Journal of Computing and Information Technology - CIT 16, 2008, 4, 345–354
doi:10.2498/cit.1001404

Using a Typed Mind Map as a Data Model in a TDD DICE System

Li-Ren Chien and Daniel J. Buehrer

Department of Computer Science and Information Engineering, Chung Cheng University, Min-Hsing, Chia-Yi, Taiwan

This paper introduces a new Typed Mind Map extension for a data model in a parse-tree-based online referee system with a TDD (Test-Driven Development) model and DALM (DICE adaptive learning model) model named DICE. Typed Mind Maps and a semantic node are defined by OWL-DL. Typed Mind Maps are now widely used in the DICE system as a uniform data model for the system, instructors, learners and materials. The Mind Maps are used for system configuration, system deployment, learner's information, organization of training materials, answer parsing and some widgets. An implementation based on this data model has been working for years in a real teaching and learning environment.

Keywords: typed mind map, data model, XML, knowledge representation, DICE, TDD

1. Introduction

This paper introduces Typed Mind Maps [1] for knowledge representation. We also use a parse-tree-based Mind Map to compare student programs within a TDD (test driven development) model named DICE [5].

We have been using our DICE system for several years now. The DICE system was implemented in an OS-independent, distributed, client-server environment, with a parse-tree-based automatic assessment system for test-based assignment tutoring and problem solving. All training work including assigned practice, turn-in and assessment can be run on the DICE system. DICE has been working in Hsing Kuo High School in Taiwan for over 2 years, exceeding 2,500 learners involved since autumn 2005. We have planned to develop a sophisticated testing mechanism TDD (test-driven development) model in DICE for the underachievers [10]; moreover, an adaptive learning environment named DALM (DICE adaptive learning model) [11] was proposed to adapt different kinds of learners to DICE TDD teaching units.

We aim to develop an explicit formal specification of the terms in the domain and relations among them [13], but within a widely-used and undemanding formalism [6]. The use of Mind Maps in the ontology development process is very attractive [14] because it is a very simple formalism with a simple user-interface that is usable for students, teachers, and administrators. A typed Mind Map is a traditional Mind Map with a semantic description that can also be expressed as a Mind Map.

From the viewpoint of teaching programming in computer science, a system like DICE should involve the knowledge representations of the learners, the instructors, the DICE system administrators and the DICE system itself. For example, DICE should represent how the system administrator should organize the system information, how an instructor should represent his teaching units, and how a learner should figure out his answer for a question. We need to integrate all of this knowledge into a unified representation.

This paper describes a typed Mind Map as a knowledge and information representation in DICE, our computer-aided assessment system. Section 2 briefly describes the Mind Map and gives formal/informal definitions of the Mind Map/typed Mind Map. Section 3 briefly describes DICE. Section 4 shows the use of typed Mind Maps in DICE. Section 5 concludes with areas for future work.

2. Mind Map

A Mind Map is based on organizing information via hierarchies and categories [7]. It has been widely used for centuries in different fields. One of the uses is "keeping a small database of something with structure that is either very dynamic or not known in advance" [3]. As Dimitry Polivaev's argument that "The main disadvantage of such approach when compared to traditional database applications is poor query possibilities". We feel that this defect of Mind Maps can be overcome by the addition of data types, so that Mind Maps can reflect their own syntactic/semantic definitions as well as user-defined types and objects.

In our view, a Mind Map is a node with recursive node collections. There is no information in a Mind Map other than the node name and its association to its child nodes. An ontology defines a common vocabulary for researchers who need to share information in a domain. It includes machine-interpretable definitions of basic concepts in the domain and relations among them. [13]

2.1. Formal Definition of the Mind Map

We describe a Mind Map by a recursive definition.

Definition 1: (Mind Map Node)

A mind map node is a pair $n = (s, < N|l >)$ *where s is a String, N is a set of Mind Map nodes and l is a link to a Mind Map.*

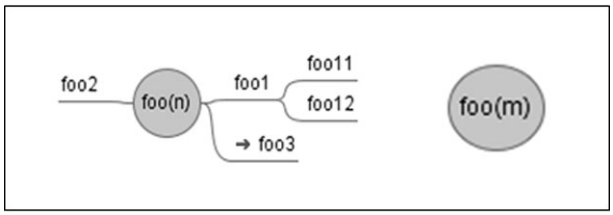

Figure 1. The foo Mind Maps.

Definition 2: (Link)

A link is a reference of a Mind Map.

Definition 3: (Child/Parent Node)

3-1 *For a Mind Map node* $\boldsymbol{n} = (s, N)$*, if* $\boldsymbol{m} \in N$ *then* $\boldsymbol{m}$ *is a mind map child node of* $\boldsymbol{n}$ *and* $\boldsymbol{n}$ *is the Mind Map parent node of* $\boldsymbol{m}$*.*

3-2 *For a Mind Map node* $\boldsymbol{n} = (s, l)$*, if* $l = MM(\boldsymbol{m})$ *then* $\boldsymbol{m}$ *is a reference child mind map node of* $\boldsymbol{n}$ *and* $\boldsymbol{n}$ *is the reference parent Mind Map node of* $\boldsymbol{m}$*.*

Definition 4: (Root/Leaf/Linked Leaf Node/null leaf node/component node)

4-1 *A Mind Map root node is a Mind Map node with no parent.*

4-2 *A Mind Map node n of the form* (s, N) *where* $N = \emptyset$ *is a Mind Map leaf node*

4-3 *A Mind Map node n of the form* (s, l) *is a Mind Map linked leaf node.*

4-4 *A Mind Map node n of the form* $(null, \emptyset)$ *is a Mind Map null leaf node.*

4-5 *A Mind Map node that does not belong to the above node classifications is a component node.*

Definition 5: (Mind Map)

A Mind Map root node $n = \{s, \{M|l\}\}$ *is denoted by* $MM(n)$*.*

Definition 6: (Category Mind Map)

$\forall y \in MM(n) \wedge$ *(y is a Mind Map null leaf node* $\vee$ *y is a Mind Map linked node) then* $MM(n)$ *is a category Mind Map.*

Example 1: The Mind Map in Figure 1 can be represented as
n = {"foo(n)", {n_1, n_2, n_3}},
n_1 = {"foo1", {n_{11}, n_{12}}}, n_2 = {"foo2"} and
n_3 = {"foo3", l}
n_{11} = {"foo11"}, n_{12} = {"foo12"}, l = MM(m)
and m = {"foo(m)"}

Example 2:
In Example 1:
The n_1 is a child node of n and n is the parent node of m for $n_1 \in \{n_1, n_2, n_3\}$.
The n and the m are root nodes, for neither of them has a parent.
The n_2, n_{11} and n_{12} are Mind Map leaf Nodes, for they all have the form $\{s, \emptyset\}$ (simplified {s})
The n_3 is a Mind Map linked leaf node that refers to m.

The n is a "foo(n)" Mind Map and the m is a Mind Map with concept "foo(m)".

Consequently, a Mind Map is a vertex-labeled directed rooted graph. The string s denotes the root node of a Mind Map representing a concept or a term that we are interested in for a specified domain. A self-reference linked leaf node can extend Mind Maps from tree structures to general labeled graphs. Such a definition attempts to provide a simple graph data structure for a data model.

A rooted Mind Map node represents a concept by a term **s**. The linked node (**l**) or the Mind Map nodes in set of Mind Map (**N**) of a root node maintain the association of other terms with the root node. A leaf Mind Map node holds the changeable information of a concept. A category Mind Map represents the schema of the Mind Map when using Mind Map as a "small database".

2.2. Informal Definition of the Typed Mind Map

A typed Mind Map consists of a traditional Mind Map and a typed Mind Map describing the types which are used in that traditional Mind Map. A typed Mind Map may also be displayed as a traditional Mind Map, and it can be edited and displayed by any FreeMind-like software, with kernel types being extended to include those used in the typed Mind Map. We classify a traditional Mind Map as either an unrestricted Mind Map (UMM) or a typed Mind Map (TMM). The ontology information can be used to translate to/from a typed node of a TMM into a sequence of un-typed nodes in a UMM.

As Figure 2 shows, a typed Mind Map is an extended Mind Map with a "special" Type Mind Map to describe the typing mechanism. The special Type Mind Map consists of a typed Mind Map to describe the schema of a Typed Mind Map, a semantic typed Mind Map to express the ontology of a Mind Map and some misc. nodes to illustrate some additional information of a Mind Map (e.g. version control, access control, category . . . etc.).

Figure 2 represents a specific Mind Map that describes the type of a type (i.e. what a typed Mind Map looks like).

Definition 7: (Syntax Type Mind Map Node)

A syntax typed Mind Map node ***s*** *is the root node of a category Mind Map that associates to a specific Mind Map* ***n****. We define s as the syntax Mind Map node of the Mind Map n.*

As Figure 2 shows, a node named "Components" shows the schema of MM(("_Type", N)). It describes the structure or the schema of a Mind Map. The typetype.mm describes itself via the "Components" node. In our implementation, we define a special Mind Map named _type as a primitive type to mark a Mind Map as a *typed* Mind Map.

Definition 8: (Misc. Type Mind Map Nodes)

As Figure 2 shows, some misc. nodes are reserved for comprehensive use of a Mind Map.

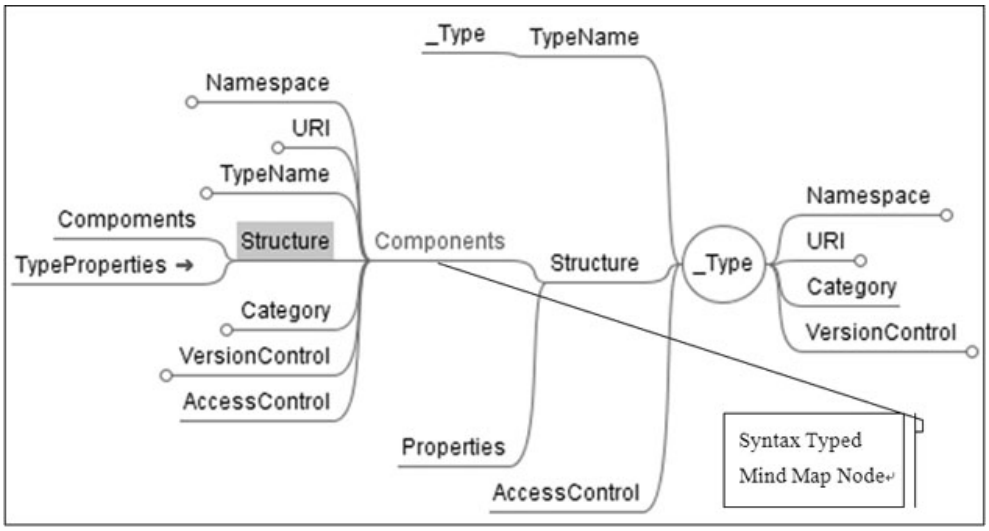

Figure 2. The kernel typed Mind Map of type, typetype.mm.

For example, a version control node can be designed for tracing the changed history of a typed Mind Map, an access control node can be added to figure out the user rights for a Mind Map, and a category node list can be used to catalog a Mind Map for multiple classification systems.

Definition 9: (Semantic Type Mind Map Node)

A semantic typed Mind Map node is designed to describe the ontology of a Mind Map.

Figure 3 shows a representative semantic typed Mind Map node. The representation of a complex object should support self-description, strong typing and information hiding. The traditional Mind Map is weak on strong typing.

We separate a concept from a single representation (e.g. OWL, SUMO. . . etc.) into a traditional Mind Map and a Typed Mind Map that consists of syntax and semantic nodes.

In our proposal, the semantic typed Mind Map node plays the role of OWL-DL, as it supports the maximum expressiveness without losing computational completeness and decidability of reasoning about data types [4]. A semantic typed Mind Map node uses OWL-DL to define the primitive types and constructor types. Meanwhile, a typed Mind Map can export to an OWL-DL document via its semantic typed Mind Map node, and so can an OWL-DL document be displayed as a TMM. OWL-DL can import a typed Mind Map.

Definition 10: (Type-Type Mind Map)

A type-type Mind Map is a Mind Map that involves a syntax typed Mind Map node and a semantic typed Mind Map node in its root. Formalizing, $MM(n)$ is a Mind Map with the root node formed (s, N), if $\{\exists\{x, y\}|x \in$ syntax type Mind Map node $\wedge y \in$ semantic typed Mind Map$\} \in N$ then $MM(n)$ is a type-type Mind Map.

Definition 11: (Kernel Type Mind Map)

A kernel Mind Map having the type of a type-type Mind Map is a typed Mind Map.

For example, Figure 2 is a typed Mind Map that uses itself as a type.

Any typed Mind Map can be built from this well-defined kernel typed Mind Map. At the same time, an unrestricted Mind Map can be assigned to an exclusive typed Mind Map automatically.

Definition 12: (Typed Mind Map Node)

A typed Mind Map node is a Mind Map node with a type. We formalize the typed Mind Map as a pair $(n, MM(t))$ where n is a Mind Map node and $MM(t)$ is a kernel type Mind Map.

Definition 13: (Typed Mind Map)

A typed Mind Map is a Mind Map with a type. We formalize the typed Mind Map as $TMM(n, MM(t))$ where n is a Mind Map root node and $MM(t)$ is a kernel type Mind Map.

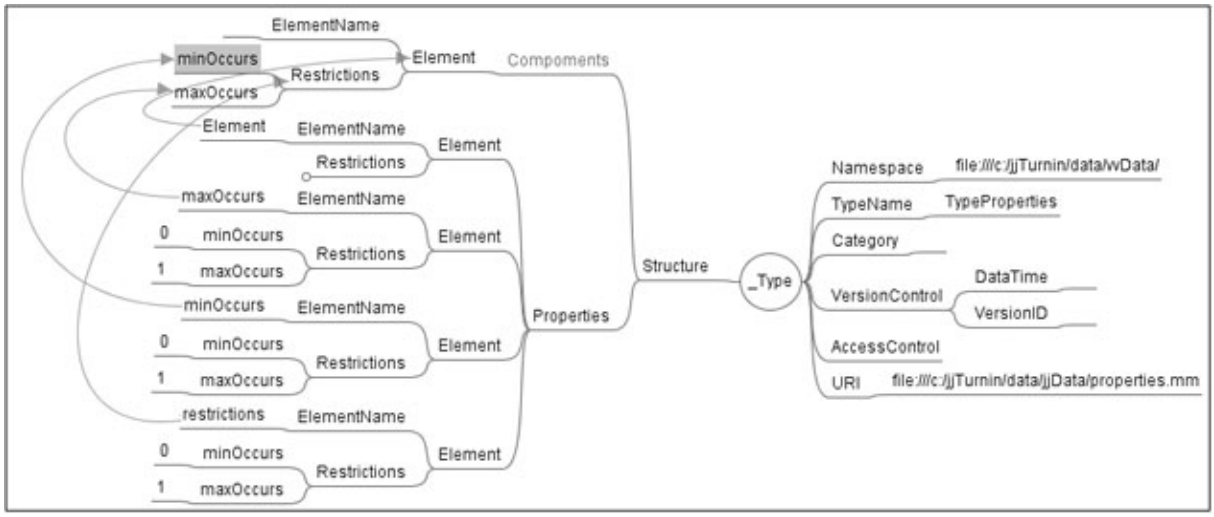

Figure 3. The kernel typed Mind Map of a semantic node.

2.3. An Implementation of the Typed Mind Map

We rely on a widely-used freeware named FreeMind, written in Java, for the creation of Mind Maps [3]. The electronic Mind Maps with extension .mm that are created by FreeMind obey the XML specification. A tree-based node collection named 'node' with a 'TEXT' attribute denoting the text in a node is enclosed by a root node named 'map'. As a knowledge representation, FreeMind-like XML schema simplify the Mind Map directly from visual image into an XML node named 'node' with a 'Text' attribute. The interesting terms in a domain are denoted by the 'TEXT' attribute in the 'NODE' node. As right side of Figure 4 shows, the recursive definition of node provides a tree-based collection to organize nodes. The 'LINK' attribute in a node extends the tree into a directed rooted graph. The links can be restricted to only form a directed acyclic graph (i.e. DAG).

The implementation of our typed Mind Map API is based on FreeMind-like XML schema. A FreeMind-like Mind Map file can be read into the API as a typed Mind Map node. A typed Mind Map algebra supports rich operations to operate on Mind Maps. A tree-based query function retrieves suitable nodes from Mind Maps.

The API was widely used on system configuration, training material organization, learner information/answers and some widgets as a common knowledge and information representation in the DICE system. The system compares the output of a student's program with standard output file to decide a score that he will get. The result is immediately sent to the student.

3. The DICE

Since 2005, we have commenced to establish DICE system for a test-based assignment tutoring and problem solving environment. [5] All training work, including assigned practice, turn-in and assessment can be run on the DICE system. DICE has been working at Hsing Kuo High School in Taiwan R.O.C. over 2 years. Figure 5 shows a running DICE System.

After running DICE for years, we found some well-known problems of a test-based grader. These caused the underachievers to be eliminated from the DICE system. One problem is that only clearly defined questions with a completely specified interface can be used. It leads students to focus on output correctness first and foremost, and it does not encourage or reward for good performance while testing [12]. Another of the perceived shortcomings is that its inflexibility prevents assessment of more complex questions [2]. When a complex question occurs, we found that some underachievers just sat before his/her computer and waited for the bell to ring. So we need a more sophisticated mechanism to help underachievers.

At the second stage, we refer to training method criteria and TDD concepts to establish a new training model for learning programming, which

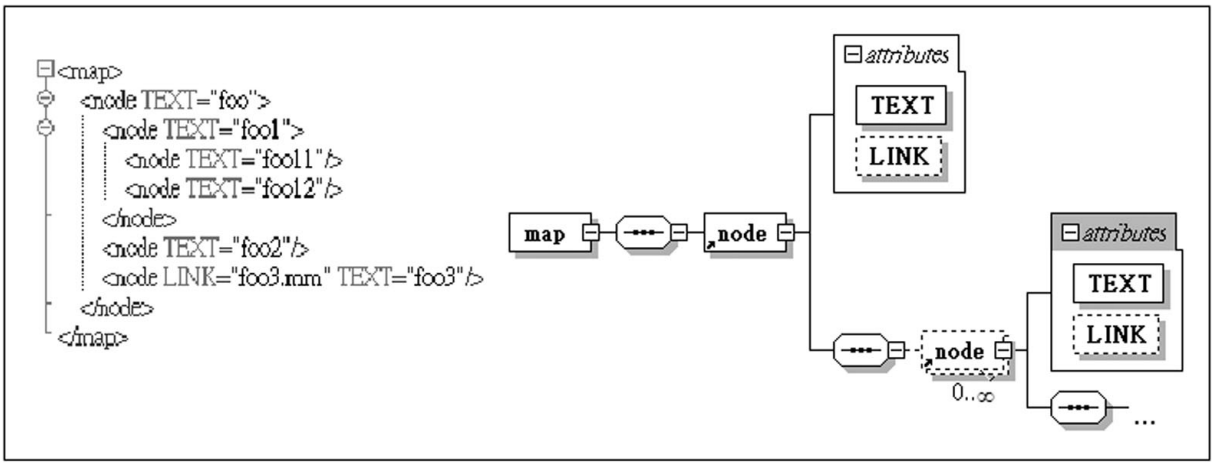

Figure 4. A simple schema for foo FreeMind Mind Map.

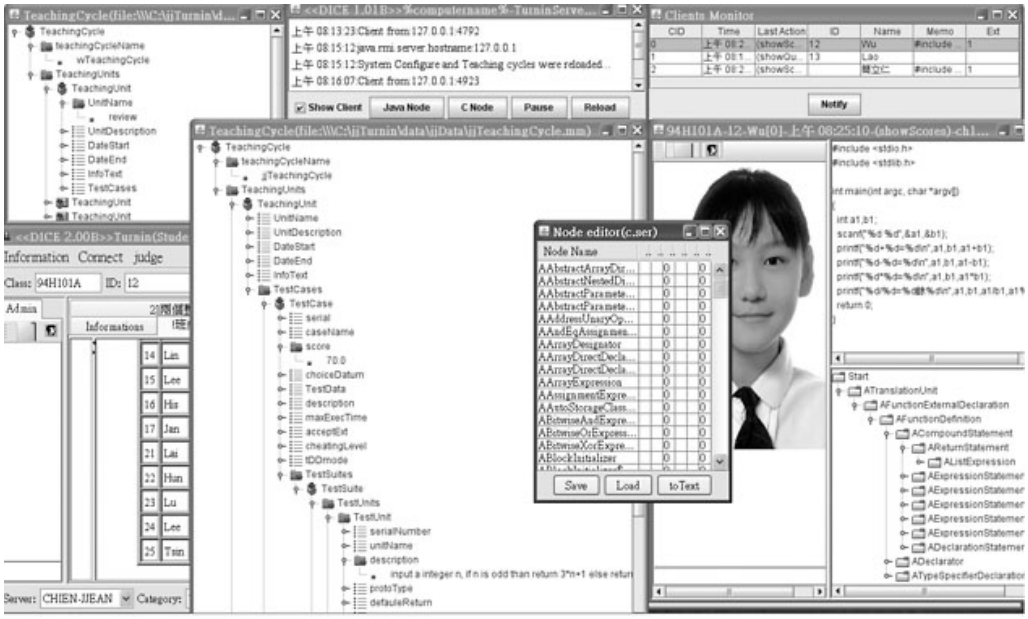

Figure 5. A running DICE.

is named the DICE TDD Model [10]. It provides sixteen kinds of training methods for learners.

At the third stage, we conduct Kolb's [8][9] learning style instrument as the test item of individual differences. We find the best fit between learning styles and training methods which will result in a satisfactory learning outcome. We will prove that different learners need different training methods in the DICE system. [11]

4. Using Typed Mind Map in DICE

Typed Mind Maps now are widely used in the DICE system to provide a uniform data model between system, instructors, learners and materials. The applications include:

- System configuration
- System deployment
- Learner's information
- Training Materials organization
- Answer Parsing
- Widgets

In the next two subsections, we will describe clearly the use of the training material organization and an electric Kolb LSI widget.

4.1. Training Material Organization for DICE TDD Model

DICE TDD is a comprehensive model in an automatic grading system for DICE. As Figure 6 shows, a program assignment was given after

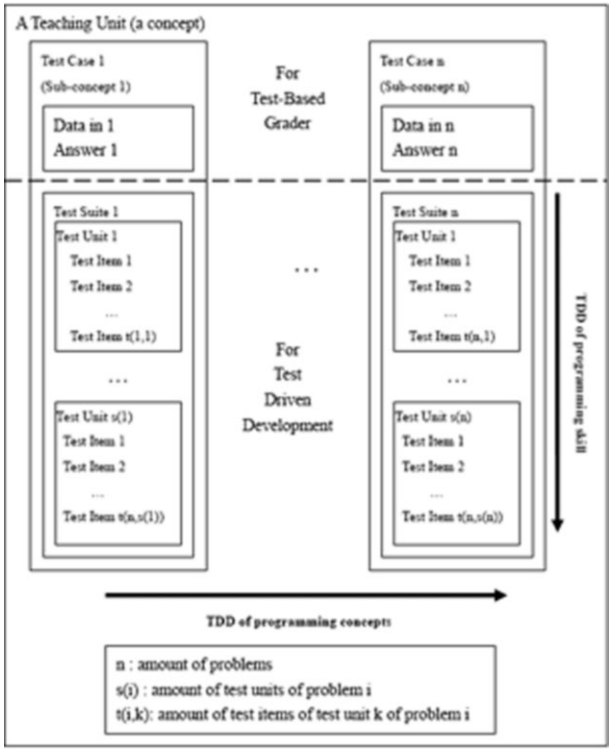

Figure 6. The TDD Model in DICE.

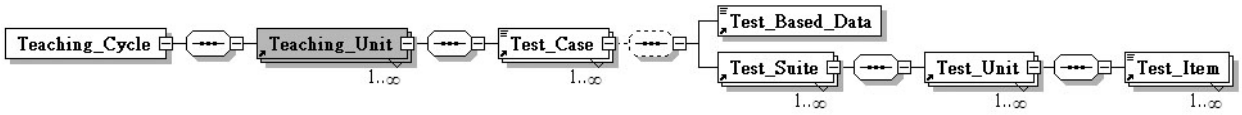

Figure 7. The schema of a test in TDD DICE.

each teaching unit. The instructor makes a test plan consisting of problems (or so called test cases). A test case was composed by the test-based grader, like data sets and TDD-like data sets.

As Figure 7 shows, a training material in TDD DICE is structured bottom up from test items, test units, test suites, test cases, and test units to a test cycle.

4.1.1. Storage Policies

Generally, there are four main policies to store the training material to a computer system:

- Storing materials to a file/directory system
- Storing materials in a relational database system
- Storing materials in an object-oriented database system
- Storing materials to serial XML files.

As Figure 8, the original design of TDD in DICE organized training materials to a file/directory system for keeping DICE independent (e.g. need not rely on heavy database system). As Table 2 indicates, a file-based storage policy is faulty on many points except economic. For example, a file-based policy is hard to distribute the training

	File Directory	Relational Database	OODB	XML files	Typed Mind Map
Distributed	−	+ +	+	+	+ +
Deployed	−	+	+	+ +	+ +
Economic	+ +	− −	−	+ +	+ +
Efficient	−	+ +	+	− −	− −
Intuitive	− −	−	+	+ +	+ +
Knowledge Represented	− −	−	+	+ +	+

Table 2. Comparison between different training material storage policies.

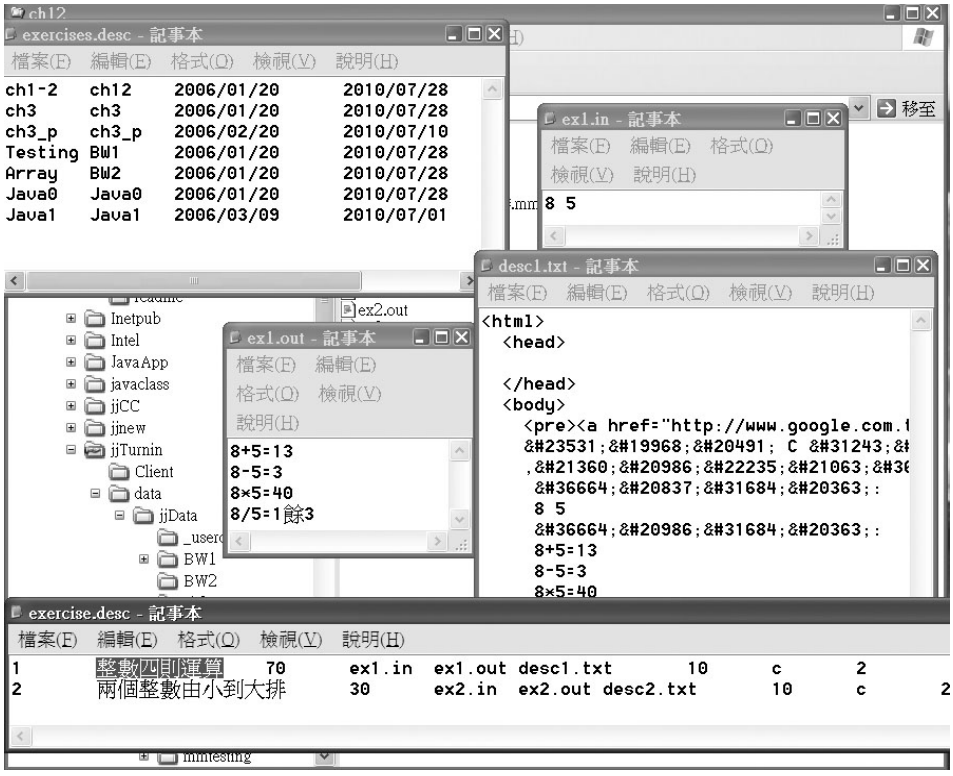

Figure 8. The original training material store used by DICE TDD.

material to instructors' space and the format of a file is non-intuitive for human understanding.

In the DICE TDD model, we take each component in Figure 6 as a "concept" of a typed Mind Map. An associated node of a concept can be a component node or a reference (i.e. linked node). In our typed Mind Map API implementation, a linked node can be point to different typed Mind Map placed on any TCP/IP device via standard URI retrieve technique or our typed Mind Map delivering layer API. The main advantage of this policy makes the deployment of training materials more flexible for different teaching and training environments than ever.

As Figure 9 shows, same material expressed in Figure 8 was transferred to several typed Mind Maps and deployed to different hosts via IP network. The instructor in host 1 planned a teaching cycle named 'jjTeachingCycle' to organize sequence of teaching units to be placed on different hosts. The teaching unit ch12 was cached to host 1 while the others were kept for reference. The host maintained an independent teaching unit type Mind Map that was cited by host 1. Meanwhile, it referred to an independent test case that was replaced on host 3.

4.2. A Widget for Evaluating Learner's Kolb Learning Style

As an adaptive learning environment, DALM (DICE adaptive learning model) based on DICE test-driven development (TDD) model was implemented in a parse-tree based automatic online grader. There are three variables in DALM. The individual differences variable (I) classifies the learners into several groups. The training method (T) tells which TDD model is chosen in DICE. The learning outcome (O) presents the learning performance of learners with an individual difference level and the chosen TDD model. The learning outcome is related to the individual difference and the training method, denoted as $O = f(I, T)$.

In DALM, we need a tool to measure learner's Kolb learning style (KLS) as individual difference. A TMM-based widget was built up for evaluating the learner's KLS before a teaching cycle. Figure 10 show TMM of the Kolb's learning style inventory-version 3.1. [8] As Figure 11 shows, our TMM-based KLS widget loads and transfers the Typed Mind Map to an electronic questionnaire. The result of this questionnaire is packed to a TMM and is delivered to DICE server as part of the learner's information.

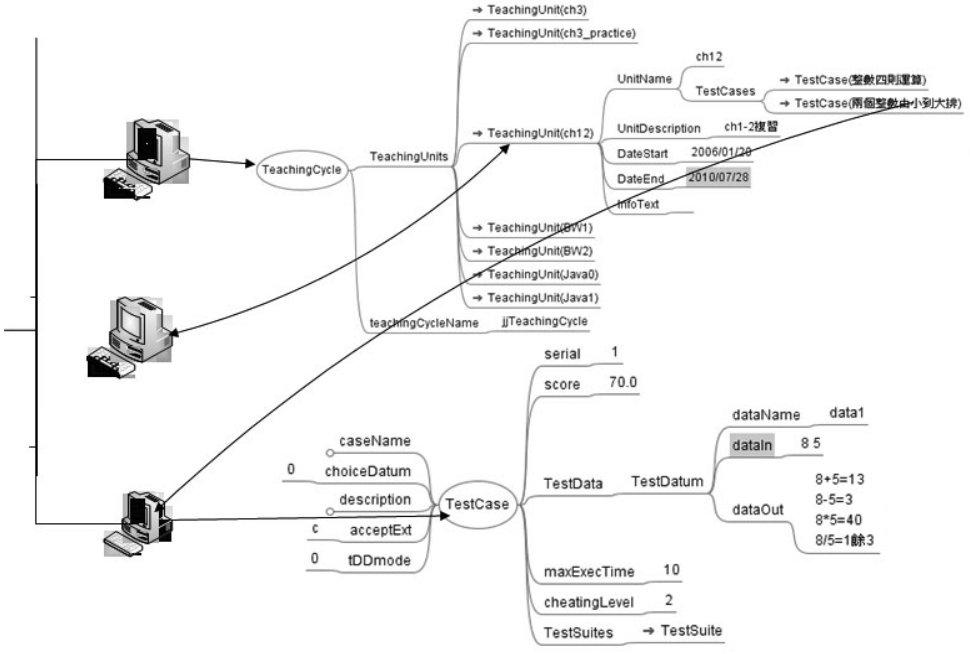

Figure 9. A simple deployment of a DICE TDD training material by typed Mind Map.

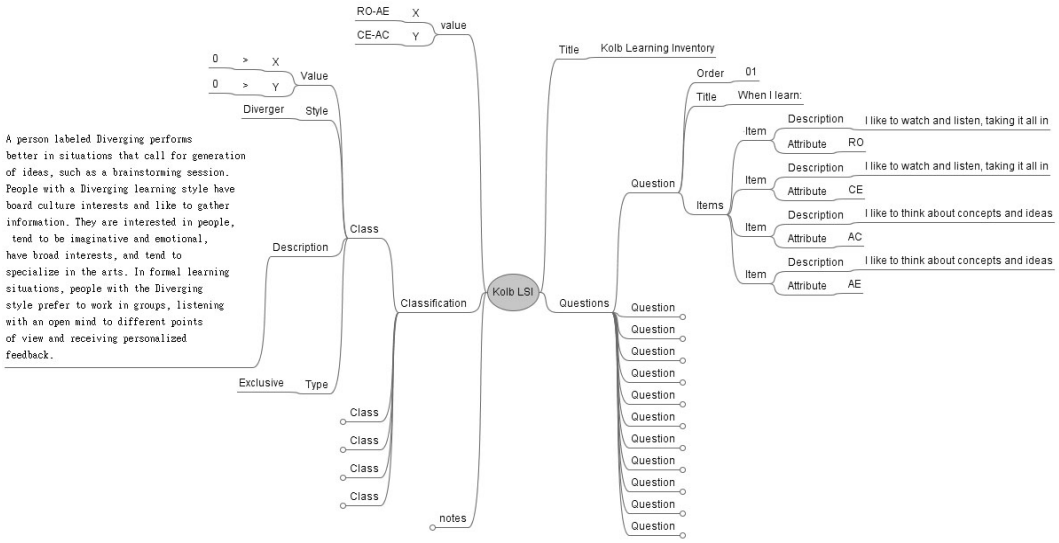

Figure 10. Typed Mind Map of Klob's learning style inventory.

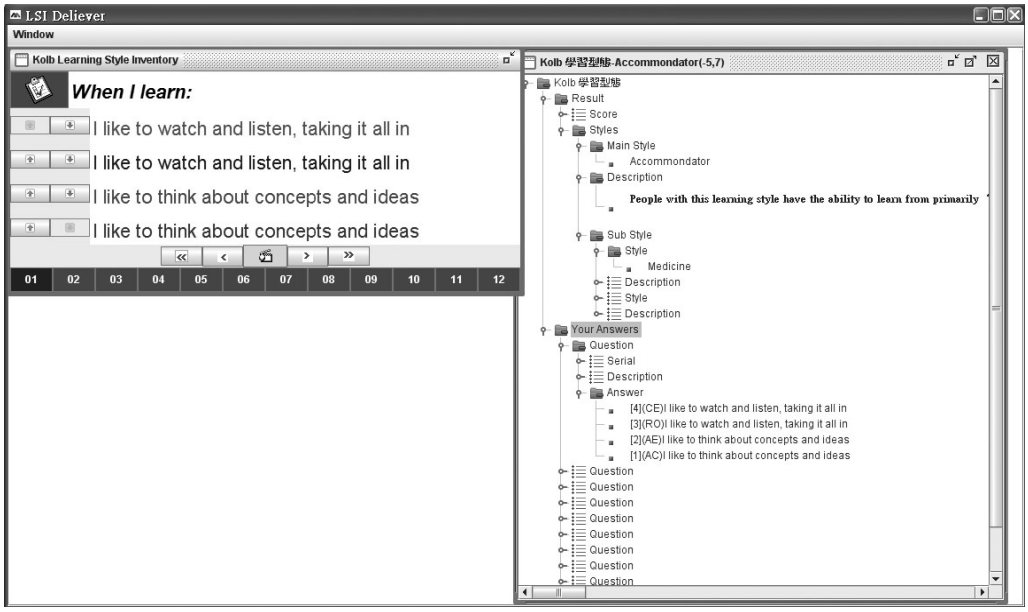

Figure 11. A widget for evaluating learner's Kolb learning style.

5. Conclusions and Future Work

This paper describes an innovative Mind Map usage via extending a kernel type Mind Map associated with the original Mind Map. The kernel type Mind Map represents the schema by a syntax node and the ontology by a semantic node. Such a policy makes the best flexibility of using a Mind Map. A traditional Mind Map is a non-typed Mind Map that can be used during system designing stage for keeping the most variant possibility of structure. When the system gets into a stable stage, it is easily extended to syntax typed Mind Map by associating the traditional Mind Map with syntax node type Mind Map. A syntax typed Mind Map can be used to do the schema checking by tree comparing. This study shows it works well in a DICE TDD system as a uniform data model.

The typed Mind Map can be used as a uniform data model in an e-learning system, between the system, instructors, learners and materials. The applications can be widely used from system configuration, system deployment,

learner's information to training materials organization. This study shows it works well in a DICE TDD system as a uniform data model.

In Section II we have defined a systematic and symbolic model for typed Mind Map. For taking the typed Mind Map as a data model, the typed Mind Map algebra is proposed. Some basic definition of elements, axioms and operations were made and implemented in our TDD API. We take the TMM algebra proposal as an important future work. Moreover, after the system has run for years in a stable stage, we can start to develop the semantic type node of the system. For we have defined a semantic type Mind Map by an OWL-DL, it's more suitable to develop the machine learning and reasoning mechanisms with artificial intelligence methods.

References

[1] BUZAN, The Mind Map Book: Mind Mapping Guidelines, Penguin, New York, 1991.

[2] D. CHRISTOPHER, L. DAVID AND O. JAMS, (2005). "Automatic Test-based Assessment of Programming: A Review", *ACM Journal of Educational Resources in Computing*, Vol. 5, No. 3, September 2005. Article 4.

[3] DIMITRY POLIVAEV. FreeMind – free mind mapping software, FreeMind Official Homepage & Wiki, Available at `http://freemind.sourceforge.net/wiki/index.php/Main_Page`, 2008.

[4] D. L. MCGUINNESS AND F. VAN HARMELEN, eds. OWL Web Ontology Language Overview, *World Wide Web Consortium* (W3C) recommendation, Available at `http://www.w3.org/TR/owl-features`.

[5] LI-REN CHIEN, D. BUEHRER AND CHIN YI YANG. Dice: A Parse-tree Based Online Assessment System for a Programming Language Course, *International Conference on Teaching and Learning* (iCTL 2007), Putrajaya, Malaysia, 2007.

[6] T. R. GRUBER, A Translation Approach to Portable Ontology Specification, *Knowledge Acquisition 5*: pp. 199–220, 1993.

[7] JOHN W. BUDD., Mind Maps as Classroom Exercise, *Journal of Economic Education*, 2004.

[8] A. Y. KOLB, D. A. KOLB, The Kolb's learning style inventory-version 3.1 2005 technical specifications, Boston, MA: Hay Resource Direct. 2005.

[9] D. A. KOLB AND R. FRY, "Toward an applied theory of experiential learning". In Thories of Group Process, G.L. Cooper(ed.), John Wiley and Sons, Inc., New York, NY, pp. 33–54, 1975.

[10] LI-REN CHIEN, D. BUEHRER AND CHIN YI YANG. Using Test-driven Development in a Parse-tree Based Online Assessment System, *IADIS International Conference e-Learning*, Lisbon, Portugal. 2007.

[11] LI-REN CHIEN, D. J. BUEHRER, CHIN-YI YANG. An Adaptive Learning Environment in the DICE System with a TDD Model, *Interactive Computer Aided Learning*, Villach (ICL 2007), Austria, 2007.

[12] STEPHEN H. EDWARDS AND MANUEL A. PÉREZ-QUIÑONES. Experiences using test-driven development with an automated grader, *Journal of Computing Sciences in Colleges*. Vol. 22, Issue 3, January 2007.

[13] T. BUZAN. Use your head. BBC Books, N. F. Noy and D. L. McGuinness. (2001) "Ontology", 1974.

[14] Y. SURE, M. ERDMANN, J. ANGELE, S. STAAB, R. STUDER AND D. WENKE. OntoEdit: Collaborative ontology development for the Semantic Web, *1st International Semantic Web Conference* (ISWC 2002), Vol. 2342 of LNCS, pp. 221–235, 2002.

Received: June, 2008
Accepted: September, 2008

Contact addresses:

Li-Ren Chien
Department of Computer Science
and Information Engineering
Chung Cheng University
#168, University Rd, Min-Hsing
Chia-Yi, Taiwan, R.O.C.
e-mail: `clj@cs.ccu.edu.tw`

Daniel J. Buehrer
Department of Computer Science
and Information Engineering
Chung Cheng University
#168, University Rd, Min-Hsing
Chia-Yi, Taiwan, R.O.C.
e-mail: `dan@cs.ccu.edu.tw`

LI-REN CHIEN is a Ph.D. candidate in the Department of Computer Science and Information Engineering at Chung Cheng University, Taiwan, R.O.C. He obtained the BS in physics from Chung Yuan Christian University and the MS in computer science and information engineering from Chung Cheng University. He has been a teacher of computer science and physics in Hsin Kuo high school, Taiwan, R.O.C. since 1989. His research interests are e-learning, automatic grading system and artificial intelligence.

DANIEL J. BUEHRER was born in Green Bay, Wisconsin, USA. He received his B.S. in physics and in mathematics from Carroll College, Waukesha, Wisconsin, in May, 1971. His M.S. and Ph.D. in computer science were received from the University of Iowa in Dec. 1973 and May, 1976, respectively. He was an assistant professor at the University of South Carolina from 1976–1979, at National Tsing Hua University from 1979–1986, DePaul University from 1986–1987. He worked at Rich, Inc. of Chicago from 1987–1988. He has been with the Dept. of Computer Science and Information Engineering at National Chung Cheng University since the beginning of the University, in 1989, and is now a full professor there.

His major interests are artificial intelligence and object-oriented technology.