-based reasoners. After a CBR system is operational, additional knowledge is learned in a demand-driven fashion, as a lazy learner [18]. Because knowledge is applied as a result of similarity assessment and retrieval, new cases do not interfere with existing cases and therefore case-based reasoners are able to integrate new knowledge to the overall existing knowledge without demanding any additional effort. Case-based reasoners meet all grounds for computer understanding from [32].

Although able to incorporate new knowledge, case-based reasoners still need to employ maintenance to guarantee good levels of effectiveness and efficiency in the case bases. A system that can capture new knowledge imposes a third source of maintenance in addition to existing flaws and environment changes. We described earlier about the mechanisms the KM framework employs to adapt to a changing world. We now discuss previous work on maintenance methods for the framework.

Verification parameters are the simplest form to control the performance of a CBR system [14]. Some of these parameters are retrieval accuracy, which is verified by observing the retrieval result when the target case is part of the case base; retrieval consistency, case duplication case coverage, retrieval time and sorting, and consistency [14]. The constant verification of these parameters can suggest maintenance needs.

The research in maintenance methods for CBR offers a great variety of techniques. A noteworthy view of current state-of-the art in maintenance methods for CBR systems is given in [33]. The methods cover both tasks of determining whether maintenance is necessary and of defining maintenance strategies. Some strategies suggest ways to determine when knowledge is needed and then recommend artificial cases for addition, we plan to use these methods as a way to estimate future maintenance needs.

Significant contributions to CBR maintenance were provided by competence-based models. Competence represents how well a case base solves a collection of potential problems. These methods are surveyed in [34]. These methods would not directly apply to the KM framework because they are designed for case-based reasoners whose cases are artificially engineered and the KM framework is composed of cases that have actually been tested in the CIS. On the other hand, these methods can suggest problems to be executed in the CIS to complement sparse areas in the case base.

Some deletion methods can also produce benefits when maintaining the KM framework. Two useful deletion strategies are based on replacing a number of cases with one more general case and on keeping track of the usefulness of stored cases during problem solving [35].

A maintenance method that is directly useful in our framework aims at maintaining knowledge in the similarity measure [36]. First, it can be used because it allows refinement of feature weights. This is useful because we will start operating the case bases even before we have a sufficient number of cases to determine final similarity parameters. Another reason for choosing this method is that it has demonstrated its benefits when a case base expands due to changes in external conditions. Finally, this method uses genetic algorithms (GA), allowing us to extend the utility of the GA module to maintenance.

Other maintenance approaches that can make use of CI methods already included in the CIS are the ones that use artificial neural networks. Two research projects have focused on using ANN to select cases[37][38]. In [38], an ANN was used for clustering cases to identify redundant and representative cases within clusters. The resulting clusters are considered to be representative of solution prototypes. Each cluster is supposed to contain representative and redundant cases, which are determined by the computation of the case density with a similarity measure. The clusters are then reduced with the

elimination of its redundant cases. The combination of ANN-based methods with others mentioned above can potentially produce highly sophisticated maintenance results.

Finally, one last group of research in CBR maintenance refers to systems that have employed maintenance in CBR for KM purposes. Obsolescence and redundancy problems are discussed in [39] in the context of a deployed case base that grew too fast given the abundant availability of new cases. The author warns for the need for unnecessary maintenance when proper maintenance methods are not included in design and implementation phases. Maintenance methods for case bases describing real experiences have also been studied from a KM perspective in the context of the experience factory [40]. The authors have investigated the tasks associated with maintaining an experience repository from an organizational perspective and proposed a maintenance and evaluation framework for experience bases.

When adopting a KM framework, we seek for assurance, not presumption. Presumption may lead to acceptance of knowledge that is not necessarily ideal. Systems need to learn, evolve, and adapt as long as they are able to guarantee high assurance levels. An important benefit of using the CBR methodology is that research in the area has already produced a reasonable number of maintenance techniques allowing a severe control of the learned experiences.

VII. Summary and Future Work

This paper introduces a knowledge management (KM) framework for integration into the design of computational intelligence systems (CIS). The main advantages of adopting this KM strategy relate to giving CIS the ability to learn from its own executions and to adapt and evolve. The KM framework also allows CIS to incorporate experiences learned in external contexts. We described the underlying methodologies used by the KM framework and the requirements for its integration. An example is given showing the KM framework integrated into the design of NISTP's CI-Tool.

We briefly discuss the role of managing knowledge in intelligent systems and how it relates to expert systems and known challenges for expert tasks. The KM framework brings new hope in the attempt to deliver intelligent methods that can replicate human skills.

The KM framework introduced here is currently under development for the first time and therefore an empirical validation of its effectiveness will soon be possible. Because implementing this KM framework includes a three step life cycle, different validation studies can be conducted to evaluate each of its phases. The validation targeting specific phases allows conclusions and following steps to be drawn to address each phase.

Nevertheless, the KM framework presented here uses two well validated methods: monitored distribution (MD) and case-based reasoning (CBR). MD [12] has been demonstrated to improve the quality of the final result produced by the system it targets. MD contributes to the framework with its capability to improve a task result by supporting it with the right knowledge at the right time. CBR, on the other hand, contributes to the framework with a solid array of techniques developed and implemented in real world applications. In addition, the extensive literature in maintenance methods for CBR provides a sound foundation for the success of our framework. Besides, the possibility of leveraging CI methods (e.g. ANN, GA) performing main tasks to also support maintenance processes has the potential to characterize the resulting CIS as not only effective and reliable, but also efficient.

Acknowledgements

The authors would like to thank Dr. Mark Last for his helpful suggestions in the course of applying this framework to the CI-Tool. This work is supported in part by the National Institute for Systems Test and Productivity at USF under the USA Space and Naval Warfare Systems Command grant no. N00039-02-C-3244, for 2130 032 L0, 2002.

References

- [1] V. L. Winter and J. M. Boyle, "Proving Refinement Transformations for Deriving High-Assurance Software," in Proceedings IEEE High-Assurance Systems Engineering Workshop, New York:IEEE Press, 1996, pp. 68-77.
- [2] R. Weber and R. Kaplan, "Knowledge-based knowledge management," in Innovations in Knowledge Engineering, R. Jain, A. Abraham, C. Faucher and B.J. van der Zwaag, Eds. Adelaide: Advanced Knowledge International Pty Ltd, 2003.
- [3] P. M. Senge, "The fifth discipline," in Fieldbook: Strategies and Tools for Building a Learning Organization, New York: Doubleday, 1994.
- [4] W. Pedrycz, "Computational intelligence as an emerging paradigm of software engineering", in Proceedings of the 14th international conference on Software engineering and knowledge engineering, New York, NY:ACM Press, 2002, pp. 7-14.
- [5] J.C. Bezdek, "Computational intelligence defined -- by everyone," in Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications, O. Kaynak, L.A. Zadeh, B. Turksen, and I.J. Rudas, Eds. Berlin:Springer, 1998, pp.10-37.
- [6] L.A. Zadeh, "Roles of soft computing and fuzzy logic in the conception, design and deployment of information/intelligent systems", in Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications, O. Kaynak, L.A. Zadeh, B. Turksen, and I.J. Rudas, Eds. Springer, 1998, pp.1-9.
- [7] J.G. Dikalakis, and K.G. Margaritis, "An experimental study of benchmarking functions for genetic algorithms,"

- International Journal of Computer Mathematics, 79(4), pp. 403-416, 2002.
- [8] A. Kandel, P. Saraph and M. Last, Test Set Generation and Reduction with Artificial Neural Networks, in "Artificial Intelligence Methods in Software Testing", M. Last, et. al. (Eds.), World Scientific, Singapore, 2004.
- [9] A. Abraham and B. Nath, "Hybrid heuristics for optimal design of neural nets," in Proceedings of the Third International Conference on Recent Advances in Soft Computing, R. John and R. Birkenhead, Eds. Berlin: Springer Verlag, 2000, pp. 15-22.
- [10] N. Roussopoulos, C.M. Chen, S. Kelley, A. Delis, and Y. Papakonstantinou, "The Maryland ADMS project: views R us," Bulletin of the Technical Committee on Data Engineering, 18(2), pp.19-28, 1995.
- [11] J. Kolodner, Case-Based Reasoning. Los Altos, CA: Morgan Kaufmann, 1993.
- [12] R. Weber and D.W. Aha, "Intelligent delivery of military lessons learned," Decision Support Systems, 34(3), pp. 287-304, 2003.
- [13] D.W. Aha, R. Weber, H. Muñoz-Avila, L.A. Breslow, and K.M. Gupta, "Lesson distribution gap," in Proceedings of IJCAI, Menlo Park, CA: AAAI Press, 2001, 2, pp. 987-992.
- [14] I. Watson, Applying Case-Based Reasoning: Techniques for Enterprise Systems, San Francisco, California: Morgan Kaufmann Publishers, Inc., 1997.
- [15] D. Leake, Case-Based Reasoning: Experiences, Lessons, and Future Directions, Menlo Park, California: AAAI Press/The MIT Press, 1996.
- [16] A. Aamodt and E. Plaza, "Case-based reasoning: foundational issues, methodological variations, and system approaches," Artificial Intelligence Communications, 7 (1), pp. 39-59, 1994.
- [17] I. Watson, "CBR is a methodology not a technology," in The Knowledge Based Systems Journal, 12(5-6), UK: Elsevier, 1999, pp. 303-308.
- [18] D.W. Aha, "The omnipresence of case-based reasoning in science and application," Knowledge-Based Systems, 11(5-6), pp. 261-273, 1998.
- [19] M.M. Richter, The Knowledge Contained in Similarity Measures: Some remarks on the invited talk given at ICCBR'95 in Sesimbra, Portugal, 10/ 25/95. <http://www.cbr-web.org/documents/Richtericcbr95remarks.html>
- [20] D.W. Aha, L.A. Breslow, and H. Muñoz-Avila, "Conversational case-based reasoning," Applied Intelligence 14, pp. 9-32, 2001.
- [21] W. Cheatham, A. Varma, K. Goebel, "Case-based reasoning at General Electric," in Proceedings of the Fourteenth Annual Conference of the International Florida Artificial Intelligence Research Society, Menlo Park, CA: AAAI Press, 2001, pp. 93-97.
- [22] D.W. Aha, I. Becerra-Fernandez, F. Maurer and H. Muñoz-Avila, Eds. Exploring Synergies of Knowledge Management and Case-Based Reasoning: Papers from the AAAI 1999 Workshop (Tech. Rep. WS-99-10), Menlo Park, CA: AAAI Press, 1999.
- [23] I. Watson, "Knowledge management and case-based reasoning: a perfect match?" in Proc. of the Fourteenth Annual Conference of the International Florida Artificial Intelligence Research Society, I. Russel and J. Kolen, Eds. Menlo Park, CA: AAAI Press, 2001, pp. 118-122.
- [24] S. Slade, "Case-based reasoning: A research paradigm". AI Magazine Spring 1991, pp. 42-55.
- [25] C. Riesbeck, and R. Schank, "Inside case-based reasoning". 1989. Lawrence Erlbaum.
- [26] I.D. Watson, Applying knowledge management: techniques for building corporate memories, Amsterdam; Boston: Morgan Kaufmann, 2003.
- [27] K.-D. Althoff, A. Birk, G. von Wangenheim and C. Tautz, "Case-based reasoning for experimental software engineering," in Case-Based Reasoning Technology - From Foundations to Applications, M. Lenz, B. Bartsch-Spörl, H.-D. Burkhard, and S. Wess, Eds. Springer Verlag: LNAI 1400, 1998, pp. 235-254.
- [28] R. Weber, D.W. Aha, and I. Becerra-Fernandez, "Intelligent Lessons Learned Systems," International Journal of Expert Systems Research and Applications, 20, No. 1, pp. 17-34, 2001.
- [29] C.W. Holsapple and K.D. Joshi, "Organizational knowledge resources," Decision Support Systems, 31, pp. 39-54, 2001.
- [30] D. Fisher, S. Deshpande, and J. Livingston, Modeling the Lessons Learned Process (Research Report 123-11), Albuquerque, NM: The University of New Mexico, Department of Civil Engineering, 1998.
- [31] R. E. Bellman, An introduction to Artificial Intelligence: Can Computers Think? San Francisco: Boyd & Fraser Publishing Company, 1978.
- [32] P. Jackson, Introduction to Expert systems, Addison-Wesley, 1998.
- [33] D. B. Leake, B. Smyth, D. C. Wilson, Q. Yang, "Special issue on maintaining case-based reasoning systems," Computational Intelligence, 17(2), pp.193-195, 2001.
- [34] B. Smyth, E. McKenna, "Competence models and the maintenance problem," Computational Intelligence, 17(2), pp. 235-249, 2001.
- [35] L. Portinale and P. Torasso, "Case-base maintenance in a multimodal reasoning system," Computational Intelligence, 17(2), pp. 263-279, 2001.
- [36] S. Craw, J. Jarmulak and R. Rowe, "Maintaining retrieval knowledge in a case-base reasoning system," Computational Intelligence, 17(2), pp. 346-363, 2001.
- [37] R. K. De and S.K. Pal, "A neuro-fuzzy method for selecting cases," in Soft Computing in Case Based Reasoning, S.K. Pal, T.S. Dillon and D.S. Yeung, Eds. London: Springer Verlag, 2001, chapter 10.
- [38] S.C.K. Shiu, X.Z. Wang, and D.S. Yeung, "Neuro-fuzzy approach for maintaining case bases", in Soft Computing in Case Based Reasoning, S.K. Pal, T.S. Dillon and D.S. Yeung, Eds. London: Springer Verlag, 2001, chapter 11.
- [39] I. Watson, "A case study of maintenance of a commercially fielded case-based reasoning system," Computational Intelligence, 17(2), pp. 387-398, 2001.
- [40] M. Nick, K.-D. Althoff, C. Tautz, "Systematic Maintenance of Corporate Experience Repositories," Computational Intelligence, 17(2), pp. 364-386, 2001.