The Foundations of the
Teaching, Training, and Learning System

Vincent P. Heuring
April 14, 2004

Objective

The objective of this research is to develop a platform-independent suite of tools that will assist in the educational process. The tool suite is known as the Teaching, Training, and Learning, TTL, System. There are tools within the suite to assist in developing and maintaining a knowledge base appropriate to the subject material, tools to assist in developing problems and solutions for student drill and examination, and tools to assist in assessing and improving student performance.

The TTL System will be based upon using web browsers for user interfaces, and the underlying software will be written in Java. This will assure platform independence for the main components of the system. The system will be designed to support additional user interface components, such as video, audio, or other haptic components, however.

The web browser is the ideal user interface to support interaction with the system. It is media-in-different, not caring whether it is pointed at a file on CD-ROM, or on a hard disk, or to a machine half a world away across the Internet With the content being developed using Java the developer is assured that it can run unchanged on any of the platforms above; content only needs to be developed once.

This system will have components to support the Teacher, who designs the information base to be used in employee training; the Trainer, who designs specific teaching and training modules from the information base, and for the Student, who interacts with the system.

Introduction and Background

This document describes the philosophy and framework that underlie the TTL System. The TTL System is classed as an intelligent tutoring system, ITS. Many ITS's have been developed in the past ten years, in areas of domain expertise ranging from algebra, computer programming, equipment operation and maintenance to models of rainfall and computer game playing. A good summary of the state of the art can be found in Polson and Richardson.[†] Although published in 1988, much of the information is still relevant.

Most of the ITSs developed in recent years have focused on the more AI-like aspects of ITSs, such as causal reasoning, natural language understanding, and realistic student modeling. As such, most of them use knowledge bases that attempt to mimic the way humans represent knowledge - by stressing goals, objectives, and rules. Many ITSs also stress the human-computer communications aspects of ITSs: developing sophisticated graphical and haptic user interfaces.

The TTL System, on the other hand, stresses student drills in areas that Richardson refers to as de-

---

[†] Foundations of Intelligent Tutoring Systems, Martha C. Polson and J. Jeffrey Richardson, Erlbaum Publishers, Hillsdale NJ, 1988.

1

clarative and procedural knowledge: the knowledge of definitions and mappings, to use our terms. Furthermore, the TTL system is designed to be useful in any domain in which knowledge can be represented by specific, quantitative definitions and mappings. Domain areas would include Mathematics, Computer Science and Engineering, and Electrical Engineering, for example. TTL System emphasis is on representing specific knowledge, extracting solutions and problems from the knowledge base, drilling the student by using those problems and solutions, and grading and diagnosing student responses. The elements of the TTL knowledge base might be considered to be "coarse-grained" compared to those in other kinds of knowledge bases. This is because in many knowledge bases the individual elements are "atomic," having little or no internal structure. The individual elements of the TTL knowledge base are blocks of definitions, executable code or presentation materials and may have considerable internal structure.

The System is general in the sense that problems and solutions are generated in abstract form, and can be formatted to use any kind of user interface, from web browsers to head-mounted displays.

The research proposed to develop the TTL System should be considered to be applied AI rather than AI research. It employs, for the large part, well-known AI principles and techniques.

The TTL System in a Nutshell

The TTL System is used by three entities: the Teacher, the Trainer, and the Student. These terms are capitalized when referring to interaction between the entities and the TTL System, as are terms that have a formal defined meaning within the TTL System. In general, the Teacher builds and maintains the Knowledge Base, KB, adding, modifying, or interconnecting concepts, k. The Trainer selects a concept, k, from KB. Using this concept and an associated concept level, L, the trainer can instantiate Solutions and Problems for individual use, or can assemble a collection of them into Homework assignments or Exams. The Trainer can design a Drill by specifying the root, or initial problem, and the leaf, or terminating problem. Drills can be static, that is, assembled ahead of time for later use, or dynamic, where the Student's Response to each problem is analyzed by the TTL system before it presents the next problem.

Each concept may have associated with it a number of Formats. Formats specify the external form of the Problem, so one Format might present a Problem as a text description, whereas another Format might employ graphics, audio, video, or virtual reality techniques. This means that the same problems can be formatted differently for distance learning, or learning for the disabled, for example. In any case, the Student submits a Response, which the TTL system coerces to an Answer. The Answer has two components, a Grade, g, and a Diagnosis, Δ. The Diagnosis can be used to provide informal or formal feedback. In particular, it can be used to guide the presentation of future Problems to the student.

Organization of this Paper

The next section of this paper justifies the approach of the TTL System by delving into philosophical questions about the nature of knowledge and the difference between static and dynamic knowledge. Following that discussion is another philosophical discourse on the nature of teaching, training, and learning, and the interactions between teacher, trainer, and student. In that section tool usage is also discussed. The two sections following that section define the TTL System, first informally and then formally.

There are many benefits to a formal description of a system as complex as TTL. Formal definitions provide unambiguous descriptions of the components of the system and their interaction. Thus they provide guidance to those who are constructing, modifying, and using the TTL System. In many cases the formal definitions can also be used as input to tools for automatic generation of at least some components of the TTL System. For clarity, along with the formal definition we provide a running example.

What is Knowledge?

Any proposal to develop a knowledge-based teaching, training and learning assistant must address the question, "What is Knowledge?" While not wishing to become mired in philosophical discourse, it is necessary to at least embark upon a short journey into that territory to explain and justify our approach.

Without apology, we adopt many of the views of John Sowa[†], whose work has influenced much of AI research since its appearance. Sowa's position, stated as early as page 2 is that

> "Knowledge is more than a static encoding of facts: it also includes the ability to use those facts in interacting with the world. A basic premise of AI is that knowledge of something is [1] the ability to form a mental model that accurately represents the thing as well as [2] the actions that can be performed by it and on it."

One might paraphrase Sowa by defining knowledge as a set of facts and skills and the abstractions that connect them. Sowa distinguishes between the static encoding of facts, in books and recordings, for example, and the use of those facts by humans in interacting with the world. Let us call these two forms of knowledge *static knowledge*. and *dynamic knowledge*.

To use the terminology of Teilhard de Chardin,[‡] we would say that dynamic knowledge resides in the *noosphere*, the thinking envelope that surrounds the planet approximately five to six feet above its surface: the knowledge that resides in human brains.

To paraphrase Teilhard, the noosphere arose at the end of the Tertiary period, in the bodies of the phylum 'homo.' The emergence of consciousness (and unconsciousness?) and the ability to *apply the conscious to the self* signalled the birth of the noosphere.[*] Then emerged speech and verbal and nonverbal communications. One important aspect of nonverbal communications was the use of tools to encode static knowledge. The result, as we all know, is an enormous store of static knowledge in places such as libraries, bookshelves, and computer disks and tapes.

Static and Dynamic Knowledge

Here we further define Static and Dynamic knowledge. Again we wish to avoid controversy by acknowledging that there are many definitions of knowledge. The best we can hope is that definitions developed here will be useful within the context of the TTL System.

[†] John Sowa, *Conceptual Structures, Information Processing in Mind and Machine*, Addison-Wesley, Reading MA, 1984

[‡] Pierre Teilhard de Chardin, *The Future of Man*, Harper and Row, New York, 1964.

[*] Vincent P. Heuring, *Diagnosing and Prescribing in the TTL System*, To appear.

We begin by tackling the more difficult of the two, Dynamic Knowledge.

Dynamic Knowledge   Dynamic knowledge is knowledge that is encoded in living beings.[†] We assume that dynamic knowledge consists of a collection of facts, skills, and the many layers of abstractions, or concepts, that connect them. Facts can be thought of as definitions possessed in the mind of an individual. In this paper we define abstractions, or concepts, as the definitions and mappings, or transformations between them of which the mind can conceive. In these terms, a skill is the ability to perform an action—purposive behavior—that demonstrates some concept.

Example definitions would include:
- the integer number system;
- Electrical resistance, capacitance, inductance;
- the instruction set of a particular computer in assembly language or in machine binary;
- sets of customer questions and customer service responses.

Example mappings would include:
- the addition of a list of numbers to form their sum;
- Kirchkoff's laws;
- The SPICE electronic simulator;
- an assembler for a certain computer;
- the relationship between a consumer question and the appropriate response.

Example concepts would include:
- the concept of integer addition;
- the concept of computing voltage and or current in an electronic circuit;
- the concept of assembly of a computer program into binary form;
- the concept of the relationship between customer questions and appropriate customer service responses.

Example skills would include:
- the ability to perform integer addition of four eight-bit numbers;
- the ability to "hand assemble" assembly language statements into binary form;
- the ability to use the Microsoft Assembler to assemble assembly language statements into binary form;
- the ability to design and analyze an electronic circuit by hand;
- the ability to design and analyze an audio amplifier using the SPICE simulator;
- the ability to provide appropriate responses to customer questions.

Some of these example skills involve only "hand work," and some involve the use of "tools." We will discuss the issue of tools *vs*. hand work later, in the section titled, "When should tools be used?"

Static Knowledge   Static knowledge is knowledge that has been encoded in non-living matter. Examples would include books, films, and computer programs. As static encodings, computer programs are of particular interest to us, because they allow us to encode facts and mappings in a way that allows simulation of concepts such integer addition, electrical circuit analysis, program assembly and disassembly, and language dialogs. These simulations can be used to generate problems and solutions for student drill and examination.

[†] Again without wishing to enter into controversy, we adopt the position that when an entity exhibits purposive behavior, it is alive.

Examples of computer programs that can compute concepts such as those described above:
- a Java program that accepts an addend and a list of augends, and which produces their sum;
- an assembler/disassembler for the Motorola MC68000 computer;
- a Java program that can compute voltages and currents in a simple electronic circuit;
- the PSPICE electronic circuit simulator;
- a Java program that can map customer questions to customer service responses.

Teaching, Training, and Learning

We humans possess within ourselves at birth the ability to acquire knowledge and skills. That is, we possess the ability to acquire facts, to organize those facts into concepts, and to demonstrate those facts and concepts. The Learning process is then the act of acquiring facts and concepts and developing skills.

How does one learn?

The individual learns, or acquires knowledge and skills, in four broad ways:

1) One-way interaction. In one-way interactions all the information flow is from the knowledge base to the individual. One-way, or passive, interactions would include reading, listening to an audio tape, or watching a videotaped presentation.

2) Two-way interaction with the static world. In two-way interactions the individual is able to stimulate the knowledge base, and in return the knowledge base produces some response. Included in this category are closed-ended experimentation, such as laboratory experiments, and more open-ended experimentation: "messing around." Computer-based tools such as browsers, simulators, assemblers, spread sheets, etc. are of particular interest, as they allow the student to practice or explore a particular skill.

3) Interaction with the living world, and in particular, with teachers and trainers. In these terms, the teacher assists with the high-level issues involved in the learning process, while the trainer assists the student in acquiring skills and more limited concepts. It is here that intelligent tutoring systems can provide leverage by assisting with skill development by automating drills and exercises.

4) Reflection: contemplation and pondering. This is where much or even most of learning takes place. Regardless of the quality and number of books or videos consulted, regardless of the number of experiments performed, regardless of the number of sessions with the simulator, regardless of the number of questions asked and sample problems solved, mastery comes only when one has internalized the facts, concepts, and skills. One does not learn, "mindlessly." As one interacts with the living and non-living world one must be mentally active: making inferences, making and testing hypotheses, *solving problems*.

One should not assume that reflection always means sitting quietly in a dimly-lit room with eyes fixed and unseeing. Much reflective activity occurs in small spurts between and during the interactions described above. These reflections would include following along with the teacher's explanation, formulating answers to questions, deciding upon values to

input to simulation programs, reflecting upon why certain responses were noted as incorrect, developing, as Sowa puts it above, "... a mental model that accurately represents the thing as well as the actions that can be performed by it and on it."

Learning takes place as one first forms, and then shapes and hones a mental model, making it conform more and more closely to a model that exists in the mind of the teacher, or the model that is described in some static knowledge base such as a book or training tape.

The popularity of books   The TTL System is designed to supplement textbooks, not supplant them. Textbooks and books in general have maintained their popularity in the face of video and multimedia training materials. Perhaps a major reason for the continued popularity of books in a pedagogical environment is that one has more opportunity to reflect when reading a book than, say, when watching a video. Books are, according to Marshall McLuhan,[†] a "cool" medium—one that is "low definition." This lack of definition induces the reader to fill in the blanks, as it were, requiring the brain to concentrate itself on the reading matter, thus making the reader an active participant. Videos would be considered "hot" relative to books, because of the greater definition, or higher bandwidth of the video signal. This higher bandwidth, along with the linear nature of the videotape consume more of the student's attention, leaving less time for reflection, and may account for the lack of great success of training videos except in areas where the "show and tell" nature of the medium is important to explaining the concept. Computers, being general purpose machines, may be programmed to be "cool" or "hot," but most current applications seem to favor the "hot" approach. Any intelligent tutoring system should consider this issue carefully during the system design phase.

Learning means integration   There is more to learning than just assimilating a certain concept, and being able to exhibit skills implied by it. Before that model and those skills can be truly useful, they must be fit in, or integrated with the other models already existing in the mind of the learner. This probably means abstracting them by generating new "connections" with existing models—putting them "in context." A child may have learned addition quite satisfactorily as measured by the ability to add columns of numbers, but until that child has linked the new concept of addition with the concept of computing the cost of a pile of candy bars, the child has a rather impoverished concept of addition. In TTL parlance we would say that the child has mastered the concept of SignMagnitudeDecimalIntegerAddition, but not the concept of DecimalAddableObjectAddition.

Parenthetically, this impoverished knowledge of addition is manifest when the student is asked to solve the dreaded "word problems." Again this is a matter that should receive great attention when designing intelligent tutoring systems and the concepts to be input to them.

When should tools be used?

In the next section we present the TTL System, which uses a knowledge base and various software tools to assist in the teaching training and learning process. Prior to that, however, we wish to address an issue related to the use of computer-based tools in student drill. The basis for that objection is that if the tools are available in the knowledge base, then rather than employ the tools to drill the student, the tools *should be used by the student* in solving problems. This issue is manifest in debates over the question, "When should school children be allowed to use calculators, if ever?"

Stated in another way, the question becomes one of when problems should be solved manually,

---

[†] Marshall McLuhan, *Understanding Media: The extensions of man*, McGraw-Hill, New York, 1964.

and when they should be solved using tools. Rather than answer this question directly, let us provide three reasons for the student to learn "hand methods" before tools are used. The three are:

Competence in the absence of tools   This is sometimes referred to as the "desert island" problem. "You are on a desert island with only a pencil and paper and you need to ..." The student who has not learned the underlying "hand method" is completely dependent on the tool. The ability to add a column of numbers manually can be very helpful to the grocery shopper who has a few items on the shopping list, a few dollars in the purse, and no calculator.

The need to know when tools should be used, and when they should not   If the student has not learned the principles underlying the operation of the tool, then the student is unlikely to know or be able to assess the region where the tool provides "good" answers, from where the tool provides "bad" answers. Bob Pease, a highly respected analog electrical engineer who has for several years written the *Pease Porridge* column in Electronic Design Magazine,[†] often devotes his column to discussing ways some of his colleagues have "gone wrong" using the SPICE analog electronic simulation program by using it in problem domains where it provides incorrect answers.

The need to create and modify tools   No tool is perfect. If the student does not understand how a tool works, he or she will never be able to modify or enhance the tool. Here the aims of the student and the course must be taken into consideration; it might be appropriate to teach tool usage without teaching the underlying principles of operation in a vocational training environment, whereas in a research university the student would be expected to understand the principles underlying the operation of the tool, and perhaps even be able to modify the tool or create a new tool.

This leads us to the observation that the proper mix of hand skills versus tool skills is determined by the pedagogical objectives of the teacher as the curriculum and syllabus are designed, and the student as he or she is deciding upon a course of study. That having been said, it is the opinion of the author that where tools are involved a skill or concept is not well and truly learned until skill can be demonstrated in three areas: the "hand method" of solving problems in the area, the "tool" method of solving problems in the area, and an understanding of where and how the two methods can and should be applied to the solution of "deeper" problems.

With this philosophical background, this paper now proposes a system to assist in the imparting of knowledge, the Teaching, Training, and Learning, or TTL System.

The TTL System

The TTL System aims to provide teacher and student with a collection of tools to facilitate the learning process. The part of the learning process that the TTL System aims to improve is asking questions, getting responses, assessing those responses, and providing feedback to the student.

The teaching process, when viewed by its practitioners, is complex. It involves designing new courses, modifying old courses, teaching courses, designing lectures, lecturing, holding office hours, answering questions, asking questions, providing drill exercises, and assessing student performance through homework, projects, and examinations. The TTL System provides a formal, automated approach to the part of this process that deals with the presentation of concepts, problems, homework assignments, exams, and drills.

[†]   Electronic Design, Penton Publishing Company, Cleveland, OH, 1992-1996.

Components of The TTL System

The TTL System posits an Knowledge Base, KB, and three entities, the Teacher, the Trainer, and the Student. The terms Teacher and Trainer are used to distinguish the more abstract, or Teacher aspects of the job, such as adding to or modifying the contents of KB, from the more concrete, or Trainer, aspects of the job such as providing drill and examination questions, answers, and grades. In reality, we are all Teachers and we are all Trainers, but we find this distinction helpful in defining the way in which the TTL System is to be used. In the next sections we define Teaching, Training, and Learning, and in so doing, we define the roles of the Teacher, the Trainer, and the Student.

Teaching   We define Teaching as the attempt to convey knowledge to others. Assisting others in their attempt to enter more fully into the noosphere, so to speak. Both verbal and nonverbal communications are employed in this attempt, though in a very real sense, the teacher can never convey knowledge to the student. The best the teacher can do is explain concepts, ask and answer questions, and measure the performance of the student toward acquisition of knowledge.

Many, including the author, are employed in the teaching profession. Most often, the entity that pays the salary of the teacher is in the business of certifying the performance of students who were taught there. This requirement of certification implies a certification method, a grading system. Hopefully most would agree that the grading system should be "as formal as possible," meaning that there are formal and informal parts to the grading process, but wherever possible rigorous, standardized methods be used to assess performance. This paper deals only with the formal part.

The Teacher builds, modifies, and augments the KB to improve its ability to assist in the acts of
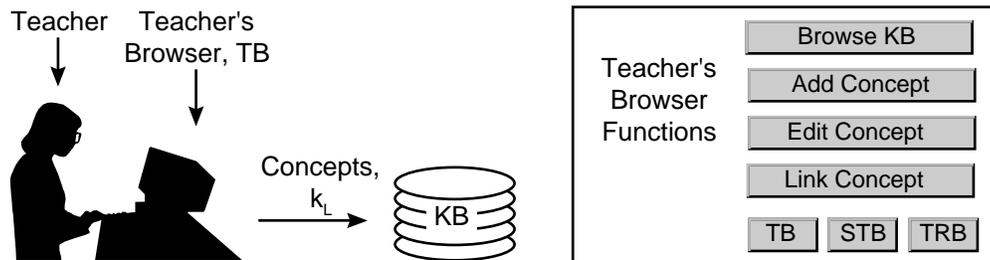

Figure 1   **Teacher's Browser and Browser Functions**

teaching training and learning. The Teacher does this by defining, linking, modifying, etc. concepts in the KB. The L subscript in the Figure refers to the abstraction level of the concept, and is discussed further on page 12. Concepts can include presentation materials, and interactive learning tools such as simulators.

Training   Training is the providing of frequent, detailed, individual instruction toward the development of some skill. One example is the training coach who schedules freethrow practice shots by the basketball team, and who keeps statistics on performance. Other examples would include the college professor who teaches mathematics, computer science, computer engineering, or electrical engineering, for example. The common threads in these areas are: 1) the need to explain knowledge and skills, 2) the need to provide drill exercises for student practice, and 3) the need to assess student performance. The training process can be thought of as one of asking questions and providing answers to questions. However, recall that by definition we are here treating only formal questions and answers, and are thereby excluding a significant and probably major factor in training and learning. Nonetheless most would agree that any assistance that the trainer can get in the way of tools that provide more and better questions and answers for student drill and assessment

are desirable, perhaps highly so.

Almost by definition, effective use of these tools will increase Teaching and Training "leverage." Leverage works in two complementary directions: on the one hand, automated drill tools provide the Student with "higher quality" learning, because the Student gets much more practice at answering questions that are likely to be asked on examinations. On the other hand, the Trainer can provide this training to more students than would otherwise be possible. (We should note here the not original observation that teachers and trainers are assessed at least in part, on their perceived ability to teach and train students, which is in turn measured at least in part by the grades given to the students. This leads, if unchecked, to an inflationary spiral in student grades and a resulting deflationary spiral in the perceived value of a grade. This topic also is outside the scope of this paper, falling as it does into the "ethics" category.)

The formal components used by the Trainer are the Knowledge Base, and extracted from it a knowledge domain, k_L. The Trainer uses the Problem-Solution Generator, PSG, to extract Solu-
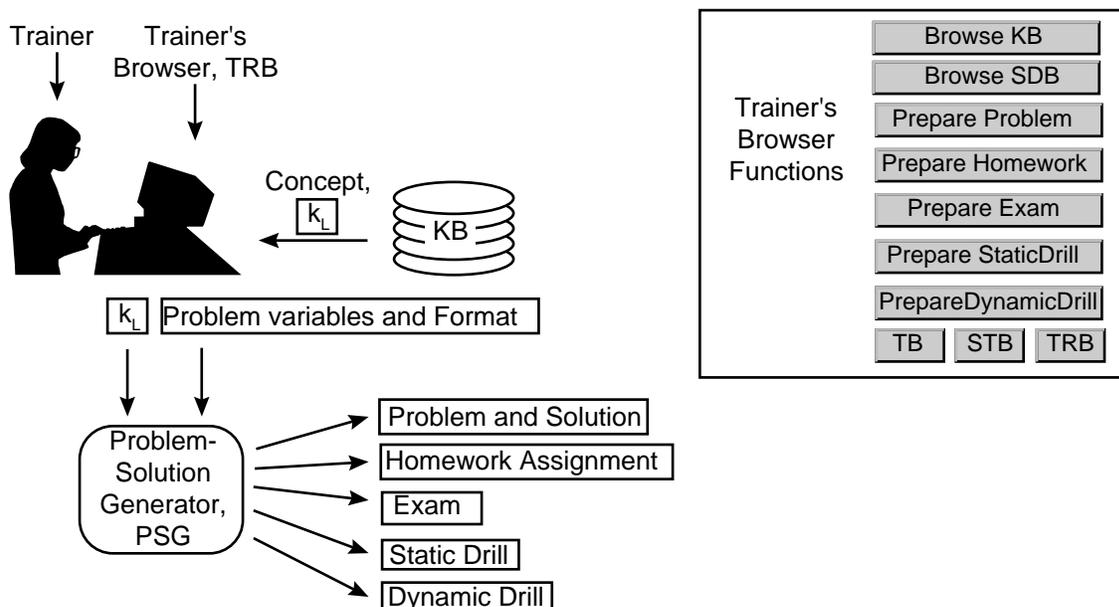

Figure 2   Trainer's Browser, Trainer's Browser Functions, Generating a Problem and Solution

tions and Problems from k_L. Problems are formatted in a way that the Trainer hopes will be comprehensible to the Student, and are presented to the Student for a Response. Problems can be collected into lists for use as homework assignments, exams, and prepared static drills, where the drill problems are generated in advance. Dynamic drills can also be prepared. Here each problem after the first, or root problem is generated as a result of Student Response, until a final leaf problem is solved correctly.

Learning: Learning is the act of acquiring skills, facts, and concepts. Let us begin with the presumption that the Student has the dual objectives of acquiring knowledge and acquiring a Grade. In a sense, "moving further into the noosphere," while also "scoring high." The learning process is of the most complex nature, but in our terms it means acquiring static knowledge, and at the same acquiring the skills required to demonstrate proficiency within the knowledge domain, k_L. While not wishing to debate the point, the author is of the view that acquiring skills is a precondition to acquiring knowledge, knowledge being both a collection of facts and skills and the abstractions that connect them.

9

Student Drill   In the TTL model of a Dynamic Drill, the Student is presented with the external form of a Problem, and provides in return a Response. The Response is analyzed by the TTL System, which, in turn, provides the Student with another problem. This process continues until the Drill is complete.
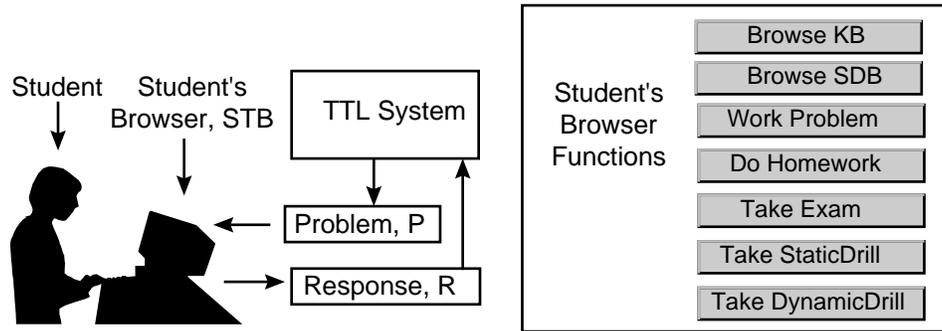

Figure 3   Student's Browser and Browser Functions

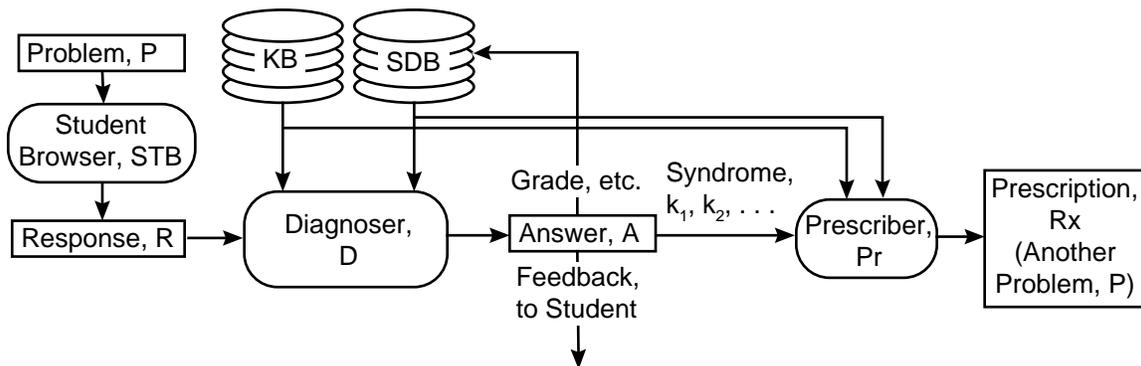The analysis of the Response takes place in two phases, as shown in Figure 4.

Figure 4   Components of the TTL System Involved During a Drill

The Response may be of any form, including no answer at all. The Diagnoser, D, with access to KB and the Student Data Base, SDB, coerces the Response to an Answer, A. An Answer has a number of parts, the three most important being a grade, g, student feedback, which may be of arbitrary form, and a Syndrome, Σ, which is a list of unmastered concepts. The grade, g, is a number between 0 and 1. A value of 0 is assigned to a totally incorrect Response, and 1 to a totally correct Response. The Syndrome includes concepts that the system infers to be missing and concepts that are found to be misapplied. In the case of a Student Drill, the Syndrome is input to the Prescriber, Pr, which, again has access to KB and SDB. The prescriber analyzes the syndrome and outputs a Prescription, Rx, which is, of course, another Problem, P´.

A Drill is initiated by the Trainer by specifying an initial, or root Problem, and a final, or leaf prob-

lem which the Student must solve correctly in order to demonstrate mastery of the concept:
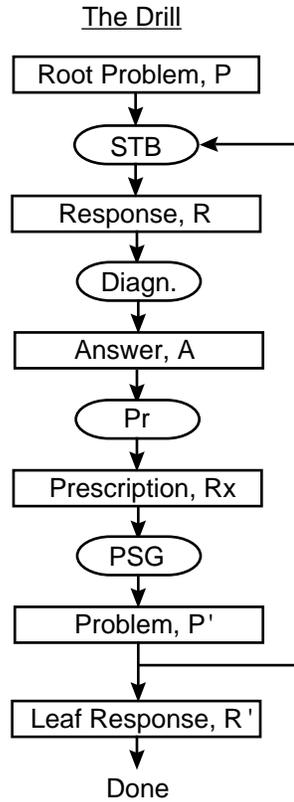
The Drill

Figure 5   The Drill

Details of these TTL System components are given below in the formal definition.

Formal Definition of the TTL System

Here is the formal definition of the TTL System, except for definitions of Teacher, Trainer, and Student, which are left as an exercise. We provide the formal definition in the left column and a simple running example of unsigned binary addition in the right column.

Definition	Example

Definition

The knowledge base The Knowledge Base, KB, is a directed acyclic graph, $KB = (K, E,$ with concepts, K for nodes and edges E connecting them:

$$KB = (K, E) \text{ , where}$$

$$K = \{k_0 \ldots k_n\} \text{, and}$$

$$E = \{ (k_i, k_j) \mid i \neq j \, , \, i, j \in 0 \ldots n\}$$

Concepts Each concept, k, is a connected directed bipartite graph with a set of $p \geq 1$ definitions, D, and a single mapping node, m, with indegree p-1 and out-degree 1.

Mapping Node Components All mapping nodes will have several components relating to node definition, formatting, diagnosis, and prescribing. All nodes have default values for these components. We will discuss these components later.

Concept Levels Concepts that are essentially identical except for their level of abstraction are organized into a list that represents the concept as a whole. All list elements have the same name but a different label, L, representing the concept level. The purpose of having several levels to a concept is to allow multiple definitions of the same concept at different levels of abstraction, proceeding from abstract to specific. Three levels defined here are Abstract, used to name the concept's nodes, Concrete, used for formal definitions, and Machine, used to generate problems and solutions.

Abstract Concepts The element at the head of the list is referred to as the abstract concept, k_{Abst}, and its level name is *Abst*, abstract. The abstract concept's definitions, D, and mapping
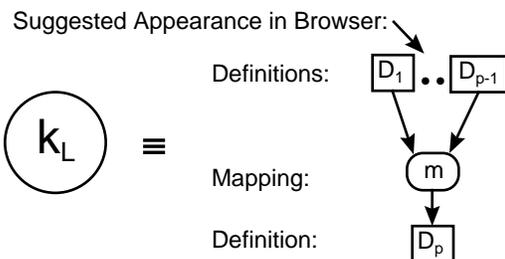
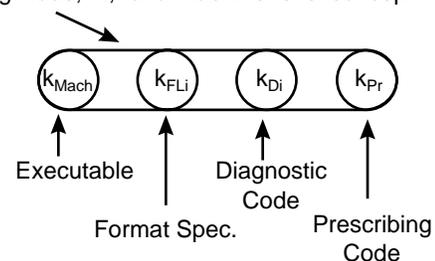## Example

Examples and Figures

The Knowledge Base, KB
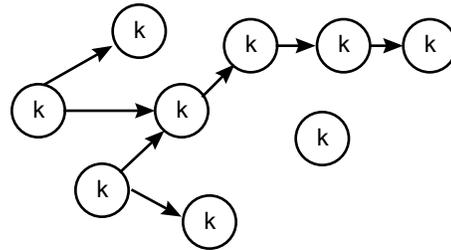
A Concept is a Directed Bipartite Graph

A Mapping Node Has Several Components

We will use a **running example** of the concept of **unsigned binary addition**. This example will show the abstract, concrete and machine level definitions, the appearance of the definitions in the Browser, and diagnostic and prescriptive code to be used in a Drill context.

Definition	**Example**

Definition

node m are of type Name. The purpose of the abstract concept is only to provide names for the definitions and mapping nodes.

A node naming convention has not yet been defined, but will be after further experience with KB. Two possibilities are function names, and compound names reflecting the inheritance of the node. The latter may simplify traversal of the knowledge base during diagnosis and prescription.

Two other levels are defined here: the concrete level and the machine level.

Concrete Concepts   The concept may have a formal definition as well. If so, the formal definition, k_{Con}, is at the concrete level, *Con*. The definition may be written using any formal specification language. One reason for providing a formal definition for a concept is to provide an unambiguous definition for the concept. A second reason is to allow automatic generation of the machine level concept. Even if the concrete concept cannot be translated automatically into code, it is still a worthwhile endeavor to write the specification at the concrete level, if only in pseudo-code, as it focuses the Teacher upon the formal meaning of the concept. The specification should match as closely as possible the execution behavior of the machine concept. In this way the concrete concept serves as documentation for behavior of the machine concept.

Machine Concepts   The purpose of the machine-level concept is to generate solutions and problems. The machine level concept, k_{Mach} at level *Mach*, is comprised of machine readable definitions and an executable mapping. As such, code and values from the domains of the definitions can be input to the Problem-Solution Generator, PSG, defined later. The PSG executes the machine code with specified inputs thus providing a Solution, s, a Problem, P, and Unknowns, U, defined below. If there is more than one
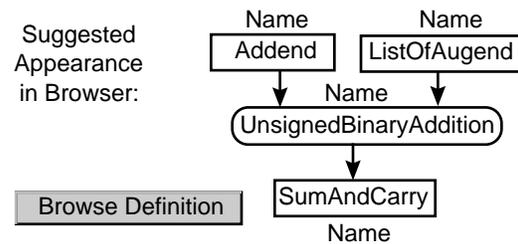
Example

### Three concepts levels for the Concept of Unsigned Binary Addition
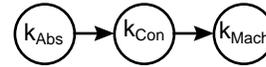
The **abstract concept** of unsigned binary addition is formed by merely naming the elements. In this model the input definitions are an addend and a list of augends, and the output definition is the sum and carry-out of the msb.

Abstract Concept: Unsigned Binary Addition

The **concrete concept** node has a slightly different node structure, with some new nodes, reflecting the presence of WS, word size, and N, number of augends, at the concrete level.

Concrete Concept: Unsigned Binary Addition
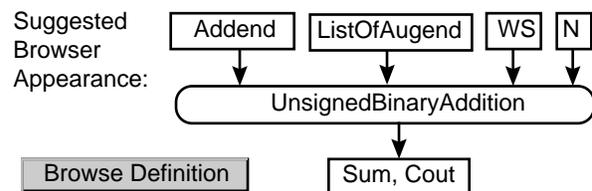
In this example the concrete concept was written in a pseudo-version of the specification language RTN, Register Transfer Notation.

RTN was chosen because of its relatively high level of abstraction and its close relationship to

| **Definition** | **Example** |

As discussed in the example column to the right, there is a uniform parameter definition and passing mechanism across all machine concepts. This is to hide the underlying function implementation from the caller. For example some machine concepts may be built upon legacy code written in another language, may be available only as the binary code, or may even be running on a remote server. In many cases the Teacher must write "wrapper" code that reformats the function call in the appropriate format.

Solutions   A Solution instance, s, is generated from k_{Mach} by selecting instances of the input nodes, I, and allowing m to produce o, the output of the mapping:

$$s = (I, m, o), I, o \in D, m \in M$$

Informally, this is the act of applying m to the input value or values, thus producing o, the output value. It is the Problem-Solution Generator, PSG, that performs this operation. The PSG will be described after the defining problems and Problems.

problems—Internal Form   The internal form of a problem, lower case p, is generated from the solution, $s = (I, m, o)$, by the Trainer. The Trainer instantiates the problem, p, by defining some proper subset of s as unknowns, $u \subset (I, m, o)$.

$$p = s - u, \text{ or } u = s - p$$

The normal instance would have $u = \{o\}$. That is, an ordinary problem provides the inputs as knowns. In actual implementation within the TTL System p is specified by the pair (s, u).

The solution is generated first so that it can be examined before a commitment is made to generate a problem. This helps avoid awkward problem instances and ensures that a solution exists.

Example

machine objects and operations:

Concrete concept for Unsigned Binary Addition written in RTN

```
WS:⟨4..0⟩:                ;Word Size in bits
Addend⟨WS⟩:
Augend⟨WS⟩:
N:                        ;Number of augends
Cout:                     ;Carry from WS addition
ListOfAugend:(Augend1, Augend2, ... ,AugendN):
Sum⟨WS⟩:
UnsignedBinaryAddn :=
Check(N>=1):              ;Add at least two numbers
Sum <- ... ;
Cout <- ... :
. . .
Sum := Sum | Cout;        ;Cout is concatenated w. Sum
```

Notice that the definition is parameterized on word size, WS, and number of augends, N. This parameterization will allow the problem domain to be narrowed or widened to suit pedagogical objectives without rewriting the underlying program.

The **machine-level concept** definition for unsigned binary addition has two components: one specifying its appearance in the browser, and the actual machine code. In the ideal case the former is generated from the latter. The browser specification provides exact information about the number and type of parameters without the need to examine the underlying machine code.

This allows the Teacher to write formatting, diagnosis, and prescription code, and the Trainer to prepare input parameters for the generation of Solutions and Problems. The obvious purpose of the machine code is to compute the output from the inputs. Its parameters might appear as follows, in a Java-type language, as-

14

Problems–Extern. Form   Before a problem can be delivered to the Student it must be converted to external form, upper case P, by including additional information for formatting, diagnosis and prescribing, information about the starting and ending problems of a drill, information about time constraints, and student information.

The Problem Data Structure

P={	(Problem)	P
k,	(concept)	k
L,	(level)	L
k_{fLi},	(format description)	k_{fLi}
(s,u),	(Solutions and unknowns)	(s,u)
(Rt,Lf),	(root and leaf of drill tree)	(Rt,Lf)
SI,	(time constraints)	SI
N }	(Student number)	N

The concept, Level, and format description are actually organized as a tuple, and P may contain a list of these. In the following data structures we abbreviate this triple as k_{Li}:

$$k_{Li} \equiv (k, L, k_{fLi})$$

Allowing lists of k_{Li} permits a diagnosis to include more than one concept.

The need for a data structure with so many components will become more obvious when we discuss data flow through the TTL System.

Formatting the Problem   The p, s, and u generated from PSG are in internal machine form. The formatting node, k_{fLi}, describes the conversion of p from internal form to external form. The external form is unspecified; it may be text, synthesized speech, video, or force feedback, for example. As the subscript *i* implies, there may be more than one formatting

suming that a list package is available.

Machine Concept: Unsigned Binary Addition
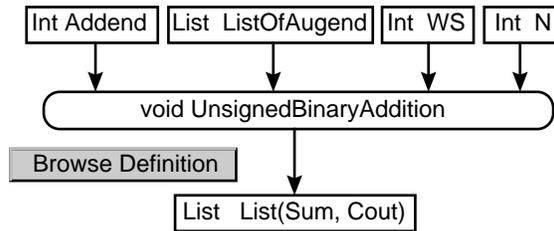
[The Trainer selects a the machine concept]

The mapping function must contain default values for all input parameters. This is so machine concepts can be invoked with any arbitrary subset of input parameters. It is the responsibility of the function caller to ensure that the specified set of input parameters will yield a valid result. For example, calling the function above with Addend = 110, Augend = 011, WS = 2, N = 2 will cause the function to return an invalid value. There need be no assertions built into the mapping functions to ensure a valid result. That responsibility falls upon the Trainer if the system is being operated manually, or upon the Prescriber if Problems are being generated for a Drill.

Notice that the number, type, and name of the definition nodes must *exactly* match the parameters expected by the underlying machine code. This is because the structure of the machine concept node serves as documentation to the programmer. Notice also that in conformity with the definition of a concept there is only a single output node. This is deemed desirable as this node may be passed around to arbitrary nodes during diagnosis and prescribing. The node may, of course, be a composite structure.

The requirement for an exact parameter match makes the use of automatic tools to build the machine concept node highly desirable.

The next Figures shows how the Trainer

15

Definition

description for a given k_{fL}. Formatting nodes are components of the concept node, k_L. In the absence of a specific formatting node at a certain concept node, there is a default format that converts the internal form to external form.
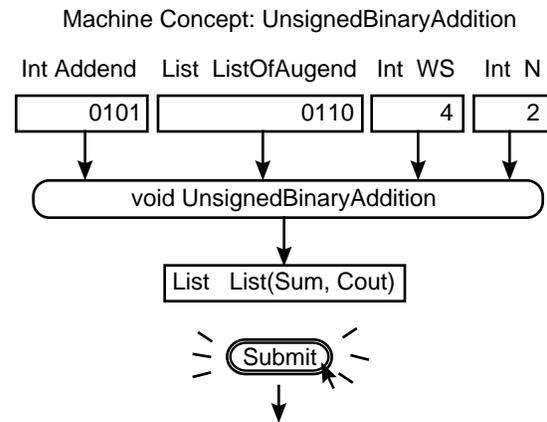
Note that k_{fLi} is not a function. It is a description of how the problem is to be converted to external form by some other function. It is the job of the browsers to accept a Problem (continuing a formatting description) and convert it to external form. One reason for using this method is bandwidth. There will nearly always be more data present in the external form than in the internal form. Synthesized speech or the generation of 3-d graphics, would be examples.

Problem-Solution Generator The PSG generates a Problem from a set of values and the other information defined above. The p, s, u, defined above are generated by specifying the concept, k, its level, L, and format definition, i, resulting in k_L and k_{fLi} being extracted from KB. A solution s is generated by employing k_L to convert input values v into s, and then specifying unknowns u, resulting in the generation of p and s. One could say that k_L "fills in

Example

Browser might appear when the Trainer is preparing a Problem by manual input of values. The Trainer first enters the input values and invokes PSG to generate a Solution.

The Trainer prepares to generate a Solution

Machine Concept: UnsignedBinaryAddition

Int Addend	List ListOfAugend	Int WS	Int N
0101	0110	4	2

void UnsignedBinaryAddition

List List(Sum, Cout)

Submit

Here is the output of the PSG:

The PSG Generates A Solution

Int Addend	List ListOfAugend	Int WS	Int N
0101	0110	4	2

void UnsignedBinaryAddition

1011, 0000

List List(Sum, Cout)
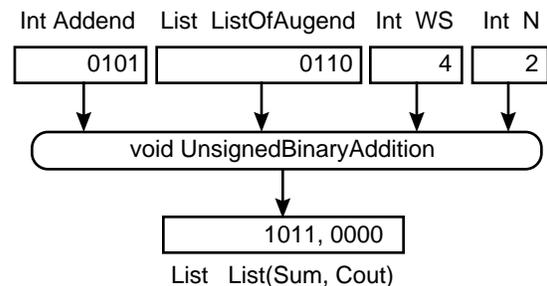
the blanks" to generate s. The figure below

The Problem Solution Generator, P SG

Inputs   **Dat. Str.**
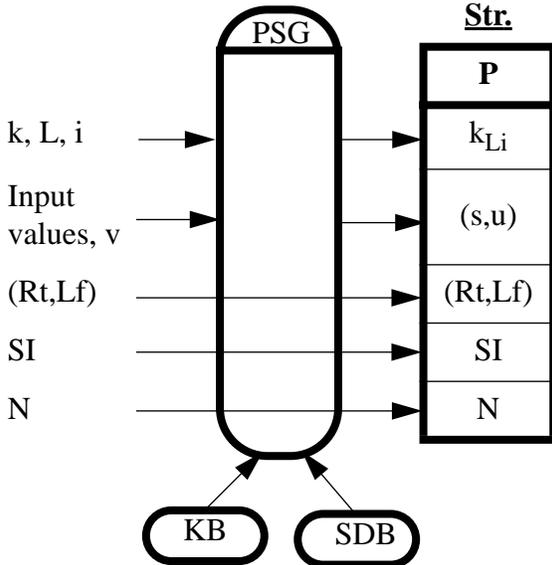
shows schematically how the parameters passed to PSG are used. Input arrows that pass through the PSG, such as those from SI, and N are not used internally within PSG, but are merely assembled into the P data structure. Input arrows that have an arrowhead at the input to PSG and another arrow going on to the P data structure are used within PSG, but are inserted into the data structure unchanged. Input arrows that terminate at PSG, such as the arrow from Input values, are used within PSG and not passed on. They are the equivalent of **in** parameters. Arrows originating at the output of PSG, such as the one going to (s,u) in the data structure are the equivalent of **out** parameters.

Browser Contexts There are a number of ways that the Browsers can be used, and the desired way is specified by the Browser Context. The Teacher and Trainer Browser Contexts must be discussed, because they affect the way Problems are handled by the System. There are five Browser contexts used in generating Prob-

---

A Problem is generated from the Solution by specifying the unknowns:
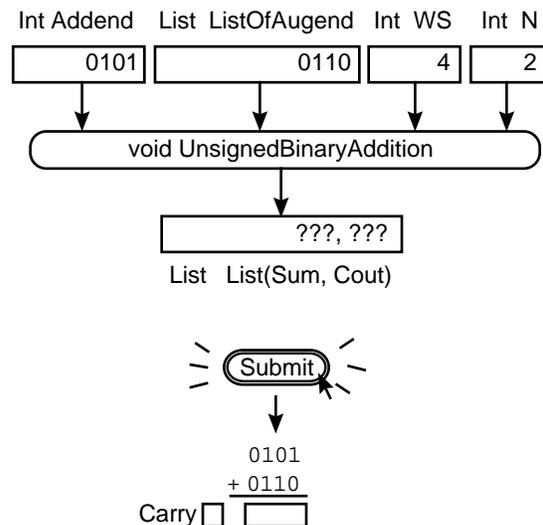
The Trainer Generates A Problem

[Trainer defines Unknowns by clicking on them and inserting "???"]
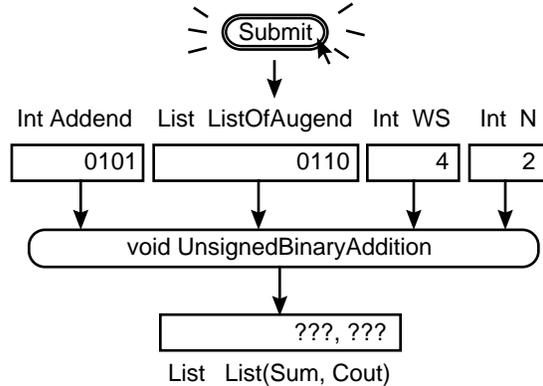
Finally a format is selected:

The Trainer Selects a Format

Notice that since the PSG computes the Solution first, the Trainer can "play" with the inputs to arrive at a Problem that illustrates or avoids some particular facet of the problem domain.

17

Definition	**Example**

lems and Solutions:

    Browser Context =

$$\{ \text{SingleProblem} \mid$$
$$\text{Homework} \mid \text{Exam} \mid$$
$$\text{StaticDrill} \mid$$
$$\text{DynamicDrill} \}$$

The first four contexts, SingleProblem, Homework, Exam, and StaticDrill result in the assembly of one or more problems into a problem list that can be viewed by browser or stored in a file, but which are not worked interactively. In these contexts responses are asynchronous, and are generally obtained by distributing the Problem, Exam, etc. for working by the Student in class, at home, etc., with Responses being returned on paper or by other equivalent means. The differences between the first three contexts are mainly in the grading semantics.

A StaticDrill is generated by the Trainer specifying Root and Leaf problems, and allowing the system to generate the static drill.

The DynamicDrill, by contrast, relies upon a Student Response at the Student Browser to each problem, and the Response is considered by the Diagnoser and Prescriber in formulating the next Problem. During the DynamicDrill the Student is "in the loop."

Responses In much in the same way as a problem, p, is composed into a Problem, P, by adding additional information, the Response,

Definition		**Example**

R, is composed from the student reply, x.

R={   (Response)

        k_{Li},   (concept, level, and format)

        (s,u),   (Solutions and unknowns)

        $(x \cup \perp)$,   (reply from Student, if any)

        (Rt,Lf),   (root and leaf of drill tree)

        SI,   (time constraints)

        SO,   (performance time)

        N }   (Student number)

Much of this complexity is due to having to carry along information needed during and after a Drill.

Student Browser   It is the Student Browser, STB, that, during DynamicDrills accepts a Problem and converts it into a Response:

       STB: $P \rightarrow R$.

Student Browser, STB

19
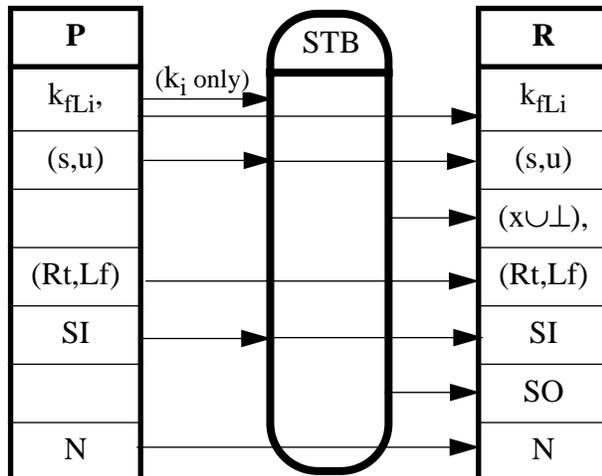
Definition

The Student Browser accepts the problem, P, and uses the formatting information, k_{fLi}, the actual problem, (s,u), and time constraints, SI, to generate a reply, x, x∈ UniverseOfValues, or ⊥ if the process times out without student submission. It does not actually use the root and leaf problems, but only passes them on in the Response. It does, however, track student performance time, and returns it in SO. One could say that STB appends a reply and performance time to the Problem.

The reason for specifying x as being in the UniverseOfValues is to avoid doing much syntax or semantic checking of x at the STB level. The question of how much syntactic and semantic checking should be done at the STB level is an open one, but one leans toward the opinion that most of that checking belongs in the diagnoser, where a more thorough analysis is possible.

Answers   The Student Response, R, is converted to an Answer, A, by the Diagnoser, D. The Answer data structure is shown below:.

A={	(Answer)
k_{Li}, Δ $k'_{Li},$ $k''_{Li},$ . . .	List of (concept, Level, format) Head has additional diagnostic information, Δ)
(Rt,Lf),	(root and leaf of drill tree)
F,	(feedback to student)
g,	(grade)
N }	(Student number)

Diagnoser   The Diagnoser compares u and SI with SO and the student reply, (x∪⊥), and returns a grade, g, and a Syndrome, Σ which is a list of k_{Li}. The head concept also has additional diagnostic information, Δ, which is local to the current k_{Li}, and which describes "moves" in the drill tree. This information is used by the Pre-

Example

Drills, Diagnosis, and Prescription   The "manual" method shown above works fine for Single Problems, Homeworks, and Exams, but in order to generate drills, the Student Response must be compared with the value expected, a syndrome generated, and then the syndrome used to provide a prescription, which is used to generate another problem.

The syndrome is generated by the Diagnoser, which has access to the KB and SDB. It generates a list of (k, L, i, Rt, Lf, SI) for further remedial drill. Sophisticated AI methods may be employed to arrive at the list. The TTL definition does not currently specify how the list is generated, but for now it is assumed that the first node to be visited is the node of the current drill.

Root and Leaf Drill Problems   Let us assume that the Root and Leaf problems are specified in pseudo-code as:

Root and Leaf Drill problems

Rt = (WS=1, N=2, Addend = RND, Augend = RND;)

Lf = (WS=8, N=4, Addend = RND, Augend = RND;)

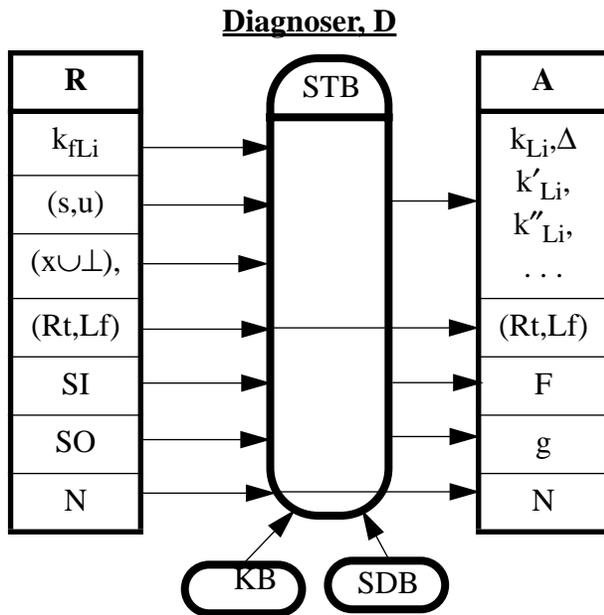;assumes that all Addends and Augends are random numbers.

Thus this drill starts with the Student adding two, single digit binary numbers, and ends with the Student adding four, 8-bit numbers.

Diagnostic Code   In the example below, all diagnosis is done by the current node, UnsignedBinaryAddition$_{Mach}$ in this case. The code fragment is written in pseudo-code:

| **Definition** | **Example** |

Definition

scriber. These are encapsulated in the Answer data structure shown above.

In making this comparison, the Diagnoser may consult KB and SDB, the Student Data Base. In any event, diagnostic code is stored at each concept level, k_L. Inputs and outputs are shown in the figure below:

Diagnoser, D

R	STB	A
k_{fLi}		k_{Li}, Δ
(s,u)		k'_{Li},
$(x \cup \perp)$,		k''_{Li},
		. . .
(Rt, Lf)		(Rt, Lf)
SI		F
SO		g
N		N

KB   SDB

The intent of the $'$ and $''$ symbols on the concept triples k_{Li} is that the syndrome may include different concepts at different levels having different formats.

The Prescription, Rx   The Prescribing code examines the Answer above, and formulates a set of new Problem input values in the domain of k_{Li}, the head of the list. These values, v, are

Example

Diagnostic code for UnsignedBinaryAddition$_{Mach}$

```
if (u ≠ x) {              ; incorrect response.
    g = 0;                ; grade of zero.
F = ("Incorrect Response. Correct Response is", u.);
SyndromeHead = (k, L, i,
            Δ = BACK);    ;simpler problem.
}
else {
    g = 1;
    F = ("Correct Response!");
    if ((WS ==WS_Lf) && (N==N_Lf))
            SyndromeHead = Nil;   ;drill is finished
    else SyndromeHead = (k, L, i,
            Δ = FWD);       ;more complex problem.
}
SyndromeTail = Nil;
```

The Prescriber reads the list head, and generates a Prescription based upon the information in the list head.

Prescriptive Code   Prescriptive code for the example might appear as follows:

Prescriptive code for UnsignedBinaryAddition$_{Mach}$

```
if (Δ == BACK) {     ; incorrect response.
            WS = (WS ==WS_Rt) ? WS_Rt : WS/2;
            N= (N==N_Rt) ? N_Rt: N/2;
            Addend = Augend = RND);
}            ;simpler problem.
else if (Δ == FWD) {
            WS = (WS ==WS_Lf) ? WS_Lf : WS*2;
            N= (N==N_Lf) ? N_Lf: N*2;
            Addend = RND);
            Addend = RND);
}
    ; In actuality more complex tests for increasing
    ;and decreasing WS and N would be required.
```

Thus this simple Diagnosis and Prescription merely increases or decreases word size and number of augends in the next problem depending upon whether the Student provided a

Definition	**Example**
inserted into the Prescription data structure:	correct or incorrect response.

The Prescription Data Structure

Rx={	(Prescription)	Rx
k_{Li},	(concept, level and fmt list)	k_{Li}
values	(for input to PSG)	v
(Rt,Lf),	(root and leaf of drill tree)	(Rt,Lf)
SI,	(time constraints)	SI
N }	(Student number)	N

There is a complication if there is a new k_{Li}, because in this case there may be a new drill tree for the new k_{Li}. In this case the old root and leaf problems must be attached to the old k_{Li}.

The operation of the Prescriber, PR, is shown below.

The Prescriber PR

A	PR	Rx
k_{Li} (list)		k'_{Li} (list)
F		v (values)
(Rt,Lf)		(Rt,Lf)
g		SI
N		N

KB   SDB
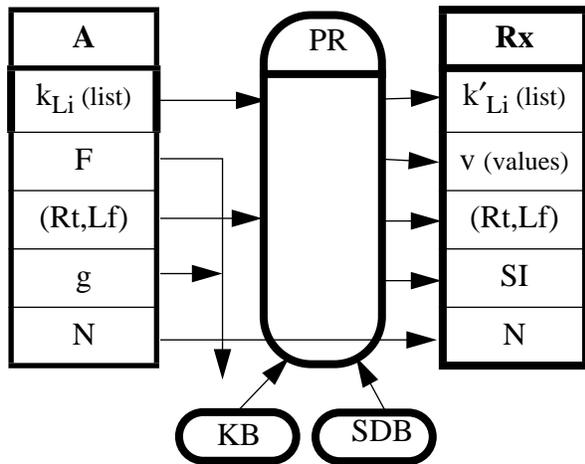
The meaning of the prime symbol in k'_{Li} (list) is that the Prescriber may remove the head of the list if the student has shown mastery of the list head concept. In this case it returns the tail of the list.

F and g are input to SDB, and F will also go to the STB if the context is Dynamic Drill.
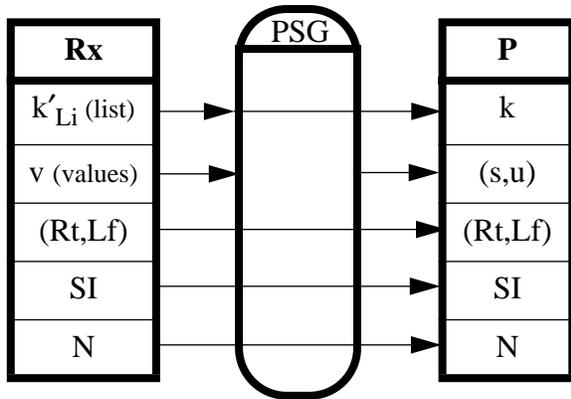
Note that Rx contains exactly the information needed for input to the PSG for the generation of another Problem, P′:

Again, The Problem Solution Generator, P SG

Rx		P
k'_{Li} (list)	→ PSG →	k
v (values)	→	(s,u)
(Rt,Lf)	→	(Rt,Lf)
SI	→	SI
N	→	N

It is this repeating process that makes a Drill, as shown in Figure 5, page 11.

A Note about Grading

It is possible to use the Diagnoser and Prescriber to grade Homeworks, Exams, and Static-Drills, by adjusting the Prescribing Code so that the Prescription is just the next problem in the series.

SI and SO

SI, the time constraints, and SO, the performance time, are merely details in the overall TTL design, but this suggests that they might

23

Definition	**Example**

be composed as follows:

    SI={   (Time Constraints)

    EarliestProblemStartTime,

    LatestProblemStopTime,

    MaximumProblemResponseTime,

    AUTH } (security information)

:

    SO={   (Time Constraints)

    ActualProblemStartTime,

    ActualProblemStopTime,

    AUTH } (security information)

Times must be coercible to [Date, GMT] with a precision of one second and an accuracy of one minute.

Definitions

Knowledge: Static knowledge and dynamic knowledge.

Dynamic Knowledge: knowledge that is encoded in living beings. The ability to use facts in interacting with the world.

Static Knowledge: knowledge that has been encoded in non-living matter.

Teaching: the attempt to convey knowledge to others

Training: providing of frequent, detailed, individual instruction toward the development of

Definition	**Example**
some skill.	

Skill: The ability to perform a specific action that demonstrates a concept.

Learning: the act of acquiring skills.

Concept: A set of definitions and the mapping between them.

323 Preface Teaching and Learning at a Distance is written for introductory distance education courses for preservice or in-service teachers, and for training programs that discuss teaching distant learners or managing distance education systems. This text provides readers with the basic information needed to be knowledgeable distance educators and leaders of distance educa- tion programs. The teacher or trainer who uses this book will be able to distinguish between appropri- ate uses of distance education. ORGANIZATION OF THE TEXT Teaching and Learning at a Distance has three types of chapters—foundation chapters, teaching and learning chapters, and managing and evaluating chapters. However, when analyzing the learner and teacher questionnaire results from a variety of schools it became apparent that the teacher-learner relationship was central to the initiation and support of autonomous learning irrespective of the curricula background of the school. The teacher's role is viewed as paramount in the development of self-regulated learning. The Importance of Autonomous, Self-Regulated Learning in Primary Initial Teacher Training. Sue Oates*. Department of Social Work, Education and Community Wellbeing, Northumbria University, Newcastle upon Tyne, United Kingdom. This article grew from the author's doctoral study into the development of self-regulated learning (SRL) at the end of the primary school. Used to support both teaching and learning, technology infuses classrooms with digital learning tools, such as computers and hand held devices; expands course offerings, experiences, and learning materials; supports learning 24 hours a day, 7 days a week; builds 21st century skills; increases student engagement and motivation; and accelerates learning. Technology also has the power to transform teaching by ushering in a new model of connected teaching. This model links teachers to their students and to professional content, resources, and systems to help them improve their own instruction and ... Online learning opportunities and the use of open educational resources and other Excellent teaching – including the development of the learner attributes – is the single most significant factor impacting on learners' academic performance and personal growth that a school can influence. Successful schools, and successful school systems, develop and nurture highly skilled teachers who are encouraged to be creative professionals working in a collaborative culture. They are the basis for the teaching and learning programmes that teachers use to express and deliver their wider expectations of learners. When planning lessons, teachers can build in opportunities for students to participate in learning activities and events that help grow and advance the development of learner attributes – consistent with the active learning approach described in the previous section. Child-oriented learning provides a basic foundation of knowledge and helps develop critical thinking skills. Adults have an existing base of knowledge and life experience. They seek out continuous learning based on personal interests, wants, and needs. We've made a review of six of the most popular adult learning theories to see how each can be used to support overall learning needs. Andragogy. Developed by Malcolm Knowles in 1968, andragogy is described by its creator as the art and science of helping adults learn. Your employees will appreciate the chance to omit redundant training and focus on job-essential skills. Your business will also benefit from this approach by gaining a productive new employee in the shortest time frame possible.