Université de Montréal

Interactive Visual Management of Curriculum

par

Ruibiao Guo

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures

en vue de l'obtention du grade de

Philosophiæ Doctor (Ph.D.)

en informatique

Avril 1999

Université de Montréal

Faculté des études supérieures

Cette thèse intitulée:

Interactive Visual Management of Curriculum

Présentée par:

Ruibiao Guo

a été évaluée par un jury composé des personnes suivantes:

DSSOULI, RACHIDA	(Présidente-rapporteuse)
KALTENBACH, MARC	(Directeur de recherche)
FRASSON, CLAUDE	(Codirecteur)
NKAMBOU, ROGER	(Membre du jury)
BOULET, MARIE-MICHELE	(Examinatrice externe)
PIERRE, SAMUEL	(Représentant du doyen de la FES)

Professeur titulaire
Département de génie électrique
et de génie informatique
Ecole polytechnique

Thèse acceptée le: 99.06.09

Abstract

A curriculum is a model for organizing and managing all kind of knowledge in computer-based tutoring systems, which includes domain knowledge, teaching strategies and didactic resources, and students' knowledge. The problem of curriculum modeling concerns the mechanism to organize and control the knowledge effectively.

Numerous works have been carried out proposing solutions. These results, on the one hand, show that most real curricula are very complex heterogeneous knowledge systems. On the other hand, interactive, integrated and simple organization models and tools should be defined to build systems useful in practice in which tutoring outcomes, tutoring activities and didactic resources can be effectively organized and manipulated.

A model is proposed to represent and organize domain and pedagogical knowledge, which provides

- A simpler and more practically useful model based on the idea of a transition network for course generation,
- A more central role of curriculum structures within an overall ITS design, and
- The practical integration of various learning theories.

This thesis approaches the modeling of knowledge and tutoring activities by relating Gagné's capability and instructional event theory to Bloom's objective theory. We propose a modeling approach of a curriculum taking into account the various aspects of domain knowledge (teaching outcomes), pedagogic (teaching) strategies and didactic (content) materials. The pedagogical aspect of the proposed system are distributed and modeled at various levels of granularity, with a particular emphasis on control exerted at the level of transition nodes at which the teaching planning is actually taking place. The organization of tutoring transition consists of modeling preliminary needs to their realization and studying the impact that can have attack of an objective on domain

knowledge. Last, didactic dimension produces a model that defines and organizes different necessary tactical means for considered resource's teaching.

A proposed model for supporting automatic generation of courses (according to learners' needs) can generate multiple alternative courses for a particular goal set. We have equally used for supporting teaching process (recommending courses, ordering tutoring activities, dynamic remedy of students' misconceptions and visual navigation).

Visualization abilities are kind of important features in our models. These abilities can help curriculum authors visually build, organize and manage curriculum. Also, the proposed system can visually display the development progress in authoring progress to raise productivity. Learners can visually set learning goals, view alternative courses covering their goals and obtain instant guidance in their learning process.

As a result, this thesis contributes to reaching the idea of an ITS both practical and useful.

Key words:

Computer-Based Tutoring Systems
Visual Curriculum
Integrated Representation and Organization of Curriculum
Visual Manipulation and Navigation
Creating Multiple Alternative Courses
Course Evaluation
Dynamic Management of Learning Process

Résumé

Cette thèse présente une façon de concevoir et exploiter des systèmes Tutoriels Intelligents (STI), essentiellement centrée sur la notion de curriculum. Un curriculum est une organisation de connaissances multiples nécessaires au bon fonctionnement du tuteur intelligent. Ces connaissances recouvrent principalement les connaissances propres au domaine à enseigner, les connaissances relatives à la pédagogie et au suivi individuel de l'apprenant utilisant le STI. Le problème qui nous intéresse ici est de proposer une structure de curriculum qui soit facile à utiliser par des auteurs de curriculums et efficace pour guider l'apprenant dans son apprentissage.

Nous nous basons sur d'importants et nombreux travaux proposant des solutions. Ce qui a motivé notre travail est que les structures de curriculum qui ont été proposées jusqu'à présent sont beaucoup trop complexes et hétérogènes pour être utilisées efficacement.

Nous proposons donc un modèle, plus simple et pratique, ayant les caractéristiques suivantes:

1 - le curriculum est une structure plus simple fondée sur la notion de graphe de transition adaptée pour la génération de cours.

2 - le curriculum tient une position plus centrale au sein d'un Système Tutoriel (STI).

3 - le curriculum intègre diverses théories de l'apprentissage.

Cette thèse aborde la modélisation des connaissances et des activités pédagogiques en mettant en relation la théorie de Gagné sur les niveaux de capacités des apprenants et la théories des évènements instructifs formulée dans le cadre de la théorie objective de Bloom. Nous proposons une approche de la modélisation qui prend en compte divers

aspects des connaissances du domaine, de la pédagogie et des stratégies pedagogique. Les connaissances et structures de contrôle pédagogique sont distribuées et modélisées à divers degrés de granularité, avec un accent particulier sur le contrôle exercé au niveau des noeuds de transitions qui constituent les pôles où l'apprentissage est planifié et réalisé. L'organisation de ces transitions tutorielles requiert un inventaire des re-requis et des conséquences de leur réalisation. Enfin le modèle permet une organisation rationnelle des ressources didactiques qui peuvent être mises en oeuvre en divers endroits du curriculum.

Notre modèle de curriculum est bien adapté à la génération de cours pour un étudiant particulier sous la forme de séquences tutorielles, incluant la comparaison de séquences alternatives pour un ensemble donné d'objectifs d'apprentissage. Nous avons également inclus divers moyen de supporter le processus d'enseignement (recommandation de cours à suivre, organisation et réorganisation des activités pédagogiques, correction des erreurs faites par l'apprenant et aides visuelles pour la navigations entre la fixation d'objectifs d'apprentissages et la réalisation d'activités pour atteindre des objectif.

Notre système fait une large place aux possibilités de gestion visuelle du curriculum. Ainsi dans le cadre de la tache de composition d'un curriculum, l'auteur peut évaluer visuellement la progression de son travail et obtenir des informations globales sur la partie du curriculum déjà réalisée et ce qui reste à faire. Similairement, l'apprenant utilisant un curriculum peut évaluer visuellement sa progression vers ses objectifs d'apprentissage et réviser ses plans pour atteindre ses (nouveaux) objectifs.

De façon générale la thèse s'organise comme suit.

D'abord nous faisons un survol rapide de l'évolution des systèmes tutoriel intelligents (STI), montrant que au cours des années les chercheurs en ce domaine ont pris conscience des facettes multiples et très variées requises par la construction d'un STI

efficace. Pour des raisons conceptuelles et pratiques il est apparu nécessaire de regrouper diverses sources de connaissances en une structure générique, c'est à dire qui puisse être utilisée pour la création de STI enseignant une grande variété de cours et de types de cours. Cette constatation fut à l'origine de la notion de curriculum, un concept qui dépasse la notion de syllabus et qui maintenant recouvre la réunion de toutes les connaissances spécifiques à un enseignement: connaissances sur la matière à enseigner, connaissance sur l'apprenant, connaissance sur la pédagogie, connaissance sur la mise en oeuvre de diverses stratégies pédagogiques et le contrôle de l'acquisition des connaissances, etc. Concrètement un curriculum se présente comme un graphe.

Les travaux les plus récents sur la structuration d'un curriculum se sont basés sur une structuration en couches fonctionnelles (connaissances du domaine, connaissance de l'apprenant, connaissance de la pédagogie, etc.). Cette approche a l'avantage de faciliter l'explicitation des diverses sources de connaissances à mettre en oeuvre dans un STI. Toutefois elle implique la nécessité d'établir des correspondances entre les divers couches, une source de grande complexité. Il en résulte que créer un curriculum selon cette structuration nécessite un effort considérable. Notre approche dans cette thèse a été de chercher à simplifier l'apparence globale du graphe représentant un curriculum sans pour autant abandonner une quelconque des fonctionnalités supportées par les curriculums précédents. Cela nous a conduit également à porter une grande attention aux moyens visuels de gérer un curriculum. Ainsi dans la thèse nous passons en revue divers modes originaux de présenter l'information et concluons sur la désidér abilité d'appliquer certaines de ces modèles au cas du curriculum. Egalement nous nous soucions de l'intégration du curriculum dans un STI. En fait dans cette thèse nous faisons du curriculum la structure de base organisant non seulement les diverses sources de connaissances à prendre en compte lors de la création d'un curriculum par un auteur (pédagogue et spécialiste du domaine à enseigner) mais également le curriculum sert à structurer et organiser l'enseignement effectif d'apprenants sous la forme d'interactions permettant une négociation entre l'apprenant et le STI portant tant sur le choix des objectifs d'apprentissage que sur des moyens et séquences d'activités nécessaires pour atteindre ces objectifs.

La partie centrale de la thèse est consacrée à la structure logicielle du système (VITCAM) que nous avons développé pour la création et exploitation de curriculums. C'est une architecture hiérarchique en couches fonctionnelles. Les niveaux fonctionnels de base créent ou exploitent les structures formant un curriculum. Du coté du processus de création et de gestion d'un curriculum (par un auteur de curriculum) la structure de base qui est obtenue est un réseau de transition (un graphe) comportant seulement deux catégories de noeuds. Certains noeuds modélisent des niveaux de capacités (niveau de maîtrise) relatives à un concept ou a une tache. Ces noeuds servent aussi à modéliser le niveau de maîtrise d'un apprenant particulier sur ce concept ou cette tache. Ces "noeuds de type capacités" sont également utilisés lors de la négociation entre le système et l'apprenant pour fixer les divers objectifs d'apprentissage.

Les autres noeuds du réseau sont appelés des "transitions". Ces noeuds correspondent à des activités dont la réalisation avec succès par l'apprenant résulte en la mise à jours pour cet apprenant des niveaux de certaines capacités. Ainsi les noeuds de type capacités servent non seulement à établir les conditions requises pour qu'un apprenant puisse participer aux activités d'un noeud de transition mais également comme réceptacles pour la mesure de l'apprentissage obtenu par suite de la traversée avec succès d'un noeud de transition. Ainsi dans le graphe un noeud de transition est toujours intercalé entre deux noeuds de capacités.

Cette structuration d'un curriculum peut apparaître assez simpliste si ce n'était que les noeuds de transition sont en fait des structures assez complexes. Un noeud de transition doit être considéré comme un mini système tutoriel intelligent. Le noeud de transition organise les activites relatives à l'acquisition de préférence d'une, mais souvent de plusieurs, capacités à divers niveaux. Il contient également une structure d'évaluation des activités de l'apprenant et peut mettre en oeuvre diverses stratégies pour remèdier à des déficiences constatées dans le processus d'apprentissage. Pour des raisons conceptuelles ainsi que d'ordre pratique il est apparu impératif de pouvoir organiser les activités

d'apprentissage au niveau d'un noeud de transition selon une nomenclature commune pour tous les noeuds de transition. Pour ce faire nous avons trouvé fort utiles les nomenclatures établies par Gagné et Bloom. Nous avons trouvé que ces nomenclatures étaient complémentaires et ainsi pour chaque noeud de transition nous avons une grille dont chaque cellule correspond à une activité élémentaire (de l'apprenant). Les diverses stratégies d'apprentissage au niveau d'un noeud de transition sont représentées par des chemins entre les cellules de cette grille. Nous avons exploré divers moyens d'exploiter cette grille pour permettre une aggrégation des noeuds de transition afin d'obtenir une représentation hiérarchique d'un curriculum.

A part ce réseau de "capacités" ed "transitions" le système VITCAM permet de générer et exploiter diverses autre structures. Ainsi VITCAM gère une structure, appelée "modèle de l'apprenant" qui regroupe des informations générales sur l'apprenant et qui est alimentée et exploitée au niveau des noeuds de transition. Une partie plus importante de la thèse est consacrée à la génération de cours spécifiques pour un apprenant. Un cours est conçu comme un sous graphe du curriculum déterminé par le choix des objectifs d'apprentissage à atteindre. Un modèle baysien est développé qui a la propriété d'ordonner diverses alternatives possibles d'un cours en fonction des caractéristiques d'un apprenant déterminé.

La thèse termine par la description du prototype créé pour valider la faisabilité du modèles de curriculum et STI proposé. Un exemple de curriculum et de cours est proposé. L'interface usager a été surtout développé pour la tache de création d'un curriculum. Pour l'instant il est encore trop abstrait pour être mis entre les mains d'apprenants réels mais nous indiquons dans la thèse comment remédier à cette situation. Le prototype a été écrit en Java; il est possible d'expérimenter avec diverses tâches d'édition et d'évaluation d'un curriculum en cours de création.

En conclusion nous avons proposé une transformation de l'architecture générique d'un STI faisant une très large place à la notion de curriculum et avons démontré sa faisabilité.

Contents

List of Figures

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Chapter 6

Chapter 7

Chapter 8

Acknowledgements

This thesis contains lots of helps and encouragement from many people.

I would like to thank Professor Dr. Marc Kaltenbach. As the supervisor of this thesis, Marc has encouraged me constantly in this work with critiques and suggestions. He proposed many good ideas to improve and extend the project during our discussions. This thesis also contains Marc's many efforts for refining the final version.

I would like to thank professor Claude Frasson who, as a co-supervisor and a deeply committed researcher for the development of ITS and the role of a curriculum in ITS systems, directed this work and provided the major financial contribution.

I would like to thank Professor Jan Gecsei who in collaboration with Marc and Claude has guided the main directions of this thesis. He emphasized the abilities of visualization for managing a curriculum. In addition with periodic advice on the progress of this work, without his financial support it would have been difficult to complete this work.

I 'd like to thank Dr. Roger Nkambou. Roger was a predecessor of this project by proposing a fairly sophisticated and convincing model of a curriculum with suggestion for further work, that provide very useful help in the current work.

I would also like to thank the Ministries of Education and Industry of Quebec Province who supported this work through the project SAFARI.

I'd like to thank researchers in the project SAFARI, including Professor Gilles Gauthier, Professor Bernard Lefebvres, Professor Esma Aimeur and Professor Susanne. Their discussions encouraged me to explore many deep issues in ITS.

Thanks an also due to Dr. Therriy Mengelle, Dr. Tang-Ho Le and Dr. Serge Tadie. We often discussed interesting issues on ITS. I'd like to thank all students in the project SAFARI. In each SAFARI meeting, their ideas reinforced my interest in curriculum management. Thank Mrs. Martinne Gemme for her various helps during my thesis. Thank Mr. Jun Zhang, Mr. Najib Rabia and Mr. Kalid Roune for their help in managing the computer environment. Thank Professor Song, Professor Nie, Professor Francois Lusman, and Professor Gill Brassard in the Department of Computer Science and Operational Research, University

of Montreal, their courses helped me think about the curriculum management from different points of view. Thank Mr. Ming Bai for his suggestions and encouragement.

I have to thank my brothers Guibiao and Wubiao. They took all responsibilities to take care of my aged mother and father when I spent all time to work on my thesis, in particular during my mother's final time.

I'd like to thank my wife, Shilu, and my son, Zhaohan. I spent too many sleepless night and weekends without staying with them.

Last, I must thank my mother and my father very much. In order to support my education, they have been working very hard and economizing on foods and clothing since my primary school time. They contributed all they had for their children. However, my mother passed during my Ph. D. study with an unsatisfied desire--just seeing me once again before her passing.

Chapter 1

Introduction

1.1 Problem Areas Related to the Project

Computer-Based Tutoring Systems have advantages over classic human teaching, such as individualized teaching, support of powerful and flexible multimedia resources, saving human efforts, etc. However, such systems are complex because they not only have to deal with the subject matter to teach but also the psychology and pedagogical characteristics of human teachers and learners. Our research involves several different directions in education, computer science and psychology. The main challenge is to get an adequate synthesis of these very diverse sources of knowledge into a practical and useful system.

From the point of view of computer science, a computer-based tutoring system is a complex well integrated software system for teaching involving knowledge and data management technology, multimedia technology, networking technology and artificial intelligence (AI).

In the educational domain, relevant theories of teaching progress (pedagogic theories) have been developed [Gagné 92, Bloom 69,78]. These theories propose methods and means to implement the knowledge acquisition and teaching processes. The integration of educational theories with actual software implementation is still a relatively new field and hence there is ground for much improvement. One of the difficulties in the teaching-learning process is to understand student behavior. One contribution of psychology (in particular Cognitive Psychology) is the development of theories that explain the learning process. This has resulted in more effective teaching and learning methods [Gagné 85, Tardif 92].

Since the beginning of the 70s', Intelligent Tutoring Systems (ITS) have been developed with the goal of experimenting, via computer software, with the integration of theories linked to previous domains (AI, education science, cognitive psychology). Some systems, such as SHOLAR [Carbonell 70], SOPHIE [Brown 75], BIP [Barr 76], Geometry-Tutor STEAMER [Hollan 84], GUIDON [Clancey 82], WordTutor [Imbeau 90], and GEOCAM [Nkambou 92], have dealt with the complexity of ITS development, notably the problems of student modeling, domain expertise, integration of teaching strategies, curriculum integration, and the problems linked to student interactions. Most of them concentrated on the learning process. Now, the remaining essential concern in the design of tutoring systems is the specification and organizations of teaching activities and materials (i.e. curriculum) as principal input in the teaching system [Finch 86]. Psychologists (humanists, behaviorists, or cognitive scientists) recognize the necessity of creating rich contexts that can support planning and teaching development [Grippin 84, Tardif 92].

The concept of a curriculum has been tackled in some works [Barr76, Halff 88, Lesgold 88, McCalla 90] but with little impact on ITS. Halff thought that the goal of a curriculum in an ITS is to formulate the representation of the material, selecting and ordering actions in particular activities. McCalla gave a more cognitive definition: the curriculum in ITS represents the selection and arrangement of knowledge with a view to realizing teaching goals that are appropriate for the current context and the current student. This definition puts the accent on the fact that a curriculum should be flexible, and should adapt to the needs of students and the needs of an author of teaching material. Other works such as that of Spector, Polson and Muraida [Spector 93] addressed the problem of constructing environments for human teaching included curriculum development and course delivery.

There are many external resources that can be managed and integrated via an ITS. They range from basic resources (such as graphics, video, etc.) for complex presentation of some subject matters and also interactive environments in which students can select objective, take decisions and observe their consequences. Integrating these resources in a unified and coherent structure is a complex task. In addition the resource management task is made more complex by the fact that these resources are constantly upgraded or changed. Many additional organizational components should be considered. Gauthier, Imbeau, and Girard [Gauthier 89, Imbeau 90, Girard 91] proposed an approach that organizes the teaching objectives and the element of a teaching domain in a hierarchy. Some other advance have been tackled in [Halff 93, Merrill 93, Tennyson 93, Gagné 93].

Nkambou proposed a curriculum model CREAM (Curriculum Representation and Acquisition Model) [Nkambou 96, Nkambou et al 98] in Project SAFARI [Gecsei and Frasson 94], in which three kinds of knowledge were identified and organized: capabilities, objectives and resources. CREAM organizes every kind of knowledge as a network with nodes and links, then another network-CKTN (Concepts Knowledge Transition Networks)- is defined to associate the above three spaces to get a transition network. CREAM dealt with more kinds of knowledge than previous models. Organizing multiple spaces is flexible for manipulating knowledge. Selection of specific nodes as learning objectives facilitates the organization of tutoring activities. However, it is difficult to manage very large transition networks effectively.

Visualization of large object space is a relatively new technology, which can enhance representation of object semantics and reduce the chance that users get lost in large object spaces. This is documented by researches such as the Internet, databases and software management [Gershon & Eick 95, Gerson & Brown 96]. Though many researchers attempted to use graphics to enhance some aspects of tutoring systems [Shen et al 88, Resiser et al 88, Feifer et al 88, Frasson et al 88, Takeuchi & Otsuti 92, Murray 96, Kabbaj et al 96, Nkambou et al 96], few researches have used visualization technology for curriculum development [Murray 98].

In summary, at present, the main problem of curriculum development is the complexity of the curriculum itself. Most existing tutoring systems are domain-specific. Too few researches focus on general-purpose development tools. On the one hand, this is because most early ITS derived from expert consulting systems that are domain-specific, in which the researchers attempted to transfer some results in expert systems to tutoring domain. On the other hand the early ITS did not recognize the difficulties created by the presence of many diverse sources of knowledge. Besides, the subject domain to teach and all kinds of relationships within it, the pedagogical, psychological, cognitive and the didactic aspects are all important for a real tutoring system. Extracting the useful knowledge in these domains is difficult and done by an experienced human teacher who is able to combine the related knowledge very flexibly based on his current understanding of his students. When this has to be generalized to creating teaching structures adapted to a variety of students, their task becomes extremely hard to manage and computer support is required.

Some researchers proposed general-purpose development environments [Merrill 93] [Gecsei & Frasson 94] [Nkambou 96] [Nkambou et al 98] [Murray 98]. These systems have shed light on the complexity of integrating various sources of knowledge and pedagogical activities. However, either they were applicable only to fairly restricts of teaching/learning tasks or too complex and hard to use practically.

Since we make the curriculum the focal point of an ITS, we identify the main problems involved in designing a general-purpose curriculum development environment:

- **How to simplify the global network structure of a curriculum in order to make the global curriculum management task easy**

In order to identify teaching outcomes of a tutoring system, a curriculum author has to divide domain knowledge into many kinds of topics. There are various relationships between topics such as prerequisite, aggregation, etc. Such topics and relationships form a topic network. The topic network in most domains is large involving hundreds, even thousands of topic nodes. In order to organize tutoring activities to teach these topic another network that connect topics and tutoring activities should be built. Each tutoring activity has to link a series of didactic resources. As a result, the final network that is able to teach domain knowledge is too big and too complicated to manage easily. For example, it has been found that a topic network for teaching "Excel" contains about 500 topics. The final knowledge transition network contains more than a thousand nodes. It is very difficult to develop, maintain and manage such a large network.

- **How to organize tutoring activities to reflect the incremental way humans learn**

Most tutoring systems form tutoring activities by attaching teaching contents or didactic resources to topic nodes. Although some systems can organize topic resources into several groups, for instance, in Eon [Murray 98], a topic's content can be organized into three levels including introduction, teach and summary, they cannot reflect the incremental way humans learn. Whatever is the introduction, the content or the summary, there are some incremental sub-process such as informing the learner of the objective(s), stimulating recall and presenting course material. These sub-processes reflect the way of humans acquire knowledge. Currently there is a great need of a curriculum tool that provides facilities to organize activities for incremental delivery of domain knowledge.

- **How to visually guide curriculum authors and learners to achieve their goals.**

As mentioned-above any real curriculum is a large network. The structural complexity of a curriculum network can easily weaken the creativity required for developing its individual components. In the development process of a curriculum, the curriculum author has to enter, edit or update the information stored in an internal database. A curriculum development tool should help the curriculum author to control the development progress without frequently opening these inner databases.

In the learning environment, the visual navigation capability offered to learners is relatively more important than that in the authoring tool, because the learner is unfamiliar with the content domain. When a learner faces what environment to a large and complicated network, some information, such as what topics he knows, what are his goals, what is his current progress status, what is the next step, etc.; all of these and more is required by the learning process. The learning environment should visually guide the learner to achieve his or her learning goals throughout the whole learning process.

- **How to create different courses to support students' evolving preferences**

One advantage of computer-based tutoring is individualized tutoring; that is, the system can organize specific tutoring content and activities for the current learner. Though some domain-specific tutoring systems dealt with the issue, it has not yet been tackled in general-purpose curriculum development environments. This concerns the issues of how to formalize learner preferences and how to evaluate the adaptability of a course to a specific learner.

- **How to dynamically manage a student's learning process**

Dynamic management of learning process refers to the mechanism to handle various possible states of a learner during the learning process. As a result of a learning sequence the learner may have grasped nothing, a little, up to everything he or she has to learn. A learning environment should be able to handle the results of various possible learner states. For example, when a learner lacks of some prerequisite concepts, the learning environment should be able to organize specific tutoring activities for remedying the learner's missing knowledge.

1.2 Thesis Objectives

We consider that a curriculum should harmoniously integrate all kinds of knowledge that meet the needs of instruction both when managing the authoring process and managing the learning process. The knowledge should come from not only the domain expertise [Anderson 88], but also the cognitive analysis of domain tasks [Polson 93, Lajoie 89] and pedagogic theories [Bloom 69, 78, Gagné 85b]. A part of our work is to find a general approach that allows constructing a curriculum that can:

- support integrated tutoring activities to simplify the curriculum structure;
- give a visual tool that will help both curriculum authors and learners;
- assist the teaching processes (recommending courses, sequencing tutoring activities, etc.);
- support individualized teaching for a particular student;
- be used in as many domains as possible.

This approach also has the advantage of proposing a learning structure that can be updated and enriched with its usage. Teachers and learners can communicate through this evolving structure.

Our concrete aims are

1. to provide the means of knowledge representation and organization in a curriculum for facilitating domain instruction by pedagogic experts or domain experts as well as knowledge acquisition by students;
2. to provide ways of aggregating knowledge about the subject domain and about learner capabilities so that the curriculum structure is practically manageable;
3. to use the visual means to satisfy the design and use of a curriculum
4. to propose approach for the generation of multiple alternative paths to reach a group of learning goals;
5. to evaluate courses to enhance individualized teaching;
6. to dynamically manage the process of remedying learners' misconceptions.

1.3 Contributions

The contributions of this thesis include:

(1) making use of visual interactive environment to satisfy the needs of curriculum authors and learners;

(2) augmenting the curriculum models (like CKTN) through an integrated transition network structure that is simpler to manage.

- **Generalizing the use of a visual interactive environment to meet the needs of both curriculum based course authoring and actual course delivery to students**

This thesis provides advanced graph editing abilities for visually building, organizing and manipulating teaching activity networks. All these functionalities are implemented based on a direct manipulation user interface. A curriculum author chooses a node template and positions a new node; the system can automatically create and display the new node. The author, then, can edit the visual properties (such as the number of capability levels) and inner (that is not visible) attributes of the new node. Other functionalities such as creating links, deleting nodes, deleting links and laying out a network can also be carried out by simple direct manipulation.

Color and shapes for the visual curriculum are used extensively to convey all kind of complex information for the authoring tasks and learning process. Visual and dynamic navigation ability is extensively exploitated in the thesis. When a curriculum author enters or edits the inner attributes of nodes and links, the system can memorize current development progress. Colors of visual cells indicate whether or not a cell contains all the necessary information. It is not necessary for the author to frequently open the inner database to view the development progress. The mechanism of visual navigation can also be used for dynamic support in curriculum development process, for example for seeing all activities a didactic resource is used for, or for viewing the distribution of one kind of didactic resource in the whole activity network. The navigation ability for learners can help a learner view the whole knowledge network, set his or her learning goals, display current states, display current recommended tutoring activities. For instance a student can very easily see what he/she has learned so far, which resources to be learned, and what are his/her next learning goal(s).

The simplification of the global curriculum network makes it possible to create multiple alternative courses for a set of learning goals. Alternatively the learner can select his/her current preferred courses. The system can recommend one course among the created multiple alternatives courses, which is the most adapted course to the learner's current profile.

This thesis proposes a mechanism for dynamically managing the learning process. It can combine various delivery modules of tutoring activities or tutors with different diagnosis abilities. If a learner fails to understand a capability that is not contained in the current course, the system can go back to the overall curriculum to create a sub-network for teaching the missing capability. If the learner doesn't acquire some capability levels that are prerequisites of a capability, the system can create a new path for the missing capability levels. If a delivery module can evaluate a learner's state as only "mastery" or "not mastery", the system can choose alternative resources to use for teaching.

- **augmenting the existing curriculum models (like CKTN) through an integrated transition network structure that is simpler to manage and yet integrates the well known instructional theories about instructional events and teaching objective levels from Gagné and Bloom**

This thesis defines a visual curriculum that organizes teaching outcomes, tutoring activities and didactic resources as a visual network with nodes and links. We absorb the overall idea for organizing transition networks from CREAM, i..e., from prerequisite capabilities to transition actions and from transition actions to output capabilities. There are two kinds of nodes in the definition: "capability nodes" representing teaching outcomes and "transition nodes" representing tutoring activities. Each node contains a group of visually manipulatable cells. A capability node can contain several incremental capability levels. A transition node consists of structured instructional events, which are organized by combining Gagne's instructional event theory and Bloom's objective level theory [Gagné, 85, 92, Bloom 69, 78]. The relationship between a capability and a transition node is either a prerequisite relationship or an output relationship. A visual curriculum includes the means to build, organize and manage tutoring activity networks. A visual curriculum has to supply facilities to help authors and learners in performing their tasks.

In order to simplify the global transition network structure (like CKTN in CREAM) of a curriculum, we propose the concepts of multiple-level capabilities and aggregated capabilities. A multiple-level capability is the capability that can be divided into several incremental levels. For example, the capability "applying Newton's law: $F=m*a$ (Force equals to mass times acceleration)" may be divided into four levels: (1) stating the law, (2) using the dimensions of variables in the law correctly, (3) applying the law to simple instances, and (4) applying the law to general instances. With the help of transition nodes, a curriculum author can organize tutoring activities to support the teaching of each capability level. An aggregated capability includes a group of simple capabilities, each of which has just one level. These simple capabilities can have the same or different types (such as a fact, a concept or a rule). For example, "identifying the definition of WWW" and "stating features of WWW" are two simple capabilities. They can be combined into an aggregated capability called "understanding WWW" with two levels: "identifying the definition of WWW" and "stating features of WWW". The mechanism for aggregating capabilities

decreases the number of nodes and links in a transition network to a great extent, making it more practical to use by both authors and learners.

These ideas are verified in an authoring and learning prototype developed in the Java programming language.

1.4Thesis Organization

This thesis is divided into nine chapters. Chapter 2 is a review of computer-based tutoring system and curricula

Chapter 3 reviews the graphical and visual aspects of existing tutoring systems and curriculum development tools.

Chapter 4 defines a visual curriculum structure called *VITCAM* (Visual Interactive Transition, Course and Activity Manager). *VITCAM* consists of three sub-models called *VTRANS* (Visual TRANSition model), *VCOURSE* (Visual COURSE model) and *VACT* (Visual ACTivity model). The *VTRANS* model provides means for organizing transition networks, whose output is a central data structure called *Tnet* (capability Transition Network). The *VCOURSE* model supplies ways to create multiple alternative courses for a group of selected goals, with a structure called *Cnet* containing multiple alternative paths as its output. The *VACT* model orders tutoring activities in a selected course for the current context and current student, which outputs a sequence of tutoring activities called *Anet*.

Chapter 5 focuses on the *VTRANS* model. In this chapter, we propose the approaches for organizing and managing capability transition networks. We define capability nodes, transition nodes and links to connect these two kinds of nodes for building a capability transition network, known as *Tnet*. Meanwhile, the mechanisms for visually creating and managing capability transition networks are described.

Chapter 6 describes the mechanisms for creating multiple alternative courses (i.e. the *VCOURSE* model).

Chapter 7 characterizes the dynamic management of a learning process (i.e. the *VACT* model) based on the proposed capability transition networks. This includes evaluating courses, identifying necessary tutoring actions for a particular student's goals, sequencing tutoring activities (i.e. *Anet*), and dynamically remedying students' misconceptions.

In Chapter 8, we present the system prototype and its application to teaching HTML.

Chapter 9, the last chapter, summarizes the thesis contributions, the limitations of the proposed models, the comparison between *VITCAM* and CREAM, and provides suggestions for further research and development.

Chapter 2

Computer-Based Tutoring Systems and Curricula

In this chapter we first review computer-based tutoring systems and authoring systems. We, then, focus on checking the existing curriculum structures as the main frames of tutoring systems.

2.1 Computer-Based Teaching and Learning

Learning is a process by which an individual acquires new knowledge. It allows a learner to develop skills, attitudes and values that can add them to his or her cognitive structure [Legendre 93]. Some researches have attempted to model the learning process, in particular those of Lansky [Lansky 75] and of Gagné [Gagné 76, 92]. The last author seems to be more explicit. According to Gagné, learning is the result of interaction between students and their environment. He had classified these interactions into an eight phrases model (Figure 2.1), in which each phrase corresponds to an internal process being able to produce a type of learning [Robidas 89]. An (internal) learning process can be brought about by the external events coming from the educational environment. When these external events are planned with the goal of supporting a sequence of learning steps, one talks of "teaching". The role of teaching (or a teacher) is to put certain factors in place to stimulate the learning process [Gagné 92]. These factors can be organized to affect the motivation of students, their attention, or any other processes that compose a

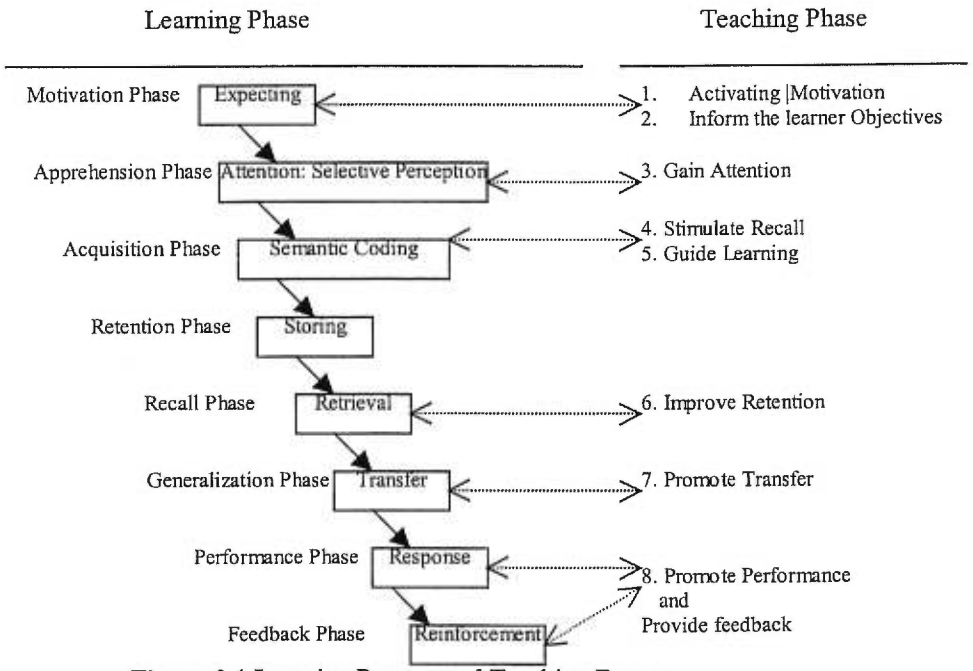


Figure 2.1 Learning Process and Teaching Events

learning action (figure 2.1). For example, a way to promote positive expectations (i.e. to establish motivation) is the communication to a student of the importance of a learning goal (that we call "teaching objective"). In short, teaching is one way of producing learning in a student.

2.1.1 Computer-Assisted Instruction

Computer-Assisted Instruction (CAI) appeared in 1950s. A CAI system, largely abandoned now, was a pre-programmed system that placed the student in a simple interactive environment. He or she answered some questions (of true / false or multiple choices) on screens concerning texts that the system presents to the student. Certain systems contained evaluation tests. The evaluation of student responses permitted the system to decide on the next frame to present. The evaluation was limited only to decide whether the response was correct or wrong. The big weakness of this approach is that there is no deep analysis of domain concepts and students responses. This type of system evolved with the use of pattern-matching techniques to analyze the responses, and later (in 70s') toward adaptive systems (generating didactic materials, in particular for arithmetic problems and vocabulary acquisition) [Uhr 69]. In such a system, the student model is usually a summary of his past behavior parameters. This evolution represents a big step toward individualized teaching, but it remains insufficiency because of the difficulty of analyzing and comprehending students' intentions as well as the inability of reasoning on the teaching domain, [Burton 82].

The curriculum in these systems is not taken into account. They use just a simple linear organization (sequences of frame screens) in a preordained order based on students responses. For example, in the system of Kimball (Integration) [Kimball 82], the advice is given to the student according to the quantity of entered data for his or her solution, rather than the validity of his or her response. Besides, there is no access to the content of a frame, so the thought domain can not be explained.

In 1970s, Carbonell [Carbonell 70] proposed a system (SCHOLAR) where for the first time artificial Intelligence techniques were used. In SCHOLAR, knowledge to teaching process was separated from domain expertise knowledge. (often called domain knowledge). Thus, the system had a semantic network that represented the fact on the geography of Southern America; and another part, a program for managing the interaction with a student. This proposal started the evolution of CAI toward the *intelligent Computer-Assisted Instruction*.

2.1.2 Intelligent Computer-Assisted Instruction

Unlike classic CAI systems centralizing on the interaction process between students and computers, intelligent computer-Assisted Instruction (ICAI) systems focus on students' knowledge and explicit representation of knowledge for the realization of tasks.

In ICAI systems, intelligence carries out the tasks that are not explicitly foreseen by the program. ICAI systems are often called Intelligent Tutoring Systems. Burns and Capps (1988) proposed that an ITS is a CAI system that is capable of following three tests:

- it should know sufficient the teaching materials in order to make inferences permitting domain problem solving;
- it should be able to deduce the knowledge level of student on the domain;
- tutoring strategies should reduce the gap between the student's performance and the expert's performance.

Based on conditions of Burns and Capps, an ITS should include three principal models (figure 2.2): an expert model, a student model and a pedagogic (tutor) model. Besides, teaching should be developed in an environment integrating a communication interface between students and the system. In the same order of ideas, Nicaud and Vivet [Nicaud 88] proposed four components as the basic components of an ITS.

Teaching requires an interaction between the agents implied, known as a tutor, a student, a domain expert and a learning environment. According to Woolf [Woolf 92b], a teacher model (Tutor) should include the methods to remedy errors, the methods for the selection of examples, of analogy and of strategies for responding to student's erroneous behavior.

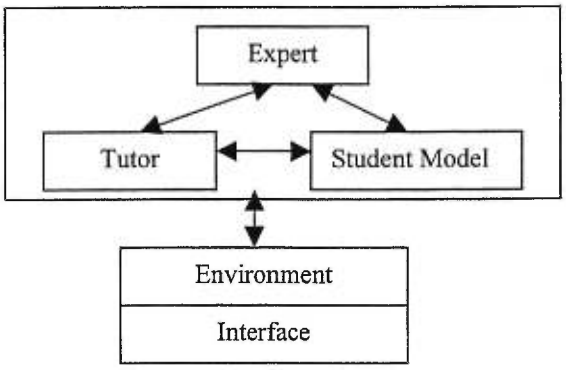


Figure. 2.2 – Basic Architecture of ITS [Burns 88]

The student model includes students' knowledge. The expert model embodies the expertise on the teaching domain; it should be able to solve the problems that are posed to the student with reasoning on the knowledge base making up the domain expertise, i.e. the teaching materials (subjects) [Nicaud 88]. The communication model, containing an environment and a communication interface, is built on the set of elements that interact with the student (simulation, menu, command language…) [Frasson 91]. A system is considered as intelligent only if the previous four modules are well implemented [Woolf 88, Kearsley 87]. ITS that ascribe the same importance to all these modules are not numerous. When some ITS put more emphasis on the student model (Sherlock Lajoie 89], Geometry-Tutor [Anderson 93, etc.), others

concentrate more on the teaching strategies, or on the interface model (Cardiac Tutor [Murray 93a]). We have summarized in table 2.1, and give the importance degree of each module in these ITS.

TAB. 2.1-Summary of the evaluation of importance of each module in ITS.

Systems	Domain	Strategy	Student Model	Interface
Cardiac Tutor [Murray 93a]	*	-	-	**
Static Tutor [Murray 93a]	**	*	*	-
Sherlock [Lajoie 89]	*	-	**	-
AMALIA [Vivet 87]	**	*	-	-
Lisp Tutor [Reiser 85]	**	*	**	-
Steamer [Holtan 84]	**	-	-	**
Geometry Tutor [Anderson 83]	**	-	**	**
SOPHIE [Brown 75]	**	**	-	-

Each type of ITS is characterized by the way it is used for teaching or for learning. We identify six types of ITS: the Socratic ITS, the task-based ITS, the simulation and demonstration systems, the exploration environment of learning, the critic systems, and the social systems. All those types of tutoring strategies around a cognitive model has been evoked by Frasson [Frasson 94a] who thought the ITS as module environment capable of supporting all tutoring strategies.

Socratic ITS allows, with the help of a dialog with student (Socratic dialog, question / response ...), the teaching of facts and necessary skills for inferring other knowledge [Allessi 85].

Task-Based Systems teach necessary skills and procedures for accomplishing a task through exercises, examples and problems.

Simulation and Demonstration Systems represent a concrete situation in which learners can experiment or experience with problem solving [Allessi 85]. They do not teach really in the sense of Socratic or task-based ITS.

Exploration-Based Systems help a student learn, by exploring the learning object with some guidance.

Critic Systems are used for criticizing the students during problem solving. The critic consists of aiding, preventing and reducing judgement errors with the goal to bring users toward a correct solution.

Social Systems are the ITS that involve, in addition to the classic agents in ITS, other agents such as several other students, one or more other teachers [Chan 90] [[Aimeur 95a, 95b]

Integrated Environments can be proven to be very useful for enhancing the power of ITS; the module environment proposed by Frasson et al [Frasson 94a] goes in this sense. An approach building integrated ITS is actually implemented in the project SATARI [Frasson 94b, Gecsei & Frasson 94].

2.2 Authoring Systems for Instructional Design

We have seen, in the previous paragraphs, that restricted domains do not pose the problems linked to the design and organization of teaching. The task of teaching design, also known as instructional design [Brien 92], is a complex process requiring dedicated experts. A theory of instructional design has been developed [Gagné 93, Merrill 91, Tennyson 93, Reigeluth 93]. Due to the difficult, repeated and costly process, it is necessary to provide effective tools. The required expertise is difficult to acquire, and the integration of such an expertise in an authoring environment is absolutely necessary. The integration leads to the availability of powerful tools well integrated in the authoring environment. The development of some systems such as ISD Expert [Merrill 87, 91, 93], IDE [Pirolli 90], ISD Expert [Tennyson 93], GAIDA [Gagné 93], fall in this category. We describe some examples of authoring systems dedicated to the instructional design in the following paragraphs.

2.2.1 GAIDA

Gagné proposed an approach to the automation of instructional design [Gagné 93]. According to him, successful teaching requires representing knowledge in terms of capabilities that one wants the student to acquire as well as the teaching strategies to use for favoring the knowledge acquisition. In particular, teaching strategies are identified for each capability [Gagné 85b, 93]. His proposed system called GAIDA (Guided Approach for an Instructional Design Advisor) does not use artificial Intelligence techniques. It uses neither an expert system nor reasoning nor a user model; it takes on simply the designer in a dialog (based on the process of development proposed by Gagné) that accustoms him to a design format appropriate for the development of effective teaching materials. This approach is not generative to some extent that all teaching knowledge is entered explicitly by expert.

2.2.2 Elaborated Frame Networks

Merrill considered the GAIDA as the first generation and its attributes are limited as follows:

- lack the integration of design activities;
- limited means of representation and acquisition of knowledge;
- weak specification for course organization;
- absence of consideration of dynamic features of teaching system development.

Thus, Merrill proposed an approach for the second generation authoring systems. According to him an authoring system should include the following functionalities:

- Being capable of analyzing, representing and leading teaching;
- Being able to produce pedagogic prescription for the selection of sequence of teaching transactions [Merrill 91]. The transactions of teaching are the tactics of teaching, the interaction models, conceived for returning the student capable of acquiring certain types of knowledge or skills. In order to be able to produce the pedagogic prescriptions for the selection and the sequences of teaching transactions, the second generation of the systems should contain the rules of prescription of teaching strategies;
- Being the opening systems, capable of incorporating new knowledge on teaching and learning and applying them to design process ;
- Integrating the teaching development phases (contrary to the first generation systems where the work in each phase is relevant independent of other phases);

Merrill's approach is generative in the sense where the teaching transactions can be generated from the rule-based expert systems [Merrill 87]. In his transaction theory, Merrill [Merrill 91, 93] had identified three types of knowledge to represent: the knowledge of entities, the knowledge of activities (the knowledge of different stages of the execution of a task), and the process knowledge (for example, the interpretation of a state of a machine).

A structure EFN (Elaborated Frame Network) was proposed for representing the three types of knowledge. The teaching transactions are built for exploiting the structure. Four types of transactions have been identified for acquiring the different types of knowledge: the transactions of components, the transactions of abstraction, the transactions of association and the transactions of enterprise.

Transaction of Components

It allows the teaching of entire or partial cognitive structure. Three kinds of transactions had been identified:

- Identifier (it has students indicate names, functions, properties and places related to all parts of an entity (knowledge of "What"));
- Executor (it has students acquire the stages of an activity);
- Interpreter (for the process).

Transaction of Abstraction

It is used to teach abstract hierarchies. Five classes were identified in this category: judge, classify, generalize, decide, and transfer.

Transaction of Association

It teaches two or more cognitive structures connected by an association relation. This type of transaction allows a student to acquire a set of components in the context of another group of competence of learning by analogy, of acquiring new competence by referring a competence having achieved, of invention new knowledge on entities or properties, or discovering new process. At least five classes had been identified in this type: spread, make analogy, substitute, conceive, and discover.

Transaction of Enterprise

It teaches all cognitive structure (and the relation among them) for a particular task.

2.2.3 Some Other Authoring Systems

Tennyson proposed an approach [Tennyson 93] somewhat more ambiguous than that of Merrill. He proposed a system (ISDExpert) that provides an adaptive interface and some ITS techniques for instructional designers, but lacking experimentation facilities to guide the conception of quality teaching. ISD Expert is used for the purpose of modeling experts' knowledge on the process of instructional design to aid the building of teaching. It should be able to not only post up the list of applied rules (like that in classical expert systems), but also support and explain its recommendation and its prescriptions in the language of instructional design. ISD Expert will thus be a sort of ITS for instructional building, integrating the aspects of coach and advice (figure 2.3) and accompanying an intelligent interface for the communication between the author users and the system.

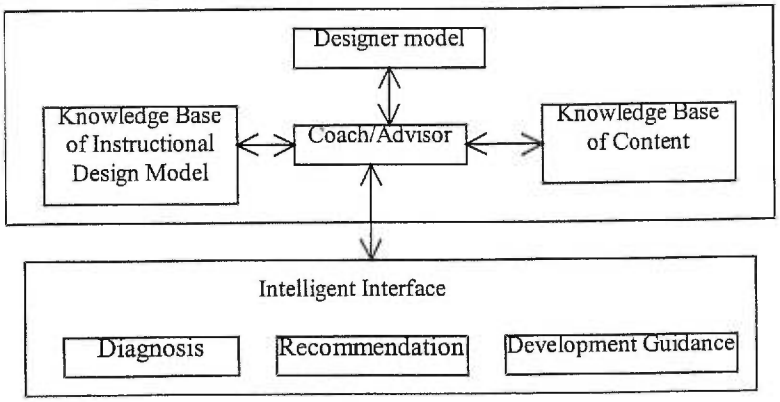


Figure 2.3 Architecture of ISDExpert (Tennyson 93])

We cite only some other examples.

- IDExpert [Merrill 87, 91, 93]: a prototype of rule-based expert systems for development of instruction, which generates recommendation based on the structure of content, on the organization of courses and on the teaching transactions.
- IDE [Pirolli 91] is a hypermedia environment for building teaching. It is a system for designers, which allows them to enter, edit and handle their analysis and specification. It allows making a knowledge structure, a cognitive analysis of tasks and a representation model of information to different levels of abstract.
- ISDExpert [Tennyson 93] is a future system of ITS for building instruction. This ITS proposes the integration of coach and advisor. It integrates an intelligent interface.
- DOMINIE [Elsom-Cook 88] is an authoring environment supporting the development of teaching content and curriculum by direct manipulation in a graphic environment. The graphical environment is however passive; no behavior of graphic objects is defined.
- RAPIDS [Towne 90] is an authoring environment able to support the development of teaching content and curriculum by direct manipulation. One prepares panoplies of a simulated equipment pieces and assembly rules that an author will manipulate to build instruction.
- DEGREE (Environment of demonstration generation) [Nkambou 95d, 95c] is a system like RAPIDS in the sense of using direct manipulation. Developed in the frame of the project SAFARI, it allows to build tasks by direct manipulation and construct scenarios that make part of demonstration to present to the student.
- SGD (Atelier De Genie Didactical) [Paquette 96], is a system for building training estimation for teachers.
- CREAM Tools [Nkabmou, Frasson and Gauthier 98] (to discuss in detail later).
- Others see IJAIED special issue on ITS authoring 1998 & 1999.

2.3 Curriculum

We introduce in this section a study on the notion of curriculum in teaching. This study is made also from the point of view of Education, of Cognitive Psychology and of Computer Science. This will clarify the concept of a curriculum and underline its importance in a teaching system.

In the educational domain, the development of a curriculum passes by three stages: the identification of teaching content, the definition of teaching objectives and the determination of necessary means for deriving the instruction steps needed to meet the defined objectives.

Several studies on these curriculum development stages have been undertaken [Finch 86]. The problem is to know by which stage it starts for obtaining an optimal curriculum. Some people propose to start by building the objectives before identifying the representation of content. The consequence of this approach

is that one can find objectives with weak content. However, when the content deriving materials precedes the identification of objectives, the result is tangible and thus is more significant.

Determination and classification of teaching content consists of identifying and defining the knowledge that is used in the teaching environment. The determination of content is based on two phases: the utilization of strategies for the identification of relevant knowledge, and the classification and representation of retrieved knowledge.

2.3.1 Strategies Determining Teaching Content

These strategies go from more subjective to more objective. We have tried to present them in the order: philosophic foundation, introspection, functional approach, and task analysis. The utilization of one or another strategy in a process of curriculum development depends however on the particular knowledge domain.

The strategy, "Philosophic Foundation", consists of using a specific philosophy or a set of philosophies as the foundation for the choice of content. It is based on verbal assertions from educators. For example, consider the following assertion: learning driving theory is the preparation of learning practical driving. Based on this assertion, all content of curriculum of driving theory does not touch the practical driving environment. Once the curriculum developers of agree with this assertion, the relevant content to this assertion has been identified. This strategy may be more subjective, but it is used widely in the development of curriculum in academic domains.

An introspection strategy is based on examining the thinking and beliefs of people while implied in the development of a curriculum relative to a knowledge domain. It implies that one or more teachers or domain experts pose the following foundational question:" whether this or that item seems pertinent to include in the curriculum?" The different contributions will be analyzed later in order to select the most pertinent content. Thus, the content follows from a discussion between a group of professors. The advantage of this approach is that one can get a curriculum with rich content. More specific approaches within this identification process can be:

- Seeing the description of domain knowledge as it is progressively linked into a curriculum;
- Identifying the general groups of implied abilities;
- Identifying the specific performance(or behavior) for each group of general abilities;
- Structuring the performance in a significant learning sequence; and
- Specifying the levels of competence for each task performance.

One the two previous strategies are based on the subjective judgement, another strategy is more objective: The latter approach is a functional approach. It is based on the definition of functions learning environment in terms of the operations executed in the environment. The content is thus identified in terms of functions.

The Task Analysis strategy involves the identification and verification of tasks executed in an environment, with a view of including them as the content of a curriculum. Several methodologies exist for task analysis [Bovair 90, Polson 93, and John 94].

The strategies for content determination are very diverse. The question is that given certain resources, which strategy could be the most appropriate for determining the content of a specific curriculum? In general, the utilization of a unique strategy cannot always cover all information implied in a domain; several strategies should be used for identifying significant content. This can render the final curriculum capable of covering the needs of all students.

2.3.2 Classification of Content

Classification of knowledge is extremely important in the cognitive design of teaching and learning since it orders the strongly different teaching strategies and the representation in human memory. We introduce the notion of capability, in the sense of Robert Gagné, for describing the outputs of learning carrying on content. By capability, what one acquires is the ability developed, allowing a person to succeed in the exercise of an intellectual, professional, or physic activity. It deals with information structure (knowledge of cognitive unit) stored in the long-term memory of the student and that give possibly a performance. It represents what is learnt (product of learning) [Grippin 84]. Several theories have been developed in the education domain, in the psychology and in the artificial intelligence for characterizing the product of learning. We introduce in the following paragraphs some of the theories.

Gagné's Capability Theory

In his theory of outcomes of learning (knowledge), Gagné identified five big categories of capabilities [Gagné 85] that can produce most of human activities: verbal information, intellectual skills, motor skills, attitudes and cognitive strategies.

- Verbal information is the kind of knowledge we are able to *state*. The possessors of this kind of capabilities can explain, describe, or name objects or facts.
- Intellectual skills enable individuals to interact with their environment in terms of symbols or conceptualizations. For example, solving an equation, programming, etc. Gagné divided intellectual skills into several sub-categories ordered by the complexity of mental operations: discrimination, concrete concepts, defined concepts, rules and high-order rules (problem-solving

- Cognitive strategies are the kind of capabilities that allow acquisition of other types of capabilities and their management in the reasoning and the resolution of problems (learning strategies, problem-solving strategies
- Attitudes are the internal states that influence the choice and the actions of a person that possess them. For example, a favorable attitude to classic music influence the selection of information that one wants to listen
- Motor skills, which remain too little consideration in the goal of learning, are a class of capabilities (output of learning) linked to the activities concerning the movement: driving a car, typing, sport activities. A curriculum can sometimes imply the acquisition of motor skills.

One of advantages of this classification is that, the particular conditions for favoring the acquisition of all types of knowledge in this taxonomy have been defined by Gangé [Gagné 85b,93].

Theory of complex learning

According to this theory [Norman 75], the complexity of learning is defined by the existing of experts in the subject. Since some experts of cooking exist, cooking is a subject of complex learning. Norman has identified five categories of capabilities:

- motor skills,
- intellectual subjects,
- subjects based on the procedures,
- subjects based on facts, and
- mixed subjects.

In this theory, capabilities (products of learning) are not linked to subjects, rather than to knowledge modules. The knowledge modules are some units of memory in the interconnected or hierarchic structure. They are acquired through the stages: (1) increase (2) re-organization and (3) adjustment.

The increase stage consists of acquiring new verbal information. The critic formulated by connecting to learning strategies of this stage is vague (for example, study and learn). The re-organization is a process of reorganization of increase, in a more significant structure. The result is the new sense given to previously learnt information. The teaching strategies used in this phase are analogy, metaphor, or examples. The adjustment phase consists of refining previously learnt knowledge. During the process, the structures get more effective. There is neither new knowledge nor new structure.

The classification of Norman does not include the teaching of learning strategies. According to him, the learning strategies can be learned through the three phases. In this classification, the three phases, in fact, are the processes of knowledge acquisition.

Extended capability theory with meta-cognitive or social competence

Grippin and Peter [Grippin 84] added a new dimension in the taxonomy of Gagné: the extension of cognitive strategies with meta-cognitive strategy. The utilization of this cognitive strategy needs a control of a student according to a situation. The capability being able to control the strategies is called meta-cognitive strategy [Tardif 92]. In fact, meta-cognitive is the strategy that manages the cognitive strategies.

The classification proposed by Chamberland and others [Chamberland 95] is a recent extension of Gagné`s classification. In the classification, six categories of capabilities are described: declarative knowledge, intellectual knowledge, motor capabilities, strategy knowledge, attitudes and social competence. Except the social competence, the other categories correspond to those of the taxonomy from Gagné.

Cognitive, Emotional and Psychomotor Theory

The taxonomy of Bloom brings up three domains of products of learning, each of which corresponds to a capability: the cognitive capability, the emotional domain and the psychomotor domain. In the cognitive domain, the capabilities go from the recall of facts to the interpretation of their values. The emotional domain handle capabilities related to attitudes; while the psychomotor domain handles capabilities relative to motor skills. This theory is very useful for the building of evaluation test. It can also cover a vast category of learning. The analysis of learning and prescriptions for learning based on this model are however too weak because of lacking the precision of strategies of acquisition.

Component Display Theory

The theory of Merrill proposed two categories of learning results: the capabilities following the type of content and the capabilities following the levels of tasks [Merrill 87, 91, 93]. From the point of view of content, we have facts, concepts, procedures and principles. From the point of view of tasks, we find the following expression: recall an instance word for word, recall an instance in other terms, recall a generalization word for word, recall a generalization in other terms, use a generalization and search a generalization.

The two dimensions forms a matrix with 24 cells (capability matrix) each of which represents a capability (a product of learning) (figure 2.4)

Some examples are:

- Example 1: division of fraction (utilization of procedure). In this example, the division of fraction represents the content and the utilization represents the task.

- Example 2: the definition of a chair (recall a concept word for word).

	Recall an instance Word for Word	Recall an instance in other word	Recall a generalization word for word	Recall a generalization in other word	Use a generalization	Search a generalization
fact						
concept	Recall a concept word for word					
procedure					Use a procedure	
principle						

Figure. 2.4 Capability matrix

The theory results in a matrix called the matrix of teaching strategies. The combination of the capability matrix and the matrix of teaching strategies is able to produce the prescription for teaching. Some rules are developed in the theory for the prescription of teaching. The result gives the prescriptions generated by the sort system: "Say" an "example" for learning a "recall an instance word for word". This explains the necessity of arranging the rules of prescription that gives the result.

This theory has been used for conceiving teaching in the commercial domain. Its complexity limits its utilization only for the users initially implied in its development.

Outcomes of learning Based on cognitive psychology: know versus know-do

Three categories of knowledge have been identified in the domain:

- Declaratives: This is the theoretical knowledge often called *know*. One talks about also the knowledge of facts, of rules, of laws and of principles. This type of knowledge does not contain action. However, they can be translated in procedures or in conditions for actions.
- Procedures: This type of knowledge describes *how* to do something. It deals with the stages to realize an action. One talks about still the *know-do* or dynamic knowledge. The acquisition of this type of knowledge demands that the student should be placed in a context of action. They develop exclusively in the action. For example, solving a series of problem of multiplication. The objectives linked to the

development of procedural knowledge demands that the student be placed continuously in a context of relation of tasks.

- Conditional: A conditional knowledge describes the condition of use of procedures. It deals with *when* and *why* of the action. They are generally represented by a series of constraints following an action, for instance, distinguishing a square from a rectangle.

In summary, seeing all the theories on the classification of knowledge, it seems that the classification of Gagné is more elaborated and include most of other theories (see Table 2.2)

TAB 2.2—Comparison of outcomes of learning

Gagné	Merrill	Anderson	Winograd	Bloom
Rules	Procedures	Procedures	Conditional	Cognitive
Concepts	Concepts		Procedures	
Principles, laws	Principles			
Proposition		Declaration	Declaration	
Cognitive Strategies	Procedures	Procedures	Procedures	
Attitudes				Affective
Motor Skills				Psychomotor

In addition, Gagné has defined for each category of capability of his taxonomy, the learning strategies and the conditions in which the acquisition may be easy [Gagné 85b, 93]. We use the capability theory of Gagné as the outcomes of teaching in our organization model of knowledge.

2.3.3 Knowledge Representation and Organization in ITS

According to Newell, Hebert and Simon [Luger 93], an intelligent activity, as much in human as in a machine, is realized by using:

- the symbolic representation for representing the significant aspects of a problem domain,
- the operation on the representation that allows generating potential solutions of posed problems, and
- the search algorithms that allow to select one solution among different solutions.

This hypothesis, still called the hypothesis of symbolic representation, allows to distinguish two types of knowledge used in the symbolic systems: declarative knowledge (facts, schemes, etc.) and procedural knowledge. The last one can infer declarative knowledge. However, intelligence is handled as the integration of a set of specific facts and a set of procedures for manipulating facts. This is true in a certain measure, when it deals with the knowledge representation in an ITS. Two big categories of knowledge are

considered in an ITS: the knowledge linked to a domain to teach (frequently called domain knowledge) and the knowledge on teaching means (strategy knowledge).

2.3.3.1 Representation of domain knowledge

Representation based on rules

This approach consists of modeling the expertise of a domain with the help of a rule base; which can represent the domain knowledge and make reasoning on the domain knowledge [Anderson 88]. The system is often modeled with the help of production rules representing domain knowledge or strategy knowledge. The domain or strategy knowledge is compiled in a rule packet. This approach has been used in several ITS like Geometry Tutor [Anderson 83], BUGGY [Burton 82], GUIDON [Clancy 82a]. The advantage of this approach is that it is easy to exploit and allows a good representation of procedural knowledge. The disadvantage is that it is difficult to represent structured knowledge such as concepts and relationships between them.

Representation based on networks

It uses network to represent domain knowledge. This representation takes into account the cognitive model of domain. It allows to representation most types of knowledge, but it is more appropriate for the representation of declarative knowledge. The advantage of this representation is intuitive for domain experts; it is easier for an expert to express his knowledge with this form than with the rule form. Several varieties of this type of representation have been proposed in AI: semantic networks [Quillian 67], conceptual graph [Sowa 91, Kabbaj 96] and the frame networks [Minsky 81].

Semantic networks in SCHOLAR were used for representing domain knowledge (facts on the geography of South America). Another example using network representation approach is the subject networks proposed by Murray and Woolf [Murray 93a]. This is a kind of semantic network that is used to represent domain knowledge and different relations among them. Knowledge in a semantic network may be facts, concepts, procedures, principles, etc., and is represented by nodes of network. Murray and Woolf identified different types of links between knowledge: familiar, deep familiar, part of, error of critic concept, simple prerequisite, typical prerequisite (figure 2.5). The system KAFITS [Woolf 89, Murray 91] provided a browser that is used for building this representation.

Representation based on Layers

Schreiber and his colleagues [Schreiber 93] proposed an approach of knowledge representation on four layers: domain, inference, task, and strategy. The theory of domain contains concepts, relations between

concepts, expressions and relations between expressions. According to the authors, the theory models the static knowledge and declarative domain. There are two types of relations between concepts: aggregation relations (of sub-components) and relations of specification. An expression represents a property of a component (a concept and its states) or a test (and result of the test). Two types of relations exist between expressions: the relation of causality and the relation of indication. If we are in the context of systems, the layers represent the static and the functional aspect, introduced by the expressions and the relations between expressions. The mixing of static and functional levels can cause some problems at the time of exploitation of representation.

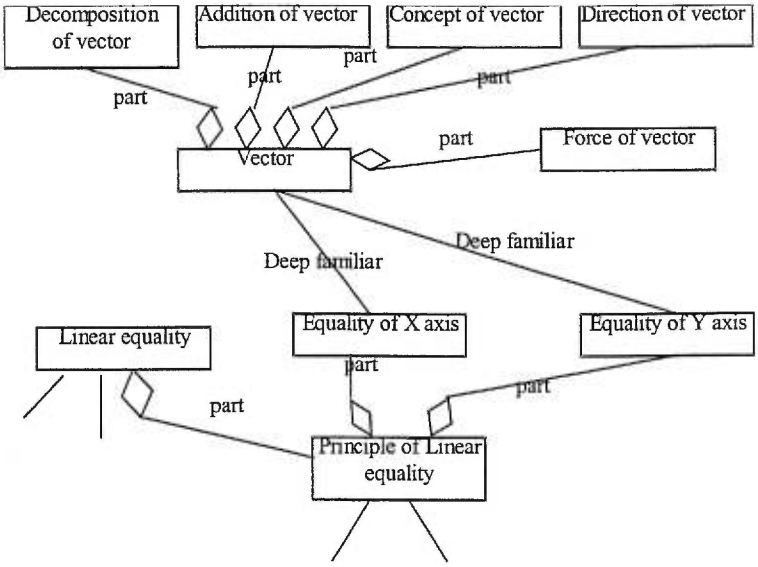


Figure 2.5 A portion of subject network in STANTIC-Tutor

Most previous representation approaches make abstraction of the aspect of knowledge organization for final teaching, yet it is necessary if one wants to teach [Merrill 88, Lesgold 88, and Webster 94].

2.3.3.2 Representation of Strategic Knowledge

The strategy knowledge from the point of view of system concerns teaching strategies (strategies for activity organization), planning strategies (choice of content to consider when to give teaching), and tutoring strategies (selection of tactics, intervention strategies and/or communication). The strategy knowledge in the point of view of the student deals with problem-solving strategies [Gagné 92, Schreiber 93] and learning strategies [Gagné 92]. From the point of view of curriculum management, the involved strategy knowledge mainly is the teaching and tutoring aspects. The learning strategies can be considered in activity deliver module in tutoring systems.

In most ITS (SCHOLAR, WEST, GUIDON, WUSOR), strategy knowledge is represented by production rules. Figure 2.6 shows an example of strategy representation by rules in GUIDON [Clancey 82b]. This example illustrates how the communication model of GUIDON guide the session to help the tutor's rules (Noted T-RULESxxx, where xxx represent the number of the rule). The language of plan and the meta rules [Vivet 88] are other examples of strategy representation to the help of rules. The figure 2.7 (a) and (b) show respectively an example of language of plan and an example of meta-rule

```
T-RULE26.03
IF: The recent context of the dialogue mentioned either a "deep subgoal" or a
factor relevant to the current goal
THEN: Define the focus rule to be the d-rule that mentions this focus topic.
```

Figure 2.6. An example of rule representing a strategy in GUIDON (Clancey 82 in BROWN 82)

```
For verify that a theme is known
      Try successively
            -ask questions on theme formula
            -propose an exercise
            -in case of failure of exercise
                  *update student model
                  *assign new tasks in the agenda on the theme
                  (a)
If the student is a child
      Then keep just the pedagogic plan
      Write in a familiar language
      (b)
```

Figure 2.7 – An Example of plan language and meta-rule

Such approach of representation is more specific in the given domain and difficult to transfer from an ITS to another [Clancey 92]. Woolf and McDonald [Woolf 84] studied the strategy representation of GUIDON (complied in about fifty tutoring rules) for transferring from a domain to another. Thus, the system MEMO-Tutor [Woolf 84, 87] goes up to an abstract level (concerning the strategies) which is higher than that in GUIDON. We introduce in the following some other formalisms for the strategy knowledge representation.

Representation of Tutoring Strategies

TUPITS [Woolf 92b] is a frame network, which represents tutoring strategies with the help of a primitive action vocabulary, such as teach, motivate, summarize, contrast, give examples, etc. A frame represents an object in TUPITS and some relations, such as prerequisites or corequisites, link each frame to other frames. These objects may be lessons, knowledge units, examples, questions, etc. The information associated with each object allows the system to dynamically reply to new tutoring situation. For example, each knowledge unit, or each subject represented as an object, has an associated procedural method: teach, teach prerequisite, test student, summary, give examples, motivate student, compare with other knowledge and give didactic explanation.

The problem in the representation is the mixing domain knowledge and strategy knowledge as if these strategies are implemented as the methods of a knowledge object. Against, it can specify clearly the strategy associated to the teaching of particular knowledge.

Management of presentation

Murray [Murray 91] proposed a response matrix that allows implementing response strategies and response tactics (figure 2.8). This matrix combines response strategies, tactics of response and actions of tutoring.

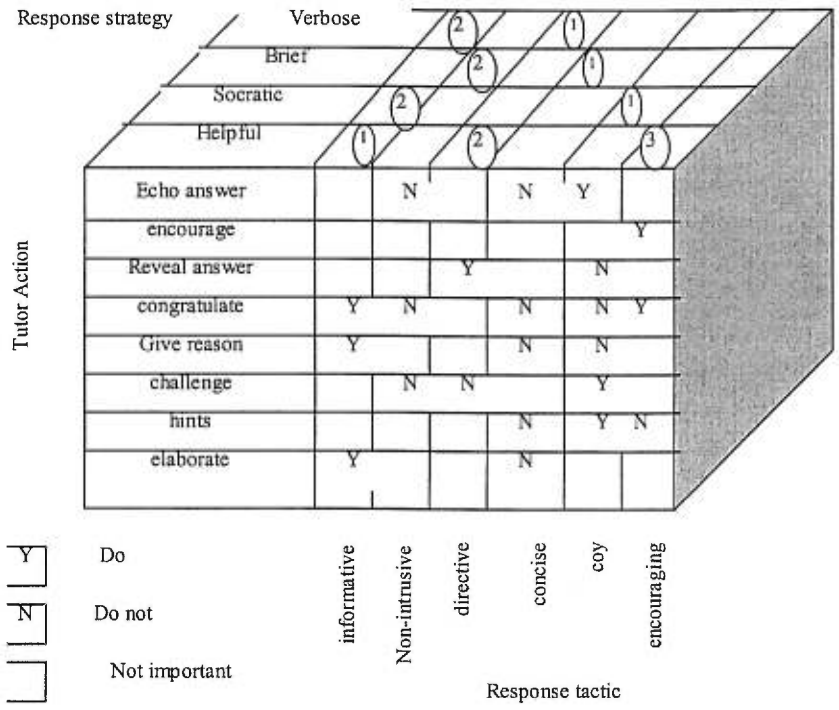


Figure 2.8 Adapted response matrix [Woolf 92b]

Several tactics can be associated with a response strategy. A priority thus can be given to each associated tactic. This is described by a value of priority in the matrix corresponding to the association, which the system should give to the tactic of the strategy. For example (figure 2.8), for being brief, the system should be prior to *non-intrusive* then *concise*.

Management of user-system interaction

The management of interaction between users and a system [Woolf 87] concerns the skills of the system to maintain the interactive discourse with a student and to adapt its response to the student according to the

levels of discourse. For example, the system should ensure that an interaction is directly linked to the student's history, to his or her learning style and experience with the system. It should reason on the student's actions, on the curriculum and on the history of discourse of dynamic means. The first formalism, which can represent the type of interaction between the system and a student, was proposed by Woolf and McDonalds and used in MEMO-Tutor. The formalism is called DWN (Discourse Management Network); by the following it has evolved with another version: PAN (Parameter Action network) [Murray 93b, 93a], DACTN (Discourse Action Network) [Woolf 92b, 92a]. These are the mechanism in ATN (Augmented Transition Network) [Woods 70], which represents and controls the dialogues between a student and a system.

A PAN represents possible a situation space of a discourse when to response a student's request. Arcs in PAN are states of knowledge and are defined as the set of predicates, while nodes in PAN provide actions to tutor. The difference from the ATN is that the actions are placed on the nodes and not on the arcs. The advantage of the method is that allows the nodes to represent the abstract actions that will be developed when the node becomes accessible during the execution of PAN.

PAN (DACTN) has been used for the representation of communication in the static domain with STATIC-Tutor, for supporting the development of management skills of time [Slovin 88] and for explaining the concepts and the process in the theory of electric networks [Suthers 90].

2.3.4 Some Typical Curriculum Systems

2.3.4.1 Curriculum in BIP

BIP (Basic Instructional program) [Barr 76] is a tutoring system of an introduction course to BASIC programming. It individualizes the teaching sequence via the appropriate selection of series tasks in a set of 100 examples of problems. Contrary to the approach of probability selection of Kimball [Kimball 82], BIP puts its selection strategy on the content information in a network called Curriculum Info Network (CIN) that connects tasks in the curriculum to domain objectives (curriculum issues [Lajoie 89])

In BIP-I, the curriculum is divided into three conceptual levels. The super level contains the principal themes of expertise called techniques, that are ordered in advanced following the prerequisite relations. These techniques are composed of knowledge units of immediately lower level called skills. The skills are not linked each other. The last level contains tasks that are connected to skills, allowing thus to exercise the last one.

The student model is a conveyance on the skills matching with a numeric value representing the student's performance. The last one is compared to a sort of interview (post-test) following each exercise. The

strategies of selection of next exercise take into account the values of student model on the skills, on the interviews and on the curriculum structure.

BIP-II [Wescourt 95] refines and increases the content information in CIN. The required skills for a technique are always regrouped in a set, but this time is organized in another network (figure 2.9). The skills are connected by the links representing pedagogic information, besides the prerequisite relations: analogy, class-object, functional dependence and relative difficult. Except for this extension of CIN, the global structure of BIP-II is identical to that in BIP-I. The task of selection should be however more refined and more opening.

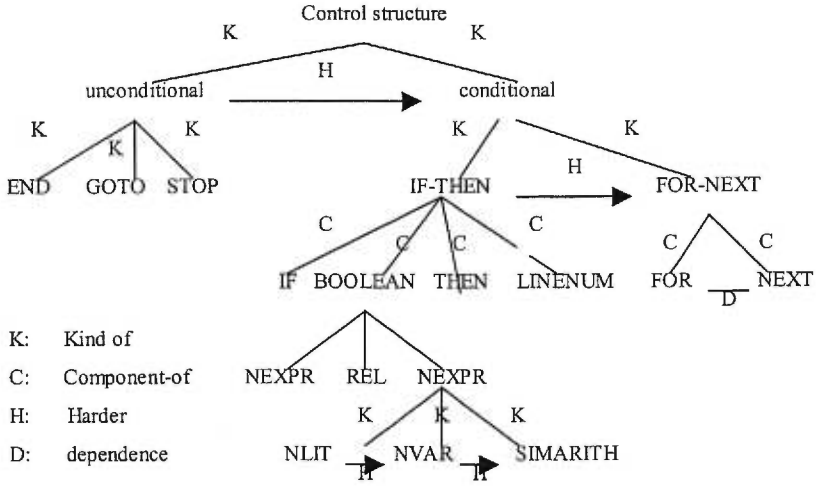


Figure 2.9 A portion of curriculum in BIP-II

A major limitation of BIP is that it does not support diagnosis and reasoning analysis of students because there is not task model. However, it is one of rare ITS which the reoccupation is the teaching of a complete course. It explains the central role of curriculum in BIP.

2.3.4.2 Organization of Content in SCENT

SCENT [Bretch 90] is an adviser system for learning of LISP. The curriculum in SCENT (SCENT-3) is considered as the result of the planning of content. This approach of curriculum corresponds to the view of McCalla [McCalla 90], according to which the curriculum corresponds to a plan of course adapting a particular student. The notion is taken back in PEPE [Bretch 90] where an extended structure of the granule theory [McCalla 94], with the introduction to the prerequisite link, in addition to existing links (aggregation and generalization), is used for generating a plan called curriculum. We believe that sequencing teaching activities dynamically should play a role in curriculum, rather than organizing materials simply for the final teaching.

2.3.4.3 Organization of Knowledge in Sherlock

Sherlock [Lajoie 89, Lesgold 92] is an advice-based tutoring system in which the goal is to supervise and to help the technician students in airplane who have taken the theoretic courses, to learn the knowledge in the detection of breakdown on the F15 of America army. The learning environment includes a simulation of test station and equipment to test. The principal strategy consists of confronting a student with a problem; including the task of giving a hypothesis on the origin of problem and proposing the solutions for the reparation.

One of the powerful points of Sherlock is that it is based on a solid analysis of tasks (done with the help of experts of domain) in detecting the breakdown that poses problems concerning learners' difficulties. The analyses of tasks allow to identify the skills (curriculum issues) that distinguish the expect technicians from novice technicians. The curriculum issues have been grouped into three categories: the strategies of detection, the strategies of repairing, and the strategies of resolution and making decision. Sherlock has thus led to 61 curriculum issues, which are implemented as the pedagogic goals.

A total of 34 problems (10 concerning the devices to test and 24 linked to the station of test) have been implemented in Sherlock following the analysis of tasks. Each problem is associated with a knowledge structure called WPS (effective problem space) that represents the possibilities of solution. A set of curriculum issues is associated to each node of an EPS (figure 2.10). This constitutes the second enters of curriculum in Sherlock.

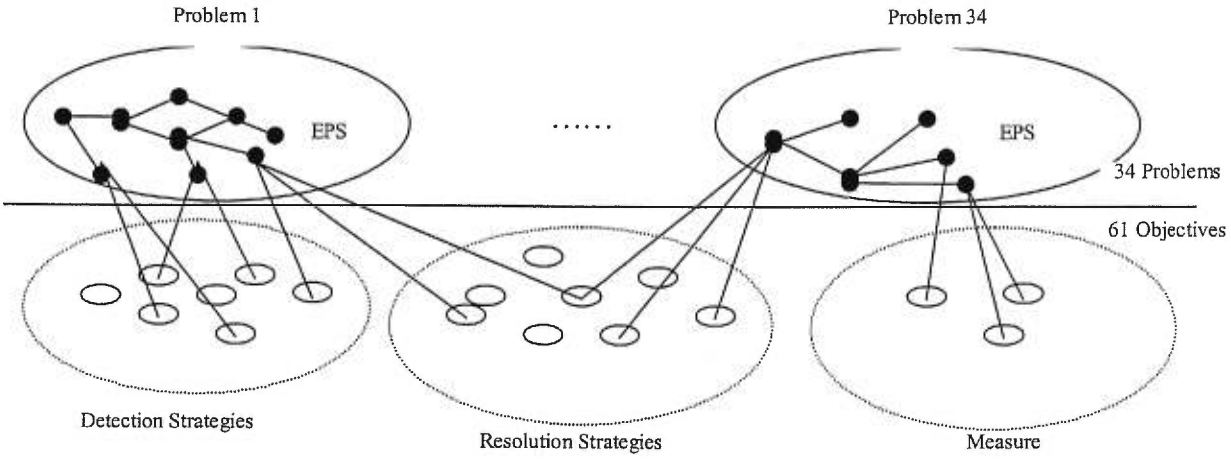


Figure 2.10 Organization of curriculum in Sherlock

The development of Sherlock II [Lesgold 92] does not aim at the improvement of curriculum in Sherlock I; its objective is especially to improve the modeling of physic system and to refine the student model. In fact,

Sherlock I can know exactly how to handle the solution of a problem, but it has some imperfect data on the student's knowledge. The concept *reflexive follow* has been introduced in Sherlock II for giving the possibility for the student to replay his or her solution, by the visualization of a diagram of real circuit simulated by the system. Another innovation in Sherlock II is the planning that of the student. The point of selection in the problem list is automatically adjusted following the performane of the student on the current problem [Katz 92]. This may be considered as an evolution of curriculum in Sherlock.

As a domain-specific tutoring system, Sherlock achieves meaningful results. It distinguishes domain problems from objectives explicitly. However, it is appropriate for only some task domains, rather than a general-purpose curriculum model.

2.3.4.4 Curriculum in SAFARI

In the project SAFARI, proposed by Gecsei and Frasson [Gecsei & Frasson 94]. Nkambou proposed a curriculum model: CREAM [Nkambou 96, Nkambou et al 98]. CREAM organizes learning outcomes, objectives and didactic resources as three separated networks first. A global association network, Curriculum Knowledge Transition Network (CKTN), is responsible of forming tutoring activities based on the three separated networks.

- **CREAM Architecture**

The architecture of CREAM is shown in Figure 2.11.

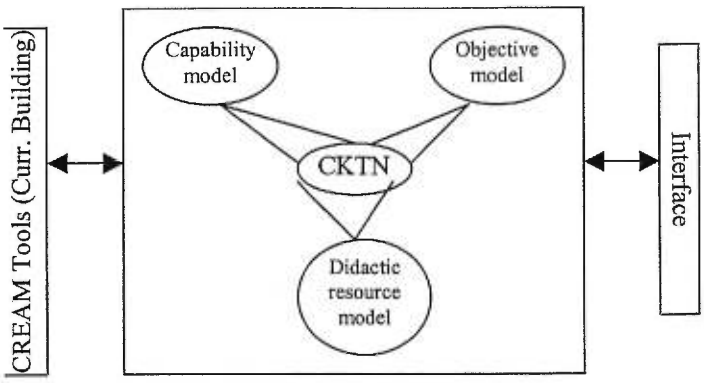


Figure 2.11 Architecture of CREAM

Four models in CREAM are shown in the central rectangle of the architecture. Capability Model represents and organizes domain capabilities as learning outcomes. Objective Model deals with objectives to teach capabilities. Resource Model contains all kinds of didactic resources. CKTN is the core of CREAM that combines the capability, objective and resource models together to get a knowledge transition network that is used for creating tutorial actions.

Two interface parts include the didactic workshop used for the acquisition of curriculum knowledge, and the interaction between CREAM and other tutorial components in SAFARI such as a planner. The CREAM based teaching development process is shown in the Figure 2.12.

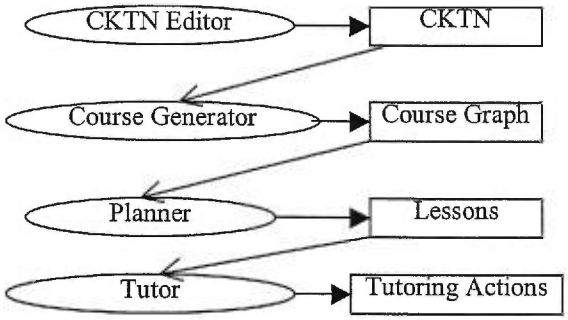


Figure 2.12 CREAM based teaching development process

- **Capability Model**

CREAM identifies learning outcomes based on the classification of capabilities by Gagné [Gagné et al. 92: intelligent skills, verbal information, cognitive strategies, motor skills and attitudes. Most of them also contain some subclasses.

In the Capability Model, CREAM defines five types of relationships between capabilities including analogy, generalization, abstraction, aggregation and deviation. When to build a capability space, a curriculum author should describe all relationships between capabilities.

- **Objective Model**

According to the instructional principles of Gagné, performance objective is to show progress toward the instructional goals. These objectives are:

(1) to provide a means for determining whether the instruction relates to the accomplishment of goals,
(2) to provide a means for focusing the lesson planning upon appropriate conditions of learning,
(3) to guide the development of measures of learner performance, and
(4) to assist learners in their study efforts.

CREAM defines an objective as a description of a set of performances that a student should be able to demonstrate after learning. It contains information about the levels according to Bloom, acquired skills, the context students should demonstrate, and success criterion.

The "context" in an objective refers to the concrete means to guide the tutoring. The attribute " Criterion of Success " gives the standard for evaluating the acquisition levels of students, which depends to the specific " context " attributes.

There are three main relationships between objectives, that is, prerequisite relationships, pretext relationships and contribution relationships. The prerequisite relationships may be also classified as obligatory prerequisite and desirable prerequisites.

- **Didactic Resource Model**

Didactic resources comprise all kinds of tutoring materials that support the tutoring to achieve certain objectives for acquiring certain capabilities.

In CREAM, didactic resources are classified into three principal types: tutorial resources, intelligent resources and dumb resources. Tutorial resources refer to the resources how to guide tutorial activities. Intelligent resources consist of problems, demonstrations, exercises, electronic documents, intelligent videos, tests, exams, etc. Dumb resources are primitive tutorial materials such as texts, images, sound, static multimedia and physical resources, as well as the dynamic resources, for instance, animation, simulators, hypermedia, and dynamic physical resources. Organization of resources is based on the following six types of relationships: similarity (analogy), abstraction, particular cases, utilization, auxiliary and equivalence.

- **Partial Association Spaces**

In order to carry effective tutoring, CREAM identifies partial associations spaces between models: Capability-->Object (C-O), Objective-->Capability (O-C), and Objective-->Resource (O-R), as shown in Figure 2.13.

The C-O association contains two prerequisite relationships, i.e. obligatory prerequisite and desirable prerequisites. Contribution relationships from objectives to capabilities, in O-C association, contain three levels: strong, medium and weak contributions. They represent the degree that an objective supports some capabilities. The O-R association reflects the relations between objectives and resources. There are two relationships in this subspace, i.e. critical resources and accessory resources. The accessory resources are used for the alternative resources when a student can not achieve certain capabilities after the critical resources have been used.

- **Global Associations**

CKTN (Concept Knowledge Transition Network) is a global association space that combines the above three models together (Figure 2.13).

In CKTN, a transition node connects capability nodes, objective nodes and resource nodes. The relationship from a capability node to an objective node is a prerequisite relationship, and from an objective node to a capability a contribution relationship. Resource nodes linked to objectives are real materials of tutoring activities.

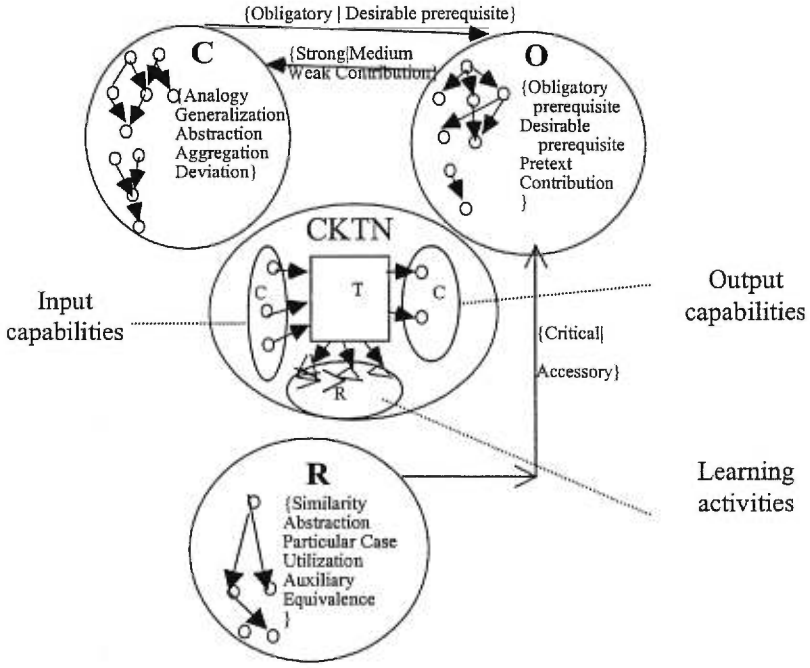


Figure 2.13 Relationships and Associations in CREAM

- **Characteristics and Limitations of CREAM**

Compared with most tutoring systems, CREAM exploits general-purpose curriculum development environment and enhances the curriculum as the central role in ITS. This motivation itself is more advanced than specific-domain systems. Learning outcomes in CREAM are capability theories of Gagné that reflects the conditions of human learning. CREAM uses multiple spaces to separate capabilities, objectives and resources explicitly, which makes it possible to develop a curriculum in parallel. Meanwhile, this enhances flexibility of knowledge utilization in teaching delivery stage. For example, analogy relationships between capabilities in Capability Model can be used by tutor module to remedy learners' errors.

However, some limitations exist in CREAM. Multiple spaces increase the complexity of system architecture and realization. With CREAM, a real curriculum would consist of a number of nodes, for instance, the curriculum for teaching EXCEL includes about 500 capability nodes. Both learners and curriculum authors are easy to lose in such large space. If the system can supplies visual navigation ability, or the system can support compounding multiple capabilities, it will enhance usability of system.

2.3.4.5 Other approaches for curriculum representation

Hartley [Hartley 90] began on the principle that acquiring a theme (subject) of teaching may be by three levels: operational level, conceptual level and procedural level. Thus, in his representation, the central object is the theme that is represented by combination of the three levels. These themes are connected by links of prerequisites, of composition or of sub-classes. The objectives of teaching are selected according to the level of acquisition of concerned theme. Some rules allow building an order on objectives. A more interesting aspect of the approach is the fact that associated success to each subject can do several levels; in fact, the stress may be also put better on the conceptual knowledge of subjects, than in the procedural or operational knowledge. Each consideration corresponds to an objective. This characteristic is specified as an attribute of subject. An operational knowledge is defined as the utilization of conceptual and procedural knowledge in a certain context. Consequently, the objectives of conceptual or procedural knowledge level associated to a subject (and the objectives associated to its sub-subjects and specialization) should be realized before its operational knowledge not becomes a realizable objective.

McCalla, Peachey and Ward [McCalla 90] proposed a representation of courses with the help of AND /OR graph, where the nodes represents the concepts to teach and the arcs, the necessary prerequisites. All successive nodes of the arcs of type AND are prerequisites. Each node of graph can be decomposed in a sub-graph comprising more fine concepts. One of advantages of the representation is that it is flexible because it can support several views (the expert can integrate different approaches of decomposition according to teaching style or the detail level to consider). Another advantage is its adaptability because it allows easily generating an individualized curriculum for a given student. However, such decomposition is not sufficient for the curriculum as the defined one: the teaching objectives should be formulated and the necessary resources should be specified. Thus the individualization of curriculum in the approach creates a mixture of the notion courses and curriculum.

Derry et al. [Derry 90] proposed a representation called knowledge model, where the teaching objectives are represented by nodes. Links between objectives may be the type of prerequisites, the competence to realize, or the attitude to know. The important aspect of this representation is the notion of teaching point that is expressed in terms of competence to acquire (objective to reach) and acquisition level of the competence.

Lesgold [Lesgold 88] proposed a knowledge representation in ITS with three levels: domain, pedagogic and strategic (figure 2.14). In this representation, a layer curriculum is represented by a specification of target structure, which guides the teaching of expertise represented by domain knowledge. The last structure is a trellis that brings out the hierarchy of objectives (which Gagné calls "learning hierarchy"). The only relation presenting the level is the prerequisite. The theory of Lesgold has separated clearly the domain knowledge from curriculum, and the strategy knowledge is also independent of domain knowledge.

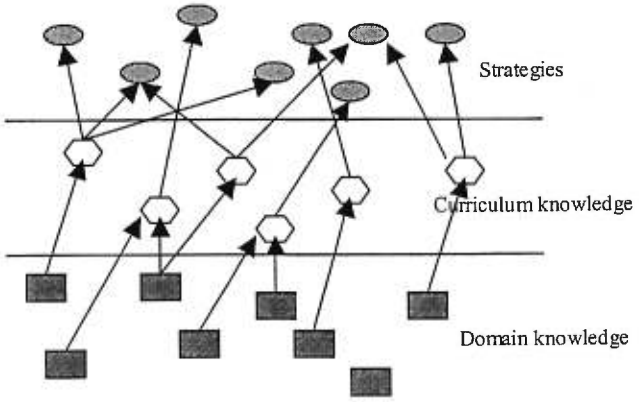


Figure. 2.14 Different Levels of knowledge in ITS (Lesgold, 1988)

Another type of curriculum is based on the help of finite state automate [Gauthier 89, Imbeau 90, Frasson 92]. In this model, two types of elements have been identified to represent the curriculum model: the knowledge unit -KU- and the teaching units –TU (figure 2.15). A KU contains learning objectives (syllabus) and performance level of student (student model). A TU represents the dynamic aspect of curriculum and is associated to the teaching content to present. One of advantages of this representation is that it considers student model and teaching aspect in a curriculum.

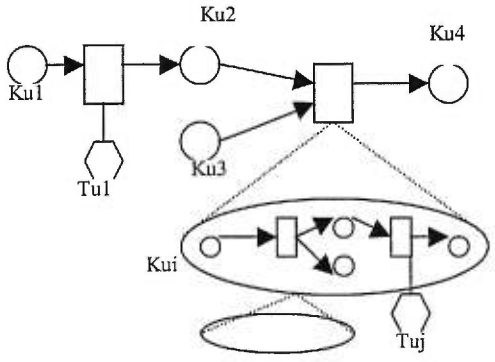


Figure. 2.15 – A representation for curriculum [Imbeau 90, Girard 91]

2.4 Conclusion

Most early tutoring systems are domain-specific. Many researchers focus on some aspects of tutoring such as reasoning on student performance, representation of domain topics, etc. Too few ITS researchers are interested in the notion of curriculum. This, in the education domain, goes to against a good teaching that requires a curriculum integrating the perspectives linked to domain, to the pedagogic and to the didactic. Certain approaches (McCalla, Imbeau, Ferraris) are proposed to deal with some issues.. CREAM is a recently proposed curriculum model that tackled more issues of curriculum and put the curriculum at the central role of ITS.

In chapter 4 we introduce a visual model of curriculum representation and organization, taking into account not only the visualization, domain and pedagogic aspects, but also the integration of teaching objectives and simplification of global transition structure. The visualization characteristics in our model will enhance the usability of the curriculum development environment.

Chapter 3

Visualization and Curriculum

Visualization of object spaces, a recently proposed discipline, can carry more semantic information, enhance interaction ability and reduce user losing in large object spaces. This chapter reviews the visualization possibilities used in curriculum development. Classical human-computer interaction tools are briefly mentioned first. The information visualization is assessed in section 2. Finally we review some tutoring systems that rely, in certain degree, on graph based representations, for instance Graphic Knowledge Base (GKB) [Shen et al 88], Graphical Instruction in List (GIL) [Resiser et al 88], Sherlock [Feifer et al 88], PIF [Frasson et al 92]., EXPITS[Takeuchi and Otsuti 92], Eon [Murray 96, 98], Synergy [Kabbaj et al 96], AGD [Paquette & Girard 96] and CREAM-Tools [Nakmbou et al 98].

3.1. Human-Computer Interaction Tools

One objective of this research is to determine how to visually manage a curriculum both at the authoring stage and when used by a learner. Relevant knowledge can be found in works concerning human-computer interactions and information visualization.

Currently, lists and outlines (or trees) are the only ways to represent large object spaces with standard user interface tools [Guo et al 95, 96a] (Figure 3.1).

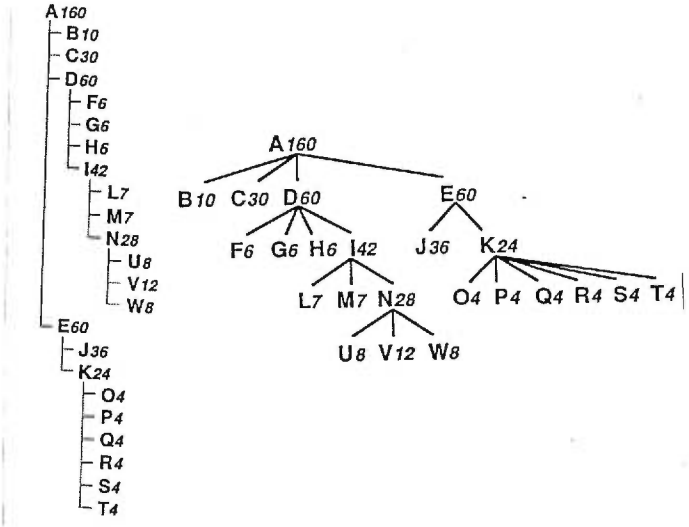


Figure 3.1 Outline (Trees) in standard user interface tools

In standard user interface tools, lists are a kind of simplest components representing a set of objects, so that almost all user interface tools provide lists for displaying collections of objects. One advantage of lists is their rapid response. Users click one button or a menu item, then all specified objects can be displayed rapidly. However, only dozens of ordered object names can be displayed with lists. Lists can not represent apparent semantics between items.

Outlines or trees are a little more advanced over lists. The directory structure in Windows 95 is an example of using outlines. It can display a graphics with vertical tree structure in which nodes are the names related to some files or directories, and links indicate part-of relationships. Though less limited than lists, this type of representation is not adequate to convey meaning associated with complex collections of items.

Standard user interface components such as the mentioned lists and outlines are basic tools for simple discrete objects, but they do not suffice in representing the complex information structures of a curriculum

A common approach to represent large information spaces by using current interface tools is the graph structure that users may create, using the buttons or icons of the standard graphic user interface. Labels and colors may be used to indicate names, types and states of information items. The most recent curriculum prototype in SAFARI is based on this approach.

However, this approach suffers from the fact that the graph is a static structure. This poses some viewing difficulties and makes it hard to modify the graph layout. Clearly the visualization of complex information spaces requires sophisticated display dynamics.

3.2. Information Visualization

3.2.1 Introduction

Information Visualization [Gershon & Eick, 95, 98] [Gershon & Brown, 96] is now the term used to visualize large structured information spaces against classical WIMP (Windows-Icons-Menus-Pointing) metaphor [Robertson et al. 93]. Classical user interface tools supply lists or outlines (trees) for displaying collections of objects. This is appropriate for only in simple cases, for example, small set of objects, single-type relationships between objects. However, in many cases, object spaces are large network structures. Furthermore, the semantic relationships between nodes are complex. Standard user interface tools fail to display such complex structures. Information visualization technology is to enhance the ability of current user interface tools. It combines visualization techniques and information management techniques to raise

the abilities of both. Some experimentation has been carried out, for example Treemaps, Focus+Context, Cone Trees, Rooms, and Dynamic Query.

3.2.2 Typical Visualization Systems

3.2.2.1 Treemaps

Treemaps [Shneiderman. 90, Johnson. 93] is a method for visualization of hierarchically structured information. Treemaps partition screen space into a collection of nested rectangular boxes representing a tree structure. The drawing of nodes within their bounding boxes is entirely dependent on the content of the nodes, and can be interactively controlled. The main objectives using Treemaps are efficient space utilization, interactivity, comprehension and esthetics (drawing and feedback must be esthetically). An example using treemaps is shown in Figure 3.2.

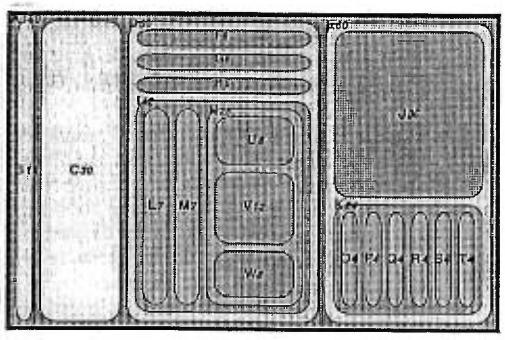


Figure 3.2 Treemaps

In Treemaps, children of a node are placed inside the display area of its parents. The structure of documents is implicitly represented. Users may assign some visual properties to indicate the document attributes, such as, colors indicating document types, sizes indicating the information amount of documents, and users may click any node to pop up a window to get details for the node.

The main contribution is the efficient use of screen space. It however has some limitations: only for tree structures, rather than general networks; and representing only implied single-type relationships, such as the inclusion relationships in file directories. Treemaps fail to represent complex networks such as a curriculum.

3.2.2.2. Focus+Context

Focus+Context [Lamping et al. 1995] is another well-known visualization technique based on the metaphor of logic structures. Focus+Context is used for visualizing and manipulating large hierarchies. Its goal is to assign more display space to a portion of hierarchy and still maintain the context of entire hierarchy. This scheme is to layout a hierarchy in a uniform way on a hyperbolic plane and to map this plane onto a circular display region. Figure 3.3 represents an example of Focus+Context.

Within Focus+Context, the overall structure of an information space may be mapped to a circular hyperbolic space. Users may change the focus of the whole space to view a particular local structure and may bring any node to the center of the space. The authors claimed 1000 nodes can be displayed in the hyperbolic space (in a 600x600 pixels window).

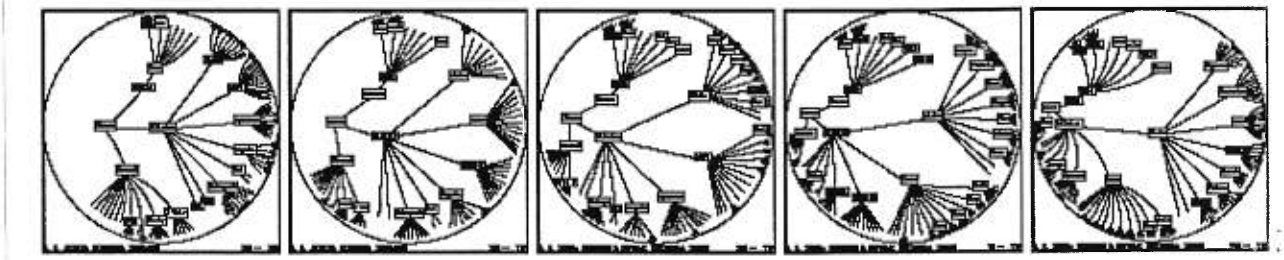


Figure 3.3. Focus+Context

Advantages of Focus+Context are that users get a sense of the overall tree structure, and with focus technique, users may find particular nodes according to the titles of nodes.

Focus+Context has some limitations:
(1) The overall structure of an information space is mapped onto a circular hyperbolic space, while actual screen space is a rectangle, so that the screen space is not used optimally;
(2) One change of focus causes both a change of the whole structure and changes in directions of links and positions of nodes, so that positions of some nodes is not stable; as a result users may be lost or find it difficult to retrieve some previous familiar data items;
(3) Computing cost is high because of animation and mapping to hyperbolic space. These limitations make it difficult to use for curriculum development and guiding learning.

3.2.2.3. Cone Trees

Cone Trees [Robertson. 1991] are a three-dimensional extension to more familiar 2D hierarchical tree structures. Cone Trees are constructed by placing the root node at the apex of a translucent cone that is near the top of the display. All children nodes are then distributed at equal distances along the base of the cone. This process is repeated for each node in the hierarchy with reducing the diameters of cones at each descending level to ensure sufficient space to accommodate all leaf nodes. Cone trees enable the related trees to bring any particular node smoothly into a focus area on screen. Figure 3.4 shows an example of cone trees at Xerox PARC.

There are some other prototypes that use cone trees to display large information spaces, such as LyberWorld [Hemmje et al. 1994], and WebQuery [Carriere and Kazman. 97].

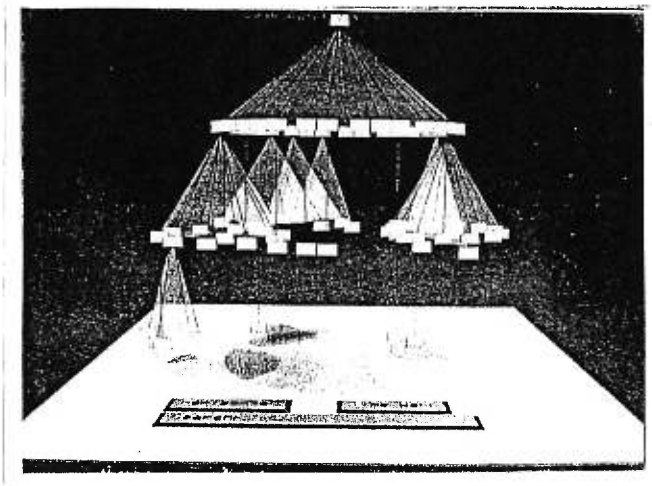


Figure 3.4 Example of Cone Trees

Cone trees are an attractive visualization metaphor. Its intuitive correspondence to the logic structure of an information space makes the users' navigation easier. However, cone trees are restricted to tree structures. Another limitation of cone tree is the high computational costs of their 3D animations.

3.2.2.4. Rooms

Rooms [Henderson and Card. 1986] are one of well-known early prototypes of workspaces. Its purpose is to prototype a workstation window manager that allows users to operate with large collections of objects. Within Rooms, the tasks of users are distributed in a collection of Rooms. A Room contains a set of window Placements, each of which indicates a window, a shape, and other presentation information. Users' work is distributed among several Rooms, and each Room contains a main task.

Rooms focus on two essential issues: simultaneous access to separated information and navigation. In order to deal with the simultaneous access to separated information, a window may be placed in multiple Rooms. This needs to reposition the shared window in different Rooms. To share collection of tools across tasks, a Room may be included in other Rooms.

The navigation in Rooms deals with four issues.

- returning to a Room,
- general orientation to find other Rooms,
- finding windows, and
- finding which Rooms are connected.

The prototype of Rooms provides an icon - Back Doors - that may be used for returning to the previous Room. In order to help users remember or discover the route to a particular workspace, Rooms provide a pop-up menu containing Room names and an overview that displays all Rooms in the system (Figure 3.5). Users can identify the particular window in other Rooms from the overview.

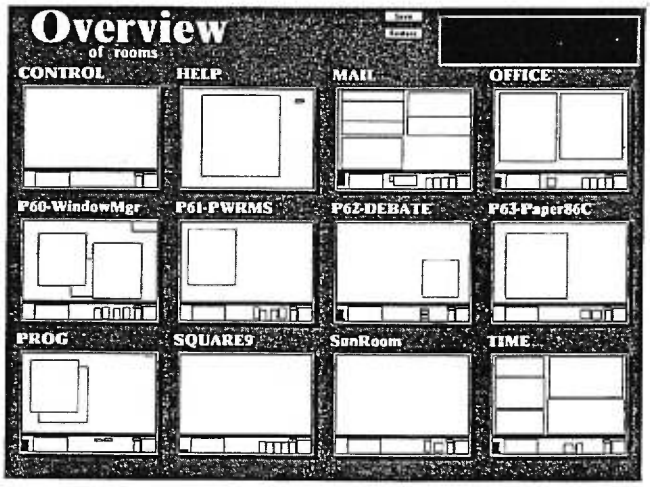


Figure 3.5. Overview of Rooms

Based on the overview of Rooms, the system can generate the links that connect different Rooms to help users find the connected Rooms.

Rooms' contribution is that it generates an easily manipulated workspace that bridges the shared windows and tasks. However, when the users' task space and shared information are complex, the navigation abilities are still weak. Although Rooms does not provide effective means to visualize large object networks, the ideas of multiple tasks and sharing information may be used for some curriculum prototype with multiple networks such as that in SAFARI.

3.2.2.5. Dynamic Query

Dynamic Query [Batafogo et al. 92, Shneiderman. 96, Tanin et al. 96. Plaisant et al. 96. Kumar et al. 97] is a typical example of using culling technique to help users understand information.

Principal objectives of Dynamic Query contain:

- visual representation of the world of action including both the objects and actions,
- rapid, incremental and reversible actions,
- selection by pointing (not typing),
- immediate and continuous display of results,
- the visual representation of query, and
- the tight coupling between visual query components and the query results.

Obviously, although many prototypes previously discussed have nice visual appearances, they can not satisfy these goals well.

Home Finder [Alberg and Shneiderman. 94] is a prototype based on the above principles (as shown in Figure 3.6). Within Home Finder, the background of display area is a geographical map of a country or a region. With a provided attribute palette, users can form desired queries by manipulating sliders to set attribute values of queries, then the system creates corresponding query results (houses satisfying certain conditions) and displays them in the geographical map with a starfield. When the user change the attribute values to query, the query results dynamically follow the change. The whole query process is incremental and direct manipulation. Furthermore, the user may also click stars (representing houses) in the display area to view the details of particular houses.

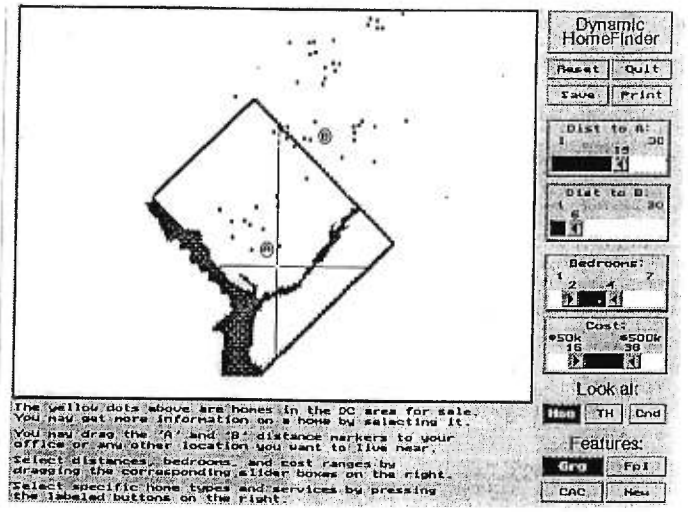


Figure 3.6 Home Finder Based on Dynamic Query

There are some other prototypes that use some dynamic techniques to help users understand information, for example, TileBars [Hearst. 95] display a rectangle that visually display the relevance statistics accompanying each title of the relevant documents.

Obviously, dynamic interactions provided by Home Finder can help users understand the whole query process. It is intuitive for the information concerning geographical areas. The dynamic query mechanism is useful for some other fields such as authoring and guiding learning. However, the knowledge elements and the relationships between knowledge elements in a curriculum are much more complex than that in these mentioned prototypes.

In sum, many ideas on visualization in the reviewed systems can be integrated into the consideration of visual curriculum model. However, the complexity of data types and semantic relationships in curriculum make it impossible to use these approaches directly to curriculum models. We have to explore the mechanism to integrate visualization techniques to tutoring fields.

3.3 Graphics and Curriculum

We, now, review some knowledge bases and curricula that use graphics in certain extent. These systems include: Graphic Knowledge Base (GKB) [Shen et al 88], Graphical Instruction in Lisp (GIL) [Resiser et al 88], Sherlock [Feifer et al 88], PIF [Frasson et al 92], EXPITS [Takeuchi and Otsuti 92], Eon [Murray 96, 98], Synergy [Kabbaj et al 96], AGD [Paquette & Girard 96], and SAFARI [Gecsei and Frasson 94].

3.3.1 Graphic Knowledge Base (GKB)

GKB (Graphic Knowledge Base) [Shen et al 88] methodology implemented a graphic knowledge base, in which the structured knowledge representation allows hierarchic and network representation of knowledge that can be used tutoring systems. This methodology includes two components; one is the representation of knowledge; and another is the architecture of the system. Knowledge representation contains two elements: objects and arrows. Each object in the representation is a knowledge entity that may contain a text file that describes the element. Arrows form a structured graphic knowledge base by linking knowledge entities together. Some shells are developed for the structured graphic knowledge base, which includes a user interface and a frame for describing element. Figure 3.7 shows an example of GKB methodology.

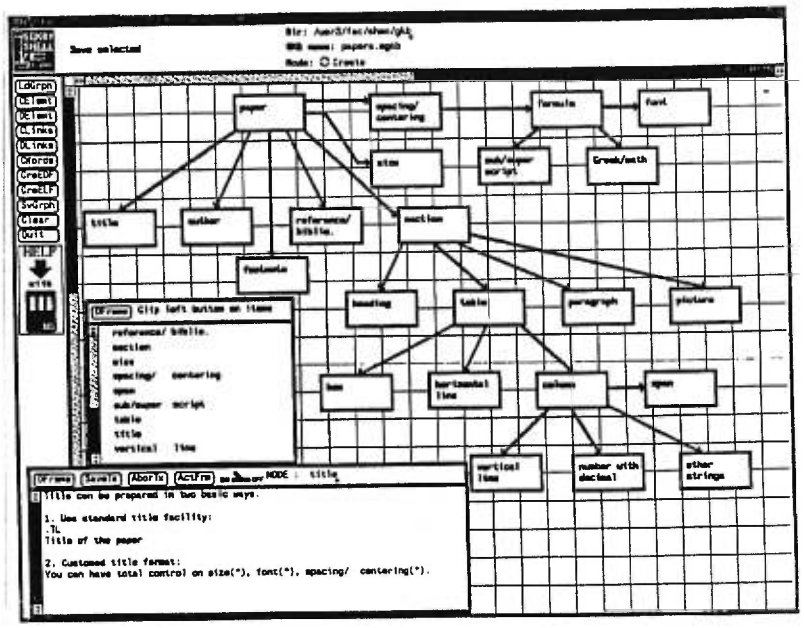


Figure 3.7 A snapshot of SGKBSA

GKB is just a query system of knowledge base, because there is neither tutoring strategies nor student model. It is a simple system relative to a curriculum.

3.3.2. Graphic Instruction in Lisp

GIL (Graphical Instruction in Lisp) is an intelligent tutoring for Lisp programming developed in Princeton University [Reiser et al 88]. The GIL tutor is embedded in a graphical programming environment. In order to build a program a student can connect objects representing program constructs into a graph. Figure 3.8

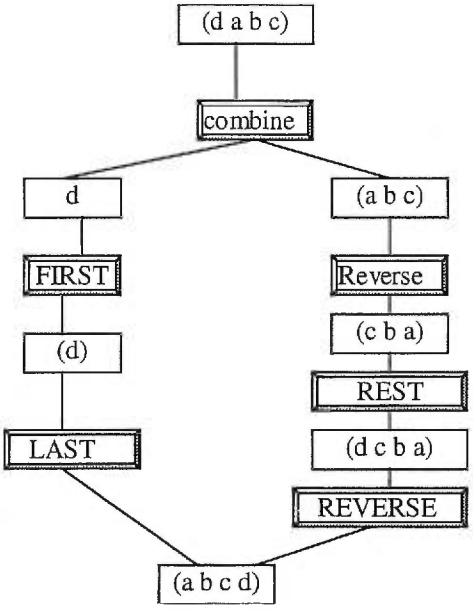


Figure 3 8 A complete problem graph for the problem *rotater*

shows an example of graph in GIL, in which each icon represents a programming operation and has one or more input nodes and one output node. The input nodes are the data operated on by the particular LISP operator, and the output node is the result of the operation. The student may select a programming icon from a menu, drag it into the Program Window, and specify its inputs and outputs. The student uses provided graphic tools to construct a program graph for a given problem. The system then checks its correctness and feeds back information to the student.

GIL is essentially a procedural simulation system. Its constructed graphs reflect explicitly LISP program structures. However, it is used especially in LISP programming. It cannot combination concept knowledge into graph, and the graphs cannot reflect tutoring strategies and student models.

3.3.3 Graphics in Sherlock

Feifer et al [Feifer et al 88] proposed a Graphic Mapping approach based on the Sherlock environment for teaching knowledge representation. It provides a learner with three components: a) a text to be represented pictorially, b) a screen containing icons representing concepts within the text, and c) a set of links which the learner can use to connect icons. Five kinds of links between icons are provided: IS-A, LEADS, EQUIV, CHAR and NOT. Learners are instructed to find icons that they believe are related and then choose the link that best represents the relationships. For example, if the learner believes that one icon represents a concept that is a generalization of a concept requested by a second icon, they should make an IS-A link between the two icons. A portion of the screen icons is pictured in Figure 3.9. Sherlock provides the learner with the capability to move icons, clone icons and link them together.

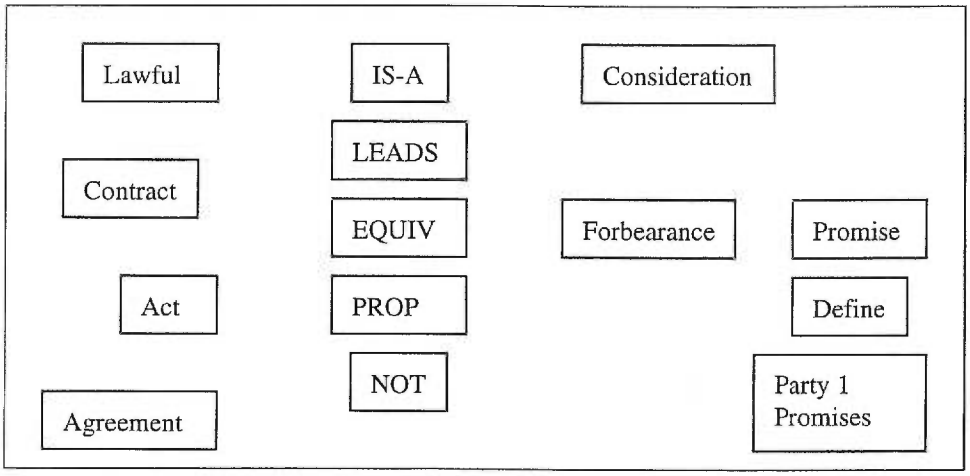


Figure 3.9 Opening screen showing some of the icons offered

The approach wants Sherlock to do two things while a learner constructs a graphic map: a) when the learner get stuck, suggests a next step and b) when the learner makes a mistake, provide appropriate tutoring.

This approach provides intuitive representation of concepts in a paragraph of text. However, it is not a curriculum of direct manipulation. All it represents are only the subjects and the relations among them. It is difficult to use to general domains. There is no visual tutoring strategy and detailed information for the represented concepts.

3.3.4 Physical, Intentional and Functional Worlds

Frasson et al proposed an approach called PIF (Physical, Intentional, and Functional Worlds) for teaching the diagnosis of breakdown of physical devices [Frasson et al 92]. This is a simulation-based tutoring. It uses three windows (called Window P, Window F, and Window I) to visual display the realistic image of a simulated physical device, a more abstract schematic diagram of the object and edit plans as well hypotheses, for detecting the breakdown of the physical devices. The system also provides the visual elements simulating repair tools. When a tool is used to a part of the devices, its affection can be visually displayed.

The PIF approach supports various tutoring control mechanisms. Though its example is a device of electronic circuits, it provides a mechanism for simulating physical devices in a wide range. However, tutoring strategies are not visualized. It is difficult to form a general curriculum development environment based on the model.

3.3.5 EXPITS

EXPITS [Takeuchi and Otsuti 92] is a direct manipulation environment to teach intelligent tutoring systems. This environment provides views and facilities to represent internal states of an intelligent tutoring system and to manipulate these states in order to study structures and functions of constituents of ITS. EXPITS divides topics in an ITS into problems-solving process, domain knowledge structure, student modeling, and teaching expertise. The system provides visualization facilities for these structures and functions in a certain extent. Figure 3.10 shows its main interactive views: resolution trees, knowledge structures and teaching expertise. A resolution tree displays a problem-solving process (upper-left window in figure 3.10). The window at the bottom-left in Figure 3.10 displays a part of a knowledge structure. The window on right in the figure 3.10 displays a list of teaching expertise, which is used for planning teaching global sequences and local teaching methods.

Though EXPITS provides some visibility, it is still a domain-specific tutoring system. There is no visual manipulation ability for the displayed structures. In addition, it has no dynamic navigation ability.

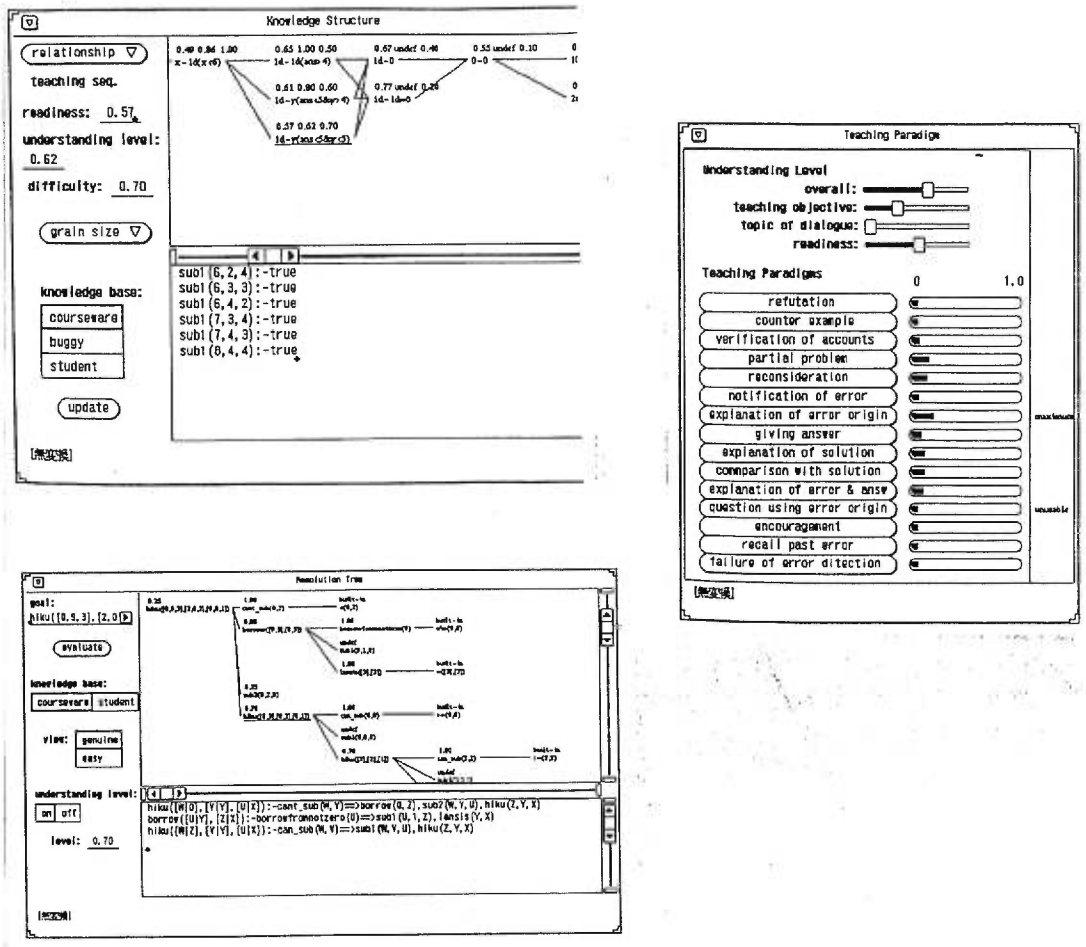


Figure 3.10 Windows in EXPITS

3.3.6 Eon

As a general authoring shell, Eon [Murray 96, 98] attempts to visualize the modules in ITS, including learning environment, domain model and teaching model. In order to support authoring and learning, Eon supplies some main tools such as Topic Net Editor, Present Content Editor, Topic Contents Editor, Interaction Editor, Flowline Editor, and Student Model Editor. With Topic Net Editor (window at the top-right corner in Figure 3.11), a curriculum author can visually build topic networks with provided graphical elements and links with different shapes and colors. These shapes and colors indicate the types and attributes of topics. The Present Content Editor (window at bottom-left corner in Figure 3.11) is used to create teaching materials. The author can connect topics with material through the Topic Content Editor.

With the Interaction Editor (window at top-left corner in Figure 3.11), interaction windows can be developed. Teaching strategies can be edited through the Flowline Editor (window at bottom-right corner in Figure 3.11), which also sustains visual ability for organizing teaching strategies.

The learning environment in Eon uses a knowledge-based paradigm [Murray 96a] for representing instruction content, in which, each screen in a reusable template, and an author creates "content" objects to

fill in these templates. Eon's interaction editor contains a hierarchical pallet of user interface components called widgets. There are simple widgets such as button, text, pictures, sliders, movies, and hot-spots, and more complex widgets such as multiple-choice dialogs, tables, and graphs.

The Eon tools, perhaps, is one of the systems with the most visual properties in existing authoring environments. Numerous visual properties in Eon make the system more usable in practice. However, Eon has some limitations. The topic nodes in Eon are graphics, rather than manipulatable components, so that we cannot get detail information of topics by clicking. Although Eon can connect teaching materials to topics, it does not visualize teaching activities. The author of Eon does not mention how to support the visual creation of specific course for a particular student. In addition, Eon does not support composite topics. In our proposed curriculum model, these limitations will be tackled.

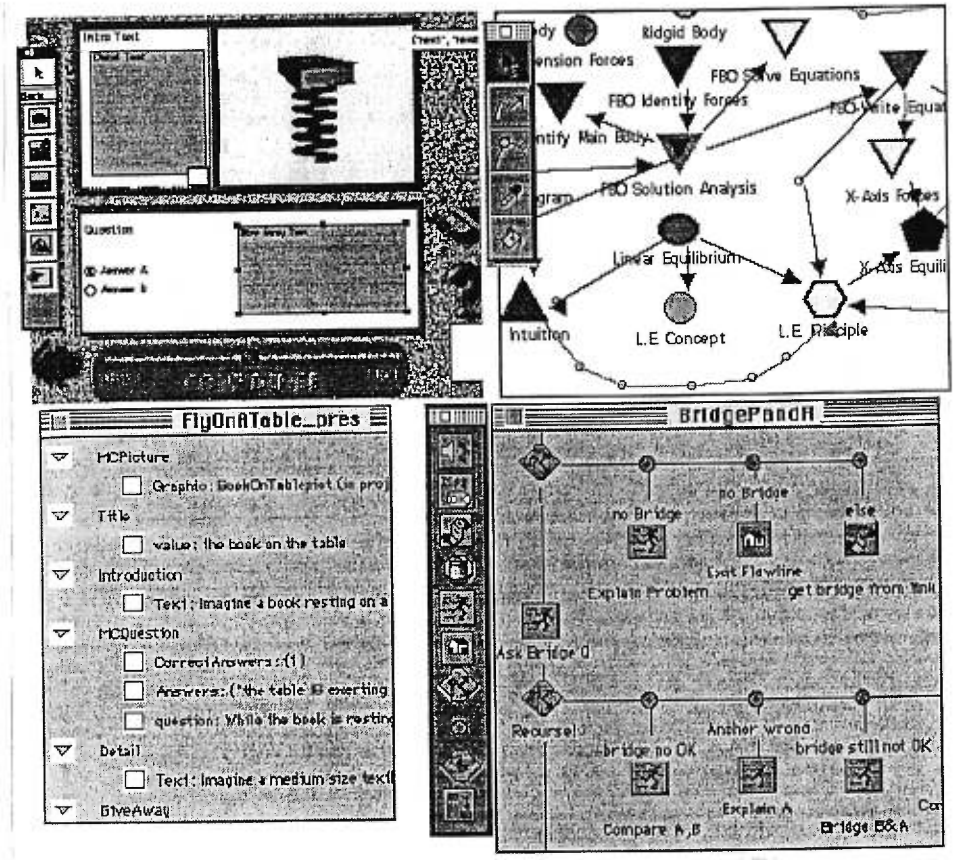


Figure 3.11 Windows in Eon

3.3.7 Synergy

Synergy [Kabbaj et al 96] is a graphical multiple-paradigm language to represent declarative and procedural knowledge. Conceptual environment of Synergy is composed of a long-term memory and of a work memory. The long-term memory is used to represent domain knowledge bases with generalization graphs. The work memory is the place where a user specifies his requests. Synergy defines a wide range of data types to support, from primitive types (such as number, string, Boolean, etc.) to custom types (such as procedures, activities). A user may use the Synergy language to write their program by drawing nodes and links. Each example of a Synergy "program" corresponds to a parallel activation of multiple semantic networks. In these semantic networks, the activation of a concept corresponds to its value's activation.

An example of application of Synergy for modeling intensive care unit is shown in Figure 3.12, which displays definitions of some types in the example

As a kind of graphical programming language, Synergy is a meaningful exploitation, because of its intuitive presentation and supporting declarative and procedure knowledge. However, as a development environment of ITS, some limitations exist. For example Synergy uses same types of nodes to display all kinds of nodes from primitive data types, such as Boolean, to very large data types, such as tutoring processes, so that the system loses the explicit structured properties, and causes new losing of users

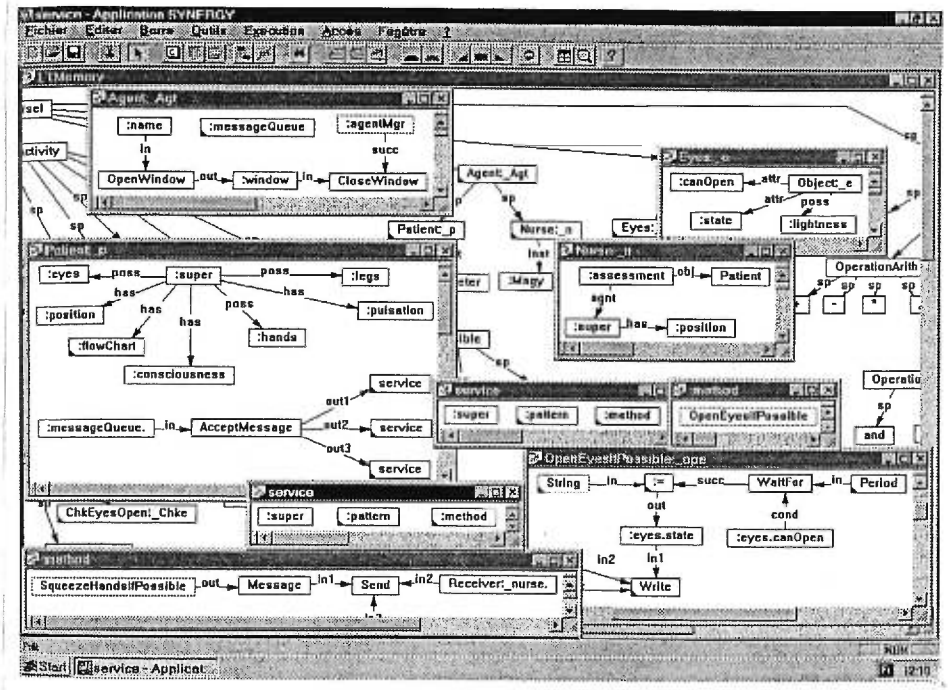


Figure 3.12 an example of Synergy application in Intensive Care Unit

3.3.8 AGD

AGD [Paquette & Girard 96] is a performance support system for content experts involved in educational design, which contains a task support system and an advisor based on the task support system. The task support system puts a strong emphasis on strategic instructional design knowledge. It identifies 156 sub-tasks for building learning systems, which are fallen in 5 categories including defining learning situation, building and distributing knowledge models, designing learning systems, managing projects and planning the development of learning system. In AGD, the knowledge model editor provides two kinds of nodes: procedure nodes and concept nodes. A procedure may be linked to several sub-procedures. Meanwhile, a concept maybe linked a procedure as input or output of the procedure. Needs of learning may be attached to knowledge units. One can transfer a selected group of knowledge units from an existing model. With AGD, an author can also define graphically learning scenarios, in which, learning components can be linked didactic to learning events. AGD, in fact, is an advisor system for building ITS. Though AGD identifies many sub-tasks of building a learning system, only a small number of ID strategic knowledge is implemented. The author of AGD in their paper do not deal with how to support various instructional strategies, how to support individualized teaching (dealing with student model) and how to manage various didactic resources.

3.3.9 SAFARI

SAFARI [Gecsei and Frasson 94] is a domain-independent development environment for ITS. Its curriculum model has been reviewed in the previous chapter. An authoring environment named CREAM-Tools [Nkambou, Frasson and Gauthier 98] has been developed to sustain the development of curriculum based on CREAM.

CREAM-Tools use three kinds of visual nodes to represent capabilities, objectives and learning resources. These nodes are created with rectangles and labels having certain sizes and fonts. Links between nodes describe relationships between corresponding components. A curriculum author can layout knowledge transition networks by dragging. Figure 3.13 shows an example of graphical transition networks in which the rectangle nodes are objective nodes and the rectangle nodes with arc corners are capability nodes. An important feature of the prototype is that it can graphically display capability spaces, objective spaces and resource spaces. Further, it can organize a new space-CKTN-that associate with the above three spaces to get a knowledge transition network (Figure 3.14).

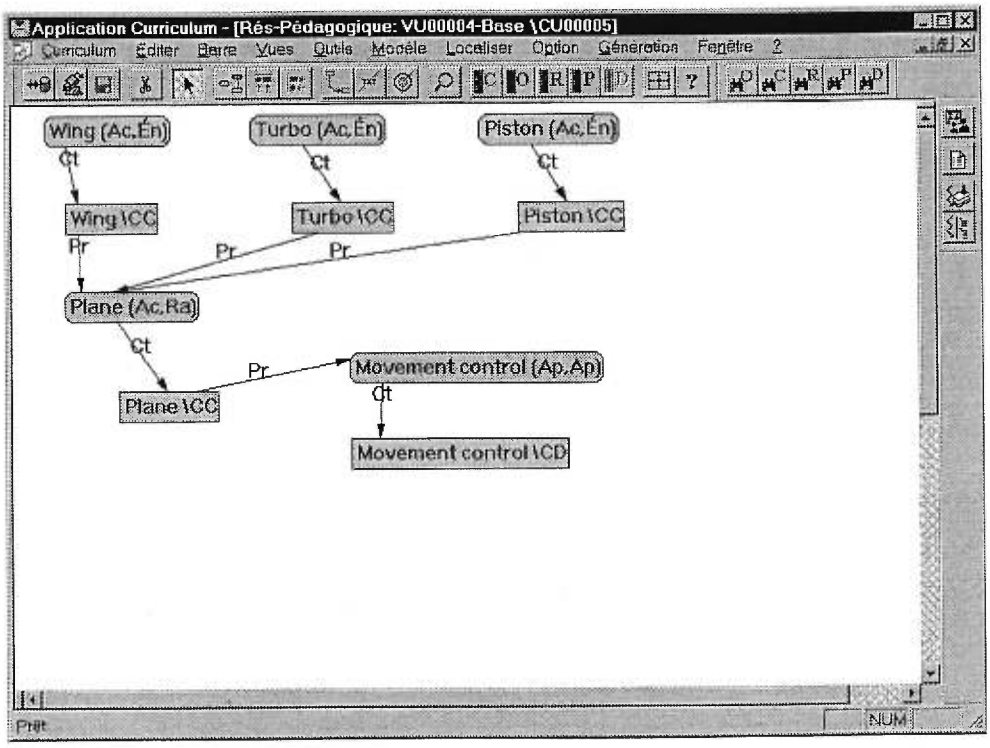


Figure 3.13 An example of transition network in CREAM-Tools

Compared to previous curriculum development environments, the graphical representation of knowledge transition networks in SAFARI is a new exploration. However, the nodes in the prototype are relevant simple because they can not carry more semantics such as capability levels, states, viewing details by simply clicking, etc. In addition, it does not support visualization of composite capabilities and integrated tutoring activities.

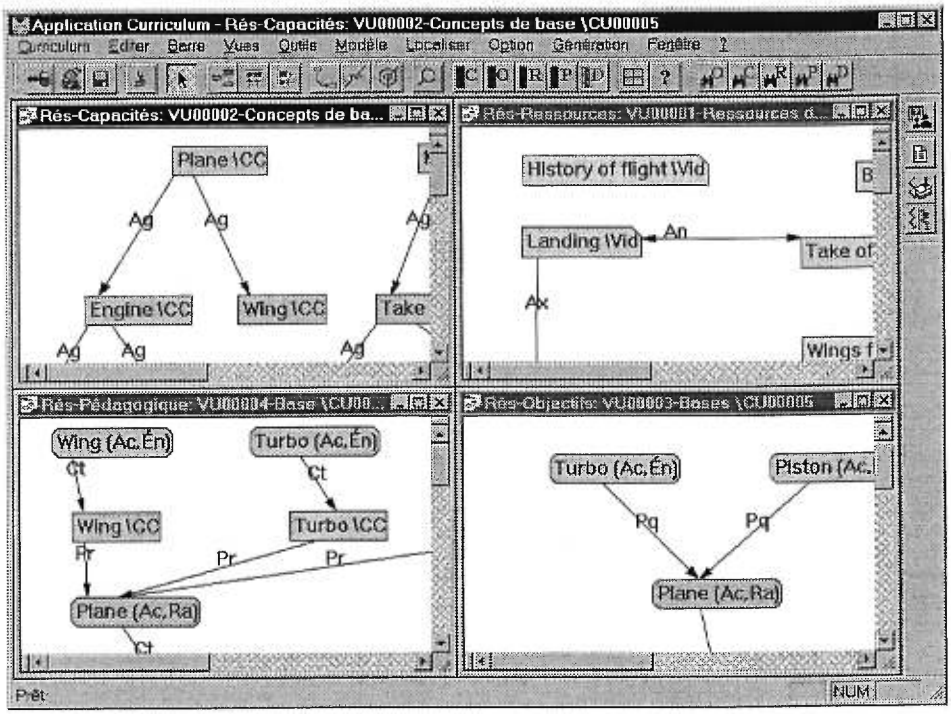


Figure 3.14 Capability, objective, resource and CKTN spaces in CREAM-Tools.

3.4 Summary

3.4.1. Characteristics of Existing Prototypes of Information Visualization

All the reviewed systems attempt to explore how to easily organize and query large object spaces from the point of view of visualization and human-computer interaction. Uniform data type, simple semantic relationships, and simple navigation ability are their common characteristics.

- Uniform Data Types

The data types in almost all reviewed prototypes are limited to some particular application domains such as documents, file management, code line management, and data pieces in structured databases. For example, Treemaps and Cone Trees initially are used for the file directory management. Home Finder based on dynamic query is specifically for the house database.

- Simple Semantic Relationships

Another feature in the discussed systems is that the relationships between data items are simple. Most of them have unique relationship. For example, the relationship between files is only the part-whole or inclusion relationship. In code line and code file management, the associations are the calling or called. Of

course, many types of relationships may be contained in database domains. However, the features of structured and exactly matching may be used to handle these relationships as nodes.

- Simple Navigation Ability

The requirements of visual navigation in the reviewed systems are relatively simple because they manage single type of data, simple relationships and tree structures. However, the structures in a curriculum are complex networks and consist of heterogeneous nodes and links. Students often lose in the unfamiliar concept spaces.

3.4.2. Characteristics of Graphical Tutoring Environments

A few tutoring and curriculum systems dealt with graphical aspects to a certain extent. These systems fall into three basic categories: material-oriented systems, topic-oriented systems and curriculum-oriented systems.

Materials-oriented systems attempt to provide a graphical environment based on teaching materials, with which a physical device or a procedure to teach can be simulated. However, this kind of systems is domain-specific. Topics, teaching activities and teaching strategies have no more visibility in such kind of system. What a curriculum model needs is a tool that can be used for more domains.

Topic-oriented systems can graphically display domain topics and relationships between topics. This is useful for viewing structures of domain knowledge. However, the topic networks provided by most topic-oriented systems are static graphics. One cannot visually manipulate these graphics for detail information. In addition, these systems cannot integrate and visualize teaching activities and strategies. This is not enough for a curriculum development environment.

Curriculum-oriented systems are closely related to our research. The reviewed Eon and SAFARI are two typical examples. The purpose of these systems is to supply a general development environment for curriculum. Their exploitations on visualization achieve success to a certain extent. However, the visualization abilities in their systems are still limited to certain ranges. These development environments can display graphically topic networks, but these topic networks are just graphics and they cannot be visually manipulated the details of each node. No prototype can support the organization of integrated tutoring activities for composite topics. Visual dynamic navigation to learners and curriculum authors is important because any real curriculum network is so large that users often lose in such a big space. Another limitation of these systems is that they neither visually create multiple alternative paths to achieve a group of particular learning goals nor visually select and order tutoring activities for instant context and learners. Our proposed model will attack these issues.

3.4.3. Problem Complexity in Our Research

A curriculum's characteristics may be summarized as multiple and heterogeneous data types, complex relationships between data items, complex information structures, multiple tasks, large and multiple spaces.

The problem is complex because of the need to deal simultaneously with visual representation constraints and semantic relationships. So we need to address problems related to both areas: visual or surface properties of complex information structures and semantic and operational properties of complex information structures.

- **Powerful navigation supporting easy generation, organization and management of large knowledge spaces**

Figure 3.15 is a part of the capability space to teach EXCEL developed in SAFARI, there are about 500 nodes in the space. Only dozens of nodes can be displayed on the screen. The users can not view the overall structure of the capability space. In order to build the capability space continuously, a curriculum author has to scroll the big graph frequently. It is very difficult to find a particular node because most nodes can not be displayed. With the larger and larger graphs, authoring efficient is lower and lower.

- **Visually making sense of, and easily manipulating information workspaces**

Current systems are well adapted to representing information and knowledge in a sequential manner even if combined with hypertext links (the sequentially being induced by the ordering of link following). Representing information and knowledge spatially has been limited to representing static structures and very few works make the representations evolve with the changing interests and needs of the viewer. This is what we intend to do in this work.

In this research the users should be able to modify easily the visual properties of their graphs in order to emphasize (semantic) features of their interests.

In a curriculum, the relationships between the various knowledge spaces are complex. We are going to provide visual tools to explicit these relationships.

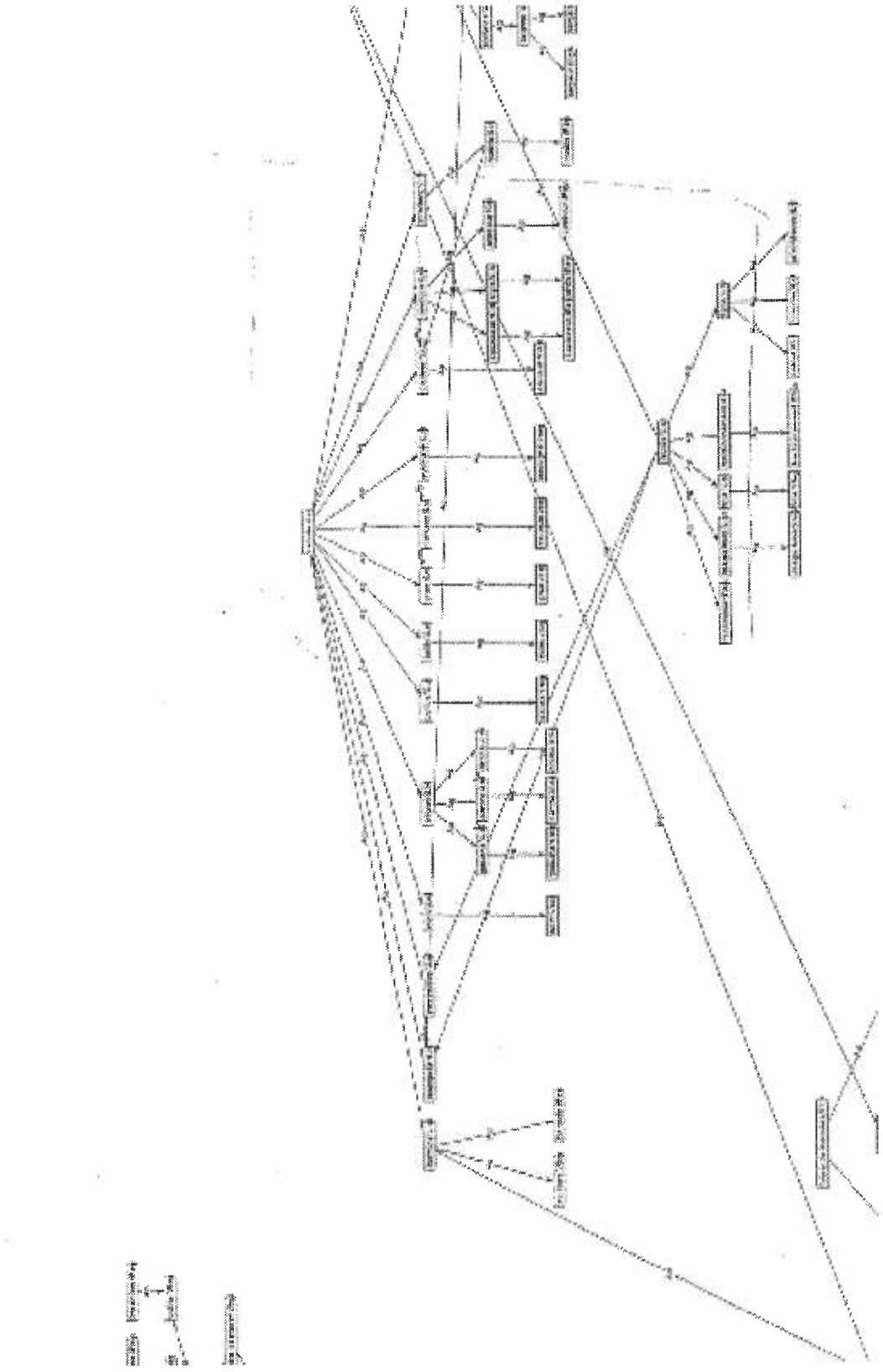


Figure 3.15. An Incomplete Capability Network for the Course " Excel "

Chapter 4

A Visual Curriculum Definition

A curriculum model [Finch 86] should includes the identification, representation and organization of didactic materials, instructional objectives, teaching events, and learning trajectories. In this chapter, we first review a few definitions of what constitutes a curriculum in our context. Then we specify what a visual curriculum can be. Last, we describe the central data structure for organizing capability transition networks in detail.

The visual curriculum definition we propose consists of three essential models each of which outputs a network structures called respectively capability transition networks, course networks and tutoring activity networks. A graph structure, called *Tnet* (Transition Network), distinguishes between knowledge states and teaching events conducing to new knowledge. *Tnet* is a structure for representing and organizing tutoring outcomes, tutoring events and didactic materials. Derived from a *Tnet* a simplified graph will represent an actual course a student may engage in. *Cnet* (Course Network)—provides a representation of the actual learning goals, events and knowledge relevant to the attainment of these goals. Further refinements on a *Cnet* actually produces a sequence of instructions called *Anet* (Activity Network). This includes only the events relevant to the student's attainment of a homogenous set of goals.

4.1 Some Existing Curriculum Definitions

In the educational domain, development of a curriculum is an important stage in the process of instructional design. Some researches [Finch 86] consider a curriculum as the input data of a tutoring system. In computer-based tutoring domain, a few works have been devoted to defining a curriculum, but too few researchers have considered the necessity of including a visual curriculum in a tutoring system, as reviewed in the previous chapter. Our purpose is to characterize a well-defined visual curriculum model to support efficient and effective tutoring.

In the educational domain a curriculum [Legendre 93] is defined as a structured set of instruction and learning experiences, in which the principal elements include content objectives, skill objectives, specific objectives, alternative learning paths, rules, didactic materials, and events of instruction and learning.

In the context of ITS research, a few works have considered the notion of a curriculum [Halff 88, Lesgold 88, McCalla 90, Nkambou 96, Nkambou et al 98].

Halff [Halff 88] considered that the goal of curriculum in ITS is to structure a representation of instructional materials and, based on the representation, to help select and order training events. Halff positioned a curriculum toward the representation of didactic materials and the selection and ordering of tutoring events based on these materials. It did not relate these materials to outcomes of tutoring and teaching strategies. McCalla [McCalla 90] gave a more cognitive definition of curriculum: a curriculum in ITS represents the selection and the ordering of knowledge in order to achieve the goals of tutoring that is appropriate for current context and current learner. This definition also indicates that a curriculum should be flexible, evolving and adaptive to the needs of learners and of the instruction development. Obviously, this definition focuses on the dynamic and more abstract aspects. Nkambou [Nkambou 96] defined a curriculum as a structured representation including the capabilities of Gagné as well as the capabilities and the didactic resources contributing to the learning objectives. Nkambou has focused on the representation and organization of knowledge in a curriculum, rather than on dynamic aspects such as the selection and ordering of tutoring events from a curriculum.

Some other works, such as that of Wipond and Jones [Wipond & Jones 88] and that of Merrill [Merrill 91], define the construction environment for human teachers to develop teaching.

Though these definitions of curriculum emphasize different perspectives, no definition of curriculum deals with more visual perspective. We think that in order to get an efficient, effective and practical curriculum, a curriculum model has to tackle the visual representation and organization of all kinds of knowledge related to teaching, the selection and ordering of tutoring events for a group of goals, and the practical interactive environment. In order to achieve these objectives, the following hypotheses are essential.

4.2 Basic Hypotheses

In order to define a visual curriculum, the following essential issues should be dealt with first:

- Focus of curriculum
- Factors affecting usability
- Evolving ability
- Systematic Methodology

In this section we tackle these issues as four basic hypotheses.

Hypothesis 1: Knowledge (including the knowledge of content, of pedagogy, of cognitive and of student model) is the focus in a curriculum.

ITS is a derived branch from Artificial Intelligence (AI). Early intelligent systems (in 50~60s) put the stress of AI on reasoning and control. Though these systems achieved success to a certain extent, but for lack of

knowledge, had difficulties when trying to move out the laboratories. In particular, the experience of the ambitious 10-year plan for intelligent fifth generation computers in Japan compels the researchers of AI to think about what is the exact focus of an AI system. At present, most AI researchers think that knowledge is the key of an AI system. A powerful intelligent system must have as much knowledge as possible.

An intelligent tutoring system includes more types of knowledge than usual AI systems. Unlike most AI systems dealing with just the domain knowledge and the expertise, an ITS concerns not only domain knowledge, but also didactic materials, pedagogic expertise, cognitive expertise, the student model and software expertise. The representation, organization, manipulation and management of all these types of knowledge are much more complicated than in general AI systems.

Hypothesis 2: the Interaction and navigation environment to a great extent determines the usability of an ITS.

According to Murray [Murray 96, 98], the usability of an ITS contains two aspects: learnability and productivity. Learnability is how easy a system is to learn to use it. Productivity is how quickly a curriculum author can enter information and produce tutoring systems. Learnability and productivity are often odds, since a system that is designed to be picked up quickly by novices may not provide the powerful features that experienced users need to efficiently produce large systems.

Classical ITS consists of three major components: the expertise in the domain to teach, the expertise of teaching, and the ability to infer something abound what students know or feel. Many ITS are developed based on the three components. However in order to develop useful systems for business or organizations, many researchers have recognized the necessity of the fourth components of ITS: the learning interactive environment [Wenger 87].

Unfortunately, the usability of Murray does not deal with another important feature, that is, the on-line navigation ability to users. The dynamic navigation to curriculum authors is an important factor affecting productivity because curriculum authors often get lost in a large network with hundreds even thousands nodes. Furthermore, the on-line navigation ability is even more essential for a learner because the learner is not familiar with the knowledge structure in the domain to teach at all. Therefore, we consider the interaction and navigation characteristics of an ITS to be of primary importance if the ITS is to be really useful.

Hypothesis 3: A flexible, evolving and adaptive curriculum must be open to its environment with ways of achieving a pertinent balance between generalization and specialization.

Early intelligent tutoring systems are domain specific because they originate from expert systems that simulate the experts' experience in some specific domains, so that, their knowledge structures and strategies are often domain specific and thus are difficult to be used for other domains. In order to reduce development costs and satisfy the needs of schools, enterprises and organizations, the general development environment for wide range of domains is necessary.

Obvious, such an environment should be flexible, evolving and adaptive for different domains, different teaching strategies and learning strategies.

Also, making the balance between generalization and specialization is difficult. If the frame is too generalized, more efforts and higher skills from users are required to develop their own system, and if the frame is too specialized, the flexibility, evolving and adaptability will be lost. One of our purposes is to explore the balance. For example, the evaluation and diagnosis to learners' reactions are ongoing research issues; our system attempts to support the activity delivery modules with different abilities for evaluation and diagnosis.

Hypothesis 4: The management of a curriculum should be systematic, that is, when to think about the representation and organization of knowledge, the feasibility of related processes in ITS should be also considered.

When we take into account a curriculum management model, the representation and organization of all kinds of knowledge is, of course, important. Meanwhile, we should also consider the feasibility for the course (a sub-network of domain network, which covers just a particular learner's goals) creation, and dynamically guiding the learning process.

4.3. Definition of a Visual Curriculum Management Model

Our visual curriculum management model absorbs three basic ideas from CREAM: capabilities, transition and teaching development process. The first idea is to use capabilities defined by Gagné as teaching outcomes that result in the appropriate teaching strategies for certain capability types. The second idea is that organizing transition networks is based on a natural atomic structure: from prerequisite capabilities to transition nodes and then from transition nodes to output capabilities. The last idea is the teaching development process, i.e. Building transition networks → creating courses → ordering tutoring actions → delivering tutoring. Based on these basic ideas, we define a visual curriculum management model with enriching visual ability, integrated components and simplified global transition structure.

In this work a *Visual Curriculum Management Model* is an architecture providing a rich graphical, interactive environment for the manipulation of knowledge relevant to a domain of learning to be developed and exploited by teachers and learners alike. The point of view is a radical departure from previous curriculum system definitions in which learners would exploit the structures only indirectly. Here the learner has access to the same representations of the domain knowledge as the author, but with varying angles of view. For instance getting views of what are the current learning goals, what has been learnt in the recent past, what possible learning alternatives there may be, etc. A visual curriculum has built-in some coherence checks that can pinpoint to authors and learners some current limitations of the learning system. This means that a Visual Curriculum is a structure in continual development. The underlying idea is to reduce the distance between the activities of teachers and of learners. By integrating into visual structures the teaching knowledge (domain, pedagogical, personal) and effective props for learning, the proposed system will adapt fast to needed changes.

- ***Definition*** A visual curriculum management model, known as ***VITCAM*** (Visual **I**nteractive **T**ransition, **C**ourse and **A**ctivity **M**anager), is a support architecture of computer-based tutoring systems that helps visually interactively organize and manage teaching outcomes, means and resources to help the authoring and learning processes. Concretely, ***VITCAM*** includes three sub-models:

- A capability transition model called ***VTRANS*** (Visual **TRANS**ition network model) that organizes and manages teaching outcomes and tutoring events (here didactic resources are viewed as elements of tutoring events); its output is a capability transition network called ***Tnet*** (Transition **net**work),
- A course creation model called ***VCOURSE*** (Visual **COURSE** model) that generates multiple alternative paths in a given capability transition network to cover a group of selected learning goals; its output is a vector called ***Cnet*** (Course **net**work) containing multiple alternative course networks, and
- A dynamic activity management model called ***VACT*** (Visual **ACT**ivity Model) that recommends individualized courses, identifies currently available tutoring activity sequences and dynamically manages the feedback from the activity delivery module; its output is a sequence of currently enable tutoring events called ***Anet*** (Activity **net**work).

Figure 4.1 shows these three models, their outputs and the relationships between them. A curriculum author can use the *VTRANS* model to visually build, organize and manage the capability transition network for a given domain. Based on the created capability transition network, a learner is able to use the *VCOURSE* model to generate multiple alternative courses, and then to activate the *VACT* model to visually create currently available activity sequences and dynamically manage learning process.

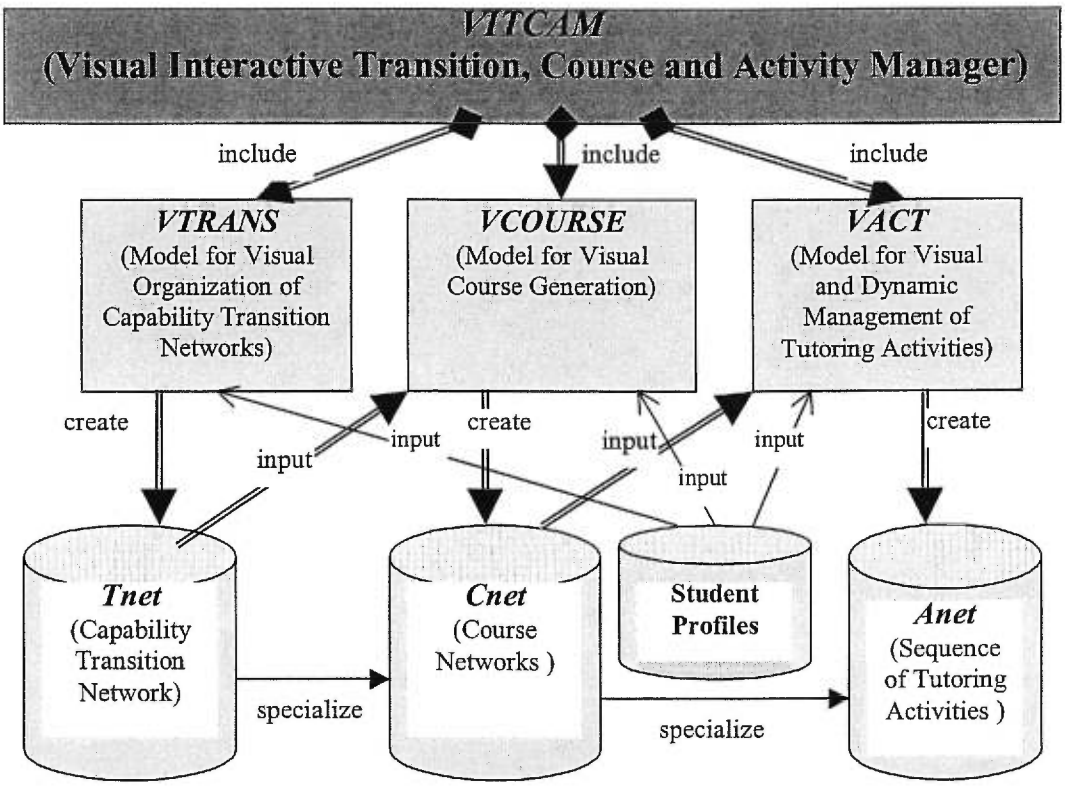


Figure 4.1 *VITCAM* and its Sub-Models

The *VITCAM* model uses or interacts with three kinds of basic objects: knowledge, interaction agents, and visual properties. Knowledge in *VITCAM* falls in three basic domains: the subject domain, the pedagogic domain and the cognitive domain. From the point of view of subject domain, the knowledge consists of learning outcomes, didactic resources and the relationships between learning outcomes. The pedagogic domain supplies the methodology of instructional design. The cognitive domain supports the methodology of instructional design by the cognitive mechanism of human learning.

Human agents are the principal interactive users of a curriculum. A curriculum author interacts with the system to visually create, to organize and to manage all kind of knowledge and tutoring events. A learner can set and achieve their learning goals with the system. The created individualized courses should reflect learner profiles. In our model, some essential student information contains initial states, current states, goal states, static attributes of the students (such as, preference, history); and dynamic attributes (such as the currently preferred strategies).

Visual properties involved in the definition of curriculum are distinct from other curriculum definitions. The main purpose using visual properties is to enhance the productivity of curriculum development and reduce learners and curriculum authors being lost in too large a structure. Colors, shapes, positions, sizes,

labels, images, and borders are some important visual signs for the purpose. A visual curriculum should supplies the following essential visualization abilities:

- Visually carrying much semantics about knowledge domain and the knowledge acquisition process,
- Providing different views for large-scale networks and using node aggregation properties to be able to visualize the curriculum global structure,
- Dynamically and visually guiding the authoring and learning processes.

The three models in *VITCAM* constitute a systematic methodology for curriculum management. *VTRANS* provides the means to organize tutoring events. *VCOURSE* supplies the statically individualized ability for learners. *VACT* model dynamically supports individualized learning process. We now briefly describe three models and the relationships between them. The details of *VTRANS* model are dealt with in chapter 5. The details of *VCOURSE* model are described in chapter 6. In the chapter 7, the details of *VACT* model are tackled.

The Capability Transition Network Model -*VTRANS*- integrates, organizes and manages various knowledge in a curriculum including domain capabilities, tutoring events supporting these capabilities, and the relationships between capabilities and tutoring events. Concretely, *VTRANS* model is able to

- support the teaching of the multiple-level capabilities and the aggregated capabilities,
- organize tutoring events with didactic resources and general strategies by combining Gagné's instructional event theory and Bloom's objective level theory,
- visualize capability transition networks with visually aggregated nodes and links to represent capabilities, capability levels, and integrated tutoring events supporting the capabilities,
- help curriculum authors to develop a practical curriculum rapidly,

Figure 4.2 shows a portion of the capability transition network for teaching HTML. The oval nodes in figure 4.2 are capability nodes, each of which contains several levels. The rectangle nodes in the figure are transition nodes that organize a group of tutoring events with two hierarchies: tutoring units and tutoring sub-units. Each tutoring unit can help learners achieve certain capability levels and consists of a sequence of tutoring sub-units, each of which corresponds to one of Gagné's instructional event. The arrows in the figure represent either the prerequisite relations from capability nodes to transition nodes, or the output relations from transition nodes to capability nodes. A *Tnet* structure implies AND/OR relationships between transition nodes and capability nodes, that is, the relationship among inputs of a transition node is an AND relationship and the relationship among outputs of a transition node is an OR relationship. The complete example for teaching HTML can be found in chapter 8.

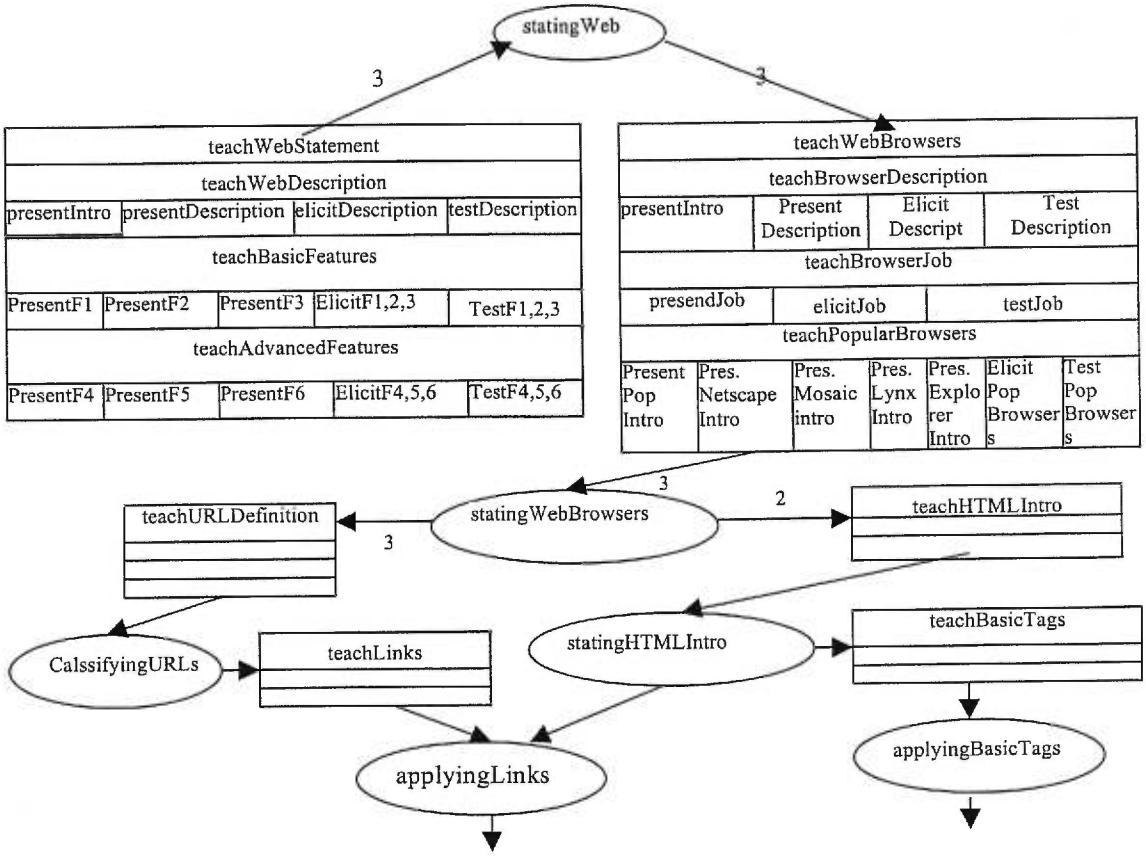


Figure 4.2 A portion of the capability transition network for teaching HTML

This visual curriculum definition also characterizes course generation and ordering tutoring events with two models respectively: *VCOURSE* and *VACT*. Their functionality includes

- visually setting known and learning goals corresponding to capability levels,
- creating multiple alternative paths called courses that cover the same group of learning goals; these alternative courses contain, besides the same group of goals, different optional capabilities, resources and efforts,
- recommending relative optimal courses to a particular student,
- dynamically sequencing available tutoring events to guide learning process,
- remedying learners' missing based on the diagnosis results of the activity delivery module in ITS, and
- visually guiding the learning process by state-driven reasoning.

The relationships within each model and between the two models are shown in Figure 4.3. Each model consists of a set of structures and a set of operation functions. We describe briefly the two models respectively as follows. Their details may be found in Chapter 6 and 7.

VCOURSE model contains a basic structure called *Cnet* and a collection of operations to create the *Cnet*. The capability transition network *Tnet* defined in *VTRANS* model is the input of the *VCOURSE* model. A *course* is defined as a minimal sub-network of *Tnet* that can cover learning goals of a particular learner. *Cnet* is the set of all possible courses. Because of the AND/OR structure of *Tnet*, for any non-empty finite set of learning goals, one or more course networks can be generated. The *VCOURSE* model defines the methodology to create various courses corresponding to the set of learning goals for a particular learner. Examples created by *VCOURSE* can be found in chapter 5.

VCOURSE supplies the following categories of operations:
--common visual operations for navigating learners,
--operations for setting known and goal capabilities and their levels, and
--operations for creating courses.

Common visual operations in *VCOURSE* model are some functions that define and support various visual manipulations. A learner can open a *Tnet* created by curriculum authors and visually set his/her known capabilities, their levels, and learning goals, by simple clicking. The operations to create courses are used for creating multiple alternative paths covering a group of learning goals. The decision switch in figure 4.3 is an internal function, which, in learning process, compares the current knowledge states and the goal knowledge states for the current learner. As a result, it determines whether to create new course networks, or to re-sequence the tutoring events, or to transfer next tutoring activity or to exit the learning process.

The dynamic activity management model *VACT* is responsible for recommending optimal courses and individualizing activity sequences.

By evaluating alternative courses with a measure model that will be defined in chapter 7, *VACT* recommends an optimal course to the learner, and then dynamically orders tutoring events to guide learning process. Each tutoring activity in the selected course owns a current state being either recommended, or enable or partial enable, or disable, or passed. The learner can select one of the recommended tutoring events as the current event to attend. In order to generalize our visual model of curriculum, activity delivery module in our model is an opening module. In order to combine the activity delivery module with different abilities, our model can effectively manage feedback of the learning results, which will be discussed in detail in chapter 7. Common visualization operations in *VACT* model are also defined to enhance usability.

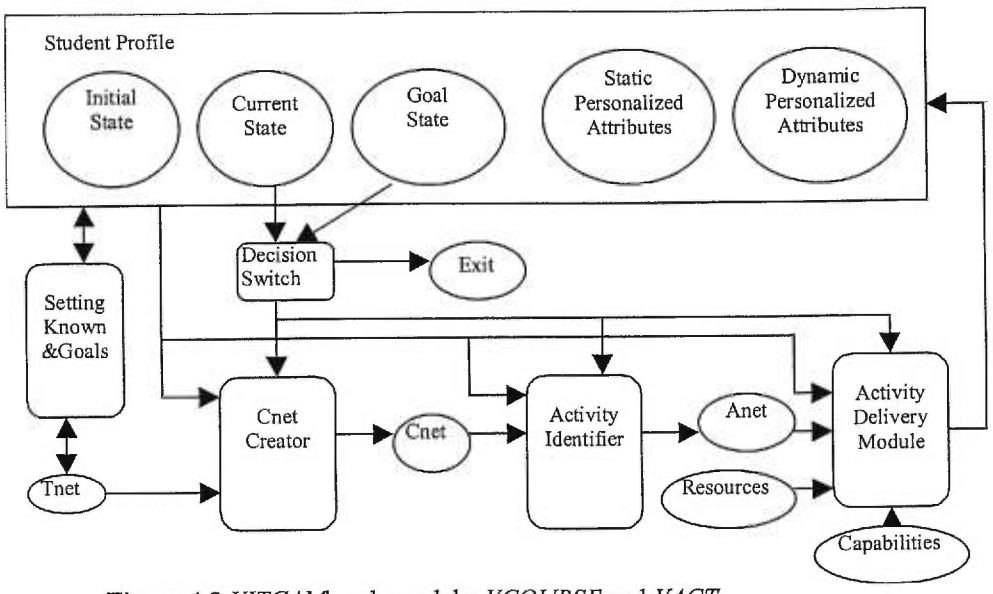


Figure 4.3 *VITCAM*'s sub-models: *VCOURSE* and *VACT*

Figure 4.3 shows all kinds of relationships and operations in the *VCOURSE* model and the *VACT* model. A learner, first, visually set his or her *known* capabilities and levels, the *goal* capabilities and their levels, based on the capability transition network created with *VTRANS* model. The learner then can activate the *Course Creator* to generate multiple alternative courses covering his/her learning goals. The learner can select one of created courses, or admit the recommended course by the system. The *Activity Identifier* orders the events and creates the recommended tutoring events in the current course. The *Activity Delivery Module*, then, be activated to deliver the recommended events and to update the learner's knowledge states. The *Decision Switch* compares the current knowledge states and the goal states of the learner to determine the next learning step: either to re-activate courses or to re-sequence the tutoring events, or to identify the next recommended activity, or exit the learning environment.

4.4 Summary

VITCAM is a visual curriculum model that can support both visual integrated representation and organization of all kinds of knowledge and tutoring events in curriculum, and selecting and ordering tutoring events to adapt current context and current students. This model absorbs the basic ideas of capability, transition and teaching development process in CREAM.

The model has four major characteristics: the central role of knowledge in ITS, the importance of interaction and navigation ability, the opening to adaptation to changing conditions of use and finally a coherent overall design.

VITCAM defines three important sub-models for visual curriculum management:

- *VTRANS,* a model that provides means to organize the capability transition networks for a given domain in which teaching outcomes, teaching events, didactic resources and general teaching strategies are integrated; *VTRANS* outputs a central structure called *Tnet*,
- *VCOURSE,* a model that creates multiple alternative courses (a course is a sub-network of *Tnet*) for teaching a group of selected learning goals; its output is a structure called *Cnet* that contains the created multiple alternative courses, and
- *VACT,* a model that dynamically orders tutoring events and manages learning process; its output is a vector called *Anet* that contains a sequence of currently available events.

The next three chapters describe in detail each one of the sub-models in *VITCAM*.

Chapter 5

VTRANS: A Capability Transition Network Model

This chapter presents the details of the capability transition network model *VTRANS*, which supports visual incremental integration, organization, management and testing of tutoring events. This model is the backbone of the overall *VITCAM* system; all other models are defined in reference to it.

In this model, domain knowledge is identified as atomic capabilities based on Gagné's instructional theory. Further, each capability node can be a complex capability with several explicitly incremental levels or an aggregation of several atomic capabilities. Tutoring events are structured to support an incremental acquisition of these capabilities.

The passage between capability nodes is modeled via transition nodes. A transition node organizes teaching events into two main levels, called respectively tutoring units and tutoring sub-units. A tutoring sub-unit is an instructional event, which can manipulate one or more groups of didactic resources. A tutoring unit consists of a sequence of sub-units, the transversal of which result in the learners' achievement of a certain capability level. A graph structure is proposed to integrate both the task of authoring and the task of providing support and guidance to learners.

We first define the capability transition model: *VTRANS* then explain in detail each component involved in the model. Last, the visual features of the *VTRANS* model are dealt with.

5.1 Definition of Capability Transition Network Model: *VTRANS*

VTRANS is a capability transition network model consisting of the following two parts:

1) a collection of structures including capabilities, resources, tutoring events and their associated networks, and
2) a set of operations implementing various functionality on these structures.

- Collections of Structures in *VTRANS* Model

We define the following five structures in *VTRANS* model:

- A capability set (given at least the prerequisite relationships between them)
- A resource base

- A transition node collection
- A visual property collection, and
- A central capability transition network: *Tnet*.

The *capability space,* the *resource base* and the *transition node collection* are basic information sources to organize domain capability transition networks. The *capability space* in *VTRANS* model covers all teaching outcomes in a domain and the relationships among them (prerequisite or aggregation relationship). Each capability node deals with one or more topics in a given subject domain. Each capability may be identified with several gradual levels, each of which may be selected as a learning goal. The *resource base* indicates the didactic resources supporting learning of all capabilities. Each resource is described with a group meta-attributes and the content of resource. A *transition node* organizes a group of tutoring events to support achieving certain capabilities. A transition node is an ordered set of two hierarchical tutoring events: tutoring units and sub-units that correspond to Bloom's objective levels and Gagné's instructional events respectively.

The *capability transition network: Tnet* is the central data structure in this model. *Tnet* (capability **T**ransition **net**work) is a structured organization of tutoring outcomes and tutoring events for a given subject domain. *Tnet* is organized as a set of connected AND/OR graphs in which transition nodes are AND nodes and capability nodes are OR nodes. The AND/OR structure means that acquiring a capability may be by means of different transition nodes, and activating a transition node requires that all its prerequisite capabilities be achieved at certain levels. The last structure in the set of structures is the set of visual properties, which consists of various visual properties both for building and management of curriculum and for learning process.

- Operations in *VTRANS* Model

All operations to manipulate various structures in *VTRANS* model fall into the following five kinds:

--Capability node operations
--Resources base operations
--Transition node operations
--View operations and
--Common visual operations

The *capability node operations*, the *resources base operations* and the *transition node operations* are used for visually constructing, organizing and managing capability nodes, the resources database, and the transition nodes. By using the transition node operations a curriculum author can organize tutoring events in a transition node. The *view operations* can create, organize, manage, test and navigate to users in capability

transition networks. Last, the *common visual operations* mark the visualization functionality for curriculum authors.

Figure 5.1 shows the relations between the collection of structures and the collection of operations in *VTRANS*. The subject content domain, the pedagogic domain, the cognitive domain and the tutoring strategies are inputs of the model. Based on these inputs, curriculum authors can use mechanisms provided by this model to organize capability nodes, transition nodes and resources. As soon as the separate resource groups, capability nodes and transition nodes are created, authors are able to use the approach supplied by the model to build and manage transition networks. Authors as pedagogues can see visually how particular students used the network, assess for a particular student the effectiveness of the strategies that were used, etc.). Learners will use the visual properties of *Tnet* to set their learning goals and assess their progress towards these goals.

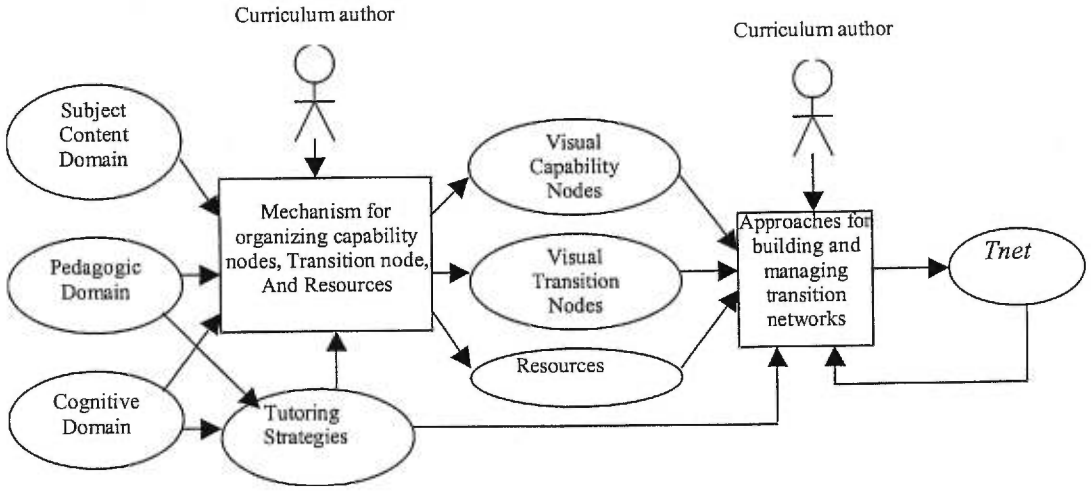


Figure 5.1 *VTRANS* Model

Though resources management is really a part of transition node management, they are so important as to need some specialized attention and organizations.

5.2 *Tnet*: A Central Capability Transition Network

Capability nodes and transition nodes are the two categories of nodes used to organize a curriculum. The capability nodes correspond to various possible levels learners may have mastered a concept, a skill, etc. Transition nodes are where the learning is taking place. This is a highly structured component that makes its possible to get a curriculum structure as rich (if not more) than earlier curriculum definitions [Nkambou, 96], while the overall network organization is much simpler and for this reason can now be used both for authoring and for teaching.

- **Definition of *Tnet***

A capability transition network *Tnet* is a structure defined by the following four kinds of components:

-- A collection of capability with one or more levels,

-- A collection of transition nodes organized based on Gagné's and Bloom's theories,

--A collection of relationships between capabilities and transition nodes, and

--Mechanism of combining capabilities and transition nodes.

A capability is a teaching outcome in a curriculum. We adopt the capability theory of Gagné to identify subjects to teach in a given domain. The distinct aspect in the definition is that it allows identifying a capability with one or more increment levels, or compounding multiple single-level capabilities into an aggregated capability with multiple levels. A curriculum author then can organize tutoring events to teach these levels respectively.

A transition node consists of two kinds of elements: tutoring units and sub-units. A sub-unit consists of a group of interaction acts refined by resource development or the activity delivery module (An activity delivery module may include certain functionality such as interaction creation, resource creation, resource selection, evaluation, diagnosis, etc). A tutoring unit consists of a sequence of sub-units. From the point of view of instructional theories, a tutoring unit relates to a level of objective defined by Bloom, and a sub-unit corresponds to an instructional event in Gagné's theory. From the point of view of capability acquisition, a tutoring unit contributes certain capability levels, and a sub-unit conducts sub-processed for achieving the levels. Didactic resources are not a kind of separated component in the definition, but necessary essential elements in sub-units.

We identify two types of different relationships between capability nodes and transition nodes: prerequisite relationship and output relationship. The relationship from a capability node to a transition node is called a prerequisite relationship, which indicates that in order to enable a transition node; all the capabilities connected to its input must be mastered at certain levels. The output relationship refers to the one from a transition node to a capability node, which means that after all tutoring events in a transition node are executed successfully, a learner should achieve the linked output capability at certain level. In this definition, a capability can be linked to multiple transition nodes and a transition node can also be connected to multiple capabilities. Such definition implies AND/OR relationships in a capability transition network. All capability nodes are OR nodes and all transition nodes are AND nodes. The upcoming sections describe in detail each element in *Tnet*.

5.3 Capability Nodes

The capability taxonomy of Gagné is essential in *VITCAM*. It sustains the teaching of the capabilities with several incremental levels and the capabilities that are compounded by multiple simple capabilities. In this section, we first introduce the classification of capabilities based on Gagné's theory. The capability levels are discussed in the section 5.3.3. Finally the representation of capability is depicted in the sections 5.3.4. Examples to identify multiple-level capabilities and aggregated capabilities can be found in the section 5.5. The visual features of capability nodes will be tackled in the section 5.6.

5.3.1 Capabilities

We think, the capability taxonomy of Gagné reflects the essence of teaching. For example, discriminating shapes of objects and applying Newton law both are identified as procedural knowledge in most ITS, while Gagné identifies them as two different intelligent skills. By using the procedural method, there is no way to teach them respectively by specific strategies. Obviously, the tutoring methods for teaching the two knowledge units are different. As a result, we adapt capability type defined by Gagné to identify domain knowledge.

5.3.2 Classification of Capabilities from Gagné

Based on the analysis of educational goals and mechanism of learning, Gagné identified [Gagné et al 92] five categories of human capabilities as teaching outcomes: intellectual skills, cognitive strategies, verbal information, motor skills and attitudes.

- Intellectual skills

Intellectual skills enable individuals to interact with their environment in terms of symbols or conceptualizations, learning an intellectual skill means learning how to do something of an intellectual sort. Generally, what is learned is called procedural knowledge.

Gagné identifies five sub-categories of intellectual skills, including discriminations, concrete concepts, defined concepts, rules, and higher order rules (problem-solving).
Discrimination is the capability of making different responses to stimuli that differ from each often along one or more physical dimensions, for example, discriminating a circle and a rectangle, distinguishing different colors, etc..

A *concrete concept* is a capability that makes it possible for an individual to identifying a stimulus as a member of a class having some characteristic in common, even though such stimulus may otherwise differ from each other markedly. Such concepts are called concrete because the human performance they require is the recognition of a concrete object. Examples of object properties are round, square, blue, three, smooth, curved, flat, and so on

The distinction between *discrimination* and a *concrete concept*; is easy to appreciate: the first is "responding to a difference"; the second is identifying something by name or other ways.

An individual is said to have learned a *defined concept* when he can demotions the meaning of some particular class of objects, for example alien, a citizen of a foreign country. A learner demonstrates the defined concept by identifying instances of concepts that are components of the definition and showing an instance of their relation to one another.

A *rule* has been learned when it is possible to say with confidence that the learner's performance has a hind of "regularity" on a variety of specific situations. Inother words, the learner shows that he is able to respond with a class of relationships among classes of objects and events. For example, "applying list tags in HTML" is a rule. "Applying Ohm's law $E = I*R$" is another rule. Obviously, processing the capability called a rule doesn't mean just to state it verbally, but to apply it to real instances.

High order rules (problem solving) are complex combinations of simpler rules. Moreover, it is often the case that these more complex or "high-order" rules are invented for the purpose of solving a practical problem or class of problems. The capability of problem solving is, naturally, a major aim of the educational process.

- Cognitive Strategies

A very special kind of intellectual skill, of particular importance to learning and thinking, is the cognitive strategy. A cognitive strategy is a control process, an internal process by which learners select and modify their ways of attending, learning, remembering and thinking [Gagné 85].Gagné identifies five sub-skills of cognitive strategies: rehearsal, elaboration, organizing, comprehension monitoring, and affective strategies.

- Verbal information

Verbal information is also called verbal knowledge; according to theory, it is stored as networks of propositions [Anderson 85, Gagné 85] that conform to the rules of language. Another name for it, intended

to emphasize the performance capability it implies, is declarative knowledge. There are three sub-categories of verbal information in the taxonomy of Gagné: labels, facts and organized knowledge.

Large bodies of interconnected facts, such as those pertaining to events of history or to categories of art, science, or literature, may also be learned and remembered. As is the case, with the learning of single fact, the networks of propositions that constitute the new knowledge become linked to the large proposition networks already existing in memory. Large bodies of knowledge are organized from smaller units so that they become meaningful wholes.

- Attitudes

Attitudes are complex human states that affect behavior toward people, things and events. For example, "I like classic music" is an attitude, and "I don't like to drive at night" is another attitude.

- Motor skills

Sequences of unitary motor responses are often combined into more complex performances called motor skills. Motor skills are learned capability that underline performances whose outcomes are reflected in the rapidity, accuracy, force, or smoothness of bodily movement. "Driving a car" is a typical example of motor skills.

An important feature of this classification is that Gagné identifies specific teaching strategies for each kind of capability. We use these categories as the foundation of organizing desired teaching outcomes in our *VITCAM* model.

5.3.3 Capability Levels

The capability theory of Gagné gives the general taxonomy of learning outcomes. In fact, many concrete capabilities may be further divided into several levels. For example, according to Gagné, "Web" is a defined concept and "stating Web" is a capability corresponding to the concept. This capability can be still divided into several levels: (1) stating the definition of Web, (2) stating the basic features of Web, (3) stating the advanced features of Web, and (4) identifying the instances of the concept "Web" in a group of given concepts.

Another example of capability levels is the Newton's law $F = m*a$ (Force equals to mass times acceleration). The law is a rule and its corresponding capability is "applying Newton's law $F = m*a$." We can identify this capability into the following levels: (1) stating the law, (2) applying the dimensions of the

three variables correctly, (3) applying the law on simple instances, (4) applying the law on complex instances, and (5) evaluating correctness of applying the law.

Bloom identifies six categories of objective levels for educational goals [Bloom 69,78], which are acquisition, comprehension, applying, analysis, synthesis, and evaluation. Though these objective levels do not directly associate the capabilities defined by Gagné, they reflect that the acquisition process of human knowledge contains gradual levels. We can associate these objective levels with the capabilities with multiple levels.

Merrill defined two mastering levels, *recite* and *verbalize*, for verbal information, and the other two levels, *posses* and *generative* for concepts [Merrill 93]. Klausmerier identified four levels for concepts, including *identify*, *recognize*, *classify* and *generalize* [Klausmerier 90]. Nkambou used Bloom's objective levels as the types of objective nodes [Nkambou 96].

There are two benefits of using explicit capability levels. One benefit of tackling with multiple-level capabilities is that it can aggregate in a node much information that otherwise would make the network structure too hard to manage. Another benefit is the relatively explicitly decomposing complex capabilities. An interesting issue in instructional design is how many capabilities should be identified in a given domain. We did not find research results on this issue. Through obviously in our framework this is ver important. For example, one author may identify 500 capabilities in the domain of teaching "Excel" and another author may identify only 300 capabilities for the same domain. This may be due to the presence of aggregated capabilities. If some capabilities can be aggregated into relatively large capabilities, the number of nodes in a given domain will be decreased, and furthermore the domain network structure covering the domain will be simplified, and the management of the whole system will be easier.

We thus assume that a capability can be organized into several incremental levels by the curriculum author. Further, a leaner may choose any assigned capability level as one of his learning goals. The system should also be able to support organizing tutoring events for achieving these capability levels.

5.3.4 Representation of Capabilities

In *Tnet* a capability refers to both a data structure for storing all detail information as a teaching outcome, and a visual component that the interaction agents (a curriculum author and a learner) may visually create, organize and manipulate.

As a data structure, a capability contains the following information: identification name, type, authoring states, learning states, set of capability levels, input relationships (links from transition nodes), output

relationships (links to transition nodes), and additional textual description. The type of a capability is based on Gagne's classification, such as a fact, a concrete concept, a defined concept, a rule, etc. The authoring states can help a curriculum author guide his/her authoring process. Due to limits or constraints, the curriculum author may have capability nodes partially unspecified; i.e. the capability levels will be just tags from the Gagne's classification. The learning states are information that is part of the student model and reflect the student's status with respect to this capability. These attributes are used for retrieving capabilities from a database, help curriculum authors navigate into the network, guiding the learners, and organizing the overall capability transition networks.

As a visual component, a capability node is a composite icon in which there is a major visual cell representing the overall information of the capability node, and one or more small visual cells representing capability levels. Every visual cell in the capability node is a manipulatable icon. The visual properties of a capability node include: start point, total width, total height, width and height of the overall visual cell, start points of level cells, width and height of level cells, border colors of all visual cells, shapes of each cell, and color of each cell. The operations of each capability node consist of (1) creating, (2) dragging, (3) adding links, (4) removing the node, (5) removing relevant links, (6) zooming in, (7) zooming out, (8) clicking a cell for editing detailed views.

5.4 Transition Nodes

In this section, we focus on the tutoring events in *VTRANS* model. We first introduce the increment mechanisms loosed Gagné's theory and Bloom's theory. Then the set of tutoring events— the transition node is defined. Tutoring units and sub-units are two kinds of components in transition nodes, which are described in section 5.4.3 and 5.4.4 respectively. The didactic resources that directly support tutoring events are characterized in the section 5.4.5.

5.4.1 Incremental Mechanisms of Knowledge Transition

Bloom and Gagné analyze teaching outcomes from different properties. Both reflect the incremental learning process.

As mentioned in previous sections, Bloom [Bloom 69, 78] defines six levels of objectives:

- acquisition,
- comprehension,
- application,
- analysis,
- synthesis, and

- evaluation.

From the point of view of educational goals, the six levels can associate with students studying at different levels. An elementary student is mainly to acquire basic knowledge, while a university student is mainly to learn the ability of analysis and synthesis. Also we can use these objective levels to the teaching of concrete capabilities.

From the point of view of the teaching of concrete capabilities, the six levels reflect that the transition of any capability is gradual from lower levels to high levels. However, it would be difficult for usual human teachers to identify them in concrete capabilities. Meanwhile, when to use these levels to capabilities, each of the six levels may be still divided into several levels. For example, "applying HTML" is a capability; we can divide the capability into three levels: (1) designing simple Web, (2) designing general Web, and (3) designing commercial Web. As a result, though we can inspire Bloom's idea about objective levels, these objective levels should be used flexibly.

Another limitation of Bloom's theory is that there is no concrete instruction strategy or event to support these objective levels. This makes it difficult to be used in computer based curriculum.

Gagné identified nine instruction events to transfer capabilities based on the analysis of cognitive process of human learning:

- gaining attention relating the reception of patterns of neural impulses,
- informing learner of the objective that activates a process of executive control,
- stimulating recall of prerequisite learning that retrieves prior learning to working memory,
- presenting the stimulus material that emphasizes features for selective perception,
- providing learning guidance that makes semantic coding;
- eliciting the performance that activates response organization,
- providing feed back about performance correctness that establishes reinforcement,
- assessing the performance that activates retrieval and makes reinforcement possible;
- enhancing retention and transfer that provides cues and strategies for retrieval

This theory provides both concrete steps of instructional actions and sub-objectives in teaching process, which is based on the cognitive mechanism of human learning.

However, the capability and instructional event theory of Gagné deal with neither identifying a capability as several incremental levels nor organizing instructional events for the capability levels.

If it were possible to combine Bloom's objective levels and Gagné's instructional events, it would be possible to support an incremental acquisition of capabilities and organize concrete tutoring event to achieve them. We are going to define a kind of transition node that makes this possible.

5.4.2 Definition of Transition Nodes

A *transition node* is an ordered set of tutoring events, which organizes a group of tutoring events to support incremental transition of a group of capability levels. Concretely, a transition node consists of a sequence of *tutoring units*. Each tutoring unit contains a group of tutoring events called *sub-units*. The transition of a tutoring unit typically will increase the level of some output capabilities. The identification of tutoring units is based on Bloom's objective level theory; that is, it reflects incremental capability levels. The idea of adopting tutoring sub-units is come from Gagné's instructional event theory, which refers to defining a sequence of concrete tutoring events to support the teaching to increase the levels of certain group of capabilities. The figure 5.2 shows the internal structure and external relationships of a transition node.

This structure combines three general tutoring strategies: Strategy A - successive refinement strategy in a given domain, Strategy B - incremental objective levels (Bloom's theory), and Strategy C - incremental instructional events according to Gagné's theory for certain capability levels. With this structure, a successive refinement strategy can be defined that organizes the acquisition of new capabilities in an orderly manner. This organization manner effectively utilizes the classification of both Gagné and Bloom. With reference to figure 5.2, the structuring of student actions in a transition node corresponds to depicting a path between cells of the grid. Other teaching strategies, for instance hints or question/answer, can be integrated into both didactic resource groups and into the activity delivery module in a tutoring system.

In *Tnet* a transition node also refers to both a data structure for storing all detail information as a group tutoring events; and a visual component that the interaction agents (a curriculum author and a learner) may visually create, organize and manipulate.

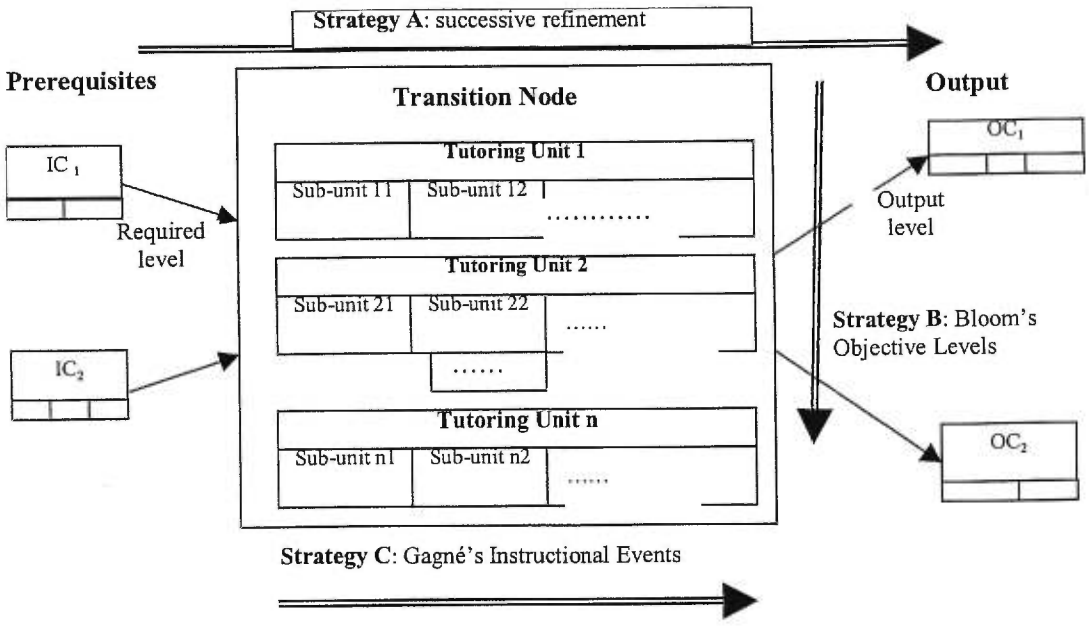


Figure 5.2 Internal structure and external relationships of a transition node.

As a data structure, a transition node consists of the following information: identification, name, scheme (preliminary information that is entitled into the node in order to facilitate the subsequent structuring of the capability transition network.), a collection of events, authoring states, learning states, a collection of input capabilities, a collection of output capabilities, description, and visual properties

Each transition node in *Tnet* contains a set of authoring states and a set of learning states. By this we mean that the nodes in *Tnet* have some ability of self-descriptive properties, such as supporting the management of self qualifies such as "incomplete", "complete", etc. These properties can give visual characteristics to the node in the visual curriculum, so that the curriculum authors can better assess the work done so far and what remains to be done. In curriculum development processes, each kind of authoring states may be either "not be developed", or "partly developed" or "fully developed". The overall state of the transition node is the combination of these states (see the section 5.6.5 for details). Learning states of a transition node include "recommended", "enable", "part enable", "disable" and "passed", which will be discussed in detail in chapter 6.

The input and output relationship of a transition node contains all prerequisite capabilities and required minimal mastered levels, as well as the maximum contributed capability levels.

As a visual component, a transition node is an aggregated icon in which, in addition to an overall visual cell and a scheme visual cell, there are multiple visual cells representing tutoring units of the transition node and each unit cell is allowed multiple small visual cells representing sub-units. Every visual cell in

the transition node is also a manipulatable icon. The visual parameters of a transition node include, start point, total width, total height, width and height of the overall visual cell, start points of level cells, width and height of level cells, border colors of all visual cells, shapes of each cell, and color of each cell. The visual operations of each transition node consist of (1) creating, (2) dragging, (3) adding links, (4) removing the node, (5) removing relevant links, (6) zooming in, (7) zooming out, (8) clicking any cell for editing detailed view.

An example of transition node is shown in figure 5.3.

<table>
<tr><th colspan="7">TeachWebBrowsers</th></tr>
<tr><td colspan="7">Definition of the Transition Node:
• Number of Actions: 3
1.TeachWebBrowserDescription, available media
2.TeachBrowserJob, available media
3.TeachPopBrowsers: Netscape, Mosaic, Lynx, & Explorer
• Prerequisite capability: statingWeb, level 3
• Output Capability: stating Web Browsers, level 3
• SuccessCriterion: >60%</td></tr>
<tr><td colspan="7">TeachWebBrowsers'Description</td></tr>
<tr><td colspan="2">PresentBrowser Introduction</td><td colspan="2">PresentBrowser Description</td><td colspan="2">ElicitBrowser description</td><td>TestBrowser description</td></tr>
<tr><td colspan="7">TeachBrowsers'Job</td></tr>
<tr><td colspan="3">presentJobDescription</td><td colspan="2">elicitJobDescription</td><td colspan="2">testJobDescription</td></tr>
<tr><td colspan="7">teachPopularBrowsers</td></tr>
<tr><td>Present intro</td><td>Present Netscape</td><td>Present Mosaic</td><td>Present Lynx</td><td>Present Explorer</td><td>Elicit Browsers</td><td>TestPopular Browsers</td></tr>
</table>

Figure 5.3 An example of transition node for teaching the concept: Web Browsers.

The transition node, "teachWebBrowsers" contains a scheme that define three tutoring units, a prerequisite capability and an output capability. These tutoring units are "teachBrowsers'Description", "teachBrowsers'Job" and "teachPopularBrowsers", each of which consists of a sequence of sub-units. For example, the tutoring unit "teachBrowsers'Job" contains three sub-units: "presentJobDescription", "êlicitJobDescription" and "testJobDescription". The following two sections characterize tutoring units and tutoring sub-units.

5.4.3 Tutoring Units

A tutoring unit in a transition node corresponds to and doesn't limit to one of Bloom's objective levels. Traversing a tutoring unit successfully by a learner means the learner has acquired certain levels of a group of output capabilities. A tutoring unit is both a data structure and a visual component in *Tnet*.

As a data structure, a tutoring unit contains the following attributes: identification, name, runner, sub-unit collection, authoring states, learning states, description, and visual properties. Most attributes are similar to that in a transition node except that they are specific for tutoring units. Here a runner in tutoring unit is a program that can deliver all tutoring events in the tutoring unit. In Figure 5.3, "teachBrowsers'description" and "teachBrowsers'Job" and "teachPopularBrowsers" all they are examples of tutoring units.

As a visual component, a tutoring unit is a group of visual cells embedded in a transition node in which there are a large cell representing overall view of the tutoring unit and several small visual cells representing its sub-units. Visual arguments and operations in a tutoring unit are similar to that in transition nodes.

5.4.4 Tutoring Sub-Units

A tutoring sub-unit is a concrete tutoring event consisting of a group of interactive actions, which is comparable to one of Gagné's instructional events.

A sub-unit is a sub-process of acquiring certain levels of some capabilities. Examples of sub-unit include "presenting a definition", and "testing mastering degree to the definition".

A sub-unit's structure is described by the following attributes: identification, name, runner, resource groups, authoring states, learning states, description, and visual properties. The runner for a sub-unit is also a program to transfer a group of interactive actions or activate the interaction functionality in the activity delivery module. The resource groups may contain one or more alternative resource collections. Each group supplies organized materials or interaction creation functionality in the activity delivery module for interactive actions in the sub-unit. In learning process, but only a current group is active. When a learner encounters difficulties with the current resource group, the system can switch the current resource group to another to remedy the learner's lacks. If the activity delivery module can dynamically create interaction activities, the resources in a sub-unit can be dynamically linked or organized based on the current states of students. The authoring states of a sub-unit cover the development states of resource groups, the runner states, and the states of other attributes. The learning states contain "recommend", "enabled", "partly enabled", "disabled", and "passed" which are described in detail later. Some other attributes of a sub-unit are similar to that in tutoring units.

As a visual component, a sub-unit is a small visual cell following a tutoring unit in a transition node. The operations a sub-unit can provide include (1) editing sub-unit attributes, (2) attaching resource groups to the sub-unit, and (3) testing the attached resource groups.

5.4.5 Resources

- **Definition and Representation of Resources**

In most tutoring systems, the term "resources" refers to the didactic materials. In order to get a relatively opening model, in *VITCAM*; a resource may be a didactic material, a resource selector, an interaction creator or a learning guidance program.

From the viewpoint of curriculum management, we classify resources as static resources and dynamic resources.

A static resource refers to various media pieces or their composition. In order to use resources inflexibly, we should consider storing more attributes, including identification, name, runner, type, media type, media format, data path, state, supported capabilities, cognitive strategies, size, inner text, description, visual properties. An identification of resource is a key in resource databases. For example a static resource may contain (1) an image for getting attention, and (2) a text for informing the learnt the objective.

Dynamic resources in our model consists of resource selector, dynamic interaction creator (a presenter of static resources, a demonstrator, a simulator, etc.) and learning guidance programs (such as diagnosis and evaluation programs). As a general-purpose curriculum development environment, *VITCAM* should be open for some intelligent and evolving aspects. For example, diagnosis of learners' errors is a difficult issue [Guo, 92]. We consider these programs as resources to enhance the flexibility of the system.

- **Resource Groups**

As mentioned above, a sub-unit contains several alternative resource groups. A resource group refers to a collection of resources that are organized by certain tutoring strategies. For example, in order to teach the proof process of a theorem, resources can be organized either by an induction strategy or a deduction strategy. We thus can get two alternative resource groups for the same teaching purpose.

In addition to the attributes defined in the resources, an important attribute of resource groups is the cognitive strategy used. When to create individualized courses or when to remedy learners' misconceptions, these resource groups serve as alternatives.

5.5 Organizing Tutoring Events to Support Capability Aggregation

As mentioned-above, *VTRANS* supports the teaching of multiple-level and aggregated capabilities. This section explains, by examples, how to identify multiple-level capabilities and aggregated capabilities and

how to organize tutoring events for teaching these capabilities. Usually, some complex capabilities for instance a rule or a high-order rule (such as a physical law or a process of problem solving) can be identified as a multiple-level capability. Some simple capabilities such as a fact or a concrete concept can be aggregated into a relevant large capability.

5.5.1 Organizing Events to Impact Multiple-Level Capabilities

The section explains, with some examples, how to use *VTRANS* model to organize tutoring events for achieving multiple-level capabilities.

Example 5.1 "Web" is a defined concept and "Classifying Web" is a capability corresponding to the concept in the meaning of Gagné. If an individual achieves the capability, he or she should be able to classify instances of the concept from a variety of concepts. We can identify the capability into two increment levels:

- "stating Web definition" and
- " Identifying instances of Web concept".

The two levels are incremental. In order to understand the Web definition, a learner should be able to state its definition first. However, if the learner can just state Web's definition, we cannot confirm whether he/she really understands the definition. If the learner can identify the instances of the concept "Web" from a variety of instances of concepts, he/she understands the definition.

After the capability's levels are identified, the curriculum author can organize tutoring events integrated into a transition node to support the teaching of the two capability levels respectively. Figure 5.4 shows a transition node that supports the achievement of the two capability levels. This transition node include two tutoring units called "teachWebStatement" and "teachWebIdentification" that support the capability level "statingWebDefinition" and "identifyingWebInstances" respectively. Each tutoring unit consists of a sub-unit sequence, for instance, the tutoring unit "teachWebStatement" consists of four sub-units: "gain attention", "present Web definition", "exercise Web definition" and "test Web definition". Each sub-unit contains a group of interaction acts with resources. For example, in order to form the sub-unit "gain attention", we may use a group of resources consisting of an animation that attract learners' attention and a paragraph of text that state the objective of this tutoring unit. An alternative is that we can activate the related functionality in the activity delivery module (if it is available) to dynamically create interactions.

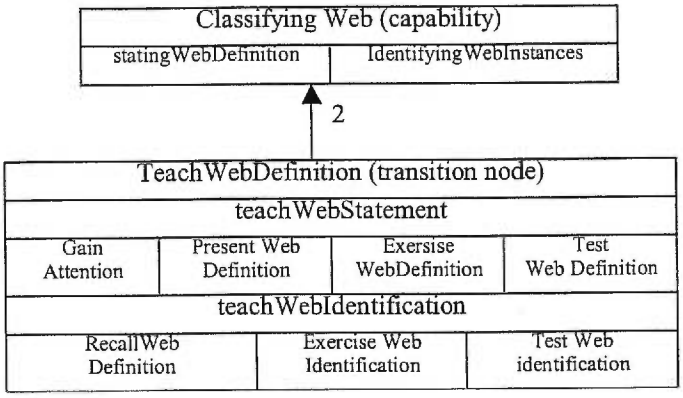


Figure 5.4 Organizing tutoring events to achieve a capability with multiple levels

Example 5.2 "Applying Newton's law $F = m*a$ (Force equals to mass times acceleration)" is a capability or rule in the meaning of Gagné. We can identify the capability into the following five levels:

- Stating the law,
- Applying the dimensions of the three variables correctly,
- Applying the law to simple instances,
- Apply the law to complex instances, and
- Evaluating the correctness of applying the law.

These five levels are incremental, too. In order to apply the law, a learner should be able state the formula first, i.e., he or she should know the three concepts involved in the formula and the relationships between them. However, just reciting the formula is not the final purpose to achieve the capability. The learner should further understand the relationships between the dimensions of the three variables. For example, the dimensions of the concept "mass" include kilogram, gram, milligram, etc., and the dimensions of the concept "acceleration" have "km/(hr*hr), km/(minute*minute), km/(second*second), m/(second*second), etc. When to apply the formula, many learners often get wrong result because of the erroneous combination of these dimensions. If the learner can correctly understand the relationships between these dimensions, he or she understands the capability to a higher level than just reciting the formula. By continuing the analysis in the above way, finally, if the learner can correctly evaluate the correctness of applying the law, we say that the learner really mastered the capability.

We can now organize tutoring events to teach these levels (figure 5.5). In figure 5.5, we use five tutoring units respectively to support the corresponding five capability levels. A learner can select any level of the five levels as his learning goal. The system, then, identifies the necessary tutoring units to support the achievement of his goals. In this example, five tutoring units and sixteen sub-units are organized to support

the teaching of the law. Each sub-unit consists of a group of interactive acts that present teaching materials, receive learners' reply, and provide guidance.

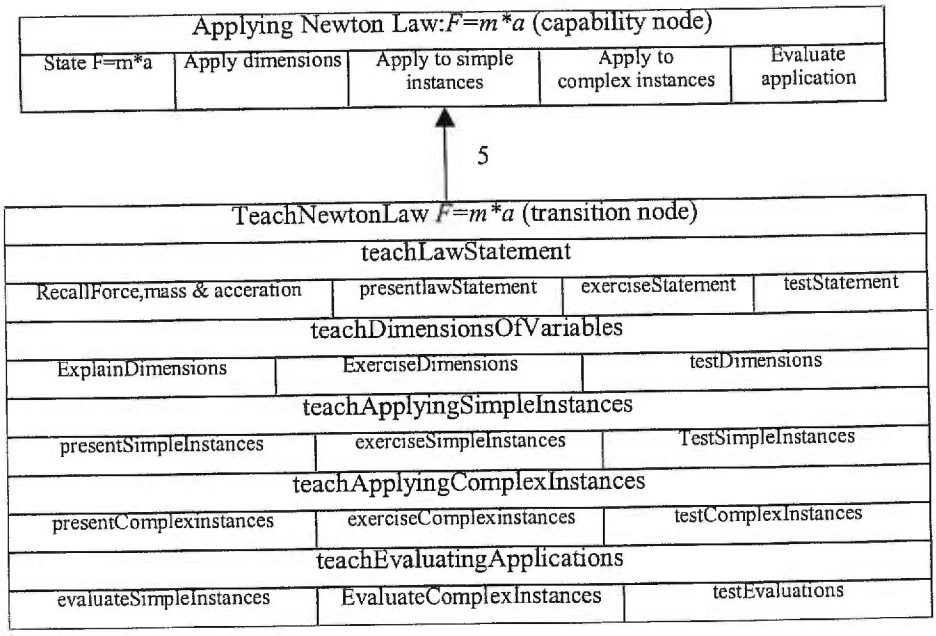


Figure 5.5 Organizing tutoring events to support acquisition of Newton's law

5.5.2 Integrating Events to Achieve Aggregated Capabilities

An interesting issue in instructional design is that how many capabilities should be identified in a given domain. There is too little research on this issue. In fact, this is very important for managing practical curriculum. For example, one author may identify 500 capabilities in the "Excel" domain, and another author might identify only 300 capabilities that are also able to cover the knowledge in"Excel", in which, some capabilities are aggregated capabilities. If some small capabilities can be nested into relative large capabilities, the number of nodes in a given domain will be decreased, and furthermore, the network structure covering the given domain will be simplified, and the management of the system will be easier.

VTRANS model can supply the ability to aggregate separate capabilities into relative large capability nodes. This is a powerful means to simplify domain network structures. The following examples explain the aggregation ability.

Example 5.3 aggregating capabilities about Web Browsers: "Definition of Web Browsers", "Web Browsers' Job" and "Popular Browsers" are three separated concepts surround the concept"Web". With existing authoring prototypes, the three capability nodes should be identified separately in its knowledge transition network. In our model, we can integrate the three separated capabilities into an aggregated capability called "Understanding Web Browsers" with three gradual levels:

- Identifying Web browsers' definition,
- Stating Web browsers' job, and
- Stating Popular Web browsers.

These three separated capabilities have different capability types. "Definition of Web browsers" is a defined concept; "Web browsers' job" and "Popular browsers" are two kinds of verbal information. Similarly, the three separated capabilities reflect an incremental sequence.

The following figure 5.6 shows a transition node that supports the acquisition of the aggregated capability.

In summary, identification of domain capabilities is a difficult task that needs human experts to complete. However, our model facilities the global structure of capability transition networks by the integration ability supporting multiple-level capabilities and composite capabilities.

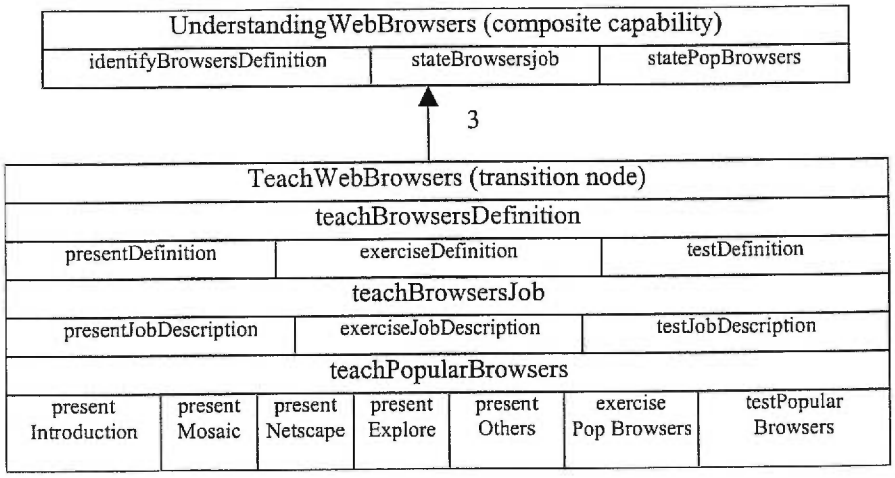


Figure 5.6 Organizing tutoring events to support an aggregated capability

5.5.3 Practical Consideration to Organize Transition Nodes

In order to organize tutoring events effectively in transition nodes, in this section, we tackle the associations among capabilities, objective levels, instructional events and resources. At first, we try to associate capabilities with objective levels. We, then, discuss the relationships among objective levels, instructional events and resources.

5.5.3.1 Associating Capabilities and Objective Levels

One advantage of Gagne's capability theory is that different capability types can use different teaching strategues to teach. Of these capabilities, some are simple such as a name and some others are complex such

as a rule or a high-order rule. A curriculum author can identify various objective levels for these capability types. Figure 5.7 shows an example to associate capability types to objective levels. In the figure, each column relates to an objective level and each row corresponds to a type of capabilities. The marked cells refer to that the type of capability in the related row may identify its objective levels as indicated in the corresponding columns. For example, a defined concept may contain three objective levels: stating, comprehension, and evaluation. The number 1,2,3... in each objective level means that the objective level can be further decomposed into some sub-objective levels.

	Stating 1,2,3...	Comprehension 1,2,3...	Application 1,2,3...	Analysis 1,2,3...	Synthesis 1,2,3...	Evaluation 1,2,3...
discriminations	✓		✓			✓
Concrete concepts	✓	✓				✓
Defined concepts	✓	✓				✓
rules	✓	✓	✓	✓	✓	✓
High-order rules	✓	✓	✓	✓	✓	✓
names	✓					✓
facts	✓					✓
Organized facts	✓	✓				✓
Cognitive strategies	✓	✓	✓	✓	✓	✓
attitudes	✓	✓	✓			✓

Figure 5.7 Example to associate capability types with objective levels

5.5.3.2 Associating Instructional Events, Objective Levels and Resources

In order to organize tutoring events in a transition node, we have to associate instructional events, objective levels and didactic resources. A curriculum author may use a table like that in figure 5.8, in which each column indicates an objective or capability level and each row relates to an instructional event. The marks inside middle cells represent didactic resource groups. For example, there are three objective levels in the figure that contains resource groups. We can organize these three tutoring units to correspond to the three capability levels. The column entitled "comprehension" includes three groups of resources: R11, R17 and R18. We can organize three sub-units for the second tutoring unit, each of which uses one group of resources. By using such method, the curriculum author can organize transition node intuitively. The table in the figure 5.8 is just an example. The curriculum author can flexibly identify objective levels and combine instructional events to meet various needs. If there is an intelligent activity delivery module available, the resource links and interactions in transition nodes may be dynamically generated. In such case, each sub-unit can be regarded as a sub-objective.

	Stating 1,2,3…	Comprehension 1,2,3…	Application 1,2,3…	Analysis 1,2,3…	Synthesis 1,2,3…	Evaluation 1,2,3…
Gain attention	R5					
Inform objective	R5					
Stimulate recall		R11				
Present materials	R7	R11	R14			
Provide guidance	R15		R19			
Elicit performance	R15	R17	R19			
Provide feedback	R15	R17				
Assess performance	R15	R17				
Enhance retention			R40			
Test	R27	R18	R37			

Figure 5.8 Example to organize a transition node

5.6 *VTRANS*-Based Visual Authoring

As a central structure of visual curriculum, *Tnet*'s visual ability enhances its usage in teaching and learning process. This section deals with visual authoring of curriculum with *VTRANS* model

5.6.1 Creating Visual Components for Capability Transition Networks

The first step of visual authoring is to be able to automatically generate various nodes and links used in the *Tnet* structure. This section describes the approaches for creating various nodes and links and for recognizing visual cells.

5.6.1.1 Mechanism for Creating Capability Icons

A capability contains its overall attributes and levels. We attempt to construct a capability node that can visually display both the capability and its levels. We define a visual capability node that consists of an overall visual cell representing the overall view of the capability and a group of small cells representing all levels of the capability. A relative large rectangle is used to stand for the overall cell and some variant shapes are used to represent the levels of the capability. The overall cells in all capabilities have the same shape, while different shapes of level cells distinguish the type of the capability from that of other capabilities. Figure 5.9 shows some example of capability icons.

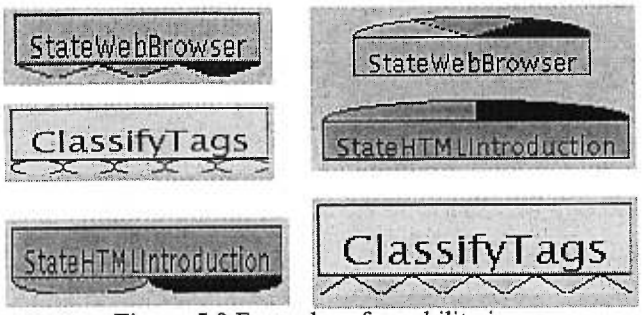


Figure 5.9 Examples of capability icons

There is no facility to directly create such defined visual nodes in existing programming languages and environments. We have to find mechanism to generate such compounded node. The Algorithm 5.1 in Appendix A describes a mechanism to create capability icons. The idea is that we draw a graphics first and then change all cells in the graphics into a Java GUI component.

5.6.1.2 Mechanism for Creating Transition Node Icons

The transition nodes are the most complicated visual components in *Tnet*. Each transition node contains four kinds of visual cells:

- An overall visual cell for the global view of the transition node;
- A scheme cell for describing preliminary information of the node;
- A group tutoring unit cells;
- Sub-unit cells involved in each tutoring unit.

Currently, there are two kinds of transition nodes in *Tnet*, one is used for inner nodes and another for leaf node. The shapes of sub-units distinguish the types of transition nodes. Figure 5.10 shows some examples of the two kinds of transition nodes.

The Algorithm 5.2 in Appendix A describes the mechanism of creating transition nodes. The idea is similar to that in capability node, but the calculation on positions and sizes of visual cells is more complex than that in capability node.

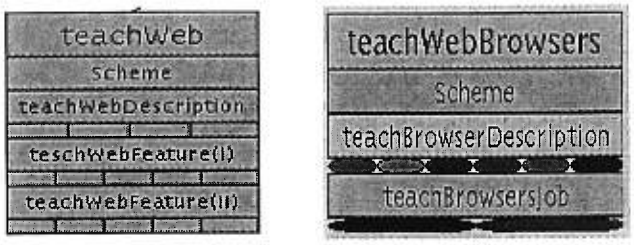


Figure 5.10 Two kinds of transition nodes

5.6.1.3 Mechanism for Creating Visual Links

A link is used for representing either a prerequisite relationship from a capability node to a transition node, or an output relationship from a transition node to a capability node. In *Tnet*, a link is both a graphic and a manipulatable visual component, which is displayed with line with an arrow at its ends. Each link has at least a source icon, a destination icon and a digit label. The digit label indicates either the required minimal mastering level in a link from a capability node to a transition node, or the contributed maximal levels in a link from a transition node to a capability. The algorithm to create links is described in the Algorithm 5.3 in Appendix A.

5.6.1.4 Recognition of Visual Cells

. Each cell in a node is a clickable element. We need an approach to recognize the visual cell clicked by a user.

The idea is that when a user click a point on screen, we check all cells in all icons on screen to see if any cell contains the clicked point. If there is a visual cell that contains the clicked point, the system activates related operations corresponding to the cell to do something. Otherwise, if there is no visual cell that contains the clicked point, then the blank area on the screen is clicked. This is the case to create a new node. The algorithm to recognize clicked visual cells is omitted.

5.6.2 Mechanism for Visual Organization

The visual organization in *Tnet* including dragging and moving nodes, deleting nodes, and deleting links.

5.6.2.1 Visually Dragging and Moving Nodes

The main steps to drag and move a node are:

- following the user's dragging to dynamically change visual parameters of the dragged icon,
- erasing the previous display from the screen,
- drawing the icon on current position,
- erasing all previous links of the icon and drawing these links to follow the dragging,
- storing the visual parameters of the finally displayed icon.

The algorithm to drag and move nodes is attached in the Algorithm 5.4 in Appendix A.

5.6.2.2 Visually Deleting Nodes

The following steps are followed to delete nodes:

- Erasing the clicked icon from the screen;
- Deleting all input links and output links of the clicked icon from the screen and the inner database;
- Deleting the related output links of the source icon of each input link, and the related input links of the destination icon of each output link;
- Deleting the clicked icon from inner database.

The corresponding algorithm 5.5 can be found in Appendix A:

5.6.2.3 Visually Deleting Links

The steps to delete links contain:

- Erasing the clicked link from the screen;
- Deleting the clicked link from its source icon;
- Deleting the clicked lick from its destination icon;
- Deleting the clicked lick from the inner database.

The corresponding algorithm 5.6 is attached in Appendix A.

5.6.2.4 Mechanism for Visual Manipulation

An important feature of visual curriculum is its strong ability of visual manipulation. That is, a curriculum author or a learner can click any visual cell in a node to do what he wants to do. In order to satisfy this requirement, we need to change all graphic cells into manipulatable icons.

The idea is that each visual cell is associated with an action handler that responses the clicking action of the visual cell. Currently, when a user creates a new node, the system automatically attach a related action handler to each visual cell in the node. These corresponding action handlers are called capability handlers, level handler, transition handler, scheme handler, unit handler, sub-unit handler and link handler respectively. These handlers are used for editing attributes, organizing resources to tutoring events, and testing tutoring events.

5.6.3 Visual Edition

Visual edition of *Tnet* consists of editing visual properties (such as adding, inserting and deleting visual cells) and inner attributes of cells.

1) Editing Visual Properties

For a capability node, adding a cell refers to simply add a capability level as its last level. In a transition node, there are tutoring unit cells and sub-unit cells. Adding a tutoring unit cell means to add a unit as the transition node's last unit, and adding a sub-unit cell implies a sub-unit cell is added to the related unit as its last sub-unit.

Inserting and deleting cells are more complicated than adding cells. For a capability icon, inserting or deleting a cell refers to inserting or deleting a capability level. For a transition node, inserting or deleting a cell means inserting or deleting either a tutoring unit or a sub-unit.

The steps to insert a cell are:

- Locate the cell to insert;
- Move all the successive cell forward a position;
- Initialize the newly inserted cell.

The Algorithm 5.7 in Appendix A describes the procedure of inserting units. Other algorithms such as deleting a unit, inserting a sub-unit and deleting a sub-unit are similar to this one.

2) Editing Detailed Information in Visual Cells

In order to create an environment for editing detailed information in each cell, a detailed view is defined for each cell, Currently, a dialog handler is provided for each kind of visual cell. They include the capability detail window, the level detail window, the transition detail window, the scheme detail window, the unit detail window, the sub unit detail window, and the link detail window. With each detail window, a curriculum author may enter or edit all detail information in the corresponding cell, and a learner may view the detail information of each visual cell. The figure 5.15 in the section 5.6.7 will give examples of detailed views.

5.6.4 Attaching Resource Groups to Tutoring Sub-Units and Testing Tutoring Actions

A sub-unit organizes a group of interactive acts. Each interactive act involves resource groups that are organized with certain tutoring strategies. Our model defines a database for resource groups with which a curriculum author can group resources into meaningful structures to organize sub-units. Within the detailed view of sub-units, the curriculum author can open the resource database groups, choose resource groups and attach them to sub-units. With the help of interactive act runners, these resource groups can be displayed to form interaction acts.

With the detailed views in tutoring units and sub-units, a curriculum author can test tutoring actions by simply clicking the corresponding sub-units in detailed views of tutoring units. If all necessary resources and interaction runners (i.e. programs to display resources and to handle interactions) work well, the related interaction session should carry out correctly.

5.6.5 Use of Visual Navigation by Curriculum Authors

Visual navigation in a large network needs to be easy and practical for both curriculum authors and learners. In this work we leave aside many questions relative to the navigation in large graph structures displayed on relatively small computer screens, as this has be dealt with by many other researchers in the past, such as that in Focus+Context [Lamping et al., 95]. In this work we concentrate on developing various ways of displaying relevant information dependent on the context in which the curriculum author or student learner is placed. In our approach it is easy to obtain different views of a structure like a *Tnet* to highlight contextual information on say, current learning goals, what are the alternatives to reach them, what are the kind of difficulties associated with each alternative, what has been seen by a student, at what levels, etc...

For the curriculum authors the information could be meta-information like the development progress of a capability node or a transition node, alternative views on the type of tutoring events that are available for students to reach their learning goals, etc. In the following those visual properties are presented in more detail.

We argued, in the previous chapter, that the usability of a curriculum should not focus only on the learnability and the productivity, as that in a usual application [Murray 98]. The learnability and the productivity are important for a tutoring system, but the on-line navigation ability is also important for such a complex knowledge curriculum system. Curriculum authors needs on-line navigation for development progress. In particular, learners prefer online navigation because they get better guidance in areas of knowledge new to them.

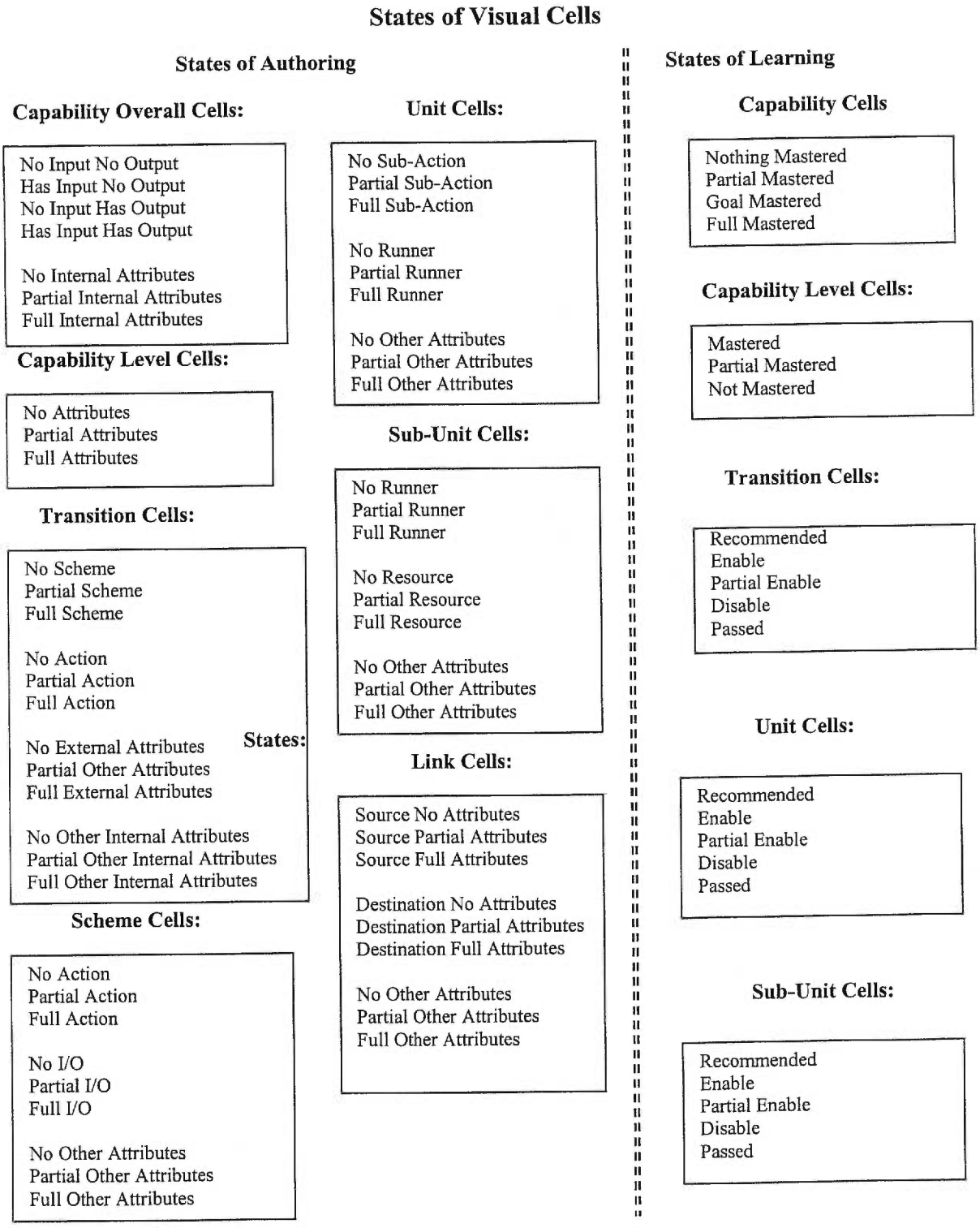


Figure 5.11 States of visual cells

In addition to visualizing the knowledge structure in a curriculum, the mechanisms of state-driven navigation, that now introduce, also supports real-time guidance of development and learning. We describe

the online navigation for curriculum authors in this section. The navigation steps available to learners will be presented in Chapter 6.

During the authoring process, the author often needs to know the development progress, that is, which nodes containe all necessary information, or which nodes lack necessary information. The navigation facilities in *Tnet* can display in real-time the current progress. This is implemented by means of various states a node can be in.

Based on the inner attributes and external relationships of nodes, we have defined several groups of states for each visual cell (as shown in figure 5.11).

A group of rules is defined for each kind of node to get a total state, then use a group of colors to map these states to visual display.

For capability nodes and their levels, the following rules define their states:

Rc1: if a capability has no input and output relationship, then the capability's relation state is capSole;
Rc2: if a capability has input relationship and no output relation, then the capability's relation state is capHasInNoOut;
Rc3: if a capability has no input relationship and has output relationships, then the capability's relationship state is capNoInHasOut;
Rc4: if a capability has both input and output relationships, then the capability's relation state is capHasInHasOut;
Rc5: if a capability has no internal attribute, then the capability's attribute state is capNoInnerAttribute;
Rc6: if a capability has part inner attributes, then the capability's attribute state is capPartInnerAttribute;
Rc7: if a transition node contains all inner attributes, then the capability's attribute state is capFullInnerAttribute;
Rc8: if a capability's state is capHasInHasOut & capFullInnerAttributes, then the capability's state is capFullAttributes;
Rc9: if a capability's state is capSole & capNoInnerAttribute, then the capability's state is capNoAttribute;
Rc10: if a capability is neither capFullAttributes nor capNoAttri, then the capability's state is capPartAttribute;
Rc11: if a capability level has no inner attribute, then the level's state isLevelNoAttri;
Rc12: if a capability level has part inner attributes, then the level's state is levelPartAttri;
Rc13: is a capability level has all inner attributes, then the level's state is levelFullAttri.

A transition node contains more rules for defining its visual cells' state. The states of a transition node are divided into three kinds: overall state, scheme state and unit state. Each kind of states contains some sub-kinds.

Rt1: if a transition node has no input and output relationship, then the transition node's relation state is transSole;
Rt2: if a transition node has input relationship and no output relation, then the transition node's relation state is transHasInNoOut;
Rt3: if a transition node has no input relationship and has output relationships, then the transition node's relationship state is transNoInHasOut;
Rt4: if a transition node has both input and output relationships, then the transition node's relation state is transHasInHasOut;

Rt5: if a transition node has no overall attribute, then the transition node's overall attribute state is transNoOverallAttribute;
Rt6: if a transition node has part overall attributes, then the transition node's overall attribute state is transPartOverallAttribute;
Rt7: if a transition node contains all overall attributes, then the transition node's overall attribute state is transFullOverallAttribute;

Rt8: if a scheme has no attribute, then the scheme's attribute state is schemeNoAttribute;
Rt9: if a scheme has part attributes, then the scheme's attribute state is schemePartAttribute;
Rt10: if a scheme contains all attributes, then the scheme's attribute state is schemeFullAttributes;

Rt11: if all sub-units' states are subUnitNoAttribute, then the unit's sub-unit state is unitNoSubUnitAttribute;
Rt12: if all sub-units' state are subUnitFullAttribute, then the unit's sub-unit state is unitFullSubUnitAttribute;
Rt13: if a unit is neither unitNoSubUnitAttribute nor unitFullSubUnitAttribute, then the unit's sub-unit state is unitPartSubUnitAttribute;

Rt14: if a unit has no runner, then the unit's sub-unit state is unitNoRunner;
Rt15: if a unit has part runner, then the unit's sub-unit state is unitParRunner;
Rt16: if a unit has all runners, then the unit's sub-unit state is unitFullRunner;

Rt17: if a unit has no other attribute, then the unit's attribute state is unitNoOtherAttribute;
Rt18: if a unit has part other attributes, then the unit's attribute state is unitPartOtherAttribute;
Rt19: if a unit has full other attributes, then the unit's attribute state is unitFullOtherAttribute;

Rt20: if a unit is both unitNoSubUnt, unitNoRunner and unitNoOtherAttribute, then the unit's state is unitNoAttribute;
Rt21: if a unit is both unitFullSubUnit, unitFullRunner and unitFullotherAttribute, then the unit's state is unitFullAttribute;
Rt22: if a unit is neither unitNoAttribute nor unitFullAttribute, then the unit's state is unitPartAttribute;

Rt23: if a sub-unit has no resource, then the sub-unit's resource state is subUnitNoResource;
Rt24: if a sub-unit has part resources, then the sub-unit's resource state is subUnitPartResource;
Rt25: if a sub-unit has full resources, then the sub-unit's resource state is subUnitFullResource;

Rt26: if a sub-unit has no runner, then the sub-unit's runner state is subUnitNoRunner;
Rt27: if a sub-unit has part runners, then the sub-unit's runner state is subUnitPartRunner;
Rt28: if a sub-unit has full runners, then the sub-unit's runner state is subUnitFullRunner;

Rt29: if a sub-unit has no other attribute, then the sub-unit's attribute state is subUnitNoOtherAttribute;
Rt30: if a sub-unit has part other attribute, then the sub-unit's attribute state is subUnitPartOtherAttribute;
Rt31: if a sub-unit has full other attribute, then the sub-unit's attribute state is subUnitFullOtherAttribute;

Rt32 if a transition's all units are unitNoAttribute, then the transition node's unit state is transNoUnitAttribute;
Rt33: if a transition node's all units are unitFullAttribute, then the transition node's unit state is transFullUnit;
Rt34: if a transition's unit state is neither transNoUnitAttribute nor transFullUnitAttribute, then the transition node's unit state is transPartUnit;

Rt35: if a transition node is transNoOverallAttribute, transNoSchemeAttribute, and transNoUnitAttribute, then the transition node's state is transNoAttribute;
Rt36: if a transition node is transFullOverallAttribute, transFullSchemeAttribute and transFullUnitAttribute, then the transition node's state is transFullAtrribute;
Rt37: if a transition node's state is neither transNoAttribute nor transFullAttribute, then the transtion node's state is transPartAttribute;

In summary, each visual cell contains three general states: no attribute, partly attributes and full attributes. We use three different colors to indicate the three states for all visual cells. Meanwhile, the curriculum author can get the details of these states with the detailed views of visual cells.

5.6.6 Visualization Large Networks

5.6.6.1 Alternative Views

Usually, the sizes of knowledge transition networks are much larger than actual screen size. In order to view the global structure of transition networks with arbitrary sizes, the system provides multiple visual views, including icon views, local views, and detailed views.

Definition We define three visual views to represent various structures of capability transition networks:

(1) An icon view is a graph containing visual nodes with the smallest size (labels with the smallest fonts);

(2). A local view is a graph containing nodes with normal sizes with which the displayed information of nodes will maintain the normal vision (labels with normal fonts);

(3). A detailed view is a graph with distorted node sizes in which the details of each node are shown.

These views may be used flexibly for various scales of networks according to the number of nodes in systems and the screen sizes. Local views and detailed views are the workspaces the users often use. When a user wants to see the global structure of transition networks, an icon view may be used. Figure 5.12 shows an example of icon view of the capability transition network for teaching HTML. Figure 5.13 and 5.14 show the local views that correspond to the icon view in figure 5.12. The example of detailed view can be found in the section 5.6.7.2.

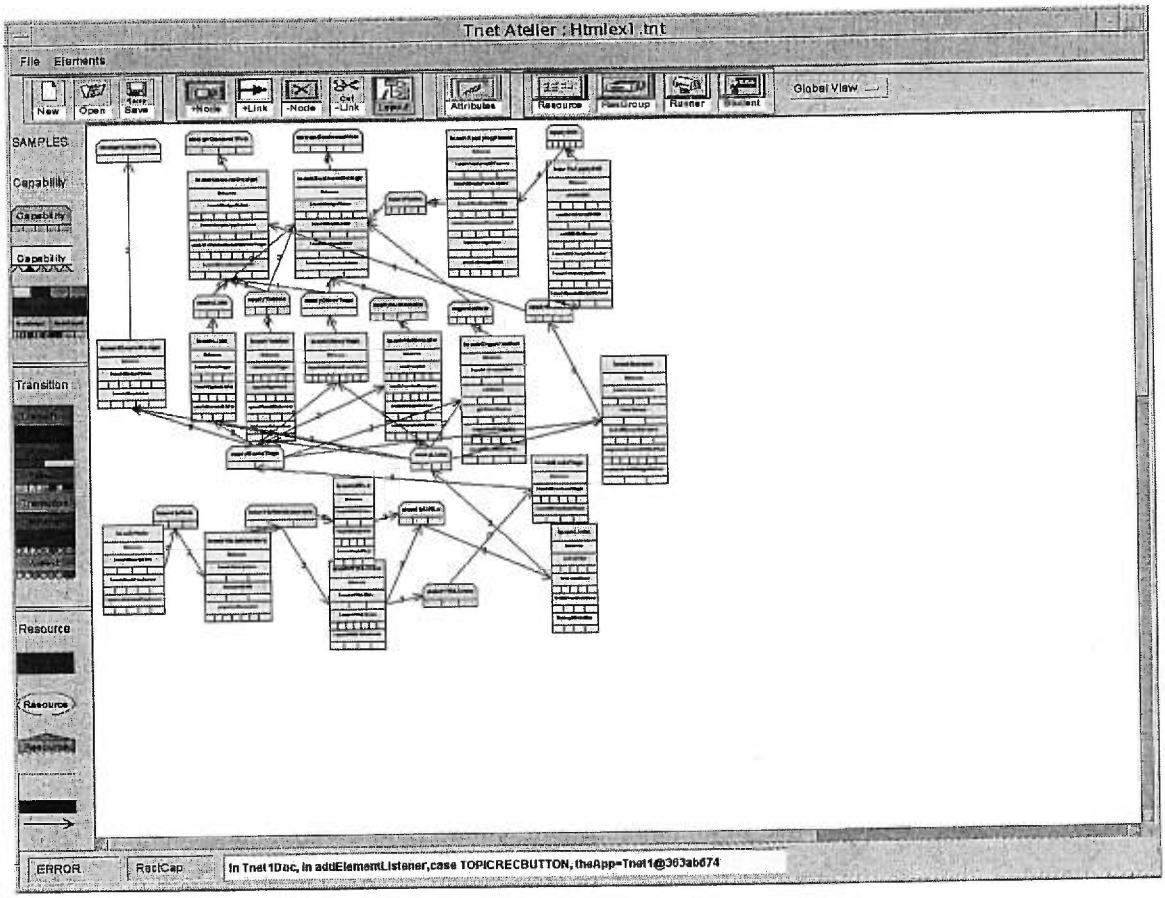


Figure 5.12 Icon view of HTML transition network

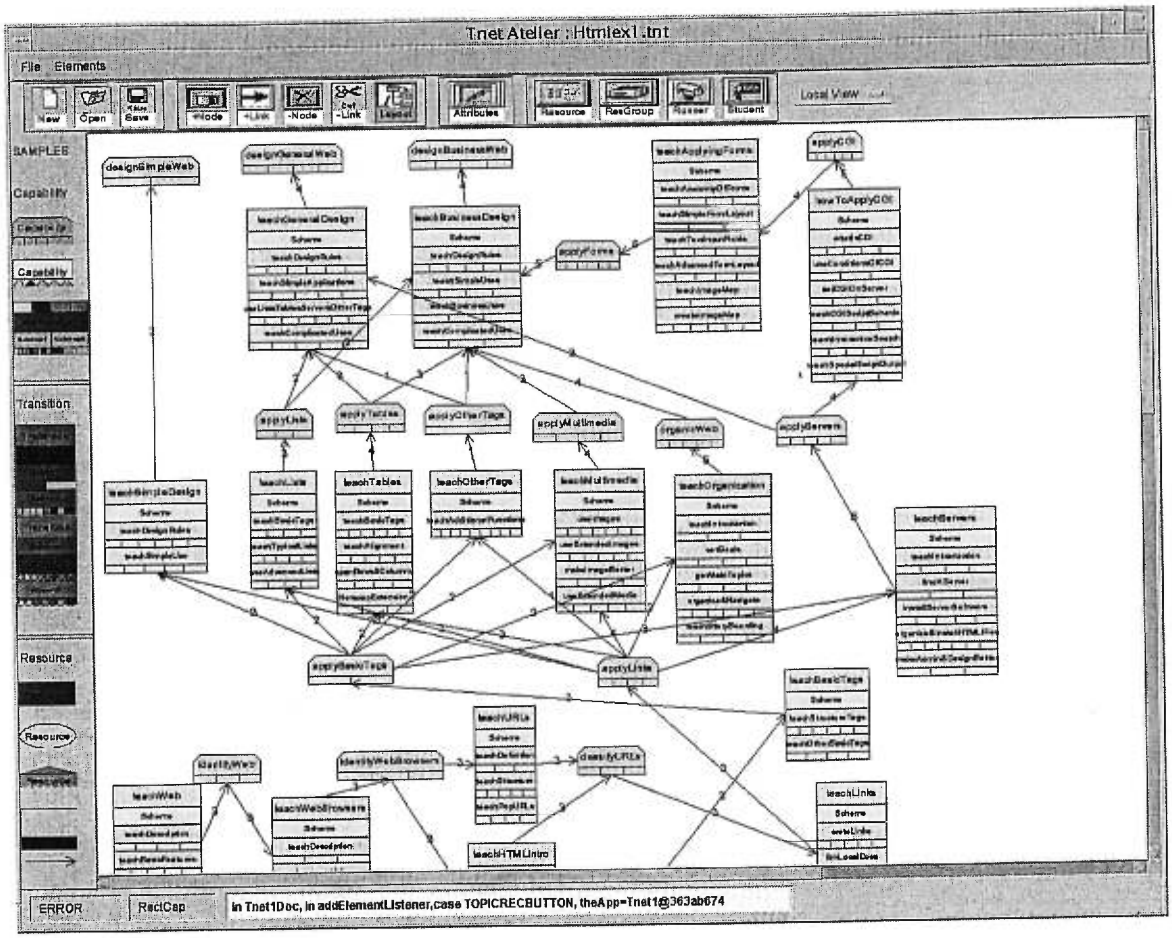


Figure 5.13 Local view of HTML transition network (1 of 2)

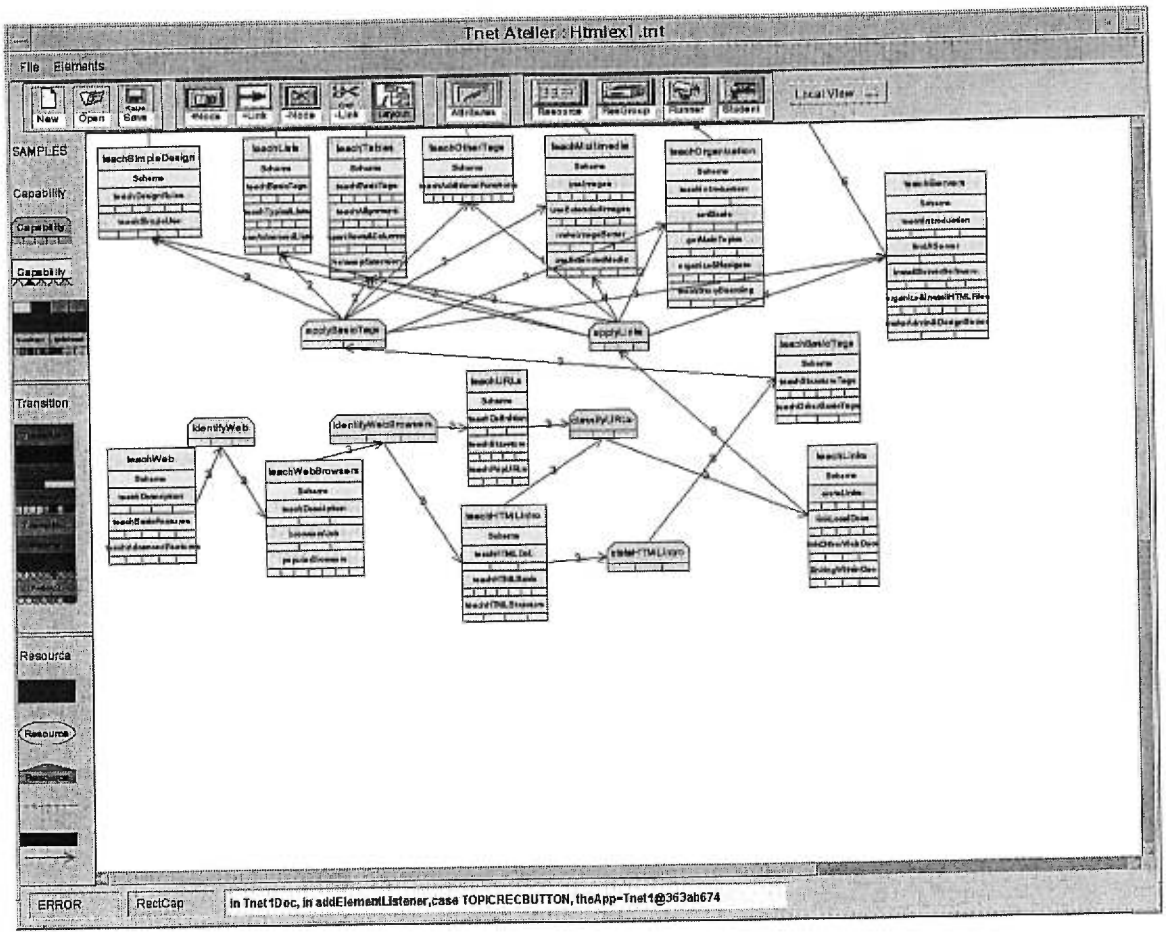


Figure 5.14 Local view of HTML transition network (2 of 2)

5.6.7 The Visual Authoring Process

The steps for visual authoring consist of identifying domain capabilities, developing resources, organizing tutoring events, building *Tnet*, organizing *Tnet*, modifying *Tnet*, and dynamic help for curriculum authors.

5.6.7.1 Identifying Domain Capabilities, Developing Resources and Defining Events

VITCAM-base prototype is a visual development environment, rather than an automation tool for building curriculum. *VITCAM* requires that a curriculum author be familiar with domain knowledge and some pedagogic knowledge, such as Gagné's classification of capabilities and instructional events and Bloom's six objective levels. Based on the knowledge, the curriculum author should identify all capabilities and their levels first, and then organize tutoring events to support the teaching of the identified capabilities. Meanwhile, the necessary resources should also be developed. The curriculum author can begin to build a curriculum with *VITCAM* prototype as soon as the above preparation is completed.

5.6.7.2 Visually Building, Updating and Organizing Capability Transition Networks

With *VITCAM* prototype, a curriculum author mainly works at two views: the local view and the detailed view. The local view contains all capability nodes, transition nodes and links, which form the domain capability transition network. The detailed view comprises all internal attributes and operations to carry out tutoring events. Usually, the curriculum author, first, builds the domain capability transition network with component templates provided by the system. Then, he or she enters and edits the details of all visual cells. With the visual navigation ability, the author can keep watch on the development progress.

Figure 5.15 shows an example of the local view and the detailed view for the capability-"statingWebBrowsers". Curriculum Authors may create, update and layout capability nodes directly by clicking and dragging. They can choose templates to represent particular types of capabilities such as facts, concepts, or rules. The difference of template shapes directly gives visual navigation information of related capability types.

The necessary information for teaching involved in capability nodes is integrated in the detailed view of each visual cell. A curriculum author can edit, update and view the information by clicking related visual cell. In figure 5.15, the window on the left is the detailed view of the overall characteristics of the capability node-"statingWebBrowsers", and the window on the right indicates the detailed view of the first level-"statingDescriptionOfWebBrowsers"-of the capability.

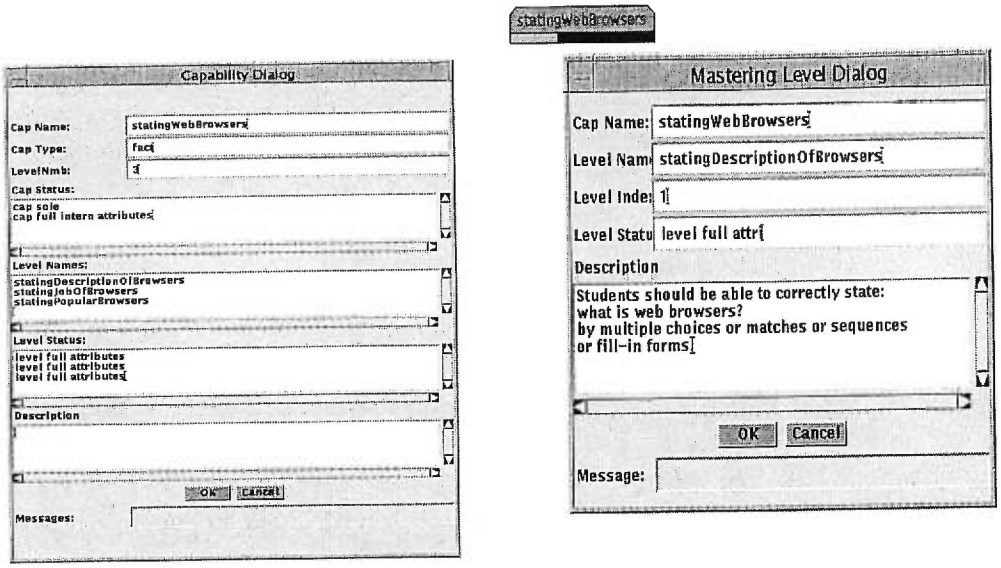


Figure 5.15 An example of visual individual capability, "statingWebBrowsers", which has three levels: "statingDescriptionOfWebBrowser", "statingJobOfBrowsers" and "statingPopularBrowsers".

Figure 5.16 shows an example of transition node, which consists of five big visual cells and fourteen small visual cells representing sub-units. The first two big visual cells labeled as "teachWebBrowsers" and "scheme" respectively, indicate the overall characteristics of the transition node and the preliminary information of tutoring events involved in the transition node. The rest three big visual cells are the sequence of tutoring units, i.e., "teachBrowserDescription", teachBrowserJob" and "teachPopularBrowsers". The small visual cells underneath each unit refer to sub-units involved in the unit. Clicking the related visual cell may get the detailed views for all internal attributes of these visual cells. The detailed views for the unit-"teachBrowserDescription"- is shown in figure 5.17 and the detailed view for the sixth tutoring sub-unit, "elicitPopBrowsers", of the unit-"teachPopularBrowsers"-is given in figure 5.18.

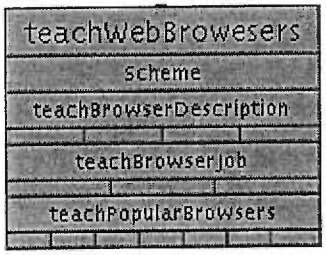


Figure 5.16. Example of visual transition node

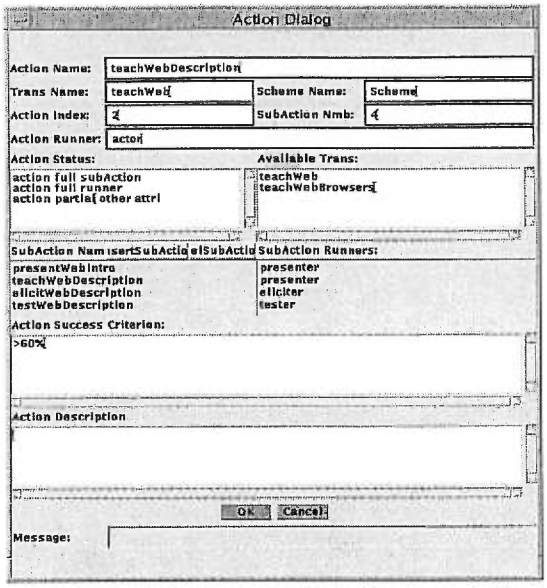


Figure 5.17. Detailed view of the unit "teachBrowserDescription"

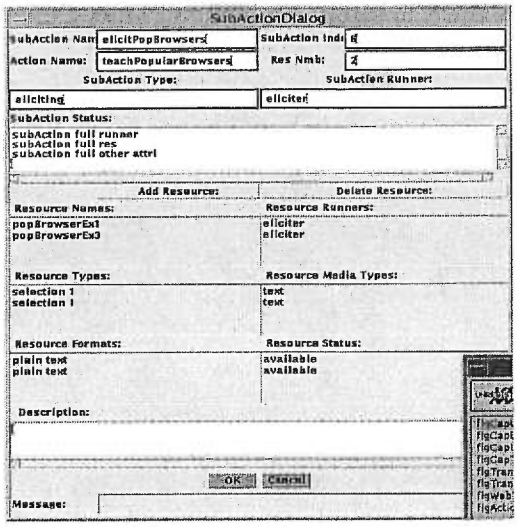


Figure 5.18. Detailed view of the sixth sub-unit "elicitPopBrowsers" in the unit "teachPopularBrowsers"

In order to help curriculum authors visually organize capability transition networks, a group of visual edition facilities are provided. Curriculum authors may visually move, remove, zoom in and zoom out visual nodes to get preferred visual structures (see figure 5.19 and 5.20).

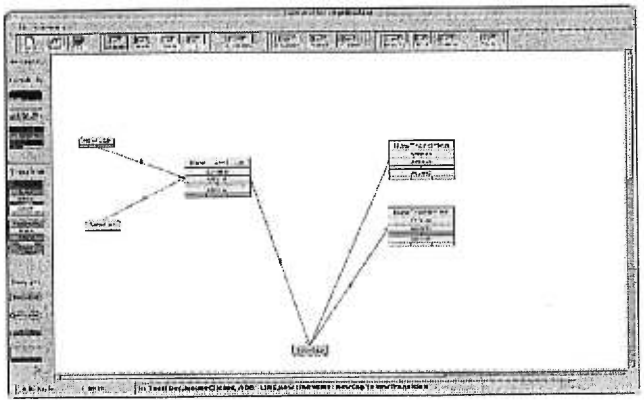


Figure 5.19 Arbitrary movement of nodes

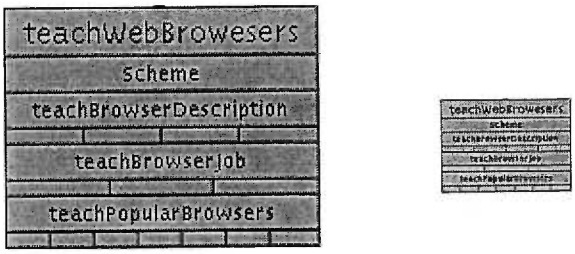


Figure 5.20 Zooming out

5.6.7.3 Dynamic Helps for Authoring

When building a curriculum, associations among capabilities, resources and transition nodes have been formed. By using these associations, a curriculum author can get dynamic helps, for instance,

- Finding all resources that support the teaching of a particular capability;
- Finding all capabilities that associate a particular resource;
- Finding all resources that have the same media type;
- Finding all resources that have the same type (e.g. multiple choice);
- Finding the distribution of a particular resource or a particular type;
- Finding all tutoring events associating with a particular resource;
- Finding all transition nodes that lack resources.

An interesting issue in dynamic support is that by adding a field indicating motivation, the transition nodes support the stages in the development of *Tnet*. For instance at the start of building a *Tnet*, a curriculum

author does not possibly have a full curriculum network in his head. He will first create some transition nodes and some capability nodes. The motivation for introducing a particular transition node may result from the availability of some teaching resources (audio, video, etc.). In order to help the curriculum author structure this network, we find it will clearly point to the curriculum author the kind of capability nodes that should be created and how to structure the network of capability and transition nodes. If the visual properties of the node can reflect these early categories by curriculum authors, the overall organization task of the *Tnet* will be greatly simplified.

5.7 Conclusion

VTRANS is the central sub-model in the visual curriculum model *VITCAM*. The model characterizes a central network structure: *Tnet* and various mechanisms and visual operations for building, organizing and managing capability transition networks. With the *Tnet* structure, the *VTRANS* model can

- support the teaching of the multiple-level capabilities and aggregated capabilities;
- organize tutoring events and general strategies by combining Gagné's instructional events and Bloom's objective levels;
- form the global capability transition networks by associating teaching outcomes (i.e. capabilities and their levels) and tutoring events;
- visualize the capability transition network in a given domain with visual composite nodes and links to represent capabilities, capability levels, and multiple level tutoring events for different learning process and sub-processes respectively;
- visual navigation to help a curriculum author develop a practical curriculum system quickly;
- associate didactic resource groups or the dynamic functionality in the activity delivery module to tutoring events.

VTRANS model provides strong abilities for visual authoring. These abilities include supporting multiple-level capabilities and aggregated capabilities, visually creating, organizing, manipulating capability transition networks, visually navigating to authoring process, and sustaining dynamic information review.

VTRANS model supports incremental representation and organization of capabilities with multiple levels,. Aggregating several capabilities into a large capability node is another feature of our model. Most real curricula are large and complex networks (e.g. hundreds even thousands nodes). To manage such a large network is a difficult task. The mechanism of capability aggregation simplifies the global network structure to a great extent.

VTRANS supports three kinds of general teaching strategies: successive refinement; objective refinement; and event driven. The strategy of successive refinement refers to that the model can organize learning outcomes according to domain knowledge structure. For all transition nodes, a curriculum author can define its prerequisite capabilities and output capabilities. In fact, the output capabilities are direct successive capabilities of prerequisite capabilities in a domain knowledge structure. The strategy of objective refinement means that the tutoring units in a transition node form an incremental sequence of objectives to help acquire certain capabilities. The strategy of event-driven corresponds to the sequence of sub-units in a tutoring unit, which are incremental instructional events defined by Gagné.

Other teaching strategies can also be integrated into resources, resource groups, dynamic interaction programs. A curriculum author can organize resource groups based on different strategies for instance induction and deduction strategies or give links to resources for the use of the activity delivery.

By visual operations provided by *VTRANS* model, a curriculum author can visually build, organize, and manage capability transition networks.

With the detailed views corresponding to each kind of visual cell, the curriculum author can visually edit internal attributes and operations of all visual cells. The author can also attach resource groups to tutoring sub-units and test tutoring events.

The visual navigation ability is a kind of distinct feature in *VTRANS*. In order to facility development process, each visual cell associates with a group of development states. With the progress of curriculum development, these states dynamically change to indicate whether the development of the cell is completed or not. It is very easy for the curriculum author to keep watch on the development progress.

In order to visualizing large networks, our model provides three views that can be used in the global, the local and the detailed stage.

In the development process of a curriculum, the internal associations among teaching outcomes, tutoring events and resources are automatically built. The system can fully use these associations to dynamically help the curriculum author get more useful information for further decision.

The coming two chapters will deal with two other sub-models: *VCOURSE* and *VACT* that are used for creating multiple alternative courses and dynamically managing learning process respectively. Meanwhile, these chapters also propose various approaches to visually navigate to learning process.

Chapter 6

VCOURSE: A Course Network Model

The capability transition network proposed in the previous chapter, *Tnet*, associates all the capabilities, tutoring events and relationships between them in a given subject domain. In the case of a particular student the *Tnet* network can be significantly simplified, taking into account capabilities the student has already acquired (and therefore may be hidden) and the fact that the student goals may involve just a small fraction of the set of transition nodes. So there is a need to be able to create sub-networks based on *Tnet*, which cover only the learning goals selected by the learner and other implied capabilities that are necessary to achieve the selected goals. We call each group of sub-networks a course. Due to the AND/OR relationship in a *Tnet*, probably, there exists multiple groups of sub-nets in *Tnet* that can cover the same group of learning goals. We call these sub-net groups alternative courses. Nkambou proposed a heuristic approach to automatically create courses [Nkambou et al 96, 98]. It attempts to create an optimal course based on the knowledge category (novice, immediate or advanced) of the current student and the strong/weak relationships of prerequisite or contribution between objective nodes and capability nodes . In *VCOURSE*, the basic idea for course generation is that a learner selects capability levels in a *Tnet* as the set of learning goals, the model then creates multiple alternative courses to cover this group of learning goals, and the learner can choose one of these alternative courses (or accept the recommendation of the system) as the current course.

In this chapter, we deal with how to create multiple alternative courses for a particular learner. First, in section 6.1, we discuss the possibility and benefits of creating multiple alternative courses. Second, based on the definition of course in section 6.2, we propose a network decomposition approach for creating alternative courses in section 6.3. Last, the visualization of networks of courses is presented in section 6.4.

6.1 Introduction

In this section, we first analyze the possibility and benefits of creating multiple alternative courses. Some problems involved in course generation, then, are discussed. Last, the main ideas to create multiple courses are dealt with. A course will exhibit a network structure of capability and transition nodes involved in guiding the student to the achievement of some capability levels chosen as learning goals. As the student is performing tasks associated with the transition nodes, the capabilities are automatically updated. So we can speak of capability node state transition.

6.1.1 Possibility and Benefits of Creating Multiple Alternative Courses

- **Benefits creating multiple alternative courses**

Tnet is an AND/OR graph, that is, an output capability may come from several alternative transition nodes and the activation of a transition node requires that all its prerequisite capabilities be mastered at certain levels. The alternative transition nodes to an output capability make it possible to create multiple alternative paths that support the teaching of the same output capability. As a result, for a given set of learning goals (capabilities), there are usually more than one group of sub-networks of the *Tnet*, which are called courses and cover the same set of learning goals. In general, the courses corresponding to the same set of learning goals contain different prerequisite capabilities, resources, media, cognitive strategies and time costs. In order to support individualized tutoring, we can define a group of criteria to evaluate and compare these courses and, then, recommend a corresponding optimal course to the learner.

However, on the one hand, it is difficult to define a standard that can exactly reflect what the learner wants or likes. On the other hand, learner preferences might change; for example, in a tutoring session, the learner may be interested in multiple choice questions, and in another tutoring session, he/she may prefer matching two groups of objects.

A good curriculum model should give users more chances to select what they like and a good deal of the control of the system. In the process of creating courses, the system should be able to present multiple alternative courses and to give the learner chances to compare and select courses. The system should also be able to rank the proposed courses according to their desirability to the learner.. Thus, one of the tasks of the curriculum model is to create multiple alternative courses for a group of learning goals.

- **Possibility of creating multiple alternative courses**

Travelling an AND/OR graph for all paths is of exponential complexity, a NP hard problem. There is no effective algorithm to decrease the complexity for a given AND/OR graph.

However, in our model, there is a possibility to decrease the real cost for travelling all paths. The idea is to simplify the given AND/OR graph by reducing the number of nodes. As mentioned in previous chapters, one benefit of the organization mechanisms of nodes in *Tnet* is that it can just simplify the overall network structure for a given domain. The capability aggregation provides the means to combine multiple relevant capabilities into a relative large capability. The transition nodes with a sequence of tutoring units provide the way to help acquire the aggregated capabilities.

By comparing with *CREAM* model, it is feasible for *VITCAM* model to create multiple alternative courses. For example, with the *CREAM* prototype in the project SAFARI, the capability space of the domain "Excel" contains about 500 identified capabilities. Although we don't adopt the "Excel" as an example, we identify the capabilities in the domain "HTML" with only 17 capabilities and 17 transition nodes. Obviously, it is possible to create multiple alternative courses on such a simple structure.

6.1.2. Main Steps to Create Multiple Alternative Courses

The main idea to create multiple courses is based on the division-and-conquer approach. This approach first marks all principal learning goals called head goals. A head goal is a selected goal capability whose successive capabilities do not contain any other selected goal. By eliminating redundant and replicate nodes, we obtain a group of connected sub-graphs, each of which contains a head capability as its root node. Each connected sub-graph is also an AND/OR graph and cover a part of leaning goals. We, then, create multiple alternative paths for each head capability based on the corresponding connected sub-graph. Any combination of alternative paths contributed by all connected sub-graphs is a course. The main steps for creating multiple alternative courses include:

Step 1: Identifying all implied known capabilities and their levels based on the initial known capabilities selected by a learner (An implied known capability is the one the learner does not click them as known capabilities, but he/she has selected one of the successive capabilities of the node as a known capability);

Step 2: removing redundant nodes and links (a redundant node or link is the one by eliminating which achievement of learning goals is not affected);

Step 3: finding all head goals (a head goal is the goal whose successive capabilities do not contain goals);

Step 4: recurrently creating all sub-networks of each head goal;

Step 5: creating the complete sub-networks of all goals' sub-networks (since all head goals are OR nodes, any sub-network of the head goal, likely, cannot cover all its sub-goals; these uncovered sub-goals should be found and combined to the sub-networks that miss them);

Step 6: combining the sub-networks of all head goals into courses.

In the next section, we first define the *VCOURSE* model and a course. The algorithms for creating multiple alternative courses are introduced in detail in section 6.3. The visualization of *Cnet* is presented in section 6.4.

6.2 Definition of the *VCOURSE* Model

VCOURSE is a model for creating multiple alternative courses that cover the same group of learning goals. A course is a sub-network of domain transition network, which covers just a group of selected goal capabilities. *Cnet* is a vector that includes multiple alternative courses for a group of selected learning goals.

Definition 6.1 Given a capability transition network *Tnet*; let *Vc* be the set of visual properties; *Fc* be the set of operation functions defined on *VCOURSE* model; *AL* be a learner agent; *K* be the set of known capability on *Tnet*; *G* be the set of learning goals, and *Cnet* be the set of all alternative courses, a course network model *VCOURSE* is defined as

***VCOURSE* = <*Tnet, Vc, Fc, AL, K, G, Cnet*>**

In the definition, *Tnet* is the input of *VCOURSE* model. All operations for setting known, setting goal capabilities, and creating alternative courses are based on the domain transition network *Tnet*. As the set of visual properties, *Vc* supplies essential needs of visual operations. *Fc* is the collection of operations on the model, which consists of two categories of operations: visual operations and internal operations. Visual operations are used for displaying transition networks, viewing details of all nodes and visual cells, setting known capabilities, setting goal capabilities and displaying alternative courses. Internal operations are mainly used for creating multiple alternative courses. Usually, the interactive agent *AL* is a learner that interacts with the system to select learning goals and create alternative courses. The set of known capabilities *K* and the set of goal *G* are intermediate results in the process of creating alternative courses. *Cnet*, the alternative course vector, is the output of *VCOURSE* model that result from applying operation functions in *Fc* on the inputs and intermediate results.

Definition 6.2 Each member of the vector *Cnet* is known as a course *Crs*:

Crs = <*Kc, G, Cc, Tc, Rct, Rtc, Vc*>

Where *Kc*: set of known capabilities, $Kc \in K$ in *Cnet*,

G: set of all learning goals established by *AL* on *Tnet*,

Cc: set of all capabilities in the course,

Tc: set of all transition nodes in the course,

Rct: all prerequisite relationships in the course (i.e. the relationships from capability nodes to transition nodes),

Rtc: all output relationships in the course (i.e. the relationships from transition nodes to capability nodes), and

Vc: set of visual properties.

The target of creating a course is to find all necessary capability nodes, transition nodes and the relationships among them, i.e. *Cc, Tc, Rct,* and *Rtc*, based on the known capabilities, goal capabilities and the operational functions in *VCOURSE* definition.

Figure 6.1 shows an example of creating courses. In the figure, the parameters in *Cnet* are:

K = {<C1,1> <C2,1> <C3,1>}
G = {<C2,2> <C4,3>}
C = {<C1,2> <C2, 3> <C3, 2> <C4,4>}
T = {<T1, 3> <T2, 2>, <T3, 2> <T4, 4> <T5, 2>}
Where *<Ci, j>* indicates the capability *Ci* at the *j-th* level. *<Ti, j>* means the transition node *Ti* including the *j* tutoring units.
The courses that meet the goals are shown in the figure 6.1 (b), (c), (d) and (e). The parameter values in these courses are:

Crs1:
Kcrs1 = {C1, 1> <C2, 1> <C4, 1>}
Ccrs1 = {C1, C2, C4}
Tcrs1={T1, T3}
Rct ={<C1 T3> <C2 T3>}
Rtc ={<T1 C1> <T1 C2> <T3 C4>}

Crs2:
Kcrs2={<C1 1> <C2 1> <C4 1>}
Gcrs2={<C2 2> <C4 3>}
Ccrs2={C1, C2, C4}
Tcrs2={T1, T2, T3}
Rct ={<C1 T3> <C2 T3>}
Rtc ={<T1, C1> <T2, C2> <T3, C4>}

Crs3:
Kcrs3={ <C2, 1> <C4, 1>}
Gcrs3={<C2, 2> <C4, 3>}
Ccrs3={C2, C4}
Tcrs3={T1, T4}
Rct ={ <C2 T4>}
Rtc ={<T1, C2> <T4, C4>}

Crs4:
Kcrs4={ <C2, 1> <C4, 1>}
Gcrs4={<C2, 2> <C4, 3>}
Ccrs4={C2, C4}
Tcrs4={T2, T4}
Rct={ <C2 T4>}
Rtc={<T2, C2> <T4, C4>}

The different courses contain different nodes, resources, tutoring units and sub-units. The learner may choose anyone he/she prefers in the four courses as his or her current course to learn.

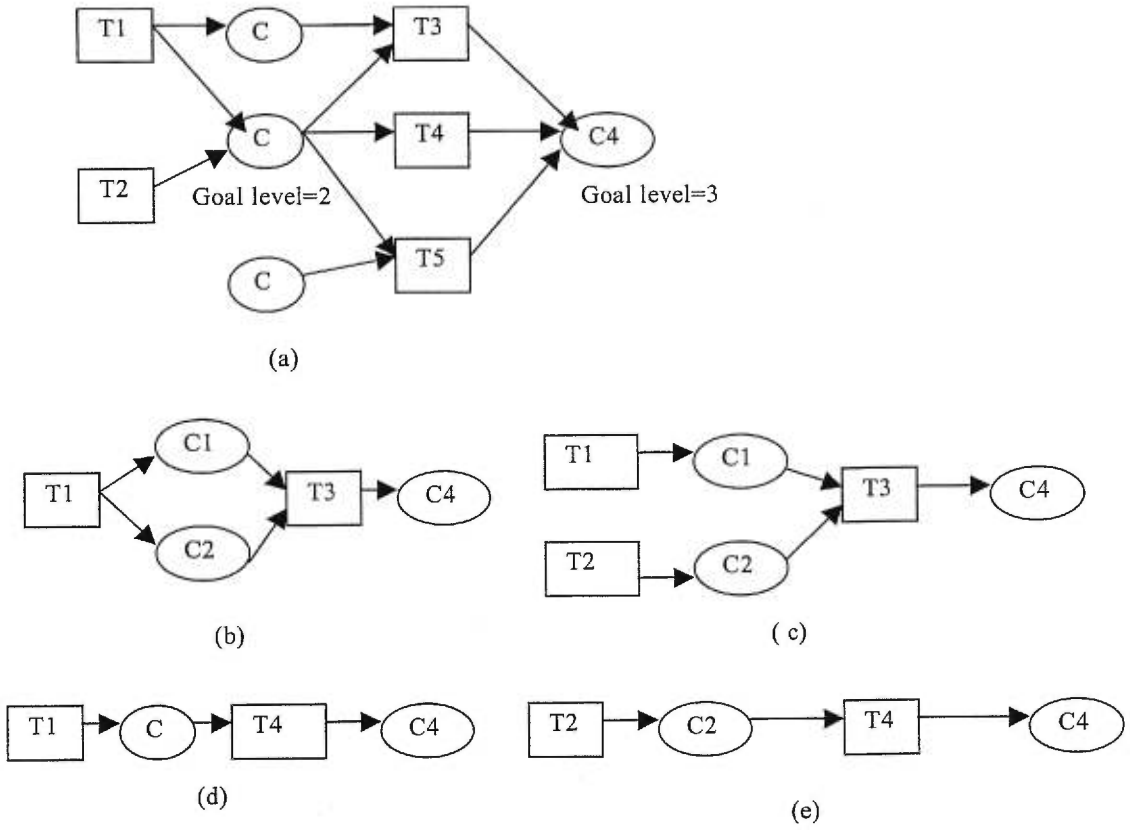


Figure 6.1 A courses Example

6.3 A Divide-and-Conquer Method for Creating Multiple Alternative Courses

The approach for creating multiple alternative courses is completely different from creating just one course. If our target were to generate one course that supports the achievement of a group of learning goal, the general search methods in AND/OR graphs would be able to borrowed. The problem is complex because we have to consider not only multiple alternative paths, but also the known capability levels and goal capability levels.

There are many alternative strategies of how to derive a course from a *Tnet*. One approach would be of the divide and assemble. For instance the system would identify all head-goals whose successive capability nodes do not contain any other goal. Each head-goal and all its predecessor goals form a goal collection. The system then would consider each of the goal collection, via backward propagation in *Tnet*, identify sub-networks of capability and transition nodes. After this decomposition stage, the various sub-networks would be superposed, eliminating redundancies and connecting elements that would appear disconnected

(by introducing elements from the rest of the *Tnet*). Another approach would consider all the goal nodes together, however, only the lower capability level of each goal node would be considered. By considering the lower level of the goal (capability) nodes, many nodes of the rest of the *Tnet* become redundant. One possibility to select among the alternative candidate nodes in order to achieve a course is to order the candidate nodes according to their contribution to satisfying the higher level goals. Once a sub-network has been obtained, the higher level goals are now considered and some repair is affected in order to keep the course (in the process of being built) consistent with these new goal levels. This goes on until the desired levels for the various goals are achieved. This gives a very general idea of possible strategies for building a course; also the implementation details are not trivial, but quite feasible. In this thesis we have considered the first strategy in more detail.

By following the steps described in the previous section 6.1, we, first, identify all implied known capabilities and their mastered levels based on the learner's selected known capabilities, and then find and remove all redundant nodes and links in *Tnet*. After some redundant nodes are removed, some nodes may become sole nodes. We should check these sole nodes or other invalid nodes and links. The next step is to divide all learning goals into several non-connected sub-graphs through finding head nodes whose successive capabilities do not contain other goals. We, then, create all sub-networks of each head goal, each of which covers some sub-goals of the goal group the head goal resides in. Some generated sub-networks of a head goal may be invalid; for example, a goal capability level is the fourth level, while the maximal contribution of the transition node supporting the capability to only the third level. We have to delete such invalid sub-networks. Owing to the existing sub-networks that do not cover all goals in the goal group, we have to let each sub-networks of each head goal cover all goals in the group the head goal resides in. As a result, each sub-network of each head goal becomes a complete sub-network that can cover all goals in the group the head goal resides in. Last, by combining all complete sub-networks of all head goals, all possible courses are gotten. The upcoming sections present each step in detail.

6.3.1 Creating Basic Relations for Courses Generation

Some basic relations between nodes are necessary through all steps of creating multiple alternative courses, which include the predecessor transition nodes of all capability nodes (allCapPreTransV), the predecessor capability nodes all transition nodes (AllTransPreCapV), the predecessor transition nodes of all transition nodes (allTransPreTransV), the predecessor capability nodes of all capability nodes (allCapPreCapV), the ancestor capability nodes of all capability nodes (allCapAncestCapV), the ancestor transition nodes of all transition nodes (allTransAncestTransV), the ancestor capability nodes of all transition nodes (allCapAncestorTransV), and the ancestor transition nodes of all capability nodes (allTransAncestCapV). These relations are defined as follows.

Definition 6.3 Let C be the set of all capability nodes in the *Cnet* after removing all redundancy and invalid elements, T the set of all transition nodes, and L the set of all links, the basic relations for creating courses are defined as follows:

(1) allCapPreTransV={<*Ci, Vt*> | *Ci* ∈*C, Vt*={*Tj* | *Tj* is the source of one input link of *Ci*}}
(2) allTransPreCapV={<*Ti, Vc*> | *Ti* ∈*T, Vc*={*Cj* | *Cj* is the source of one input link of *Ti*}}
(3) allCapPreCapV={<*Ci, Vc*> | *Ci* ∈*C, Vc*={ *Cj*| *Cj* ∈*C*, ∃*Tk* such that <*Tk, Cj*> ∈allTransPreCapV }}
(4) allTransPreTransV={<*Ti, Vt*> | *Ti* ∈*T, Vt*={ *Tj* | *Tj* ∈*T*, ∃*Ck* ∈*C* such that <*Ti, Ck*> ∈allTransPreTransV and ∃<*Ck, Tj*> ∈allCapPreTransV}
(5) allCapAncestCapV = allCapPreCapV$^+$ (transition closure of allCapPreCapV)
(6) allTransAncestTransV = allTransPreTransV$^+$ (transition closure of allTransPreTransV)
(7) allCapAncestTransV = {<*Ci, Vt*>| *Ci* ∈*C, Vt* is a vector containing all *Tk* such that <*Ci, Tk*> ∈allCapPreTransV}
(8) allTransAncestCapV = {<*Ti, Vc*> | *Ti* ∈ *T, Vc* is a vector containing all *Ck* such that <*Ti, Ck*> ∈allTransPreCapV}

For each of the relations, we need an algorithm to create it. The Algorithm 6.1 and 6.2 in Appendix C gives two of them for creating allCapPreTransV and allCapAncestCapV respectively.

6.3.2 Identifying Implied Known Capabilities and Levels

An implied known capability is the one that is not selected by a learner as one of his or her known capabilities, but its successive capabilities have been selected as his or her known capabilities. As a practical curriculum model, *VITCAM* should not expect a learner to check all details in a domain before he or she can set learning goals. For example, "applying link tags" is a capability in the curriculum for teaching HTML. If a learner says that he or she knows the capability "applying link tags", the system should be able to infer that the learner has mastered prerequisite capabilities of the capability "applying link tags". The prerequisite capabilities (such as the capability "applying the <HTML> tags") of the capability "applying link tags" are implied known capabilities.

The mechanism to find an implied known capability is based on backtracks from the selected known capabilities. As soon as a capability is selected as a known capability, we assume the predecessor capabilities of the known capability are known, and the predecessor capabilities of the predecessor capabilities are also known, and so on.

6.3.3 Removing Redundant and Invalid Elements

We first define redundant nodes and links, then give the algorithm for removing redundancy.

Definition 6.4 If a capability node *C* or a transition node *T* satisfies one of the following conditions, then it is a redundancy for creating courses:

1) the learning state of *C* is *FULLY_MASTERED*;
2) the state of *T* is *PASSED*;
3) the node *C* has one or more successive transition nodes; *C* is not a goal and the mastered levels of *C* satisfy the requirements of all its successive transition nodes;
4) *C* has successive transition nodes, *C* is a goal, *C*'s goal level is mastered and *C* satisfies the requirements of all its successive transition nodes;
5) *C* is a root but is not a goal;
6) *C* has no successive transition node; *C* is a goal and the state of *C* is *GOAL_MASTERED*;
7) *C* has no successive transition node, and *C* is not a goal.

If the capability *C* has been completely mastered, the capability should not be contained in the new created courses. If a transition node is passed successfully by the student, the node is unnecessary for course generation. If a capability is not a learning goal, it has been mastered to a certain level and the level has been larger than the levels required by the capability's all successive transition nodes, and then the capability is redundant for course generation. If the capability is a learning goal, its goal level has been achieved, and the mastered levels of the capability have satisfied the requirement of all its successive transition nodes. If a capability has no successive transition node and the capability is not a goal, it is a redundancy. If a capability *C* has no successive transition node, *C* is a goal and the state of *C* is *GOAL_MASTERED*, then the capability *C* is unnecessary for creating a course.

After redundant nodes are removed, some invalid elements are caused, for example, a sole capability node. We define an invalid element as follows:

Definition 6.5. If a capability node *C* or a transition node *T* or a link *L* is one of the following cases, then the element *C* or *T* or *L* is an invalid element:

1) *C* has neither input nor output link,
2) *T* has no output capability node,
3) *L* has no source element,
4) *L* has no destination element,
5) If *L*'s destination element is a goal capability, the level of L is less then the goal level of the goal capability.

The algorithm for removing redundancy and invalid elements is shown in the Appendix C.

6.3.4 Find Head Goals and Identifying Implied Goals

This section attempts to divide all learning goals into several non-intersection sets, each of which includes a root node (without any goal in its successive capabilities) and its prerequisite goals. We call the root node a head capability. When to create a course, we can generate sub-networks of the head capability to cover all sub-goals in the divided set.

Definition 6.6 A head capability or head node, called *headCap*, is the capability that satisfies the following two conditions:

1) it is a goal;
2) its successor capabilities do not contain any goal capability

The Algorithm 6.5 in Appendix C is to find the maximal goal capability (i.e. a head capability whose predecessor capabilities contain more goals than any other head capability). And the algorithm 6.6 looks for all head capabilities in a given goal collection.

Identifying all Implied goals

An implied goal is the capability that is not selected as a goal by a learner, but it is a prerequisite capability of a selected learning goal and the learner does not master it. In order to achieve the selected learning goal, before the goal capability is achieved all its prerequisite capabilities must be mastered. We call such prerequisite capabilities implied goals. The mechanism to find implied goals is similar to that to find implied known capabilities.

6.3.5 Creating a Head Goal's Sub-Networks

We have divided all learning goals into several groups, each of which is a connected sub-graph of *Tnet*. The head capability in a group of goals is the root of the corresponding connected sub-graph. Since the connected sub-graph is still an AND/OR graph, several paths may exist for supporting the achievement of the root capability. The Algorithm 6.7 in Appendix C recursively creates all sub-networks of a given head capability. The idea is that we, first, find all predecessor transition nodes of the head capability. Each predecessor transition node is viewed as an alternative path. Then recursively generate all sub-networks of all prerequisite capabilities for a selected transition node, and combine them with the selected transition node and the head capability. By combining all sub-networks of all predecessor transition nodes of the head capability, the sub-networks of the head capability are gotten.

6.3.6 Creating the Complete Sub-Networks of a Head Capability

Definition 6.7 Let H be a head capability, and G be all sub-goals of the head capability H, N be one of sub-networks of H, if the sub-network N contains all sub-goals in G, we say that the sub-network N is a complete sub-network for H, otherwise, N is a incomplete sub-network for H.

Since all sub-goals of a head capability are distributed in AND/OR branches in the connected sub-graph where the head capability resides in, probably, some sub-network of the head capability can not cover all its sub-goals. As a result, if we use one of the sub-networks of the head capability as a sub-course for the head

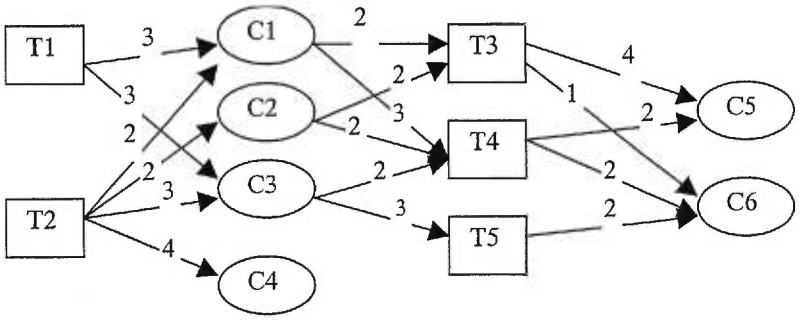
capability and all its sub-goals, some sub-goals of the head capability are probably lost. Thus, we have to check each sub-network of the head capability, and, if necessary, to expand the sub-network into a complete sub-network that can cover all sub-goals of the head capability.

The Algorithm 6.8 in Appendix C shows how to create all complete sub-networks of a sub-network for a head capability.

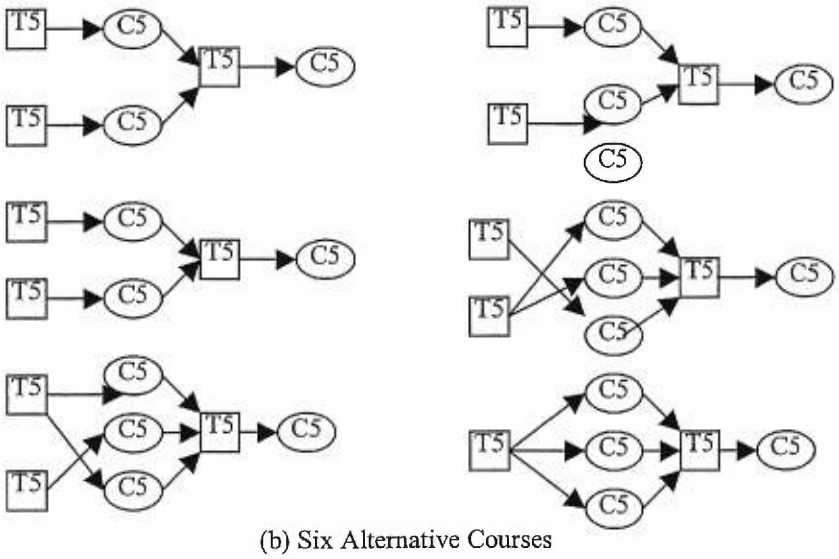
6.3.7 Putting it All Together—Creating a *Cnet*

By combining the steps through section 6.3.1 to the section 6.3.6, we get the algorithm (Algorithm 6.9 in Appendix C) to create multiple alternative courses, *Cnet*.

The figure 6.2 gives an example of course generation. Figure 6.2 (a) shows the corresponding *Tnet*. The known capabilities include C2 at level 3, C5 at level 1, and C7 at level 1. The goal capabilities are C5 to level 2 and C8 to level 3. That is



(a) Tnet



(b) Six Alternative Courses

Figure 6.2 An Example of Course Generation

K={<5, 1> <C7, 1>} and

G={<C5, 2> <C3, 3>}.

All six possible courses are shown in figure 6.2 (b). Some screen shot examples can be found in the section 6.4.4 and chapter 8.

6.4 Visualization of *VCOURSE*

Though most efforts for creating courses are to run numerous internal algorithms, we are still able to visualize the learners' states and created various courses to enhance the usability of the system. The visualization of *VCOURSE* model includes defining the states of capabilities and their levels, displaying known capabilities and goal capabilities, visualizing course networks as well as displaying the overall information of courses.

6.4.1 Definition of Capability States for the Learning Process

The following definition 6.8 describes various possible states of capabilities in learning process.

Definition 6.8 Let *C* be a capability, *L* be the ordered set of all levels in *C*, *Li* (i = 1,..., n) be an element in *L*, *S* be a student, *G* be the set of goal capabilities, and g (0<g<=n) be goal level of the capability *C*.
The following rules 1) to 10) define the states of a capability for learning process:

1) C's state =CAP_UNCONCERNED if C is neither a known capability nor a goal capability nor a necessary capability for achieving goals;
2) C's state =CAP_NOTHING_MASTERED if no level is mastered by the student S;
3) C's state =CAP_PARTLY_MASTERED if at lease *L1* is mastered and *Ln* is not mastered;
4) C's state =CAP_GOAL_MASTERED if C is a goal and the goal level g is mastered and g <n;
5) C's state =CAP_FULL_MASTERED if Ln is mastered and all i<n, Li is mastered;
6) For all i<n, Li's state =LEVEL_UNCONCERNED if C's state =CAP_UNCONCERNED;
7) For all i<n, Li's state =LEVEL_NOT_MASTERED if the student S does not mastered the level Li;
8) For all i<=n, Li's state =LEVEL_PARTLY_MASTERED if the student S partly masters Li;
9) For all i<n, Li's state =LEVEL_MASTERED if the student S has mastered Li, and
10) For all i<n, Li's state = LEVEL_GOAL if Li is a goal and Li's state =LEVEL_MASTERED.

We will use these rules through the following sections and upcoming chapters.

6.4.2 Visualizing Known Capabilities and Levels

In a visual curriculum, a learner should be able to set his known capabilities and levels by direct manipulation. In *Cnet*, the known capabilities and their levels can be established by simply clicking the corresponding visual cells. Some resources are used to explain to the student what the capability means.

When the learner clicks the overall capability cell in a capability icon, the state and the corresponding color of the capability icon will be changed. Meanwhile, the states of all levels in the capability should follow the change. In order to make consistency when the learner clicks the overall capability cell or one of level cells; a group of rules are defined for the consistency.
These rules are:

Definition 6.9 Let C be the overall cell in a capability icon, L be the ordered set of all level cells in the capability icon, Li (i = 1, …, n) be a level cell, Lg be the index of a goal level in the capability,

Rk1: if C is clicked & (C.state=CAP_UNCONCERNED or C.state=CAP_NOTHING_MASTERED), then C.state ←CAP_FULL_MASTERED & all Li (i<n).state=LEVEL_MASTERED;
Rk2: if C is not a goal & C is clicked &(C.state=CAP_FULL_MASTERED or C.state=CAP_PARTLY_MASTERED), then C.state←CAP_NOTHING_MASTERED;
Rk3: if C is not a goal & Ln is clicked, then C.state←CAP_FULL_MASTERED& all Li.state←LEVEL_MASTERED;
Rk4: if C is not a goal & i<n & Li is clicked, then C.state←CAP_PARTLY_MASTERED & (all j<=i, Lj.state ← LEVEL_MASTERED) & (all k>i, k<=n, Lk.state←LEVEL_NOT_MASTERED);
Rk5: if C is a goal & C is clicked & (C.state==CAP_FULL_MASTERED or C.state==CAP_GOAL_MASTERED), then C.state←CAP_NOTHING_MASTERED, Lg.state←LEVEL_GOAL and (for all i<n, except for the goal level Lg, Li.state←LEVEL_NOT_MASTERED);
Rk6: if C is a goal & C is clicked & C.state==CAP_PARTLY_MASTERED, then C.state←CAP_NOTHING_MASTERED, & (for all i<n, except the goal level Lg, Li.state←LEVEL_NOT_MASTERED);
Rk7: if C is a goal & the goal level g is clicked, then C.state←CAP_GOAL_MASTERED, & (for all i<=Lg, Li.state←LEVEL_MASTERED), & (for all j>Lg, j<=n, Lj.state←LEVEL_NOT_MASTERED);
Rk8: if C is a goal & Ln is clicked, then C.state ← CAP_FULL_MASTERED & (for all i<=n, Li.state ← LEVEL_MASTERED);
Rk9: if C is a goal & Li is clicked & i< Lg<n, then C.state ← CAP_PARTLY_MASTERED, & all j<i, Lj.state ← LEVEL_MASTERED, Lg.state←LEVEL_GOAL, & (for all k<Lg, k<=n, Lk.state ← LEVEL_NOT_MASTERED);
Rk10: if C is a goal & Li is clicked & i> Lg & i<n, then C.state ← CAP_GOAL_MASTERED, & all j<=i, Lj.state ← LEVEL_MASTERED & (for all k>i, k<=n, Lk.state←LEVEL_NOT_MASTERED).

This group of rules can meet the needs of all operations for setting, canceling and changing known capabilities and their levels in learning process.

6.4.3 Visualizing Goal Capabilities and Their Levels

In order to visually set up and change the states of learning goals, the following definition 6.10 gives a group of rules.

Definition 6.10 Let C be the overall cell of a capability icon, L be the ordered set of all level cells in the capability, Li (i= 1, …, n) be a level cell, and Lg be the index of the goal level in the capability icon, the rules for state transformation of goal capabilities are defined as follows:

Rg1: if C is not a goal & C is clicked & C.state==CAP_UNCONCERNED, then Ln.state←LEVEL_GOAL & (for all i<n, Li.state←CAP_NOTHING_MASTERED);
Rg2: if C is not a goal & C is clicked & C.state==(CAP_PARTLY_MASTERED or CAP_NOTHING_MASTERED), then Ln.state←LEVEL_GOAL;
Rg3: if C is not a goal & Ln is clicked & C.state==CAP_UNCONCERNED, then C.state←CAP_NOTHING_MASTERED & Ln.state←LEVEL_GOAL & (for all i<n, Li.state←LEVEL_NOT_MASTERED);
Rg4: if C is not a goal, Ln is clicked & C.state≠ CAP_UNCONCERNED, then Ln.state←LEVEL_GOAL;
Rg5: if C is not a goal & i<n & Li is clicked & C.state==CAP_UNCONCERNED, thenC.state←CAP_NOTHING_MASTERED, Li.state←LEVEL_GOAL, for all j≠i, j<=n, Lj.state←LEVEL_NOT_MASTERED;
Rg6: if C is not a goal & I<n & Li is clicked & C.state ≠CAP_UNCONCERNED & Li.state≠LEVEL_MASTERED, then Li.state←LEVEL_GOAL;
Rg7: if C is not a goal & i<n, Li is clicked & C.state≠CAP_UNCONCERNED & C.state≠CAP_FULL_MASTERED & Li.state==LEVEL_MASTERED, then C.state←CAP_GOAL_MASTERED;
Rg8: if C is a goal & C is clicked & Lg≠LEVEL_MASTERED, then Lg.state←CAP_GOAL_MASTERED;
Rg9: if C is a goal & Ln is clicked & Lg<n & Lg.state≠LEVEL_MASTERED & C.state≠CAP_FULL_MASTERED, then Ln.state←LEVEL_GOAL & Lg.state←LEVEL_NOT_MASTERED;
Rg10: if C is a goal & i<n & Ln is clicked & i<Lg & Lg.state≠LEVEL_MASTERED & Li.state==LEVEL_MASTERED, then C.state←CAP_GOAL_MASTERED & Lg.state←LEVEL_NOT_MASTERED;
Rg11: if C is a goal & i<n & Li is clicked & i>Lg & Lg.state≠LEVEL_MASTERED, then C.state←CAP_PARTLY_MASTERED, Li.state←LEVEL_GOAL & Lg.state←LEVEL_NOT_MASTERED.

These rules constitute a frame for setting learning goals.

6.4.4 Visualizing Courses

There are two aspects in visualizing courses; one is to display alternative courses, and the other is to display the overall information of the course network being displayed.

- Visually displaying course networks

After alternative courses are generated, the system provides two operations controlled by the learner: displaying the predecessor course and displaying the successive course on the screen. All states of visual cells are also displayed to navigate to the learner.

- Visually display overall information of courses

The transition nodes, non-goal capability nodes, links, resource types, resource sizes, media types and cognitive strategies are, usually, different from one course to another course corresponding to the same goal group. In order to guide the learner to compare courses and select his preferred course, Our model provides a group of overall information about a course being displayed, including the number of various nodes, resource types, media types, cognitive strategies, and the resources sizes.

As soon as the learner selects a course, the system assigns the selected course as the learner's current course, and records necessary information about the selected course to the learner's profile file. Then with the help of tutoring delivery module in ITS, the learner can leaner the course.

The following figure 6.3 ~6.7 are an example of creating multiple alternative courses. The figure 6.3 shows the *Tnet* for teaching a part of UML. A learner sets the first level of the output capability "createSequenceDiagram" as his learning goal. The figure 6.4 ~ 6.7 are four created courses supporting the learning goal.

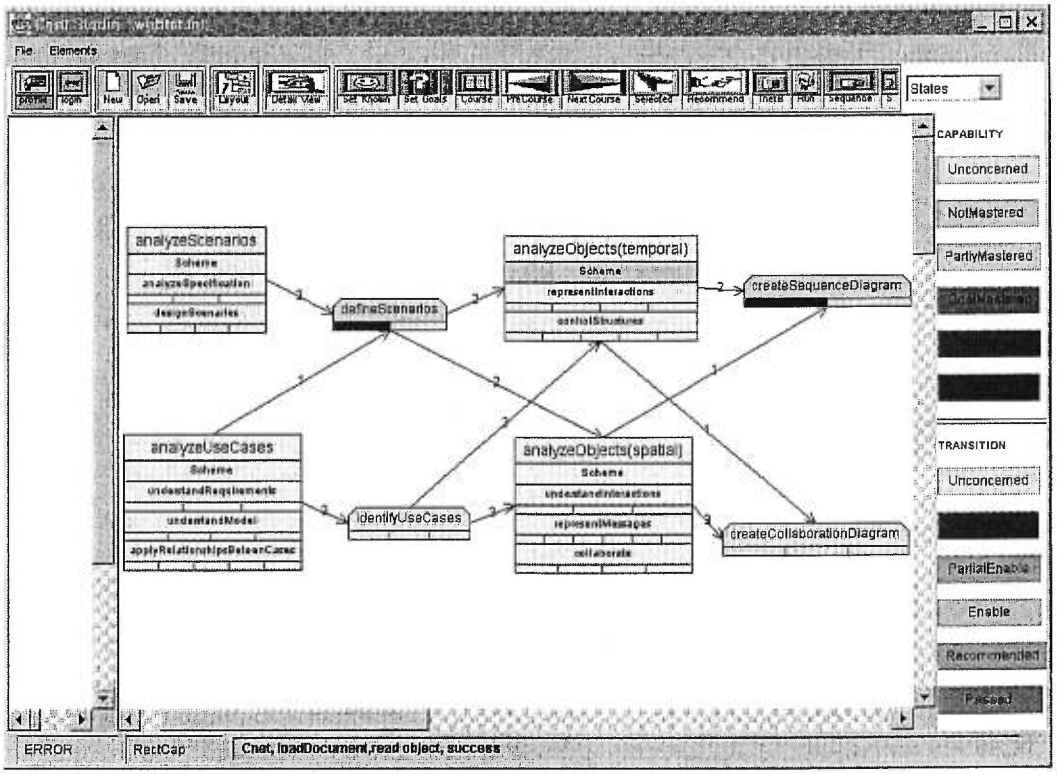


Figure 6.3 *Tnet* for teaching a portion of UML

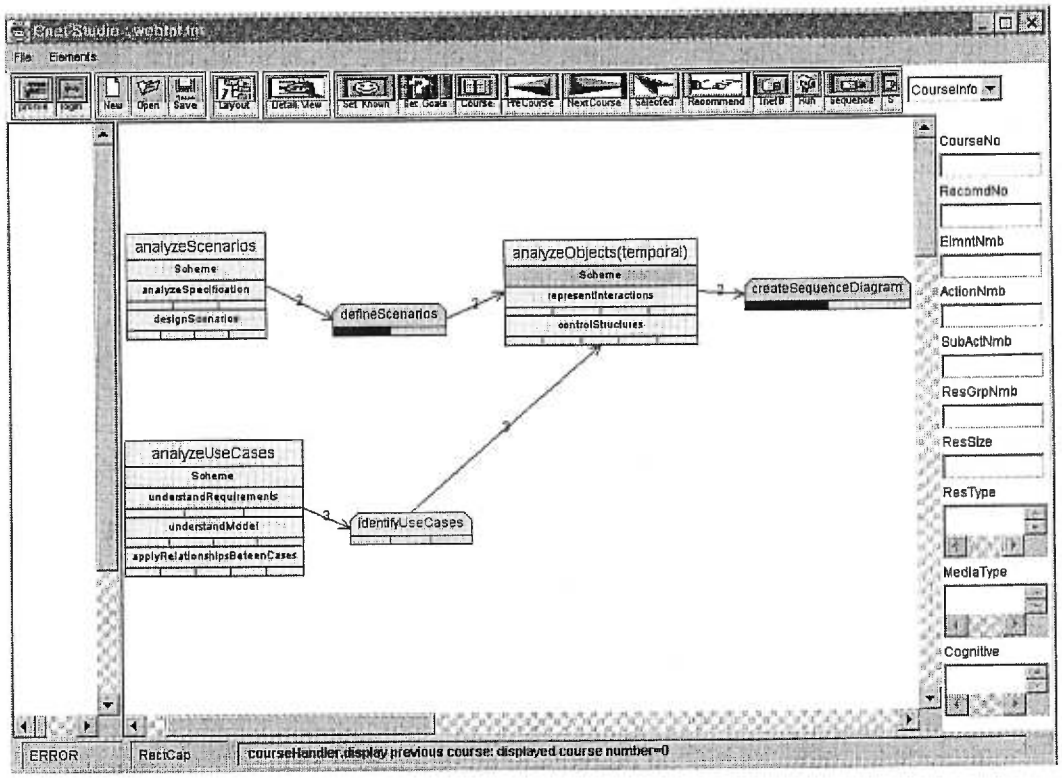


Figure 6.4 The first UML course

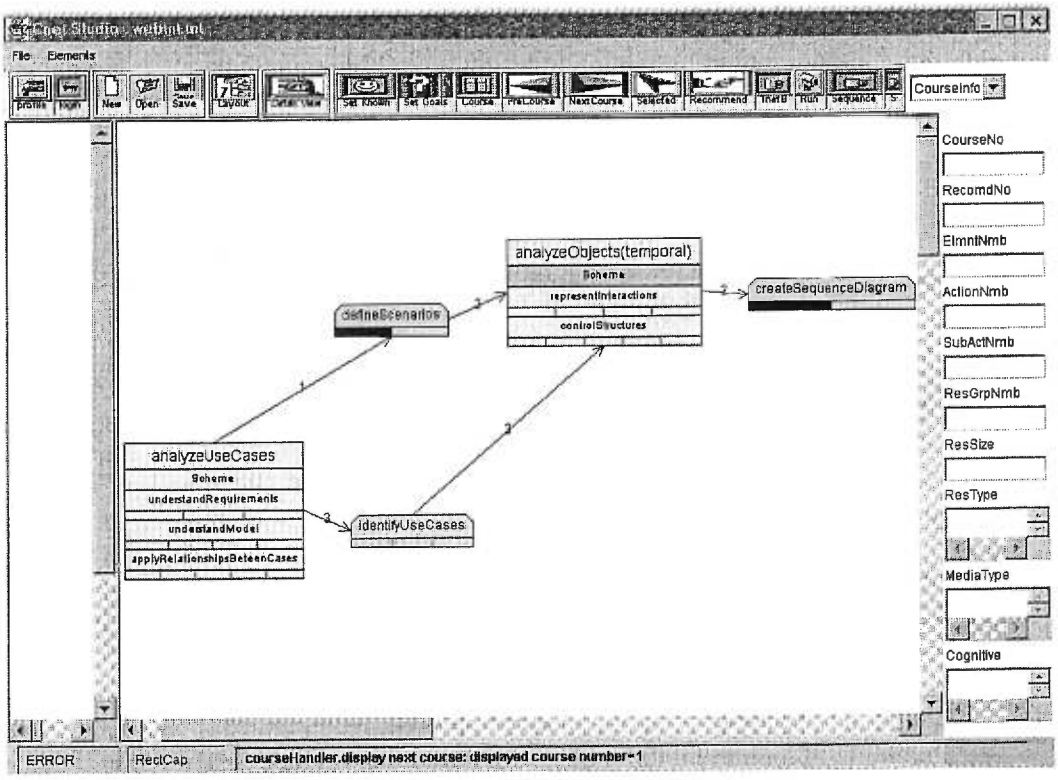


Figure 6.5 The second UML course

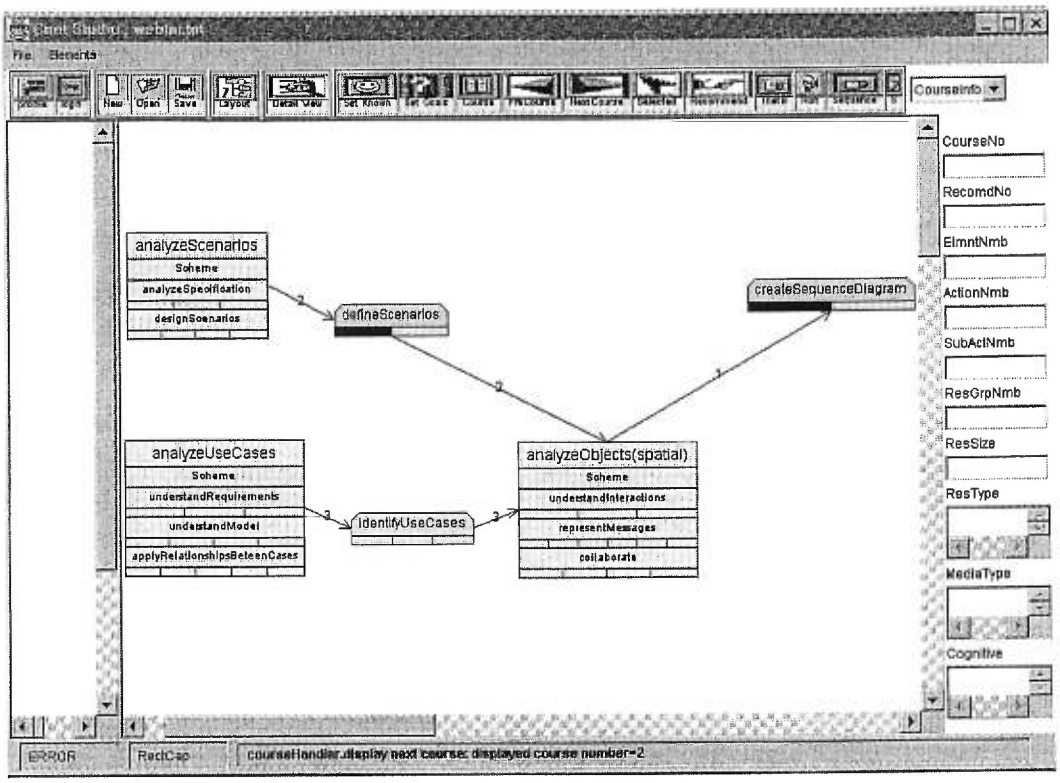


Figure 6.6 The third UML course

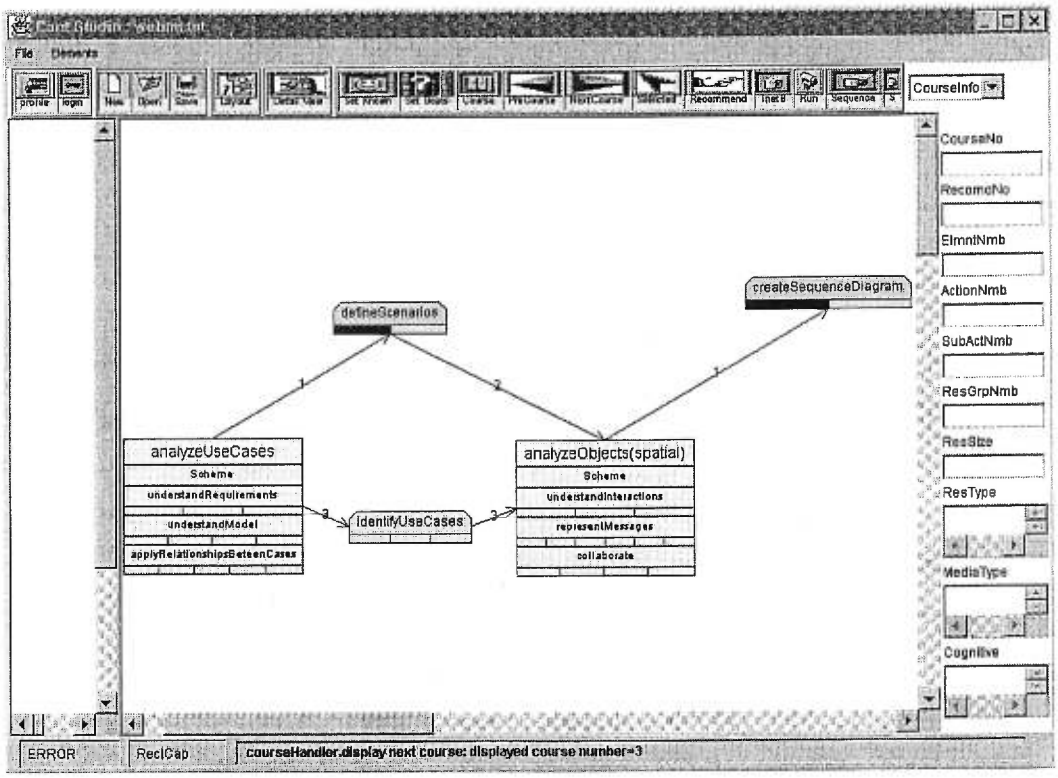


Figure 6.7 The fourth UML course

6.5 Conclusion

VCOURSE model provides approaches for setting known capabilities, establishing learning goals, creating multiple alternative courses and dynamically displaying courses.

When to set known capabilities, if the learner clicks a capability level, a group of reasoning rules are activated for making consistent state change of the capability and its levels. When to set learning goals, another group of reasoning rules are defined for making consistency of node states.

The approach for creating multiple alternative courses (*Cnet*) is based on a divide-and-conquer method by grouping learning goals to divide the *Tnet* into several disjointed sub-graphs, by removing redundant elements, recurrently creating all sub-courses in each sub-graph and combining all sub-courses in all sub-graphs.

With *VCOURSE* model, a learner may visually set known and goal capabilities. He/she can compare multiple alternative courses based on the summaries of courses, and then select his preferred course to learn.

In the next chapter, we shall describe the dynamic management of learning processes with which learners can be helped in selecting a particular course, in case they did not select one by themselves, and in guiding learning processes.

Chapter 7

Dynamic Management of Capability Transition (*VACT*)

One distinct feature of computer-based tutoring systems is adaptation to individual needs. A tutoring system should be able to identify a particular content in the current context, to sequence the identified content [Nkambou 96, Le 98] and to use particular tutoring strategies for a particular student. In the *VCOURSE* model, multiple alternative courses corresponding to a particular student's known capabilities and goal capabilities are generated. In this chapter, the courses and tutoring activities are dynamically individualized providing help in recommending recourses, sequencing tutoring activities and dynamically managing the current learner sates.

The first section analyses essential issues in individualizing courses and tutoring activities. A measure called *utility* for evaluating course adaptability is defined in section 7.2. With this measure, a method for recommending courses is introduced. In the section 7.3, we describe how to identify necessary tutoring activities for a group of learning goals. In order to sequence tutoring activities dynamically, a state-driven reasoning approach is presented in section 7.5. In section 7.6, we deal with the dynamic management for activity delivery Last, in section 7.7, the visualization of *VACT model* is tackled.

7.1 Overview of the Dynamic Teaching Process

7.1.1. Essential Issues in Dynamic Management of Capability Transition

In the process of individualizing courses and tutoring activities, some essential issues include evaluating courses, identifying necessary tutoring units in transition nodes for goal levels of output capabilities, sequencing tutoring activities, and providing dynamic feedback in tutoring process.

7.1.1.1 Evaluating Courses

The quality of a course depends on various factors, for instance, the well-organized structure of domain knowledge, good tutoring strategies, the coverage degree to learning goals, and the adaptability to a particular student. The quality of organization structure of domain knowledge, in current ITS, depends on both the mechanisms of representation and organization of various knowledge, and the competence of the curriculum authors.

Some issues have been dealt with in the previous chapters. The characteristics of the organization structure proposed in *VITCAM* have been summarized in chapter 4 and 5, in which Gagné's and Bloom's theories are combined. All alternative courses created by the *VCOURSE* model can cover all learning goals established by a learner. We classify the tutoring strategies into two categories: general strategies and individual strategies. The *VTRANS* model can support general strategies in its organization structure and individualized strategies in resources.

In *VACT* model, in order to evaluate the course adaptability degree, we focus on satisfying the preference of the learner and needed efforts. We think that the learner's preference is mainly reflected in the three aspects:

- Types of didactic resources,
- Media types in resources, and
- Cognitive strategies in resources.

There are various types of didactic resources that can be developed with current computer systems, for instance statements, multiple choice questions, matching two or more groups of objects, filling missed objects, asking/answering questions, sequencing a group of objects, true/false questions, etc. Different learners may have different interests in the resource types. If a learner is very interested in some special resource types, he/she will achieve some capabilities more effectively with the preferred resources.

The media types supported by a computer system vary from platform to platform. They may include text, graphics, images, audio, animation, video, html, vrml, etc. Usually, complex media help learning more effectively. Similarly, a learner can achieve more progress with the media types he/she prefers.

Cognitive strategies are another type of learners' preferences. They are often very important to affect learners' studying. Gagné defined five kinds of cognitive strategies:

- Rehearsal (repeating, underlining, copying, ...),
- Elaboration (associating, paraphrasing, summarizing, note taking, generating questions with answers, ...),
- Organizing (classifying, outlining main ideas, generating new organizations, comparing, collecting, describing, ...),
- Comprehension monitoring (meta-cognitive: setting goals, estimating success),
- Affective (focusing and maintaining attention, controlling anxiety, managing time,...).

Some other strategies such as induction and deduction may be also used in the development of courses.

Learning cost contains two factors: the difficulty degree of content and the access time of the content. The developer of didactic resources may define a difficulty degree for each concept. The access time of a resource can be automatically computed by the system.

We will use the above four kinds of information (i.e., resource types, media types, cognitive strategies and learning efforts) to define a measure for evaluating the individualized degree of courses. This measure, called *utility*, is characterized in detail in section 7.2.

7.1.1.2 Identifying Necessary Tutoring Units for Learning Goals

In *Tnet* structure, a transition node consists of a group of tutoring units, and each tutoring unit corresponds to certain levels of output capabilities. If a student wants to learn certain output capability levels, it is not necessary for the student to learn all tutoring units involved in the related transition node. The following figure 7.1 shows an example. In figure 7.1, the capabilities *C3* and *C4* are output capabilities of the transition node *T*. There are five levels in *C3*, and four levels in *C4*. Assume that a learner has mastered the first two levels of *C3* and the first two levels of *C4*. His leaning goals are the fourth level in *C3* and the third level in *C4*. We should decide which tutoring units in the transition node have been passed by the learner, and which units

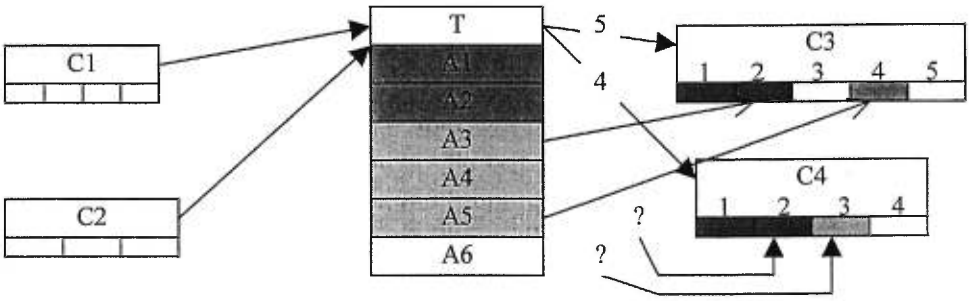


Figure 7.1 The problem in identifying necessary tutoring units

are necessary for the learning goals in *C3* and *C4*. Since, usually, the corresponding relations between tutoring units and output capability levels are not one-to-one; we use probability method to identify necessary tutoring units for certain goal levels in section 7.3.

7.1.1.3 Dynamically Sequencing Tutoring Activities

Sequencing tutoring activities is a dynamic process to identify and estimate the instant availability of transition nodes and tutoring units in a course. The instant availability of a tutoring activity depends on the requirement of domain knowledge structure, the current states of the leaner, and some other factors, such as favoring the association and remembering in cognitive strategies.

There exist some heuristic methods for ordering tutoring activities. For instance, in order to meet the structured characteristic of domain knowledge, the tutoring activities that are far from the root goals in a course should be recommended first. Considering the current learner states, the direct successive tutoring activities related to the re*cent*ly mastered capabilities should be learned first. In order to favor the association and remembering in cognitive strategies, the successive tutoring activities of the last executed tutoring activities should be given first. So, when we sequence available tutoring activities, these factors should be synthetically considered.

7.1.1.4 Dynamical Adaptability of the Learning Processes

A dynamic feedback mechanism is proposed in *VACT*, which determines the next step of the system according to the feedback information provided by the tutoring delivery module (or tutor). This means that the system's decision for next step can be either to execute next recommended activity, or to re-sequence tutoring activities, or to re-create tutoring activity networks, or to re-create new courses to adapt the learner's current states.

7.1.2 Overview of Dynamic Capability Transition

The dynamic process delivering teaching refers to that, given a student's known and goal capability levels, and a group of alternative paths covering the student's goals, how to dynamically identify and deliver teaching activities to achieve the student's goals based on the student's dynamic reaction in learning process.

The figure 7.2 shows an approach to handle dynamic feedback based on different tutors' abilities. At the beginning of the dynamic management process, the function *Course Creator* generates multiple alternative courses covering a student's goals. Another function: *Course Recommender,* then, evaluates these alternative courses and recommends an optimal course to the student. After a course is selected as the current course, the function *Unit Identifier* calculates all necessary tutoring units in transition nodes to meet the needs of achieving goal capability levels. Then, the *Unit Sequencer* identifies all currently enabled tutoring units based on the domain knowledge structure and the student's states. Last, it orders these enable units to indicate the currently recommended tutoring units.

VACT model allows the *Activity Deliverer* module (or a Tutor, an opening module in our system) with various abilities, which may be from the simple evaluation of success/failure to more advanced diagnosis abilities. The system may manage different feedback of learners' current states

If the *Activity Deliverer* module can just evaluate the student's success or failure, and cannot diagnose the cause of failure, the system can change didactic resources or teaching repeatedly.

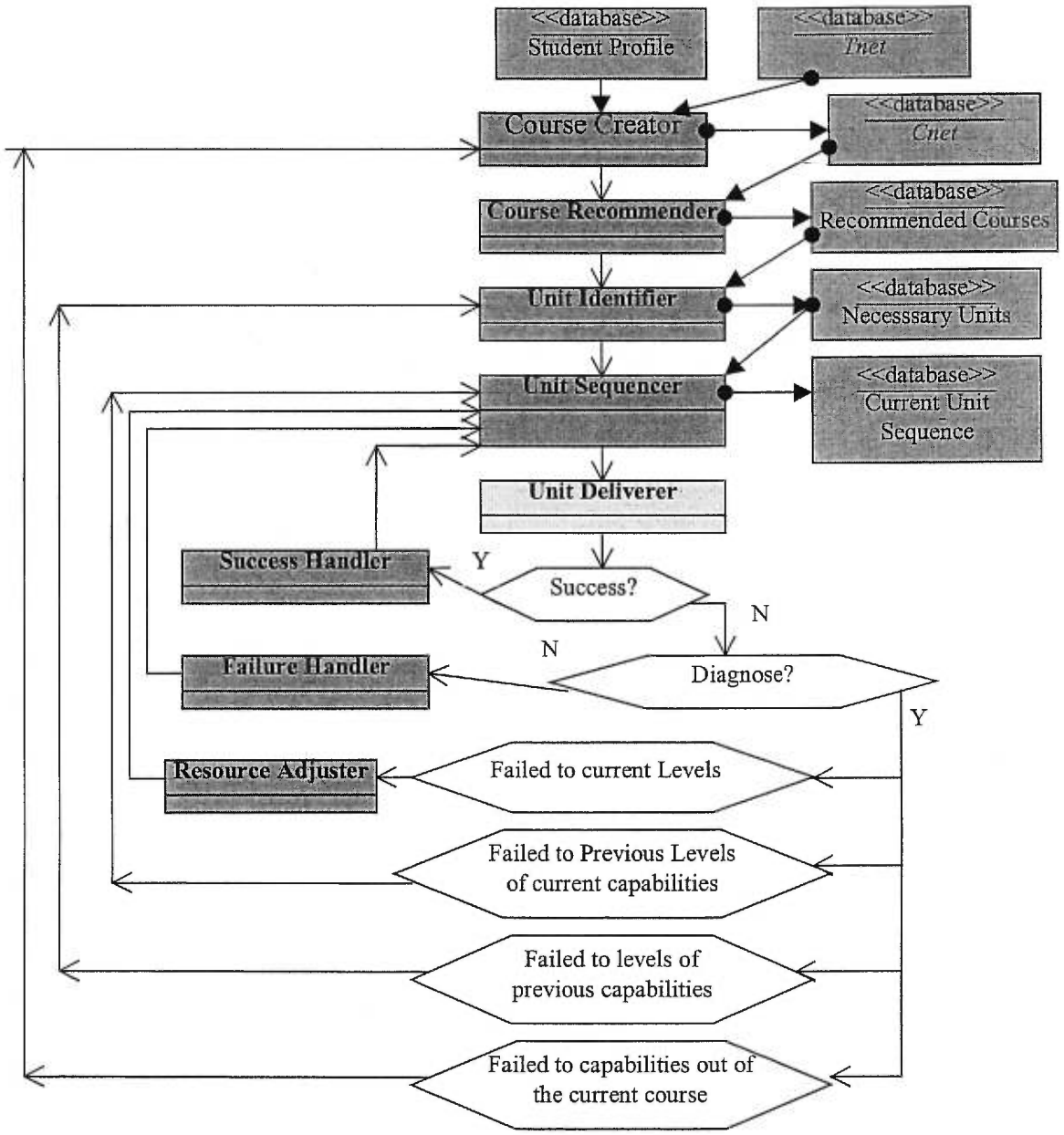


Figure 7.2 Dynamic management process for delivering capabilities

If the *Activity Deliverer* module has some advanced abilities. For example it can distinguish from:

1) Failing to current levels
2) Missing previous levels of the current capability;
3) Missing some levels of previous capabilities;
4) Missing some capabilities out of the current course.

This system dynamically handles these feedback of learner states: If the *Activity Deliverer* can just identify the student's failure to the current capability levels, the system tries to change the possible resource group in the corresponding sub-units based on the student profile. If there is no alternative resource group, the system supports learning repeatedly. If the *Activity Deliverer* can diagnose that the student misses some previous levels of the current capabilities, the system goes back to the *Unit Sequencer* to generate new unit sequence that contains the units corresponding to the missing levels. If the *Activity Deliverer* can diagnose that the student misses the levels of previous capabilities, the system goes back to the *Unit Identifier* to identify the corresponding units to learn further. If the *Activity Deliverer* can diagnose that the student misses capabilities that are out of the current course, the system, then, goes back to the *Course Creator* to generate new courses for remedying the missing capabilities. In addition, if the *Activity Deliverer* can diagnose that the student's some particular characteristics, such as, particular preferences, the system's *resource adjuster* can look for the corresponding resources to remedy the student's missing. The remainder sections will characterize the mechanisms that support the above functionality in detail.

7.2 *Utility*—A Measure for Evaluating Courses

As mentioned in section 7.1, students' preferences, learning efforts and resource features are the essential factors to evaluate the course adaptability. We model the factors by defining a measure, called *utility*, for evaluating courses. We first deal with the relationships between student profiles, learning efforts, resources and course evaluation. We, then, define the measure to evaluating courses.

7.2.1 Basic Requirements for Student Profiles and Resources

In *VTRANS* model, some basic resource types are identified, for instance statement, multiple choice questions, matching two groups of objects, sequencing objects, filling the missing in object sequences, asking/answering questions, true/false questions, etc. We will use these resource types as a kind of learner preference.

Media types are another kind of resource features. Some available media types in current computer platforms include text, graphics, image, audio, video, animation, html, vrml, etc. When to evaluate courses, these media type can be used to characterize the preference degree of a particular student.

Gagné identified five kinds of cognitive strategies: rehearsal, elaboration, organization, comprehension monitoring, and effective. In addition, other strategies such as induction or deduction may be also used in the development of resources and interactions. These strategies are very important when to create a course that is adapted to a particular student.

We identify three kinds of student information to reflect students' preference: resource types, media types and cognitive strategies. A particular student might prefer certain types of resources, such as, multiple choice questions or matching two groups of objects. Some students might prefer graphics, while some others might like audio. Some cognitive strategies are effective for some students, while some other cognitive strategies may be better for some other students. Thus, the system requires that student profiles contain such information.

In addition, in order to evaluate the efforts of tutoring activities involved in a course, the difficulty degree, the importance degree and the size of a resource will be also important parameters. In our system, a curriculum author can assign a difficulty degree and an importance degree to a resource or a capability. The system automatically computes the size of each resource. The three parameters will be also used in course evaluation.

7.2.2 *Utility* Definition

To evaluate a course is to determine whether it is the most appropriate for a particular student, i.e. whether the course can satisfy the preference of a particular student to a maximal degree. In order to do so, the following factors are taken into account:

- Types of resource (*resType*),
- Media types (*mediaType*),
- Cognitive strategies (*strategyType*),
- Preferred degrees to resource types (*preferredResTypeDegree*),
- Preferred degree to media types (*preferredMediaDegree*),
- Preferred degree to cognitive strategies (*preferredStrategyDegree*),
- Importance degree of the capabilities the resource supports (*importanceDegree*), and
- Difficulty degree of the capabilities the resource supports (*difficultDegree*).

In order to define precisely a measure for evaluation, we use the following vectors to describe these factors:

resType = [T1, T2, ..., Tu]

where *Ti* may be any type of resources such as statement, multiple choice questions, etc.,

mediaType = [M1. M2, ..., Mv] where *Mi* may be text, graphics, audio or video,

strategyType=[S1, S2, ..., Sx} where *Si* may be rehearsal, elaboration, etc.,

preferredResTypeDegree, perferredMediaDegree, and *preferredStrategyDegree* are vectors in which each element is an integer between 0 and 10, and corresponding to the degree a student prefers the related resource

type, media type, and cognitive strategy respectively. The other two parameters: *difficultDegree = [D1, D2, ..., Dy]*, and *importanceDegree = [I1, I2, ..., Iz]* are also vectors with the values between 0 to 10.

We identify the other three parameters *ResTypeDistribution*, *MediaTypeDistribution* and *StrategyDistribution* for indicating the distributions of each resource type or media type or strategy type in all resources in the current course. We define the distribution of a kind of resource *Ri* as the ratio between the number of resources of this type of resources and the total number of resources in the current course. We can define two other distributions similarly. The formulas for computing these distributions are as follows.

$$ResTypeDistribution = <numberOfRi/totalNumberOfResourcesInCourse \mid i=1,2,...>$$
$$MediaTypeDistribution = <numberOfMi/totalMediaNumberInCourse \mid i=1,2,...>$$
$$StrategyDistribution = <numberOfSi/totalStrategyNumberInCourse \mid i=1,2,...>$$

Now we define the utilities of resource types, media types and strategy types respectively as

$$ResTypeUtility = preferredResTypeDegree * resDistribution$$
$$MediaTypeUtility = preferredMediaTypeDegree * mediaTypeDistribution$$
$$StrategyTypeUtility = preferredStrategyTypeDegree * strategyTypeDistribution$$

Based on these parameters, we define the utility of a tutoring unit (*Ai*) as

$$U(Ai) = \max_{resGroup} \{ \frac{(resTyprUtility * mediaTypeUtility * strategyTypeUtility) * importanceDegree}{\sum (dificultyDegree * sizeOf\ Resource)} \}$$

The utility of a course is defined as

$$U(course) = \sum_{i=1}^{n} U(Ai)$$

Where *Ai* (i = 1,2,…,.) is a necessary tutoring unit for certain goal capability levels.

7.2.3 Algorithm for Computing Course *Utilities*

The algorithm 7.1 in Appendix D is used to compute the utility of a course. For all alternative courses generated in the previous chapter, we can compute their utilities based on the above algorithm. The system then sorts these utilities to get the optimal course and recommend it to the current student.

7.3. Identifying Necessary Tutoring Activities

For any transition node *T* in a course, there may exist several output capabilities. Each output capability contains two important levels: the mastered level and the goal level. The mastered level implies that some

tutoring units have been learned. Since the goal level may be less than the maximal level of the output capability, some tutoring units that support the higher level than the goal level are unnecessary for acquiring up to the goal level.

Thus, the necessary tutoring units for acquiring the goal level have to be identified. This section describes a method based on the Bayesian technique for identifying necessary tutoring units. We, first, analyze the characteristics and problems in identifying tutoring units. Then the identification approach is proposed.

7.3.1 *Course* Characteristics

There are distinct characteristics involved in a course, including uncertainty, heterogeneous nodes and links, causal propagation, and bi-directional inference.

(1). Uncertainty

There are several uncertain factors involved in *Tnet*.

The contribution degrees of a transition node to some capabilities are uncertain. A transition node may have very strong contribution to one capability, say to mastering level 10, while there may be very weak contribution to another capability, say to mastering level 1.

The initial mastering levels of capabilities are uncertain, which may be from 0 to 10. The desired mastering levels of capabilities are uncertain. Unlike the CREAM model, *Tnet* allows the goal capability levels to be any designed level.

Another uncertainty in *Tnet* is the uncertainty of the levels of tutoring units involved in transition nodes. For example, the transition node T_1 may contain three tutoring units based on the current resources, while T_2, probably, contains six units with available resources.

In addition the origin of contributions is uncertain. For example, the current mastering level of capability$C$ is 2. The transition node $T1$ may contribute maximal mastering level 6 to C, and T_2 may contribute maximal mastering level 4 to C. The question is that whether the current mastering level 2 comes from the contribution of T_1 or that of $T2$.

There are other two important uncertainties, one is that the mapping relations from prerequisites to transition nodes are not exact. Another is that the mapping relations from transition nodes to output

capabilities are not one-to-one relations. These uncertainties make that any existing graphical search technique can not be used for course generation.

(2). Heterogeneous Nodes and Links

Tnet includes two different kinds of nodes and links. Capability nodes and transition nodes have very different meanings in *Tnet*. Similarly, input links and output links of transition nodes do not have the same semantics.

(3). Characteristics of Causal Propagation

In *Tnet*, a capability acquisition process may be viewed as the causal propagation process. For example, the causal of acquisition of one capability is the result of some activated activities, while the activation of one tutoring activity is the causal that some precedent capabilities are acquired, and so forth.

(4). Bi-directional Inference Characteristics

A process of knowledge transition may be considered as an inference process. For example, if the prerequisites of transition node T are satisfied, then the tutoring units in T may be activated, and if these tutoring units are activated, then a student can acquire certain capabilities.

However, when the current mastering levels of capabilities are given, and we want to confirm which transition nodes gave these contributions, this may be considered as a contrary inference process, i.e., the reasoning from effects to causes. Such bi-directional inference can not be dealt with by most existing uncertainty reasoning techniques.

7.3.2 Identifying Necessary Tutoring Units by the Bayesian Technique

In this section, we briefly review some approaches for uncertainty reasoning first. Then, the approach to identify necessary units in transition nodes technique is dealt with.

7.3.2.1. Introduction

Well-known uncertainty reasoning techniques includes non-monotonic reasoning, fuzzy logic, Certain-Factor theory, and Bayesian techniques.

Non-monotonic reasoning attempts to extend first-order logic by confirming non-determined propositions by admitting them until negative evidences are given [Guo 96]. However, non-monotonic reasoning suffers from the drawback in weak ability to handle multiple values reasoning.

Fuzzy logic deals with some vague propositions that are difficult to be measured numerically. However, fuzzy logic can not handle bi-directional reasoning and complex dependence relationships between transition activities and capabilities because one transition activity may support the acquisition of multiple capabilities and the acquisition of one capability may relates to multiple transition activities.

Certainty-Factors theory (CF theory) [Shortliff, 76] is a well-known uncertainty reasoning method that was successfully used in the medical diagnosis system MYCIN. CF theory uses the credible degrees of evidences and rules to infer the credible degrees of consequences. Though CF theory has no well-found mathematical foundation, it is effective for the disease diagnosis of. However, one defect of CF is that it supports only diagnosis issues, i.e., the reasoning only from premises to consequences, and fails to sustain the predictive reasoning, i.e., the reasoning that, given some observation of consequences, how to decide the credible degrees of premises. In *Tnet*, a typical predicative reasoning is that, given the current mastering levels of output capabilities related to a transition node *Ti*, how to determine which level of tutoring activities in *Ti* has been executed successfully? Therefore, CF theory is not appropriate for the bi-directional reasoning in *Tnet*.

Bayesian network [Pearl, 1988] is another well-known uncertainty reasoning method based on probability theory. It is most appropriate for the kind of problems in which the observations of random variables can be measured numerically. It also sustains bi-directional reasoning.

A Bayesian network is also called a cause-effect network, i.e., the parent nodes stand for the causes and the child nodes indicate the effects. However, in *Tnet*, the relationships from parent nodes to child nodes are not pure cause-effect relationships because the cause of enhancing the mastering levels of output capabilities is explicitly the effect of activating tutoring activities, but the relationships from prerequisites to transition activities are preconditions rather than the causes like that from transition nodes to output capabilities. Therefore, the two distinct types of relationships in *Tnet* make the reasoning of course generation complex.

Owing to Bayesian network's distinct advantages over other uncertainty reasoning methods, for example, having well-founded mathematical foundations, supporting bi-directional reasoning, and the low costs for handling the dependency of multiple variables. We use Bayesian network techniques to identify tutoring units for goal capability levels.

In upcoming sections, we build up the association between *Cnet* and basic Bayesian techniques to identify the necessary tutoring units for the goal capability levels. Some relevant prerequisite probability formula can be found in Appendix D.

7.3.2.2 Computing Corresponding Units by the Bayesian Rule

By the Baysian rule (see Appendix D for details), we can compute the posterior probability that a tutoring unit is delivered given the mastered levels of output capabilities. We begin from the simplest case, i.e. a unit with an output capability level.

1) Single Unit with One Output Capability Level

Example: Given that the transition activity A_{i1} supports the acquisition of capability level C_{j1} (as in Figure 7.3), assume $P(C_{j1}=1|A_{i1})$ =0.8, and $P(A_{i1})$=0.15.

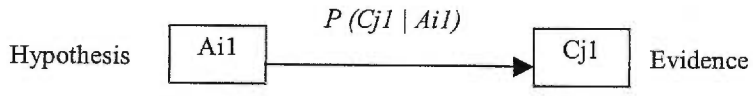


Figure 7.3 Tutoring activity A_{i1} relates to mastering level $C_{j1}=1$

We obtain the posterior odds by Basian rule

$$O(A_{i1}|C_{j1}=1) = L(C_{j1}=1|A_{i1})O(A_{i1}).$$

$$= \frac{P(C_{j1}=1|A_{i1})}{P(C_{j1}=1|\neg A_{i1})} \cdot \frac{P(A_{i1})}{P(\neg A_{i1})}$$

$$= \frac{0.8}{0.2} \cdot \frac{0.15}{0.85} \quad =0.7059$$

This posterior odds indicates that if the output capability *Cj1* is mastered, the probability that the tutoring activity *Ai1* is passed successfully is 0.7059.

2) Conditional Probability Matrix

Let the conditional probability matrix corresponding to the link $A \rightarrow C$ is

$$M_{C|A} = P(A \mid e) = \begin{array}{c} \\ A \downarrow \end{array} \overset{C \longrightarrow}{\begin{bmatrix} P(C1 \mid A1) & P(C1 \mid A1) & \dots & P(C1 \mid A1) \\ P(C1 \mid A1) & P(C1 \mid A1) & \dots & P(C1 \mid A1) \\ \dots & \dots & \dots & \dots \\ P(C1 \mid A1) & P(C1 \mid A1) & \dots & P(C1 \mid A1) \end{bmatrix}}$$

For the single link $A \rightarrow C$, if the evidence $\{C = c\}$ is observed, then from Bayes Rule, the belief distribution of A is given by

$$BEL(a) \equiv P(a \mid e) = \alpha\, P(a)\, \lambda(a) \tag{7.1}$$

where a is a possible variable of A,

$\alpha = [P(e)]^{-1}$ is the normalizing constant,

$P(a)$ is the prior probability of A, and

$$\lambda(a) = P(e \mid a) = P(C=c \mid a) = M_{c|a}. \tag{7.2}$$

For the chain $X \rightarrow Y \rightarrow Z$, we still can write

$$BEL(x) \equiv P(x \mid e) = \alpha\, P(x)\, \lambda(x).$$

The likelihood vector $\lambda(x)$ can no longer be directly obtained from the matrix $M_{y|x}$, however, but must reflect the matrix $M_{z|y}$ as well. Conditioning and summing on the values of Y, we can write

$$\begin{aligned} \lambda(x) = P(e \mid x) &= \sum_y p(e \mid y, x)\, p(y \mid x) \\ &= \sum_y p(e \mid y)\, p(y \mid x) \\ &= M_{y|x} \bullet \lambda(y). \end{aligned} \tag{7.3}$$

where $M_{y|x} \bullet \lambda(y)$ is the dot products of $M_{y|x}$ and $\lambda(y)$.

Eq.(7.3) may be generalized to the chains with arbitrary length.

In order to get *P(Cjkj|Ai)*, i.e., the probability that the *kj-th* level of the capability *Cj* is mastered, given that the tutoring unit *Ai* is taught, w use the distribution of notes of several real courses in the department of computer science in University of Montreal including computer architecture, database, data structure and computer graphics as *P(c1k1|An) P(c2k2|An) ... P(cmkm|An).*

We divide the note range 100 into 10 ranges, then the number of students whose notes fall in each range is viewed as the probability of the levels the student has mastered after the course is taught. Then we divide the note range 100 into 9 ranges, the number of students whose notes fall in each range is reviewed as the above probability. By continuing the division, a group of result probabilities is gotten (as shown in the figure 7.4).

0	0	1	2	3	4	5	6	7	8	9	10
1	0.0038	0.996									
2	0.0038	0.147	0.9496								
3	0.0038	0.105	0.203	0.688							
4	0.0038	0.049	0.0977	0.2894	0.5602						
5	0.0038	0.041	0.0789	0.1162	0.3233	0.4699					
6	0.0038	0.023	0.0489	0.0752	0.1617	0.3534	0.3346				
7	0.0038	0.019	0.0376	0.0564	0.0752	0.1955	0.3835	0.2256			
8	0.0038	0.015	0.0301	0.0451	0.0564	0.1053	0.1842	0.3571	0.203		
9	0.0038	0.014	0.0248	0.038	0.0513	0.0624	0.1152	0.1921	0.3199	0.1789	
10	0.0038	0.011	0.0226	0.0338	0.0451	0.0564	0.0677	0.1241	0.1992	0.282	0.1541

Figure 7.4 Conditional probabilities corresponding to different division of note levels

We can use these probabilities to compute a transition node's probable contributions to different capabilities.

3). Single Unit with Multiple Output Capability Levels

Assume that the transition activity $Ai1$ supports the acquisition of a set of capabilities C=$\{C_1, C_2, ..., C_m\}$ in which each capability C_j with a current mastering level C_{jk} (i.e., the capability C_j at the k-th mastering level), as shown in Figure 7.5, and the acquisition degree of each mastering level C_{jk} is characterized by the probabilities ($PC_{jk} \mid A_{i1}$) and $P(C_{jk} \mid \neg A_{i1})$, the likelihood ratio

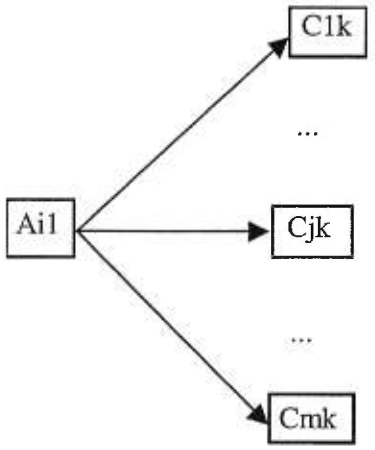


Figure 7.5. Tutoring unit A_{i1} supporting multiple capabilities

$$L(C_{jk}|A_{i1}) = \frac{P(C_{jk}|A_{i1})}{P(C_{jk}|\neg A_{i1})}, \tag{7.4}$$

Now we have the posterior odds

$$O(A_{i1}|C_{1k},\ldots,C_{mk}) = L(C_{1k},\ldots,C_{mk}|A_{i1}|O(A_{i1}). \tag{7.5}$$

Assume that each current mastering level C_{jk} depends only on whether the tutoring activity A_{i1} took place and is thereafter independent of other tutoring activities, we can write

$$P(C_{1k},\ldots,C_{mk}|A_i) = \prod_{l=1}^{m} P(C_{lk}|A_{i1}) \tag{7.6}$$

and

$$P(C_{1k},\ldots,C_{mk}|\neg A_i) = \prod_{l=1}^{m} P(C_{lk}|\neg A_{i1}) \tag{7.7}$$

which leads to

$$O(A_{i1}|C_{1k},\ldots,C_{mk}) = O(A_{i1})\prod_{l=1}^{m} L(C_{lk}|A_{i1}). \tag{7.8}$$

Example: Given tutoring activity A_{i1} and the output capabilities $C=\{C_1, C_2\}$ with the mastering levels $\{C_{13}, C_{26}\}$, $P(A_{i1})=0.25$, $P(C_{13}|A_{i1})=0.7$, and $P(C_{26}|A_{i1})=0.4$, the posterior odds

$$O(A_{i1}|C_{13},C_{26}) = O(A_{i1})\prod_{l=1}^{2} L(C_{lk}|A_{i1}).$$

$$= \frac{P(A_{i1})}{P(\neg A_{i1})} \cdot \frac{P(C_{13}|A_{i1})}{P(C_{13}|\neg A_{i1})} \cdot \frac{P(C_{26}|A_{i1})}{P(C_{26}|\neg A_{i1})}$$

$$=(0.25/0.75)(0.7/0.3)(0.4/0.6) =0.5185$$

This means that the posterior odds of A_{i1} given C_{13} and C_{26} is 0.5185.

4). Multiple Tutoring Units with Multiple Output Capability Levels

Eq.(7.8) is appropriate only for one tutoring unit in a transition node. However, in fact, each transition node includes multiple tutoring units. The following deals with this issue.

Let A_i=(A_{i1}, ..., A_{in}) stand for the vector of tutoring units in transition node Ti, the set of output capabilities related to Ti be C=$\{C_1, \ldots C_m\}$, the observation of current mastering levels on C be the vector LC=$\{C_{i1}, \ldots, C_{m1}\}$, as in Figure 7.6,

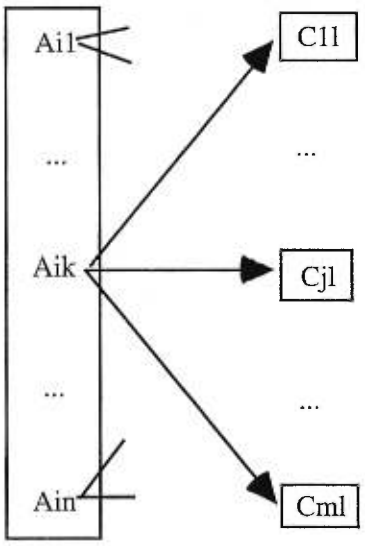


Figure 7.6. Multiple level activities relating to multiple output capabilities

by basic Bayes rule, we have

$$P(A_{ik}|C_{1l}, C_{2l}, \ldots, C_{ml}) = \alpha P(C_{1l}, C_{2l}, \ldots, C_{ml}|A_{ik})P(A_{ik}) \tag{7.9}$$

where α =1/*P(C1l, C2l, ..., Cml)* is a normalizing constant.

By Eq.(7.6), we obtain

$$P(A_{ik}|C_{1l}, C_{2l}, \ldots, C_{ml}) = \alpha \prod_{j=1}^{m} P(C_{jl}|A_{ik})P(A_{ik}) \tag{7.10}$$

which indicates the probability that the tutoring unit *Aik* has been successfully carried out given *C1l, ..., Cml*. By combining n levels of tutoring units in *Ai*, we obtain the belief vector

$$P(A_i|C_{1l}, C_{2l}, \ldots, C_{ml}) = \alpha P(A_i) \prod_{j=1}^{m} P(C_{jl}|A_i) \tag{7.11}$$

where

$$P(A_i|C_{1l}, C_{2l}, \ldots, C_{ml}) = [P(A_{i1}|C_{1l}, C_{2l}, \ldots, C_{ml}), \ldots, P(A_{in}|C_{1l}, C_{2l}, \ldots, C_{ml})],$$

$$P(A_i) = [P(A_{i1}), \ldots, P(A_{in})],$$

$$\prod_{j=1}^{m} P(C_{jl}|A_i)=[\prod_{j=1}^{m} P(C_{jl}|A_{i1}), ..., \prod_{j=1}^{m} P(C_{jl}|A_{in})], \text{ and}$$

α is obtained by $\sum_{jk=1}^{n} P(A_{ik}|C_{1l},\ldots,C_{ml}) = 1$

Example: Assume that

$A_i=(A_{i1}, A_{i2})$,

$C=\{C_1, C_2, C_3\}$,

$LC=(C_{15}, C_{22}, C_{37})$, where C_{jl} means that the current mastering level of C_j is l,

$P(A_i)=(0.3, 0.4)$,

$P(C_{jl} \mid A_{ik})$ is represented by the following matrix

$$M_{C_{jl}|A_i} = \begin{bmatrix} 0.3 & 0.2 & 0.4 \\ 0.4 & 0.3 & 0.45 \end{bmatrix},$$

we have

$P(A_i|C_{15}, C_{22}, C_{37})= \alpha(0.3, 0.4)\ (0.3*0.2*0.4, 0.4*0.3*0.5)$

$= \alpha(0.3, 0.4)\ (0.024, 0.054)$

$=(0.25, 0.75)$

which means that the probability that A_{i1} is executed is 0.25, and the probability that A_{i2} is carried out is 0.75.

For the case that there are n units in a transition node:

$P(A|\ c1k1, \ldots cjkj, \ldots, cmkm)$

$=P(A1, \ldots Ai, \ldots An|C1k1, \ldots Cjkj, \ldots Cmkm)$

$=[\ P(A1|\ C1k1, \ldots Cjkj, \ldots Cmkm),$

$P(A2|\ C1k1, \ldots Cjkj, \ldots Cmkm),$

.....

$P(An|\ C1k1, \ldots Cjkj, \ldots Cmkm)]$

$$= \alpha[P(A1), P(A2),\ldots, P(Ai),\ldots, P(An)] \begin{bmatrix} P(C1k1,\ldots, Cmkm \mid A1) \\ P(C1k1,\ldots, Cmkm \mid A2) \\ \ldots \\ P(C1k1,\ldots, Cmkm \mid An) \end{bmatrix}$$

$$= \alpha[P(A1), P(A2), \ldots, P(An)] \begin{bmatrix} \prod_{j=1}^{m} P(Cjkj \mid A1) \\ \prod_{j=1}^{m} P(Cjkj \mid A2) \\ \ldots \\ \prod_{j=1}^{m} P(Cjkj \mid An) \end{bmatrix}$$

$$= \alpha\left[P(A1)\prod_{j=1}^{m} P(Cjkj \mid A1) \quad P(A2)\prod_{j=1}^{m} P(Cjkj \mid A2) \quad \ldots \quad P(An)\prod_{j=1}^{m} P(Cjkj \mid An)\right]$$

The unit corresponding to goal capability levels is the index in the result matrix that has the maximal value.

Now the question is that, given a transition node containing multiple tutoring units and with a contribution *Lc*, how to compute *P (Ai|lj)*? We define a heuristic function to modify the table shown in figure 7.4. Let *Lui* be the maximal contribution level of *Ai*

n/m = i (i=1,2,3) / j (j=1,2,...,m-1)

So *j = (m*i)/n.*

Where *n*: number of tutoring units in the transition node,

m: number of output capability levels,

i: the index of tutoring unit, and

j: the index of output capability level.

We will use this function to the compute the maximal contribution level *j* based on *m,n,*and *i*.

Example: As seen in the figure 7.7

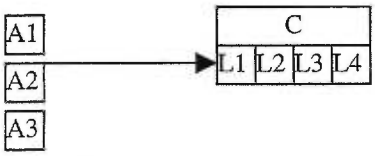


Figure 7.7 Computing conditional probability *P (Ai|Lj)*

When *i* = 1, the maximal contribution level *j* = (4*1) /3-> *j* = 2,

i = 2, the maximal contribution level j = (4*2)/3 -> *j* = 3.*i* = 3, maximal contribution level *j* = (4*3)/3 ->*j* = 4.

We cut some columns in front of the corresponding level row in the figure 7.4, and then add the cut value approximately to the remaining column s after cutting. If a value is greater than other values in the remaining column, then the column will get more cut value. The following figure 7.8 shows an example: a transition node with six tutoring units supporting a capability with maximal four levels.

	0	1	2	3	4
0	1				
1	0.3406	0.6594			
2	0.3406	0.6594			
3	0.1031	0.3055	0.5912		
4	0.0491	0.0981	0.2905	0.5623	
5	0.0038	0.0528	0.128	0.3501	0.4653
6	0.0038	0.0489	0.0977	0.2894	0.5602

Figure 7.8 Conditional probability *P(Ai|Cj)*

7.3.2.3 Identifying Tutoring Units in Transition Nodes

This section describes the algorithms for identifying tutoring activities. The algorithms are based on the Bayesian techniques introduced in section 7.3.2.2.

Give a transition node *T*, and a set of output capabilities of the transition node: *C*. The conditional probability vectors from the maximal tutoring unit to each capability in *C* are based on the rows in figure 7.9. The figure 7.9 shows the relations between the transition node *T* and each capability in *C*.

Let the tutoring units in *T* be *A1, A2, ..., Ai, ..., An, C={C1, C2, ..., Cj, ..., Cm}*; the set {*C1k1, C2k2, ..., Cjkj, ..., Cmkm*} refers to either known levels or goal levels of all the capabilities.

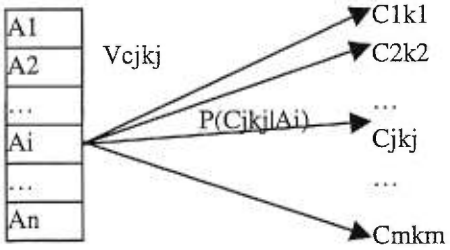


Figure 7.9 General relation between a transition node and output capabilities

In figure 7.9, *Vcjkj* is the conditional probability vector in which each element *P (Cjkj|Ai)* refers to the probability that the level *kj* of the capability *Cj* is acquired after the tutoring unit *Ai* is taught. In the figure 7.4, we can find the vector of condition probability *Vcjkj|An*. Based on the vector, the linear imitation method is used for other conditional probability vector. Then we get the conditional probability matrix *Mcjkj|A*: is

$$\begin{bmatrix} P(C1k1 \mid A1) & P(C2k2 \mid A1) & \ldots & P(Cmkm \mid A1) \\ P(C1k1 \mid A2) & P(C2k2 \mid A2) & \ldots & P(Cmkm \mid A2) \\ \ldots & \ldots & \ldots & \ldots \\ P(C1k1 \mid An) & P(C2k2 \mid An) & \ldots & P(Cmkm \mid An) \end{bmatrix}$$

Applying *McjkjlA* on Bayesian rules, we get the probability: *P (A|C1k1, ..., Cmkm),* that is, the probability that each tutoring unit is taught after *C1k1, C2k2, ..., Cmkm* are acquired. The tutoring unit that has the maximal probability value among all tutoring units in T is the boundary of tutoring unit having been taught. The algorithm 7.2 and 7.3 in Appendix D is used for identifying the boundary.

7.4 Sequencing Tutoring Activities by State-Driven Reasoning

In order to identify the availability of a tutoring unit at the current context in learning process, we define five kinds of states for tutoring units and sub-units, including *recommended, enabled, partlylEnabled, disabled* and *passed.* If a tutoring unit or sub-unit's state is *recommended,* the learner may learn the unit or sub-units. If a tutoring unit or sub-unit's state is *enabled,* it is not the best tutoring unit or sub-unit though it can be learned currently. If a tutoring unit or sub-unit's state is *partlyEnabled,* only parts of the unit or sub-unit may be learned. If a tutoring unit or sub-unit's state is *disbaled,* the learner cannot learn the unit or sub-unit. If a tutoring unit or sub-unit's state is *passed,* the unit or sub-unit has been passed successfully by the learner.

The following rules are both definitions of the states and the foundation of reasoning for identifying the state of a tutoring unit or sub-unit.

Ra1: if transition node *T* contains recommended unit, the *T*'s state is *recommended,*
Ra2: if the set of current recommended units is null and all units in the transition node are *enabled,* then *T* is *recommended,*
Ra3: if *T* is *enabled, T*'s successor contains more goals than other *T*s, then *T* is *recommended,*
Ra4: if all units in *T* are *enabled,* then *T* is *enabled,*
Ra5: if parts of units in *T* are enabled and some other unit is disabled, the *T* is *partlyEnabled,*

Ra6: if there is no *enabled* unit, the *T* is *disabled,*
Ra7: is all units are *passed,* the *T* is *passed,*
Ra8: if a unit *A* is *enabled* & *A* is a direct successor of last executed unit, then *A* is *recommended,*
Ra9: if *A* is *enabled* & *A* and the last executed unit have the same parent, then *A* is *recommended,*
Ra10: if *A* is *enabled* & both *A* and the last executed unit have same descendants, then *A* is *recommended,*
Ra11: if the required levels of all prerequisite capabilities for *T* are satisfied & all sub-units in *A* are available, then *A* is *enabled,*
Ra12: if the prerequisite levels of all input capabilities are greater than assigned percent, then *A* is *partlyEnabled,*
Ra13: if not all prerequisite levels of input capabilities are greater than assigned percent, then A is *disabled;*
Ra14: if the sub-unit *S* is available, *A* is *recommended* & *S* is the first sub-unit the learner does not learn, then *S* is *recommended,*
Ra15: if *S* is available, *A* is enabled & *S* is the first sub-unit that is not given to the learner, then *S* is *enabled,*

Ra16: if *S* is available, *A* is *partlyEnable* & *S* is the first sub-unit that is not given, then *S* is *partlyEnable*,
Ra17: if *S* is not recommended, not enabled, and not *partlyEnabled*, then *S* is *disabled*, and
Ra18: if *S* contains necessary resources and necessary runners, then *S* is available.
Ra19: if *S* associates necessary runners and necessary dynamic interaction functionality in the activity deliverer, then *S* is available.

7.5 Dynamically Transferring Tutoring Activities

In this section, we, first, propose the protocol between *VACT* and the tutoring delivery function. Then, the dynamic transferring process combining *Cnet*, *VACT* and the tutoring delivery function is described.

7.5.1 Basic Protocols between *VACT* and the Activity Delivery Module

In *VITCAM*, we attempt to provide the adaptability that can combine various tutors with different abilities. When we define *VITCAM*, the tutoring delivery module or tutor is considered as an open module. The ability of tutoring delivery module concerns some difficult issues in ITS, such as, diagnosing learner errors and evaluating learner responses.

We view the following tutor abilities as the protocols among *Cnet*, *VACT* and the activity delivery module (or called a tutor), except the ability to dynamically create interactions in the activity delivery module.

- The tutor can only display resources to learners and simply give a yes/no evaluation to learner responses;
- The tutor can determine what capabilities that are not in current course are needed for remedying the learner lacks;
- The tutor can determine what capability levels that are in current course are needed for remedying the learner lacks;
- The tutor can determine what alternative resource groups in current tutoring activities are needed for remedying the learner lacks.

In next section we introduce the dynamic transference process of tutoring activities based on the four protocols.

7.5.2 Dynamic Transference Process of Tutoring Activities

In *VACT*, the transference process of tutoring activities is dynamic; i.e., the course creation module and the sequencing module may be re-activated based on certain feedback information provided by the tutoring delivery module. The figure 7.10 shows the dynamic process.

Based on the input (*Tnet)* of *VACT* and a student's profile from student model, the *Course Creator* creates multiple alternative courses for the student, then *Anet Creator* is activated to evaluate and recommend courses. As soon as the student selects a course, *Anet Creator* sequences all tutoring activities in the selected course. The tutoring activity sequence is passed to the tutor module: *Activity Deliverer*. The tutor, then, selects a recommended tutoring unit interactively with the student and passes the selected tutoring unit to the student. The output of the tutor contains the following possibilities:

- The student passes the tutoring unit successfully;
- The student fails to the unit;
- The student lacks some capability levels in the current course;
- The student misses capabilities out of the current course.

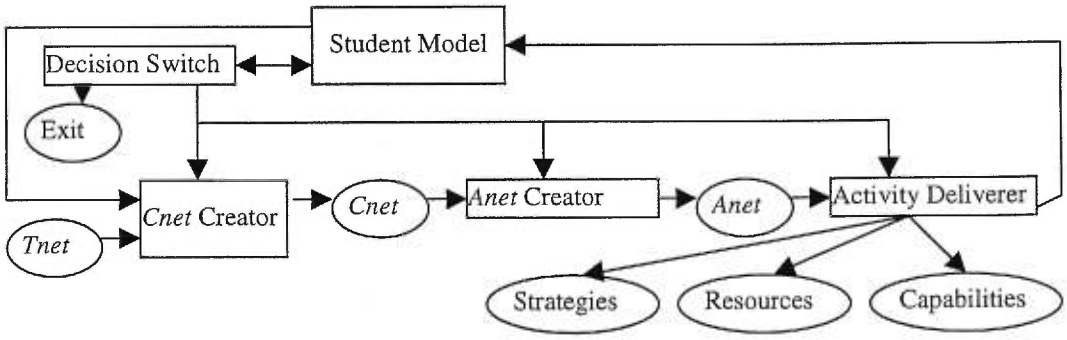


Figure 7.10 Dynamic Transference of Tutoring Activities

If the student passes the tutoring activity successfully, the *decision switch* determines the next step: either exiting when the student's all goals are achieved, or activating *activity deliverer* to give next recommended activity, or creating new recommended activities when no recommended activities. If the student fails to the unit and there are alternative available resource groups, then the system changes alternative resource groups in the tutoring unit to deliver the unit again.

If the student lacks some previous capability levels in the current course, the *Anet Creator* is activated to re-compute the necessary tutoring units in the course, and then passes the activities to the *activity deliverer.*

If the student misses some capabilities out of the current course, the *Cnet Creator* is activated to create the remedy course for the missed capabilities, then the tutoring activities in the remedy course are sequenced, and are passed to the activity deliverer to offer the remedy activities first. If the student fails to a tutoring unit, and the *activity deliverer* has no ability to determine what the student lacks, the sequenced tutoring activities may be re-learned by the student.

7.6. Visualization of *VACT*

VACT model can display the evaluated course sequence and the recommended tutoring unit sequence, dynamically map the visual element states to colors and navigate to learners.

- Displaying the evaluated course sequence

As soon as the *VACT* model creates alternative courses, the system can dynamically display the courses ordered by their utilities. Meanwhile, a student may select one of the recommended courses as his current course

- Dynamically display the recommended tutoring unit sequence

The system can dynamically display currently recommended tutoring unit sequence that is obtained by a group of reasoning rules according current state of the learner.

- Mapping element states to colors and navigating to learners

As soon as necessary tutoring units are identified, the states of all visual cells are initially determined. Various colors are used for possible states of capabilities and their levels, and some other colors are used for possible states of visual cells in transition nodes.

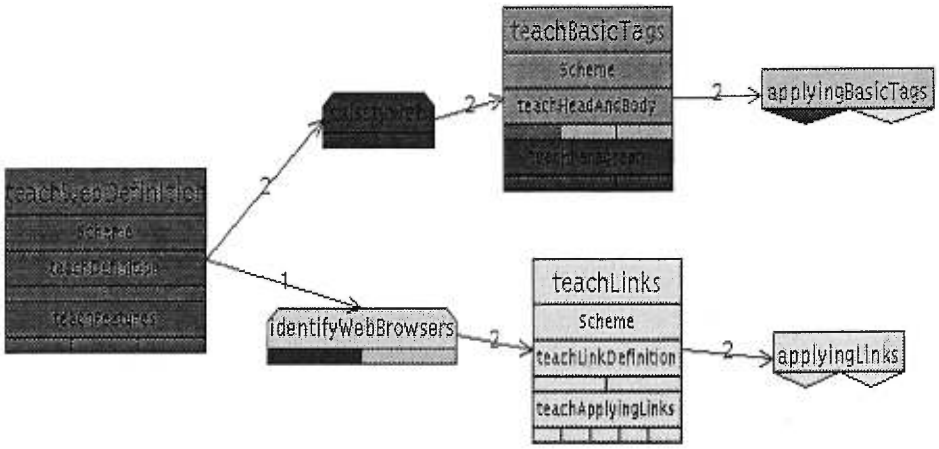


Figure 7.11 Dynamic state navigation

Since a dozen of states and colors are displayed on screen, novice users may have difficulty to recognize these states and colors. A state-color bar is provided in the information palette. The user can switch on the state-color bar at any time for the navigation to learning process.

In the learning process, the states of visual cells will real-time changed with the change of the student states. *VACT* model dynamically displays the corresponding changes of visual element states to supply navigation to learners.

The figure 7.11 shows an example of state navigation. The learning goal is the capability "applyingBasicTags". The currently recommended tutoring unit is "teachHeadAndBody".

7.7 Conclusion

This chapter deals with the dynamic management of capability transitions including approaches for evaluating courses, sequencing tutoring activities, remedying student misconceptions, and visual navigation by learners.

A measure, *utility*, is defined for comparing the course adaptability to the current student, which is based on the students' preferred resource types, media types and cognitive strategies, as well as the needed efforts for learning goals. By comparing the utilities of alternative courses, a course sequence sorted by *utility* is obtained. The course with the maximal utility is the recommended course.

In order to create courses that are adaptable to a particular student, the necessary tutoring units are identified by Bayesian techniques that can tackle the uncertainty between the output capabilities and tutoring activities.

A state-driven reasoning approach is proposed for sequencing tutoring activities in a selected course. Six states are defined for each tutoring activity (either a transition node, or a tutoring unit, or a sub-unit), that is, *recommended*, *enabled*, *partly enabled*, *disabled*, *unconcerned*, and *passed*. A group of reasoning rules is proposed for the dynamic state change of each visual cell.

One feature of *VITCAM* is the dynamic transition of tutoring activities. After a tutoring unit is transferred the system can make decision for the next step, either exiting the system, or changing alternative resource groups, or re-sequencing the tutoring activities in the selected course, or re-creating new remedy courses.

The visualization of *Anet* consists of displaying sorted course sequence, displaying currently recommended tutoring units, mapping states of visual cells to colors, and the real-time navigation by the state-driven reasoning

In the next chapter we present a prototype developed so far that supports the feasibility of the proposed curriculum based ITS.

Chapter 8

Prototype and Application

In this chapter we, first, briefly describe the prototype of *VITCAM* system including its system architecture and principal functionality. Based on the prototype, an example of curriculum for teaching HTML is introduced. The prototype of *VITCAM* model is implemented with 120,000 lines of Java codes (about 700 classes).

8.1 Architecture of the *VITCAM* Prototype

The Figure 8.1 shows the system architecture of *VITCAM*. This architecture includes three parts: a visual component creator, an authoring workshop and a leaning workshop.

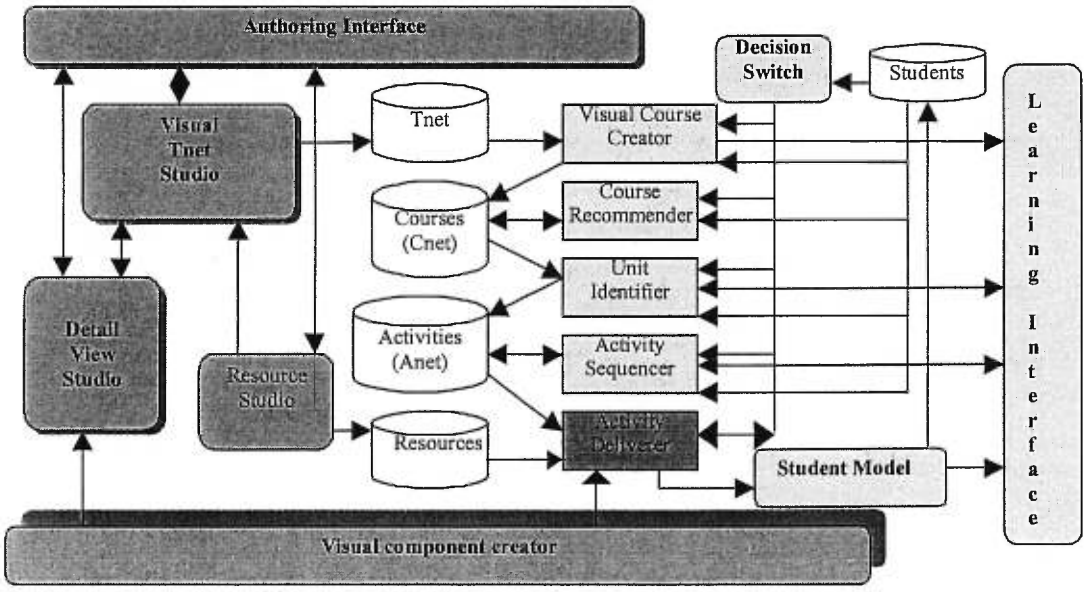


Figure 8.1 Architecture of *VITCAM* prototype

8.1.1 Visual Component Creator

Visual Component Creator (Figure 8.2) is an essential module in *VITCAM* prototype. It is a program library containing dozens of component creation programs. The component library is used for creating four categories of visual components: a capability node group, a transition node group, a resource group and a

link group. A great number of operations are provided in each kind of components to support creating, modifying and managing various visual and internal attributes as well all kinds of manipulations.

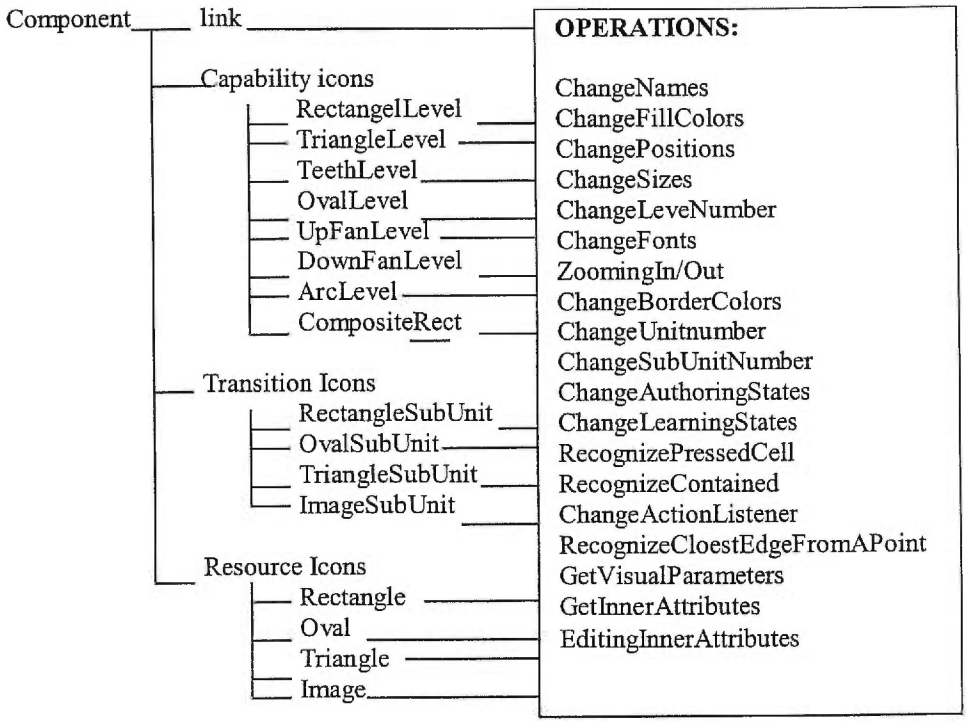


Figure 8.2 Visual components and the operations applied on them

Each group of visual components includes some node templates with different shapes. These shapes can be used to indicate the types of nodes. For example, we use a kind of shapes to represent concepts and use another kind of shapes to represent rules. The visual component library contains another kind of icons for resources (which are not used yet in current prototype). Node shapes can be used to distinguish different resource types. There are two kinds of transition nodes; currently, they are used to identify an intermediate transition node and a leaf transition node in a capability transition network. The inks in *VITCAM* include prerequisite links and output links. We use the types of source and destination nodes to indicate the types of links, for instance if the source of a link is a capability node, we say that this link is a prerequisite link.

The operations on these visual components fall into two categories: getting information and setting/updating information. These operations will affect the inner attributes of the nodes as well as their visual properties. Each operation listed in figure 8.2 consists of a group of concrete methods; for example, the operation "changeSizes" includes "changeTotalWidth, changeTotalHeight, changeCellWidth, changeCellHeight, changeTotalBounds, and changeCellBounds." Many operations in the authoring environment and learning environment are based on these basic operations on visual elements. The coming sections explain in detail the above three parts of *VITCAM*.

8.1.2 Authoring Workshop

In the Authoring Workshop of *VITCAM* all visual interactive activities between a curriculum author and the system take place in the authoring Interface. A curriculum author can directly activate one of three studios including the *Tnet* Studio, the Resource Studio and the Detailed View Studio. The visual *Tnet* Studio is used to create, organize and layout the domain capability transition networks. A curriculum author can use the Resource Studio to develop text resources and connect images to text resources. The author can also open a dialog window (detailed viewer) for each visual cell in nodes to edit the internal attributes of the cell.

8.1.3 Learning Workshop

The learning workshop is more complicated than the authoring workshop. There are four basic databases, including course network base, an activity network base, a resource base and a student profile base.

There are seven modules in this workshop; that is, a *visual course studio*, a *course recommender*, a *unit identifier*, a *unit sequencer,* an *activity deliverer* (opening module), a *decision switch* and a *learner interface*. With the course studio, a learner may visually set known capabilities and learning goals (i.e. capabilities and capability levels to learn). The learner, then, may activate the *course creator* to create multiple alternative courses covering his/her learning goals. Meanwhile, the course creator can display all generated courses and their details to let the learner select his or her preferred course. The *course recommender* can recommend an individualized course that is currently the most appropriated course for the learner. The learner can accept the recommended course or select anyone else he currently preferred as his current course to learn.

The upcoming sections describe in detail the functionality of the authoring environment and the learning environment.

8.2 Authoring Environment of *VITCAM*

The authoring environment of *VITCAM* consists of seven groups of functionality as shown in figure 8.3:

-File access
-Local view editor
-Detailed view editor

-Resource base editor

-Component templates

-System state navigator

-Student profile viewer

Their functionality is briefly described as follows.

- File Access

The File Access Group provides all file operations. They include creating a new capability transition network *Tnet*, opening a *Tnet*, saving a *Tnet*, and saving a *Tnet* in different paths or under a different name.

- Local View Editor

The local view editor is used to create and organize the capability transition network *Tnet*. A visual node is created and displayed on the screen based on twice mouse clicking, one for selecting a node template and the other for positioning the component on screen, as the editor's state is "Add Node". A link can be created to connect two visual components by clicking the source and then the destination node. To delete a node or link, the user can simply click the component to delete. The system can automatically insure the consistency of the deleting operation, such as, if a node is deleted, all its input links and output links should be deleted, too. The user can also layout the *Tnet* on screen by simply dragging.

- Detailed view Editor

With this editor, a curriculum author can edit visual properties and inner attributes of visual cells. In addition to some visual properties that are automatically changed by the system such as sizes, colors, etc., the curriculum author can change the number of sub-cells, for example the number of levels of a capability, the number of tutoring units in a transition node. Each kind of visual cells has an inner attribute editor for editing the internal attributes related to this kind of cells. These editors consist of the overall capability attribute editor, the capability level editor, the overall transition attribute editor, the scheme editor, the tutoring unit attribute editor, and the sub-unit attribute editor.

A group of sub-editors can be used for visually inserting or deleting capability levels, tutoring units and sub-units. In order to attach resource groups to sub-units, with the detailed view editor, the author can open a

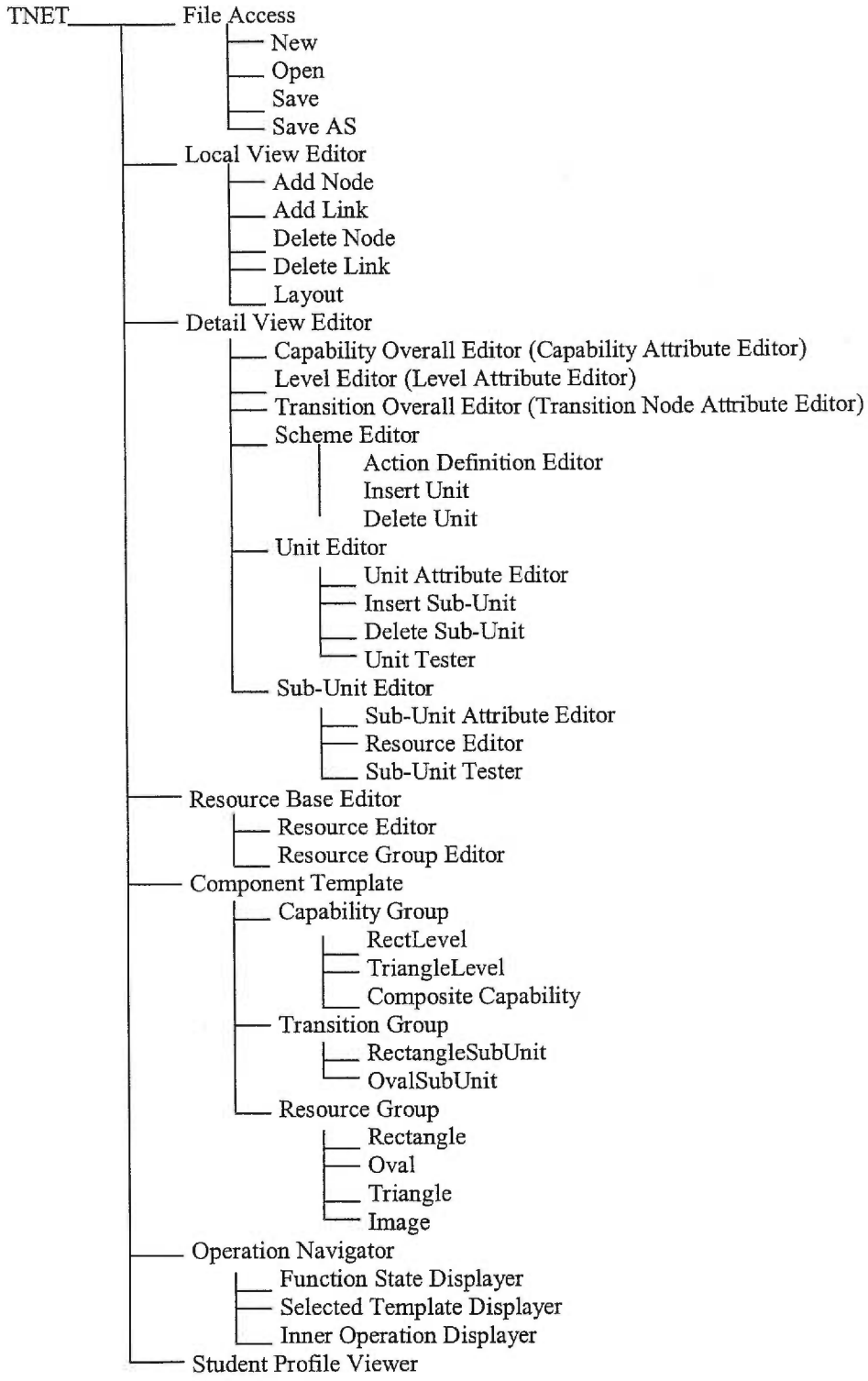


Figure 8.3 Functions of Authoring Environment in *VITCAM*

database of resource groups, visually click the wanted resource group, and the clicked resource group will be automatically attached the selected sub-units.

With the detailed view editor of tutoring units or sub-units, a curriculum author can test tutoring activities in tutoring units and sub-units. Both tutoring units and sub-units are associated with a *runner* that can deliver the real tutoring acts. If the necessary resource groups have been attached to sub-units, the runners of tutoring units or sub-units will present the attached resources. The interaction activities between the learner and the resources take place on the interaction windows of runners.

- Resource Base Editor

Currently, *VITCAM* prototype supplies dialog-based editors for creating resources and resource groups. With these two editors, a curriculum author can generate resources and resource group databases.

- Component Templates

The visual component templates contain various visual components for constructing capability transition networks (*Tnet*). They are divided into four groups: the capability group, the transition node group, the resource group and the link group. When an author wants to create a new node, he simply clicks the desired template (if the currently selected template is not he/she wants) and then clicks the position to display it on screen. The new node corresponding to the selected template is created and displayed. The author, then, can organize the transition network using tools for laying out and for deleting components. Finally, the author can edit visual properties and inner attributes of the transition network with the local view and detailed views.

- System State Navigator

The system state navigator dynamically displays current system states. These states include the selected template (for a curriculum authoring session), the current system tool, and the current inner operations (which will for instance report what the system is doing).

- Student Profile Viewer

An author can use the student profile viewer to get the current student information stored in the system.

- User Interface of Authoring Environment

The figure 8.4 shows the user interface of the authoring environment. In this user interface, the toolbox at the top of the window consists of all operations the author can select. The component templates are placed at the left of the window. The state bar at the bottom is the system state navigator. The central area of the window is the user workspace where the real authoring and learning take place.

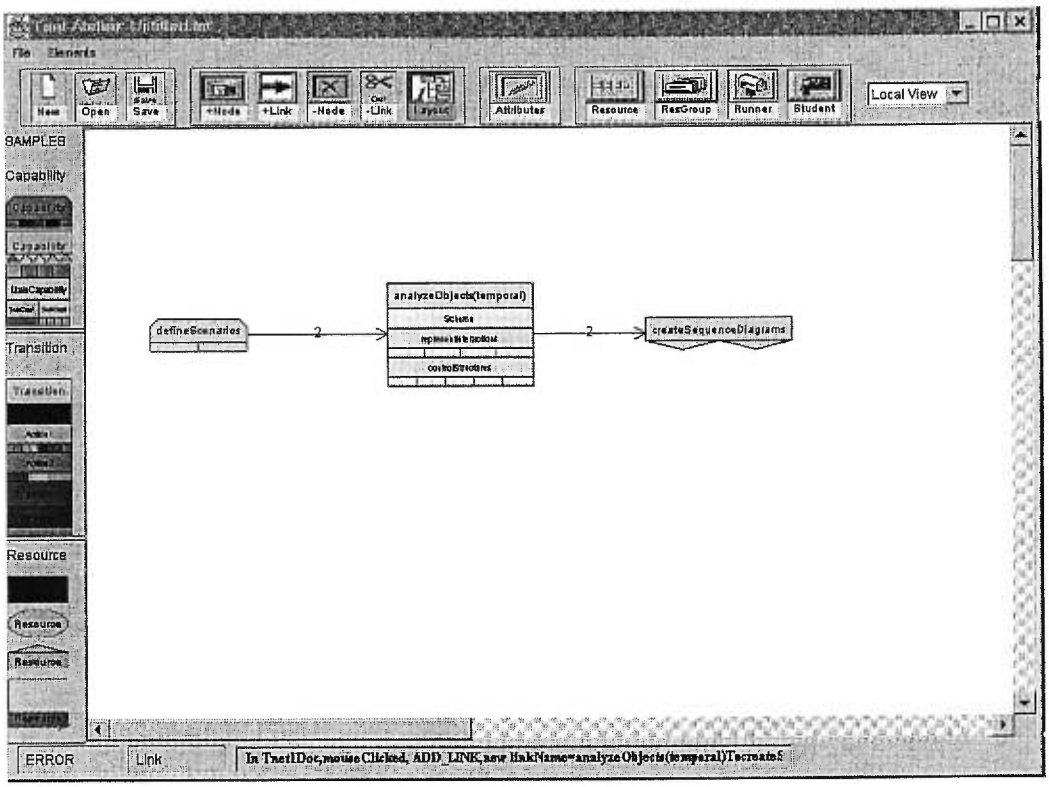


Figure 8.4 User interface of the authoring environment

8.3 Learning Environment—*VCOURSE* and *VACT* Prototype

The *VCOURSE* and *VACT* prototype provides all preparations for the activity delivery module (or tutor) in tutoring system. These preparations facility viewing the details of domain transition networks, setting known capabilities and their levels, setting goal capabilities and their levels, creating multiple alternative courses, recommending courses, sequencing tutoring activities, and visually navigating to a learner. Meanwhile, the prototype also supports remedying learners' missing by re-creating courses, or re-sequencing tutoring activities based on the feedback information from the activity delivery module.

The functional architecture of the *VCOURSE* and *VACT* prototype is shown in figure 8.5. The prototype contains more inner operations than the *VTRANS* prototype. We summarize the main functionality as follows.

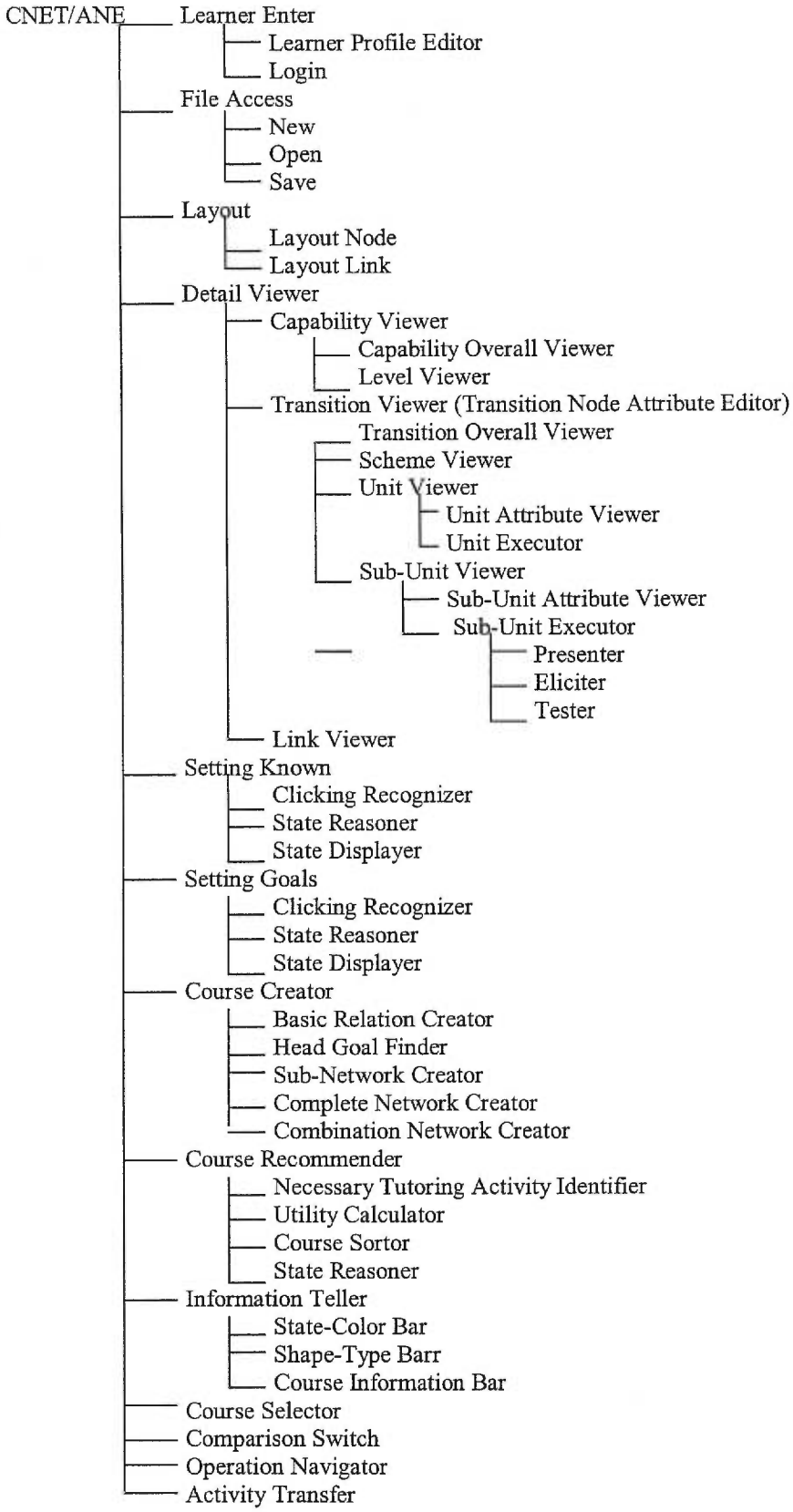


Figure 8.5 Functionality modules of *VCOURSE* & *VACT* prototype

- Leaner initialization

Prior to a learning session, the student must provide his or her profile to the system. This can be done through two processes: the login entrance and the learner profile entrance. When a learner uses the system at the first time, he/she interactively establishes his/her individual profile. In later learning sessions, the learner can view his learning history and update his individual information profile.

- File Access

This part provides the operation for opening a new capability transition network, opening a selected course, and storing selected course.

- Layout

Since a course is a sub-network of a *Tnet*, the layout of a created course is based on the original layout in the *Tnet*. The learner may re-layout the selected course by dragging.

- Detailed Viewer

The detailed viewer is supplied to help the learner understand the overall structure of the domain knowledge and explain what the capabilities and their levels are.

With the detailed viewer of transition nodes, learners can freely explore tutoring activities. Four kinds of viewers are provided for transition nodes: the transition overall viewer, the scheme, tutoring unit and sub-unit viewers. The learner can get the overall information of a transition node through the transition overall viewer. Similarly, the scheme viewer gives the detailed definition the transition node. The unit and sub-unit viewers consist of their attribute viewers and tutoring runners. The attribute viewer may be used to obtain detailed information on a tutoring unit or sub-unit. The tutoring runner offers an environment for freely exploring some particular teaching sequences present in the transition node.

- Setting Known Capabilities

By the functionality of setting known capabilities, a learner can visually set his known capabilities and their levels. There are three operations involved in setting known capabilities: a *clicking recognizer,* a *state reasoner* and a *state displayer*. When a learner clicks a visual cell in a capability node, the *clicking recognizer* is activated to determine exactly which visual cell is clicked. And then, the *State Reasoner* infers the states of the overall cell of the clicked capability and all its levels based on the reasoning rules

defined in section 6.4. After the states of all visual cells in the clicked capability are determined, the *State Displayer* changes the colors of visual cells in the clicked capability to reflect the corresponding states.

- Setting Learning Goals

The process of setting learning goals is similar to that of setting known capabilities. The only difference is that they use their own reasoning rules defined in section 6.4.

- Course Creator

The course creator is composed of five principal modules: a basic relation creator, a head goal finder, a sub-network creator, a complete network creator and a combination network creator.

The *basic relation creator* generates all necessary basic node relations. Owing to the two kinds of nodes in a *Tnet*, the capability node relations and the transition node relations should be found respectively. These relations include predecessor relations and ancestor relations.

The *head goal finder* attempts to divide a network into several connected sub-networks. The root of each sub-network is a learning goal and each sub-network contains a group of learning goals that are ancestors of the root nodes. With such division mechanism, there is no learning goal that across two connected sub-networks.

After all head goals are found, the *sub-network creator* generates all possible sub-networks of each head goal.

As soon as the complete sub-networks of each head goal are achieved, alternative courses can be obtained by combining the complete sub-networks of each head goal.

Figure 8.6 ~ 8.10 show an example of course generation. Figure 8.6 is a capability transition network *Tnet* for teaching part of UML. The selected learning goal is the capability "createSequenceDiagram" to the first level. Figure 8.7~10 show four created alternative courses.

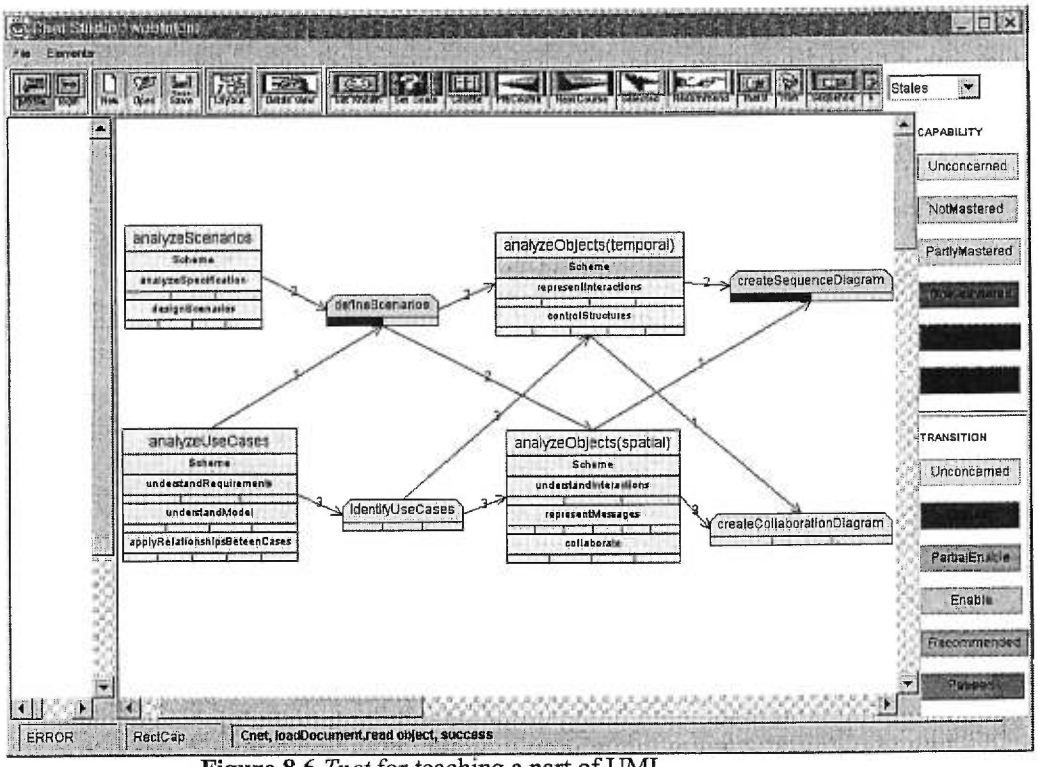


Figure 8.6 *Tnet* for teaching a part of UML

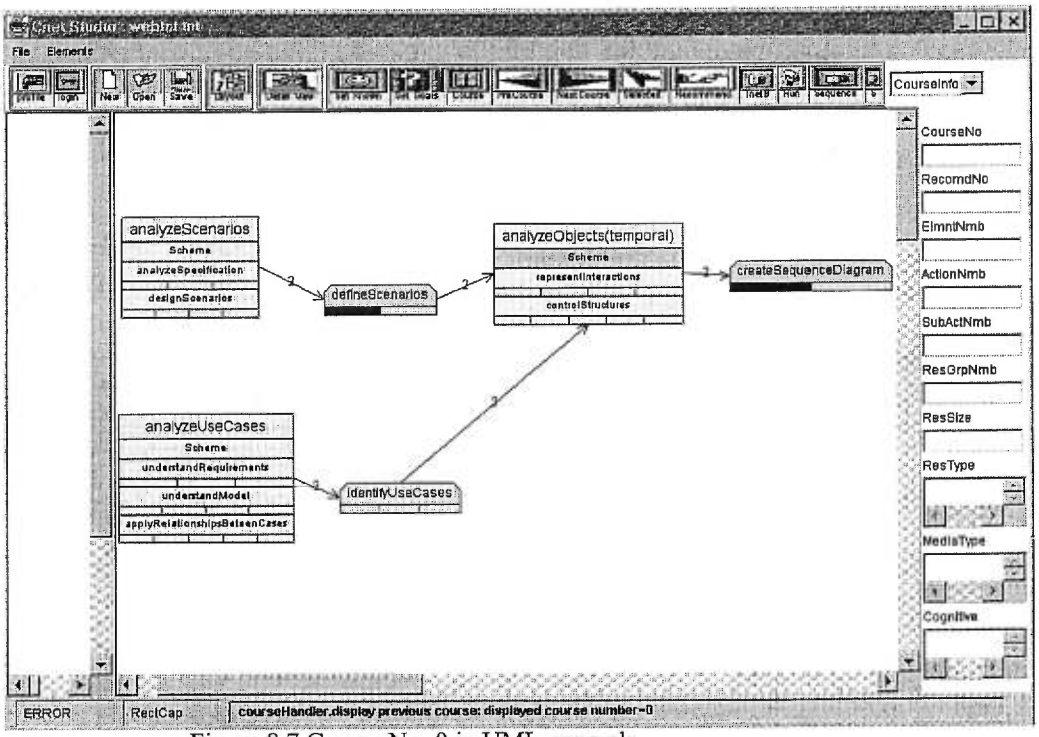


Figure 8.7 Course No. 0 in UML example

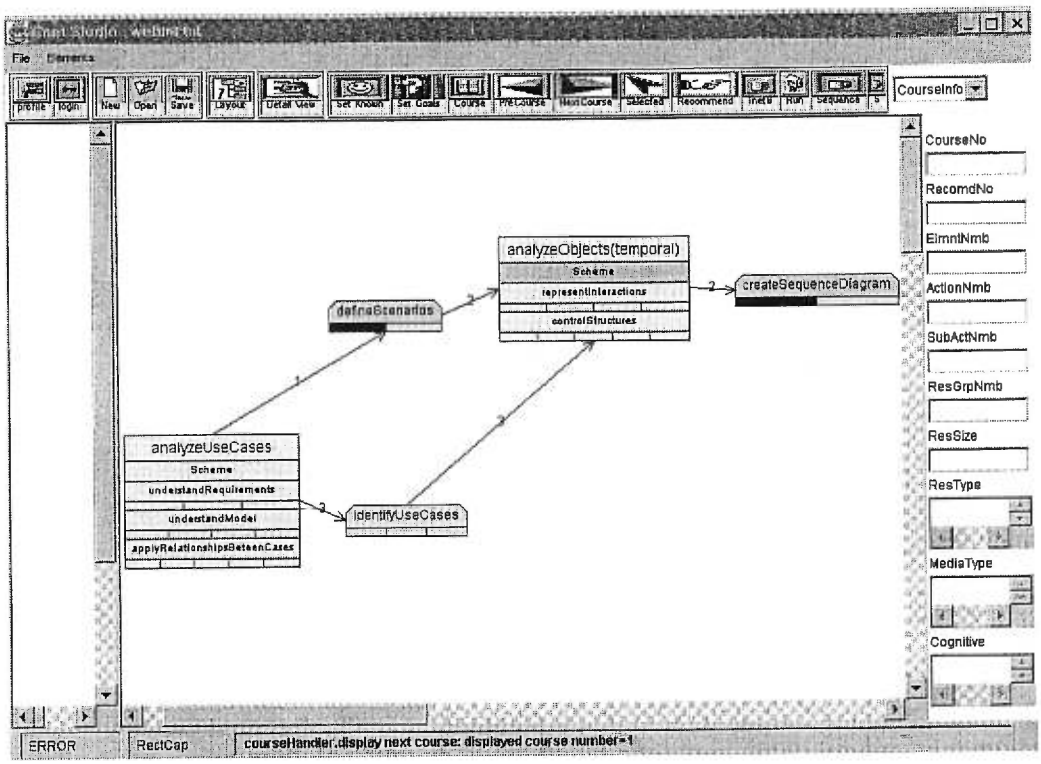


Figure 8.8 Course No. 1 in UML example

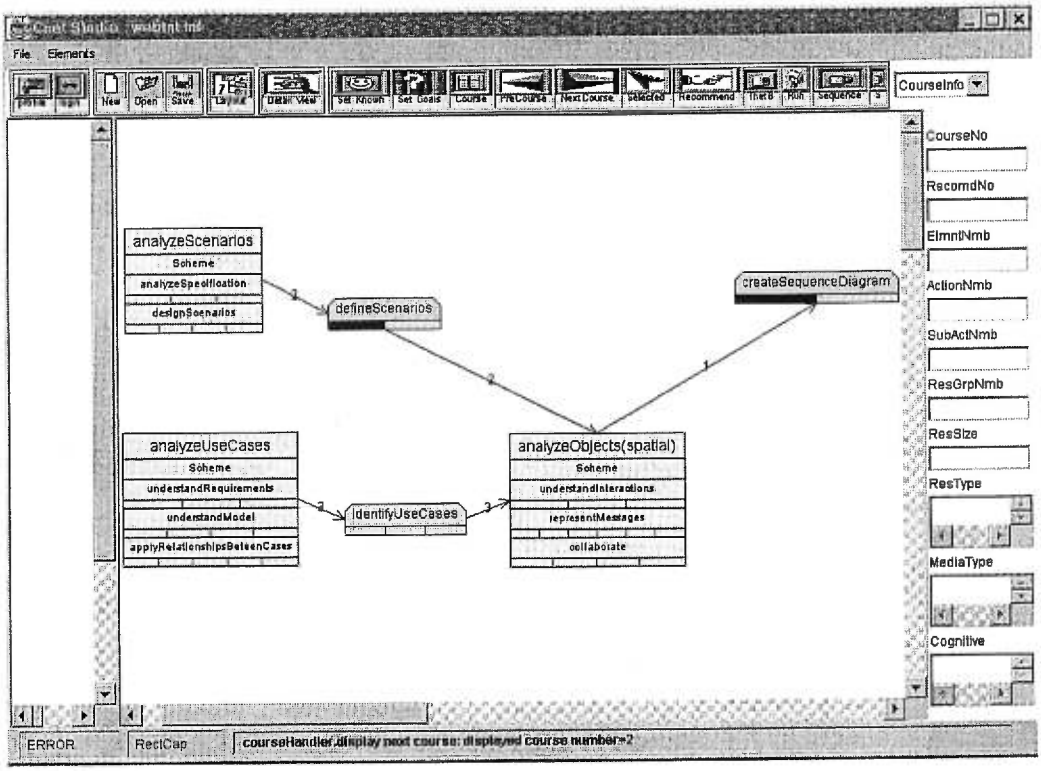


Figure 8.9 Course No. 2 in UML example

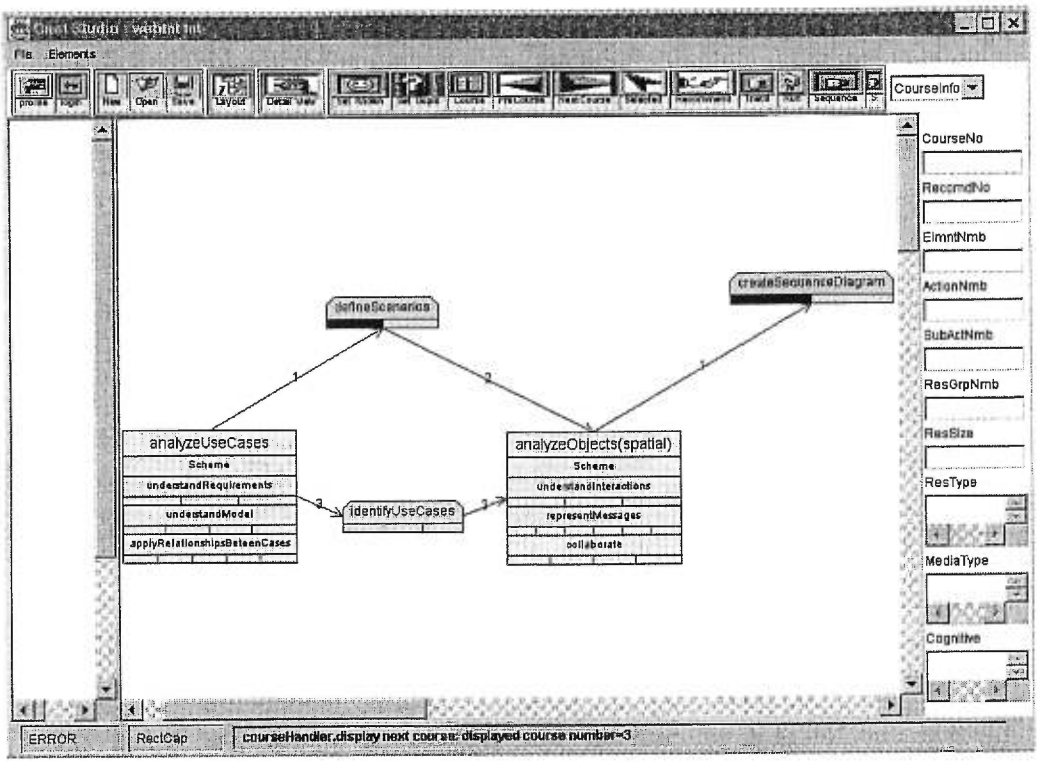


Figure 8.10 Course No. 3 in UML example

- Course Recommender

There are four steps for recommending courses, that is, identifying necessary tutoring units, calculating utilities of all alternative courses, sorting courses with utilities and inferring activity states.

The *tutoring unit identifier* compute the range of the tutoring units in a transition node based on the known levels and the goal levels of all output capabilities. The central approach for computing necessary tutoring activities is based on the Bayesian techniques described in chapter 7.

In order to evaluate the adaptability of courses, the *course recommender* module computes the utilities of all necessary tutoring activities in all courses by using the approach defined in chapter 7. Combining the utilities of all necessary tutoring units in the course may get the utility of each course. All courses are sorted by their utilities. The course with the maximal utility is the recommended one.

The *state reasoner* is used to identify the state of all transition nodes, tutoring units and sub-units in the selected course. A group colors are adopted for displaying the states of all tutoring activities. The learner can choose currently recommended tutoring activities to learn.

- Information Teller

Since numerous visual properties are adopted in *VCOURSE* and *VACT* prototype, novice users need indicators for recognizing the visual properties. The *information teller* consists of three information bars: a state-color bar, a shape-type bar and a course information bar. A learner can switch on any one of the three information bars as the current information bar. The state-color bar displays the mapping from the states of visual cells to colors. The shape-type bar gives the relationships between component shapes and component types. The course information bar displays the overall information contained in currently displayed course, such as node number, resource number, media types, cognitive strategies, resource sizes, etc.

- Course Selector

When a learner activates the course selector, the current displayed course is considered as the selected course to learn. All information in the selected course is stored to the learner's profile.

- Decision Switch

The decision switch is the handler for the feedback information provided by the activity deliverer module in tutoring system. It switches the control of *VCOURSE* and *VACT* prototype either to the *course creator* or to the *state reasoner* in the *course recommender* to activate the corresponding modules.

- System State Navigator

Its functions are similar to the system state navigator in *VTRANS* prototype.

- Activity Deliverer

The activity deliverer module is viewed as an external module of *VITCAM* prototype. The protocol between the module and *VITCAM* supports both simple activity deliverer and powerful activity deliverer from the point of view of diagnosis abilities.

8.4 Application: Example for Teaching HTML

An example for teaching HTML is developed with *VITCAM* prototype. In this section, we first, describe the identification of capabilities in HTML. And then the capability transition network, *Tnet*, and the course network *Cnet* as well as the activity network Anet are dealt with.

8.4.1 Identification of Capabilities

Based on the knowledge structure of HTML. Seventeen capabilities are identified as shown in figure 8.11. Most of the capabilities are composite capabilities; that is, each of which contains several small capabilities. For example, the capability "Applying basic tags" includes "Applying Head Tag", " Applying title tag", "Applying body tag", and " "Applying paragraph tag". For the capabilities without too much levels, we combine them together to get a composite capability, which simplifies the global structure of domain transition network in a great extent.

StatingWeb:3
1. statingWebDescription
2. statingWenBasicFeatures
3. statingWebAdvancedFeatures

statingWebBrowsers:3
1. statingBrowsersDescription
2. statingBrowsersjob
3. statingPopularBrowsers

classifyingURLs:3
1. statingURLDefinition
2. identifyingURLStructure
3. identifyingPopularURLs

statingHTMLIntroduction:3
1. classifyingHTMLDefinition
2. statingHTMLBasis
3. identifyingURLStructure

applyingBAsicTags: 2
1. applyingBasicStructureTags
2. applyingAdvancedBasicTags

applyingLinks:4
1. creatingLinks
2. linkingLocalDocuments
3. linkingOtherWebDocuments
4. linkingSpecificPlaceWithinDocument

applyingLists:3
1. applyingBasicListTag
2. applyingTypicalLists
3. applyingAdvancedLists

applyingTables:4
1. applyingBasicTags
2. aligningTables
3. spanningRowAndColumn
4. applyingNescapExtension

applyingOtherTags:1
1. applyingOtherTags(formating, horzonRule,BR,BlackQuoted, address, fontSize,background)

applyingMultimedia:4
1. usingImages
2. usingExtensionImages
3. makingImageBetter
4. applyingExtendedMedia(extended Image, sound, video)

organizingWeb:5
1. stating main steps
2. settingGoals
3. gettingMainTopics
4. organizingAndNavigating
5. usingStoryBorder

applyingWebServer:5
1. statingServerBasis
2. findingAServer
3. organizingAndInstallingYourHTMLFiles
4. applyingBetterServerAdministrativeRules

applyingCGI:6
1. classifyingCGI
2. statingUserConditionsOfCGI
3. settingCGIOnServer
4. statingCGIScriptBehavior
5. applyingInteractiveSerach
6. applyingSpecialScriptOutput

applyingForms:6
1. applyingFormBasis (input tag, script to present form)
2. layoutingSimpleForms
3. applyingTextInputFields
4. layoutingAdvancedForms
5. identifyingImageMap
6. creatingImageMap

designingSimpleWeb:2
1. applyingBasicRules
2. designingSimpleWeb

designingGeneralWeb:4
1. applyingDesignRules
2. designingSimpleExamples (basic tags, links)
3. designingWebWithListsAndTables
4. designingWebWithOtherTagsAndServer

designingBusinessWeb:4
1. applyingDesignRules
2. designingSimpleExamples
3. designingGeneralBusinessWeb
4. designingComplicatedBusinessWeb

Figure 8.11 Identified capabilities and levels in HTML

8.4.2 Constructing Capability Transition Network—*Tnet*

Identifying tutoring activities in a capability space is an action that needs human intelligence. *VITCAM* provides an environment for curriculum authors to construct the capability transition network, *Tnet*. The capability transition network, *Tnet*, for HTML is shown in figure 8.12, in which the circle nodes indicate the capabilities and the rectangle nodes refer to the transition nodes. The number with the input links of transition nodes are the required minimal capability level, and the numbers with the output links of transition nodes are the maximal contribution levels to the output capabilities.

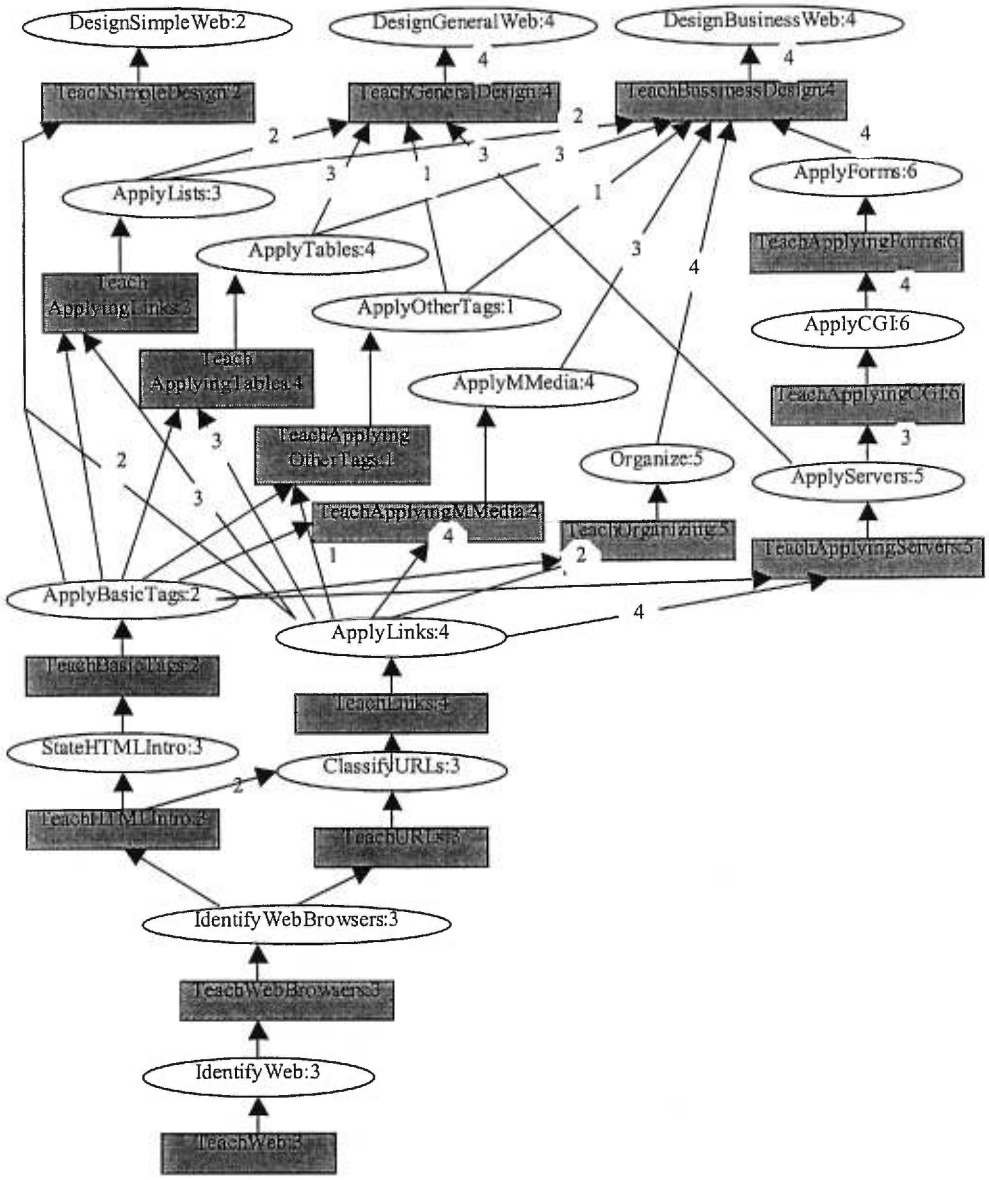


Figure 8.12 *Tnet* of HTML

The details of tutoring units in each transition node are given in Appendix A. Figure 8.13 shows the details of the transition node: "teachApplyingServers". Figure 8.14~15 redraw the screen appearance of this example as in chapter 5.

Transition node: "teachApplyingServers"
- Unit 1: teach server introduction
  - Sub-unit 1: present definition of servers
  - Sub-unit 2: present the job of servers
- Unit 2: finding a server
- Unit 3: installing server software
- Unit 4: organizing and installing your HTML files
- Unit 5: making better server administration & design

Figure 8.13 Details of the transition node "teachApplyingServers"

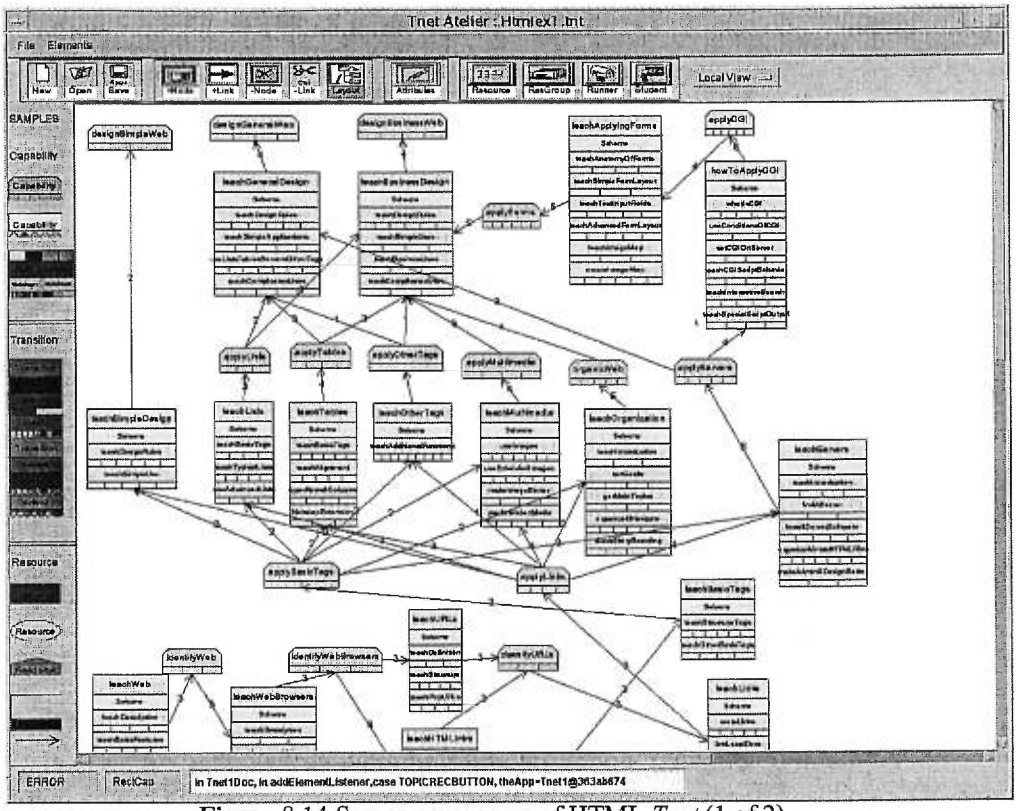


Figure 8.14 Screen appearance of HTML *Tnet* (1 of 2)

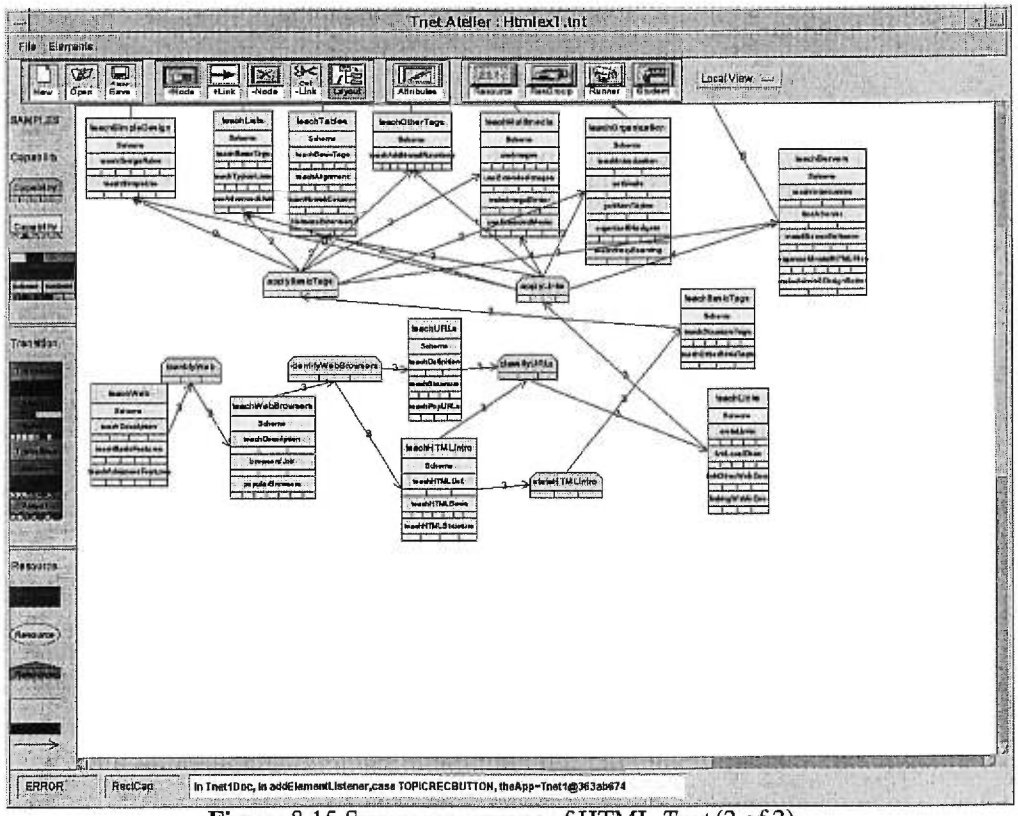


Figure 8.15 Screen appearance of HTML *Tnet* (2 of 2)

8.4.3 A *Cnet* in HTML

The following is an example of course network corresponding to certain input.

Given that a learner mastered the capabilities:

ApplyingList at level No. 3

ApplyingOtherTags at level No. 1

ApplyingBasicTags at level No. 2

And the learner's goals are:

DesigningGeneralWeb to level No. 3

ApplyingMultimedia to level No. 2

The following are the steps and results of creating courses:

1. All possible known capabilities:
   ApplyingList at level No. 3
   ApplyingBasicTags at level No. 2
   ApplyingLinks at level No. 3
   ApplyingOtherTags at level No. 1
   StatingHTMLIntroduction at level No. 3
   StatingWebBrowsers at level No. 2
   StatingWeb at level No. 3
   ClassifyingURLs at level No. 2

2. Removing redundancy and invalid elements, and create all courses (figure 8.16~17).

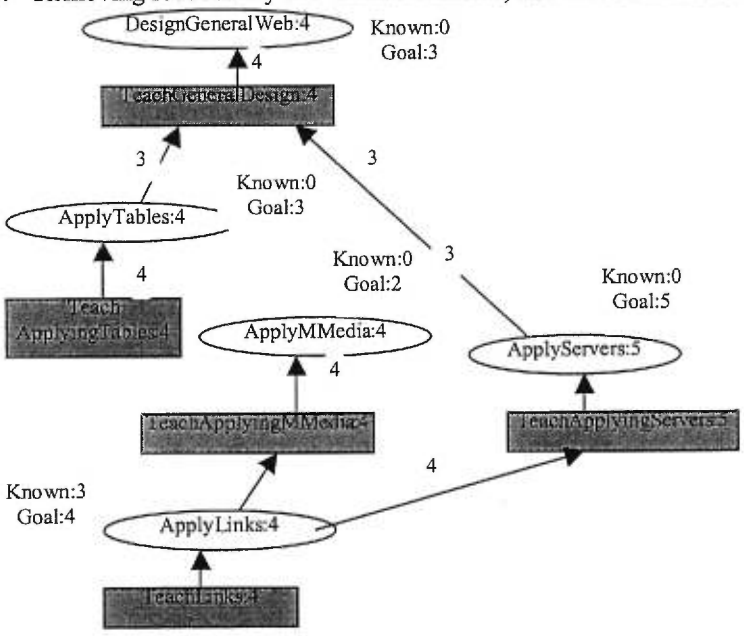


Figure 8.16. A course for a particular learner

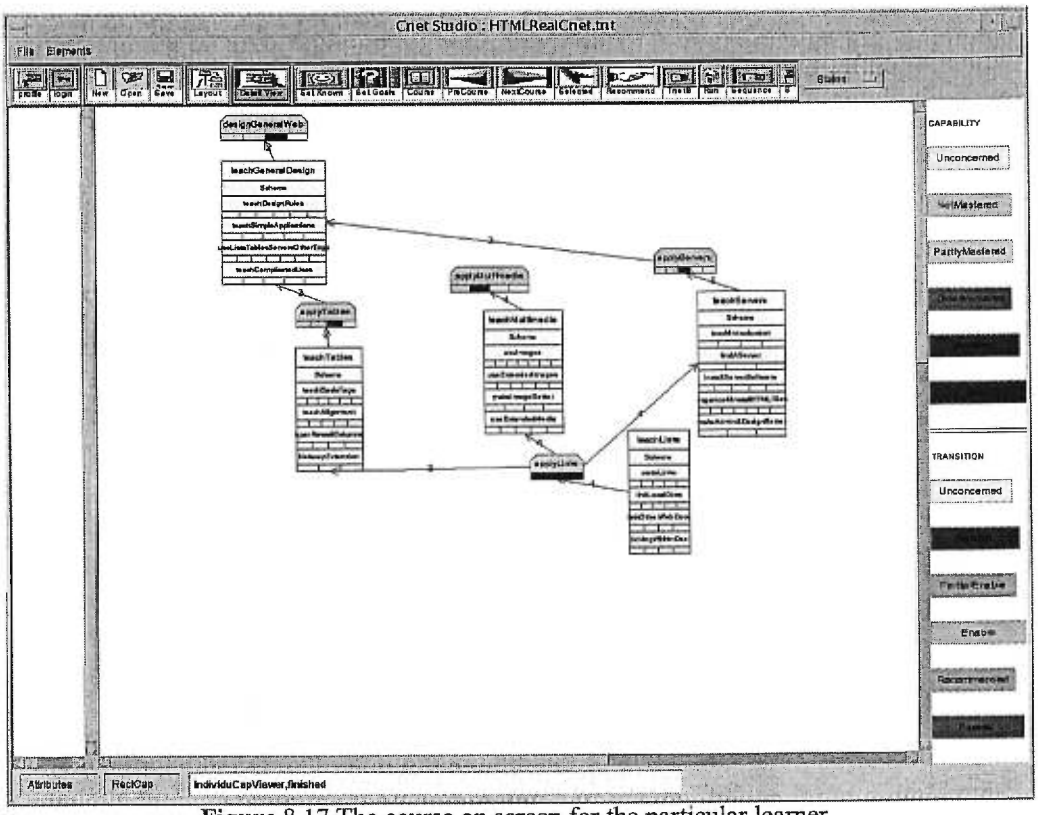


Figure 8.17 The course on screen for the particular learner

In the figure 8.16, the numbers beside capability nodes indicate the goal levels and the maximal levels of capabilities.

8.4.4 An *Anet* of HTML

This section describes the *Anet* corresponding to the example in 8.4.3. The figure 8.18 indicates necessary tutoring units. The figure 8.19 shows the practical tutoring activity sequence (*Anet*) in the example.

There are two recommended transition nodes and three disabled transition nodes at this moment:

Recommended Transition Nodes: teachLinks (unit 4), and TeachTables (unit 1~3).

Disable Transition Nodes: teachMultimedia (unit 1~2), teachServer (unit 1~3), teachGeneralDesignWeb (unit 1~3).

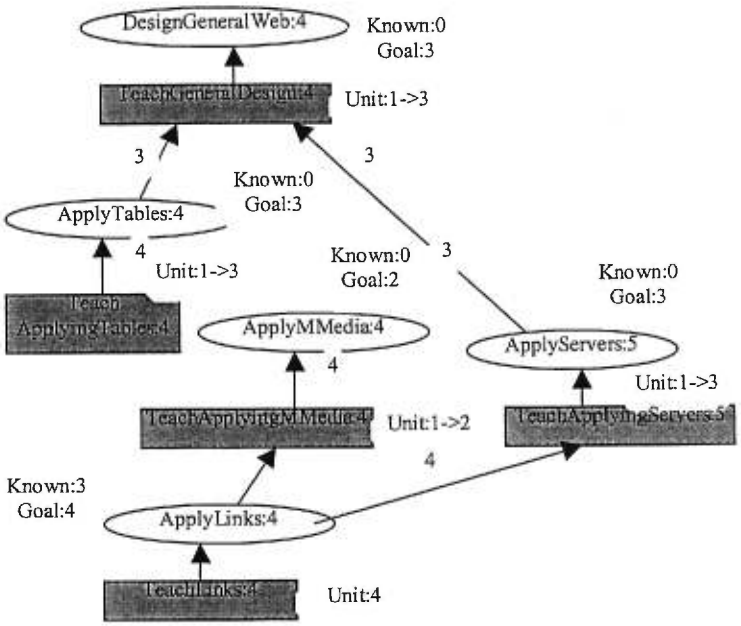


Figure 8.18 Identifying Necessary Tutoring Units

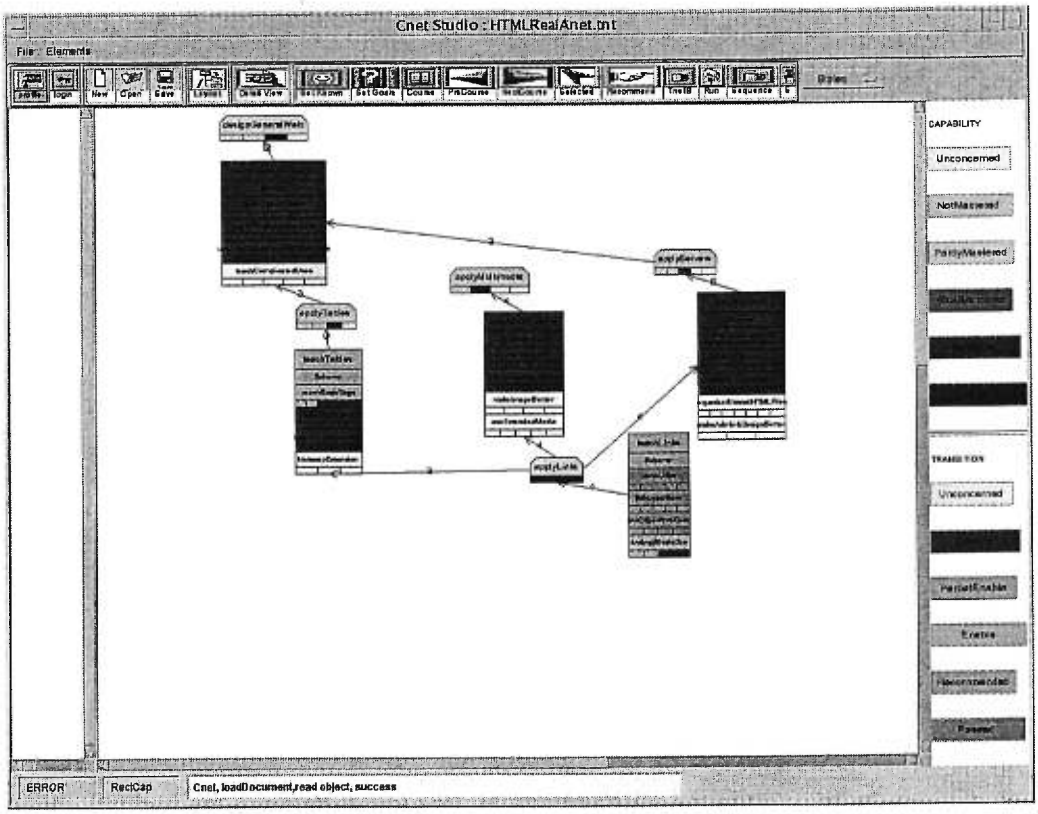


Figure 8.19 Sequenced tutoring units for the particular learner

8.5 Conclusion

The prototype of *VITCAM* consists of three parts: a visual component creator, an authoring environment and a learning environment.

The visual component creator is used to generate a library of visual components used for representing capabilities, transition nodes and links. Each kind of visual component contains numerous methods for editing the visual properties, the inner attributes and the manipulation. These methods are foundations of advanced operations in both the authoring environment and the learning environment.

Using the tools in the authoring environment, a curriculum author can organize all kinds of knowledge in curriculum to visually construct the capability transition network, *Tnet*, in a given domain. Meanwhile, the author can visually organize and test the capability transition network.

The learning environment consists of more operational categories, including

--Learner entrance
--File access
--Layout
--Detailed viewer
--Setting known capabilities and their levels
--Setting goals capabilities and levels
--Course creator
--Course recommender
--Information teller
--Course selector
--Decision switch
--System state navigator

A learner can establish and edit his own profile for learning. He/she can set known capabilities and learning goals by simply clicking. When the learner activates the course creator, multiple alternative courses are generated. The course recommender then suggests the most adaptable course for the learner.

There are three groups of navigation facilities for learners:

--Information teller: indicating the meanings of visual properties and the overall information of a course,
--On-line state navigation: using colors of visual cells to represent the current states of a learner and
--System state navigation: telling the learner what the system is doing.

As a practical application, the example for teaching HTML is developed. This chapter describes this application's capability transition network *Tnet*, and the examples of its *Cnet* and *Anet*.

Chapter 9

Conclusion

In this chapter we summarize our contributions so far, the comparison between *VITCAM* and other systems, current limitations, and future research objectives.

9.1 Contributions

The contributions of this thesis include

- Augmenting the existing curriculum model, so that it is simpler to manage globally than the ones proposed in the past.
- Proposing a visual useful curriculum management model that integrates the well known instructional design theories about instructional events and objective levels from Gagné and Bloom and generalizes the use of visual interactive environments to meet the needs of both curriculum based authoring and learning processes.

9.1.1 Presenting a Globally Simpler Curriculum Representation Structure

This thesis augments the existing curriculum models by presenting a visual curriculum representation structure (*Tnet* in the *VTRANS* sub-model) that is simpler to manage and manipulate than the ones proposed in the past.

Our curriculum model includes two kinds of nodes: capability nodes and transition nodes. Capability nodes indicate what a learner can acquire after certain tutoring activities. In fact, a capability node in our model is an ordered set of atomic capabilities or sub-capabilities identified by Gagné. Transition nodes organize interactive events and tutoring resources (including interaction creation or guidance programs) to support the acquisition of capabilities. The aggregation ability for organizing capability nodes and transition nodes is one of distinctions between *VITCAM* and other models.

There are just two types of links to represent the relationships between capability nodes and transition nodes: prerequisite links and output links. A prerequisite link defines a prerequisite relationship from a capability node to a transition node. The prerequisite link is one of the preconditions to activate a transition node. An output link represents a contribution relationship from a transition node to a capability node. That

contribution means that after the tutoring activities in the transition node are carried out successfully the learner should have acquired the transition node's output capability.

This approach is a great simplification of the previous curriculum models. Now the complexity of the curriculum is placed essentially at the level of the nodes and not at the level of the overall structure. This idea makes it possible to further reduce curriculum networks via node aggregation and make it much more practical to use curriculum visualization tools in both curriculum authoring and actual teaching. The *VITCAM* model is well adapted to knowledge representation and organization. This model supports acquiring multiple-level capabilities and aggregated capabilities. Most practical curriculum structures are large complex networks (e.g. hundreds even thousands of nodes). To manage such large networks is a difficult task. The mechanism of capability and transition node aggregation makes the global network easily manageable. For example, the capability transition network for teaching HTML contains just seventeen capability nodes and seventeen transition nodes with our prototype, in which 61 concrete capabilities, about 289 instructional events (an event is, in fact, an interaction session), and thousand resources are integrated. This makes the transition network save more than 70% of the number of nodes, comparing *Tnet* with a single-capability transition network. Such a simple network provides great favorites for the global management and course generation.

9.1.2 Proposing a Visual Useful Curriculum Management Model

The usability of the presented curriculum model comes from the following aspects [Guo, Kaltenbach, Frasson and Gecsei 99] [Guo 98] [Guo, Kaltenbach, Frasson and Gecsei 98a, 98 b] [Guo, Kaltenbach and Frasson 98a, 98b]:

(1) Organizing tutoring transitions and general strategies by combining Gagné's instructional event theory and Bloom's objective level theory to help the curriculum authors get good guidance and alternative choices when designing a curriculum.
(2) Visualizing the capability transition network in a given domain with visual composite icons and links. This is obtained by way of visually representing capabilities, capability levels, multiple level tutoring events and the relationships between capabilities and tutoring events respectively;
(3) A dynamic management of the authoring and learning process helps both the curriculum authors navigate and assess a curriculum with ease and the learners achieve their learning goals quickly.

- **Organizing capability transition by combining well known instructional theories**

This thesis proposes a mechanism that combines Gagné's capability and instructional event theory with Bloom's objective level theory. On the one hand, the transition structure reflects the incremental

mechanism of human learning. On the other hand, it increases the flexibility in establishing learning goals by learners, at the same time relatively simple and useful for curriculum authors.

Both Gagné's capability and instructional event theory and Bloom's objective levels are based on the analysis of general human teaching and learning, so that they can be used to guide teaching design in most domains. From the viewpoint of *VITCAM* structure, some of general teaching strategies are distributed in its component organization and processes. Various other teaching strategies, whether they are generic or domain specific, are open and can be integrated in resource development and the activity delivery module in *VITCAM*. Based on the future models and tools on resource development, interaction generation and teaching delivery, *VITCAM* based tutoring systems can be used to more domains.

- **Visualizing capability transition networks**

VITCAM defines three kinds of visually aggregated components to represent capabilities, transition nodes and resources. A first kind of node is a classical icon with various shapes, which can represent many types of resources. A second kind of node is a composite icon with a big visual cell representing overall attributes of capabilities and several small visual cells representing capability levels. A third kind of node integrates the lower level ones and represents a transition node with tutoring units and sub-units. In our model, a capability transition network is a graph with composite icons and links.

With the visual operations available in the *VITCAM* model [Guo et al 99, 98a, 98b], a curriculum author can visually build, organize, and manage capability transition networks from various perspectives: the icon view for the global network structure, the local view for building local network structure and the detail view for editing and managing the details of visual cells.

- **Helping curriculum authors and learners**

The visual dynamic properties of *VITCAM* make it possible for a curriculum author to assess globally particular properties of a curriculum, for instance how the use of a practical resource type is distributed in the network.

In the development process of a curriculum, the links among teaching outcomes, tutoring activities and resources are automatically built. The system can fully use these links to dynamically help the curriculum author get useful information to modify or restructure a curriculum.

The visualized capability transition networks developed by a curriculum author can be directly input to the learning environment. The actual displays shown to students are still in development stages but the existing

prototypes demonstrate the feasibility of the approach. A learner then starts his/her learning process based on the domain transition network. The visual interaction process for learning contains:

- visually setting known capabilities
- visually establishing learning goals
- displaying multiple alternative courses and recommended courses
- dynamically displaying students current states of learning
- visually recommending the next tutoring activity to engage in.

- **Dynamically managing learning process**

In addition to creating alternative courses, evaluating courses and recommending courses to learners, this system can dynamically order and reorder tutoring activities in response to the evaluation of transition node activities by an activities delivery module in ITS.

The proposed state-driven reasoning approach can order tutoring activities in a selected course. Each tutoring event or a group of tutoring events is associated with six states: *recommended*, *enabled*, *partly enabled*, *disabled*, *unconcerned*, and *passed*. A group of reasoning rules is presented for dynamically and visually changing the states.

9.2 Comparing *VITCAM* to Other Systems

There are two existing systems that are the closest to *VITCAM*, i.e. *CREAM* {Nakmbou 96] [Nkambou et al 98] and Eon [Murray 96, 98]. Now we can compare *VITCAM* to them.

9.2.1 Similarity between *VITCAM* and *CREAM*

There are three basic ideas in *VITCAM* that are absorbed from *CREAM*: i.e. using Gagné's capability theory and Bloom's objective levels, the basic idea the overall transition structure and the basic idea for teaching development process.

- **Using Gagné's capability theory and Bloom's objective levels**

The importance of Gagné's capability theory lies in that, in teaching and learning processes, the achievement of certain type of capabilities has its own internal and external conditions [Gagné 85]. These internal conditions form distinct learning strategies and the external conditions are the basis of designing teaching strategies. Using capabilities as teaching outputs is one of *CREAM's* features. *VITCAM* absorbs this feature.

- **The basic idea of the overall transition structure**

In *CREAM*, the basic idea for construction transition networks is an atomic process:

prerequisiteCapabilities→ transition→ outputCapabilities.

The benefit of introducing transition nodes is that it associates domain subjects, tutoring objectives and actions. *VITCAM* directly regards this atomic process as its start point.

- **The basic idea for teaching development process**

The *CREAM* based teaching development process is

building transition networks,
generating courses for a particular student,
planning action sequences, and
delivering teaching.

This process supports the individual course generation based on a domain transition network. *VITCAM* also inherits this idea.

9.2.2 Differences between *VITCAM* and *CREAM*

There are four aspects that *VITCAM* are different from *CREAM*: visualizing as many as structure components, visually and dynamically navigating, differences in structure, course generation, and planning action sequences

- **Visualizing as many as structure components**

CREAM-Tools is an example that uses more visual properties to replace the classical means for representing object spaces. In *CREAM*, capability, objective and resource nodes are visualized as buttons with labels and different background colors. Visual links indicate the relationships between nodes. A node in *VITCAM* is a composite icon with structured visual cells. Each visual cell is a clickable button. In capability icons, visual cells include an overall attribute button and several capability level buttons. A curriculum author can click the overall cell to enter or edit the internal attribute of the capability, and click a level cell to enter or edit the internal attributes of the related capability level. A transition node icon is a structured button group with three kinds of cells: an overall cell, tutoring unit cells and sub-unit cells. A curriculum author can click the overall cell to enter or edit the overall attributes of the transition node and define the number of tutoring units. If he/she clicks a unit cell, the internal attributes and the number of sub-units can be defined. A sub-unit cell corresponds to an internal event, where resource groups (including interaction generation and guidance programs) can be attached and tutoring actions can take place. Meanwhile, *VITCAM* use more visual properties such as shapes, colors and sizes to carry more internal semantics to the screen.

- **Visually and dynamically navigating**

CREAM-Tools use background color to distinguish the category of nodes. In *VITCAM*, a basic idea is to use colors of visual cells indicate the current states of authoring and learning. For example, in authoring process, the states of capability level cell might be "All necessary information are entered", "Part of attribute values are entered" and "Nothing is entered". In the curriculum development process, the system dynamically follows the development progress and displays the current development states. The dynamic navigation in learning process is more important then that in authoring process because the students don't know the subject domain. We define some states for each visual cell in a transition node. For example, a sub-unit might be one of the following six states: recommended, enabled, partly enabled, disabled, passed, and unconcerned. A reasoning engine is defined to dynamically follow the learning process and automatically change the states of each visual cell. The corresponding colors also automatically. Such navigation ability can reduce the lost of authors and learners.

- **Differences in structure**

As mentioned above, we use the basic idea in *CREAM*, i.e. prerequisiteCapabilities→transition→outputCapabilities. However, the components in such transition networks are different. A capability node in *CREAM* is a capability defined by Gagné. This means that a capability node corresponds to one capability type. A transition node in *CREAM* mainly contains an objective node and association links to capability space and resource space. In *VITCAM*, a capability node is a ordered set that contains one or more capabilities, allowing the aggregation of different capability types. Further, a curriculum author can specify explicitly the incremental capability levels. These capability levels may be a decomposition of a complex capability or one concrete capability in an incremental capability sequence. The transition node, in *VITCAM*, is a two-level structured set of tutoring events, which is organized by integrating Gagné's instructional events and Bloom's objective levels. These instructional events, in the teaching process, are interaction sessions. Concrete resource groups, or resource selector, or interaction creator, and/or evaluation programs, and/or diagnosis programs can be attached to instructional events to form interaction session.

CREAM defines rich relationships between nodes, for example, required or contributed level between capability and objective nodes, and strong contributions and weak contributions from objective nodes to output capabilities. Such relationships contain more heuristics. *VITCAM* uses just prerequisite and output relationships with explicit levels.

- **Course generation**

Owing to the defined heuristic relationships between nodes, *CREAM* based teaching development process attempts to automatically and heuristically create an optimal course for a particular student, based on the student's knowledge category (e.g., novice, intermediate and advanced) designed by a curriculum author.

In *VITCAM*, the basic idea for course generation is that, if it's the first time, a student tells (or evaluated by an evaluation module in ITS) the system about his/her current knowledge state (a set of known capability levels) and establishes his/her learning goals (another group of capability levels) by viewing the domain transition network provided by the system (or evaluated by an evaluation module). Another idea is that *VITCAM* attempts to create multiple alternative courses for the particular students and then let the student choose (or accept the system's recommendation) a course he/she preferred.

- **Planning action sequences**

The *CREAM* based teaching planner orders tutoring activities in a given course to satisfy certain requirement such as duration, etc. [Le 98]. Owing to the difference of transition structure components between *CREAM* and *VITCAM*, *VITCAM* orders tutoring actions based on two steps: identifying necessary units and dynamically sequencing units and sub-units. One feature of *VITCAM* is that the model for ordering tutoring actions can reactivate the course creator to generate remedy courses.

9.2.3 Comparing *VITCAM* to Eon

Eon [Murray 96, 98] is another system *VITCAM* is close to. The similarity between Eon and *VITCAM* is the rich visual properties. Using shapes, colors and other visual properties to enhance node semantics. The main differences including

A visual node in Eon is a single-component picture, while in *VITCAM* each node is a composite clickable icon.

In Eon, topic networks and transition networks are separate, while in our model; the central network is the transition network that associates capabilities and structured set of instructional events.

As a prototype with long-term development, Eon provides rich resource development environment and interaction editor. *VITCAM* supplies the major management framework of curriculum and much more development efforts are needed to complete a really practical teaching development environment.

9.3 Limitations of *VITCAM*

Owing to the huge development efforts of a really practical teaching development environment, the *VITCAM* completes just its main framework. Some limitations includes

Only using 2D composite components with basic shapes to constitute transition networks,

Lack of some other support modules in ITS, for instance resource and interaction development environments, to verify some features by real users, and

Lack of more examples from different domains to verify its features.

These limitations require more efforts and provide some possible start points for further researches.

9.4 Future Objectives

The future work in this project includes

- Making the interface more convivial possibly by the use of iconic compositions that represent a curriculum network [Kaltenbach & Preiss 94].
- Testing the system in a setting for a real course with real students. Admittedly at present the prototype needs a better interface for this testing.
- Providing tools that simplify the creation of pedagogical resources to be used in the system. Providing a unified interface for these tools.
- Developing a tool to look for relevant resources on the Internet, curriculum management will become much more efficient and appealing.

- Exploiting the activity delivery module

We would like to add an activity delivery module that is an additional open module in the current *VITCAM* prototype. This will be in the future of an agent system, tutor or companion that will assist the learner in progressing in a course, by providing summary statement, environment, about the current activities the student is engaged in. Also, there are many issues in the activity delivery module that are still open, for instance, creating a generic structure for diagnosis and evaluation. Some ideas in *CREAM*, such as some heuristic approaches, likely, can be reused for the dynamic interactions in the activity delivery module.

In addition, more helps can be provided for curriculum authors and learners, for instance, a tool for helping curriculum authors aggregate atomic capabilities into capability nodes, and a tool for helping learners establish learning goals.

Bibliography

Ahlberg, C. and Shneiderman B., 1994. Visual information seeking: tight coupling of dynamic query filters with starfield displays. *CHI'94*. 313-317.

Aimeur, E. & Frasson, C., 1995a, Eliciting the learning context in co-operative tutoring systems. *In IJCAI-95 workshop on modeling context in knowledge representation and reasoning,* Montreal, QC, 1-11.

Aimeur, E. & Frasson, C., 1995b, Towards new learning strategies in intelligent tutoring systems. In *Brazilian conference on artificial intelligence SBIA'95*, Berlin, Springer-Verlag, 121-130.

Allessi S. M. & Trollip S. R., 1985, *Computer-based instruction: methods and development.* Prentice-Hall, Englewood Cliffs, NJ.

Anderson, J. R., 1988. The expert module. In Polson, M. C. & Richardson, J. (eds). *Foundations of Intelligent Tutoring Systems*. Lawrence Erlbaum Associates, Hillsdale, NJ, 21-53.

Anderson, J.R., 1983. Acquisition of proof skills in geometry. In Mickalski, R.S., Carbonell, J.G., & Mitchell, T. M. (eds), *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann Publishers, Inc., Los Altos, CA, 191-219.

Barr, A., Bear, M. and Atkinson, R.C., 1976. The computer as a tutorial laboratory; the Stanford BIP project. *International Journal of Man-Machine Studies*, vol. 8, 567-596.

Bloom, B. S., 1978. *Taxonomie des objectifs pédagogiques. tome 2: domaine affectif.* Education Nouvelle, Inc.

Bloom, B. S., 1969. *Taxonomie des objectifs pédagogiques. tome 1: domaine cognitif.* Education Nouvelle, Inc.

Botafogo, R.A., Rivlin, E., and Shneiderman, B., 1992. Structural analysis of hypertexts: identifying hierarchies and useful metrics, *ACM Trans. on Information Systems.* Vol. 10, No.2, 142-180.

Bovair, S., Kieras, D.E. & Polson, P.G., 1990. The acquisition and performance of text-editing skill: A cognitive complexity analysis. *Human computer interaction*, Vol. 4, 1-48.

Bretch, B.J., 1990. Determining the focus of instruction: content planning for intelligent tutoring systems. *Ph D Thesis,* University of Saskatchewan, Saskatchewan, SA.

Brien, R., 1992. *Design pedagogique*, Edition Saint-Yves, Ottawa, ON.

Brown, J.S.,Burton, R.B. and Bell A.G., 1975. SOPHIE: a step towards creating a reactive learning environment. *International Journal of Man-Machine Studies*, Vol. 7, 675-696.

Burns, H.L. and Capps, C.G. 1988. Foundations of intelligent tutoring systems. In M.C. Polson & J.J. Richardson (eds), *Foundations of intelligent tutoring systems,* Lawrence Erlbaum Associates, Hillsdale, NJ, 21-53.

Burton, R.B. and Brown, J.S., 1982. An investigation of computer coaching for informal learning activities. In D. Sleeman & J.S.Brown (eds.,), *Intelligent Tutoring Systems*, Academic Press, New York, NY, 79-98.

Carbonell, R., 1970. AI in CAI: an artificial intelligence approach to computer aided instruction. IEEE Transactions on Man-Machine Systems, Vol. 11, 190-202.

Carriere, J. and Kazman, R., 1997. WebQuery: searching and visualizing the Web through connectivity. *Proc. Sixth International World Wide Web Conference.* Also at *http://Proceedings.www6conf.org/index_by_topic.html#browser.*

Chamberland, G., Lavoie, L. & Marquis, D., 1995. *20 Formules pedagogiques*, Presses de l'Université du Québec, Montréal, QC.

Chan, T. W. & Baskin, A. B., 1990, Learning compagnion system. In C. Frasson & G. Gauthier (eds), *Intelligent Tutoring Systems: at the crossroad of AI and education.* Ablex Publishing Corp., Norwood, NJ, 6-33.

Clancey, W.J., 1992. Intelligent tutoring systems: a tutorial survey. In Stephen, J.A. & Gerald, W.H. (eds) *Applied Artificial Intelligence: a Sourcebook.* McGraw-Hill, Inc., San Fransisco. 250-279.

Clancey, W.J., 1982. GUIDON. In Barr & Feigenbaum (ed) *The Handbook of Artificial Intelligence.* William Kayfmann, Inc. Los Alto, CA. 267-278.

Derry, S.J., 1990. Learning strategies for acquiring useful knowledge. In Jones, B.F. & Idol, L. (eds), *Dimensions of thinking and cognitive instruction*, Lawrence, Erlbaum Associates, Hillsdale, NJ, 15-51.

Elsom-Cook, M., Spensley, F., Byerley, P., Brooks, P., Mhende, M., Federici & Scaroni, C., 1988. Using multiple teaching strategies in an ITS. In Frasson, C. & Gauthier, G. (ed.), *Proceedings of the first international conference on intelligent tutoring systems,* Montréal, QC, 286-290.

Feifer, R. G., Dyer, M. G. & Baker, E. L., 1988. Learning procedural and declarative knowledge. In Frasson, C. & Gauthier, G. (ed.), *Proceedings of the first international conference on intelligent tutoring systems,* Montréal, 499-505.

Finch C.R. & Cukilton, J. R., 1986. *Curriculum Development in Vocational and Technical Education: Planning, Content and Implementation*. Allyn and Bacon, Inc. 2nd Edition.

Frasson, C., 1994, Environnement modulaire pour les STI. *Rapport technique, Project SAFARI,* Laboratoire Heron/Multimedia, Université de Montréal, Montréal, Québec.

Frasson, C. & Gauthier, G., 1994. A gradual software environment for developing tutoring systems. In *Advances in artificial intelligence—theory and application II*, Windsor, ON IIAS, 73-78.

Frasson, C., Gauthier, G. & Imbeau, G., 1992. Architecture d'un système tutoriel intelligent base sur un curriculum. *Revue Génie Éducatif*, Vol. 1. No. 4., 21 pp.

Frasson, C. 1991. Systèmes tutoriels intellgents: état et perspectives en Amérique du nord. *Revue Génie éducatif*, Vol. 1, No. 1, 7-15.

Frasson, C., Kaltenbach, M., Gecsei, J. & Djamen, J-Y., 1988 An iconic intention-driven ITS environment, in Frasson, C. Gauthier, G. & McCalla, G. I. (eds) *Intelligent Tutoring Systems, Proceedings of the Second International Conference (ITS'92),* Montreal, Canada, June, Springer-Verlag, 66-75.

Gagné, R. M., 1976. *Les principes fondamentaux de l'apprentissage: application à l'enseignement.* Les Éditions HRW Ltée, Montréal, QC, 4th edtion.

Gagné, R.M. 1993. Computer-based instructional guidance. In M. Spector, P.C. Polson & D.J. Muraida (dir.), *Automating instructional design: concept and issues*, Educational Technology Publications, Englewood Cliffs, NJ, 133-146.

Gagné, R.M., Briggs, L. and Wager, W., 1992. *Principles of Instructional Design*. Harcourt Brace Jovanovich College Publishers, Orlando, FL. 4th edition.

Gagné,.R.M., 1985. *The conditions of learning and the theory of instruction*. CBS College Publishing, 4th edition.

Gauthier, G. & Imbeau, G., 1989. Un système tutoriel intelligent: modèle théorique et réalisation. In *6e Symposium Canadien sur la Technologie Pédagogique.* Halifax, NS, Conseil National de Recherche du Canada, 110-113.

Gecsei, J. and Frasson, C., 1994. SAFARI: an environment for creating tutoring systems in industrial training. *Proc. of the World Conference on Educational Multimedia and hypermedia (ED-Media).* Vancouver, BC, Canada., 15-20.

Gershon, N. and Brown, J.R., 1996. The role of computer graphics and visualization in the GII. *IEEE Computer Graphics and Applications*. Vol.16, No.2, 61-63.

Gershon, N. and Eick, S. G., 1995. Visualization's new tack : making sense of information. *IEEE Spectrum* Nov., 38-56.

Girard, J., 1991. Système tutoriels intelligent: une architecture multiagent et sa composante curriculum. *Mémoire de maîtrise (M.Sc.),* Université du Québec à Montréal.

Grippin, P. and Peters, S., 1984. *Learning theory and learning outcomes: the connection.* University Press of America, Inc., Lanham, MD.

Guo, R., Kaltenbach, M., Frasson C. and Jan Gecsei, 1999. Visual navigation for the curriculum based learning process, *technical report of Heron & Multimedia Laboratory*, DIRO, University of Montreal, (submitted to *ICCE'99*).

Guo, R., Kaltenbach, M., and Frasson C., 1999. A visual navigation approach for curriculum development *technical report of Heron & Multimedia Laboratory*, DIRO, University of Montreal, (submitted to *INTERTECH 2000*).

Guo, R., 1998. Knowledge Transition in Curriculum, *Proceedings of ICCE'98*, Oct., 1998, Beijing, China, 57-66.

Guo, R., Kaltenbach, M., Frasson C. and Gecsei J., 1998a, Visual Manipulation for Development of Curriculum, *Proceedings of NTICF'98,* INSA de Rouen, France, Nov., 85-96.

Guo, R., Frasson, C., Kaltenbach, M. and Gecsei, J., 1998b. *VITCAM*: A Visual Interactive Management System for Curricula, *Abstracts of Demo in TeleLearning'98*, Nov. Vancouver, 14-17.

Guo, R., Kaltenbach M. and Frasson C., 1998a. Integration and Organization of Knowledge in Intelligent Tutoring Systems, *Technical Report,* Heron & Multimedia laboratory, DIRO, Université de Montréal, accepted by *EXPERSYS'98*, Oct., USA.

Guo, R., Kaltenbach M. and Frasson C., 1998b. Organizing integrated tutoring activities for curriculum. *Technical Report,* Heron & Multimedia laboratory, DIRO, Université de Montréal, Accepted by *CAEE'99.*

Guo, R., Aimeur E. and Frasson C., 1996. User Interface: A Survey, *TR-1053, Department of Computer Science and Operational Research, University of Montreal,.*

Guo, R. and Hou Z., 1996. Issues of circulation in nonmonotonic reason maintenance systems, *IEEE International Conference on Systems, Man and Cybernetics,* October, Beijing, China, (also accepted by *IEEE SMC'95*, Vancouver, Canada), 1430-1435.

Guo, R., 1995. SAFARI User Interface: Analysis and Improvement, , *Technical Report (SARARI* Project), Heron & Multimedia laboratory, DIRO, Université de Montréal, QC, 55pp.

Guo, R., 1992. An environment for diagnosing students' errors in ICAI systems and their development tools, *Computer Engineering and Applications*, No. 5, 8-11.

Halff, H., 1993. Prospects for automating instructional design. In M. Spector, P.C. Polson & D.J. Muraida (dir.), *Automating instructionaldesign: concept and issues,* Educational Technology Publications, Englewood Cliffs, NJ, 67-131.

Halff, H., 1988. Curriculum and instruction in ITSs. In *Foundations of Intelligent Tutoring Systems,*. Lawrence Erlbaum Associates, Hillsdale, NJ, 19-108.

Hartley, R., 1990. The curriculum and instructional tasks: Strategies, and Tactics for interactive learning. In *Adaptative Learning Environments: Foundations and Frontiers.* Berlin, Springer-Verlag, 123-146.

Hearst, M.A., 1995. TileBars: visualization of term distribution information in full text information access. *Human Factors in Computing Systems (CHI'95)*, 59-66.

Hemmje, M., Kunkel, C., and Willett, A., 1994. LyberWorld--a visualization user interface supporting full text retrieval. *SIGIR'94*, 249-259.

Henderson Jr, D.A.and Card, S.K., 1986. Rooms: the use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Trans. on Computer Graphics*. Vol.5, No.3 (July), 211-243.

Hollan, J.D., Hutshins E.L. and Weitzman, L., 1984. STEAMER: an interactive inspectable simulation-based training system. *AI Magazine*, vol. 5, no. 2, 15-27.

Imbeau, G., 1990. Modélisation er réalisation d'un système tutoriel intelligent. *Thèse de Ph.D.,* DIRO, Université de Montréal, Montéal, QC.

John, B.E. & Kierras, D.E., 1994. The GOMS family of analysis techniques: Tools for design and evaluation. *Technical Report cmu-chi-94-106,* Carnegie Mellon University School of Computer Science, Carnegie Mellon.

Johnson, B. and Shneiderman, B., 1993. Treemaps: a space-filling approach to the visualization of hierarchical information structures. in *http://www.cs.umd.edu/Document/UMCP-CSD:CS-TR-2657.*

Kabbaj, A., Khalid, R. & Frasson, C., 1996. The use of a semantic network activation language in an ITS project. in Frasson, C., Gauthier, G. & Lesgold, A. (Eds) *Intelligent Tutoring Systems, Proceedings of the Third International Conference (ITS'96),* Montreal, Canada, June 12-14, Springer, 279-287.

Kaltenbach M., Preiss B., 1994. Iconicase: a visual system for rapid case review. *Proceedings of the ED-Media Conference,* Vancouver, B.C., AACE, 305-310.

Katz, S., Lesgold, A, Eggan, G., Gordin, M. & Greenberg, L., 1992. Self-adjusting curriculum planning in Skerlock II. In Tomek, I. (ed.*), Computer Assisted Learning*, Berlin, Springer-Verlag, 343-366.

Kearsley, G.P., 1987. *Artificial Intelligence & Instruction.* Addison-Wesley Publishing Company.

Kimball, R., 1982. Self-improving tutor for symbolic. In Sleeman, D. & Brown, J.s. (eds) *Intelligent Tutoring Systems*. Academic Press, London.

Kumar, H. P., Plaisant, C. and Shneiderman, B., 1997. Browsing hierarchical data with multi-level dynamic queries and pruning. *Int. J. Human-Computer Studies.* Vol. 46, No. 1(Jan.) , 103-124.

Lajoie, S. Lesgold, A. et al., 1989. A procedural guide to the avionics troubleshooting tutor development process. *Technical Report.* Learning research and development center, University of Pittsburgh, Pittsburgh.

Lamping, J., Rao, R. and Pirolli, P., 1995. A focus+context technique based on hyperbolic geometry for visualization large hierarchies. *CHI'95.*

Lansky, P., 1975. L'informatique éducationnelle. In *École d'été UCODI,* Genève.

Le, T., 1998. Planification de l'enseignement individualisé dans un système tutoriel intelligent à grande échelle. *These de Ph. D*, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal.

Legendre, R., 1993. Dictionnaire actuel de l'éconnaissance. *Mémoire de maîrise (M.Sc.),* Université du Québec à Montréal.

Lesgold, A.,1988. Toward a theory of curriculum for use in designing intelligent instructional systems. In Mandl, H. & Lesgold, A. (eds) *Learning Issues for Intelligent Tutoring Systems*. Springer-Verlag, Berlin, 114-137.

Lesgold, A. Lajoie, S. Bunro, M. & Eggan, G., 1992. A coach practice environment for an electronics troubleshooting job. In Larkin, J. L. & Chabay, R. W. (eds) *Computer-Assisted Instruction and Intelligent Tutoring System*: *Shared Goals and Complementary Approaches.* Lawrence Erlbaum Associates, Hillsdale, NJ, 201-238.

Luger, G.F. & Stubblefield, W.A., 1993. *Artificial Intelligence and the Design of Expert Systems.* Benjamin/Cummings, 2nd edition.

McCalla, G. and McCalla, I., 1990. The search for adaptability, flexibility, and individualization: approaches to curriculum in intelligent tutoring systems. In *Adaptive Learning Environments: Foundations and Frontiers,* Berlin, Springer-Verlag, 91-112.

McCalla, G.I., Greer, J.E., 1994. Granularity-based reasoning and belief revision in student models. In Greer & McCalla (ed.), *Student Modeling: The Key to Individualized Knowledge-Based Instruction.* NATO ASI Series. Springler-Verlag, Berlin.

Merrill, M. D., 1993. An integrated model for automating instruction design and delivery. In Spector, M. Polson, P. C. & Muraida, D. J. (eds.) *Automating Instructional Design: Concept and Issues.* Educational Technology Publications, Englewood Cliffs, NJ, 147-190.

Merrill, M. D., Li, Z., and Jones, M. K., 1991. Instructional transaction theory: an introduction. *Educational Technology*, Vol. 31, No. 6, 7-12.

Merrill, M. D., 1987. An expert system for instructional design. *IEEE Expert*, Vol. 2, No. 2, 25-37.

Minsky, M., 1981. *A framework for representing knowledge*. The MIT Press, Cambrige, MA, 95-128.

Murray, T., 1998. Authoring knowledge-based tutors: tools for content, instructional strategy, student model, and interface design. *The Journal of the Learning Science,* 7(1), 5-64.

Murray, T., 1996a. Having it all, maybe: design tradeoffs in ITS authoring tools. , in Frasson, C. Gauthier, G. & Lesgold, A. (Eds*) Intelligent Tutoring Systems, Proceedings of the Third International Conference (ITS'96),* Montreal, Canada, June 12-14, Springer, 93-101.

Murray, T., 1996b. From story boards to knowledge bases: the first paradigm shift in making CAI intelligent. *ED-MEDIA'96,* Boston, MA, July.

Murray, T. and Woolf, B. P., 1993a. Design and implementation of an intelligent multimedia tutor. In *AAAI'93 tutorial.*

Murray, T. & Woolf, B.P., 1993b. Tools for teacher participation in ITS design. In Frasson, C., Gauthier, G. & McCalla, G.I. (ed.), *Intelligent Tutoring Systems,* Berlin, Springer-Verlag, 593-600.

Murray, T. & Woolf, B.P., 1991. A knowledge acquisition tool for intelligent computer tutor. *SIGART Bulletin,* Vol. 2, No.2., 1-133.

Nicaud, J.F. & Vivet, M., 1988. Les tuteurs intelligents: réalisations et tendances. *TSI,* Vol. 7, No. 1, 21-45.

Nkambou, R., Frasson, C. & Gauthier, G., 1998. A new approach to ITS-curriculum and course authoring: the authoring environment, *Computers & Education,* 31, 105-130.

Nkambou, R., Frasson, M.C. and Frasson, C., 1996. Generating courses in an intelligent tutoring system. In *Proceedings of the 9th international conference on industrial and engineering applications of artificial intelligence and expert systems*; Gordon and Breach Science, New York, 261-267.

Nkambou, R., 1996. Modélisation des connaissances de la matière dans un système tutoriel intelligent: modèles, outils et applications. *Ph. D Thesis*, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal.

Nkambou, R., Quirion, L., Frasson, C. & Kaltenbach, M., 1995. A simulation-based authoring environment for demonstration design. In Fraisse, Garzotto, F., Nanard, N. & Nanard, J (Ed.), *Hypertext Design,* , Springer-Verlag, Berlin, 143-145.

Nakmbou, R., Quirion, L., Kaltenbach, M &. Frasson, C., 1995. Using multimedia in learning about process; DEGREE: a simulation-based authoring environment for demonstration building. In Chua, T.S., Pung, H.K. & Kunii, T.L. (ed.), *Multimedia Modeling: towards Information Superhighway,* World Scientific Publishing, Singapore, 365-378.

Nkambou, R., 1992. Planification dans un sti: une approche basées sur l'expérience appliquée à geocam. *Mémoire de maîtrise (M.Sc.), (Mémoire d'Ingénieur),* Institut Africain d'Informatique.

Norman, D.A. & Romelhart, D.E., 1975. *Exploration in cognition.* Freeman W.H. and Compagny, San Francisco, CA.

Paquette, G. & Girard, J., 1996. AGD: a course engineering support system. in Frasson, C. Gauthier, G. & Lesgold, A. (Eds), *Intelligent Tutoring Systems, Proceedings of the Third International Conference (ITS'96),* Montreal, Canada, June 12-14, Springer, 382-391.

Pearl, J., 1988. Probabilistic *Reasoning in Intelligent Systems: Networks of Plausible Inference,* Morgan Kaufman Publishers, Inc., San Mateo, CA.,

Pirolli, P. & Russell, D.M., 1991. Instructional design environment: Technology to support design problem solving. *Instructional Science,* Vol. 19, No. 2, 121-144.

Pirolli, P. & Russell, D. 1988. Toward theory and technology for the design of intelligent tutoring systems. *Proceedings of ITS'88,* Montreal, Canada, June 1-3, 350-356.

Plaisant, C., Milash, B., Rose, A., Widoff, S. and Shneiderman, B., 1996. LifeLines: visualizing personal histories. *CHI' 96.*

Polson, M.C., 1993. Task analysis for an automated instructional design advisor. In M. Spector, P.C. Polson & D.J. Muraida (dir.), Automating instructional design: concept and issues, *Educational Technology Publications,* Englewood Cliffs, NJ, 219-248.

Quillian, M.R., 1967. Word concepts: A theory and simulation of some basic semantic capabilities. Behavioral Science, Vol. 12, 410-430.

Reigeluth, C. M., 1993. Functions of an automated instructional design system. In Spector, M, Polson, P.C. & Muraida, D.J. (Eds), Automating instructional design: concept and issues, *Educational Technology Publications*, Englewood Cliffs, NJ, 43-58.

Reiser, B. J., Friedmann, P., Kimberg, D. Y. & Ranney, M., 1988. Constructing explanations from problem solving rules to guide the planning of programs. *Proceedings of ITS'88,* Montreal, Canada, June 1-3, 222-229.

Robertson, G.G., Card, S. K. and Mackinlay, J.D., 1993. Information visualization using 3D interactive animation. *Comm. ACM.* Vol. 36, No. 4, 56-71.

Robertson, G.G., Mackinlay, J.D., and Card, S.K., 1991. Cone Trees: animated 3D visualizations of hierarchical information. *Proc. Human Factors in Computing Systems(CHI'91*), ACM Press. 189-202.

Robidas, G., 1989. *Psychologie de l'apprentissage: un système d'apprentissage-enseignement personnalisé.* Behaviora, Inc., Montréal, QC.

Schreiber, G., Widinga, B. & Breuker, J., 1993. *Kads: a principle appraoch to knowledge-based system development.* Academic Press.

Shen, Stewart, N. T., de Mora, C., & Liu, L., 1988. The GKB methodology as an intelligent tutoring builder, *Proceedings of ITS'88,* Montreal, Canada, June 1-3, 90-96.

Shneiderman, B., 1996. The eyes have it: a task by data type taxonomy for information visualizations. *Technical Report, CS-TR-3665*, Dept. of Computer Science, University of Maryland.

Shneiderman, B., 1990. Tree visualization with tree-map: a 2-d space-filling approach. *ACM Trans. on Graphics*. Vol.11, No.1, 92-99.

Shortliffe, E. H., 1976. *Computer-Based Medical Consultation: MYCIN.* New York: Elsevier.

Slovin, T. & Woolf, B.P., 1988. A consultant tutor for personal development. In Frasson, C. & Gauthier, G. (ed.), *Proceedings of the First International Conference on Intelligent Tutoring Systems,* 162-169, Montreal, QC.

Sowa, J.F., 1991. Principles of semantic networks. *Explorations in the representation of knowledge.* Morgan Kaufman Publishers, San Mateo, CA.

Spector, M., Polson, P.C. and Muraida, D.J., 1993. *Automating instructional design: concept and issues.* Educational Technology Publications, Englewood Cliffs, NJ.

Suthers, D. & Woolf, B.P., 1990. The epistemological structure of explanation. In *Proceedings of the AAAI'90 workshop on Explanation,* Boston, MA.

Takeuchi A. & Otsuki, S., 1992. EXPITS: an experiment environment on ITS, in Frasson, C., Gauthier, G. & McCalla, G. I. (Eds) *Intelligent Tutoring Systems, Proceedings of the Second International Conference (ITS'92),* Montreal, Canada, June12-14, Springer-Verlag, 124-131.

Tanin, E., Beigel, R. and Shneiderman, B., 1996. Incremental data structures and algorithms for dynamic query interfaces. *SIGMOD Record.* Vol.25, No.4, 21-24.

Tennyson, R.D., 1993. A framework for automating instructional design. In M. Spector, P.C. Polson & D.J.Muraida (dir.), Automating instructional design: concept and issues, *Educational Technology Publications,* Englewood Cliffs, NJ, 191-217.

Tardif, J., 1992. *Pour un enseignement stratégique: l'apport de la psychologie cognitive.* Les éditions Logiques, Inc. Montréal, QC.

Towne, D.M., Munro, A., Pizzini, Q.A., Surmon, D.S., Coller L.D. & Wogulis, J.L., 1990. Modeling-building tools for simulation-based training. *Interactive Learning Environments,* Vol.1, 33-50.

Uhr, L., 1969. Teaching machine programs that generate problems as a function of interaction with students. In *Proceedings of the 24th national conference*, 125-134.

Vivet, M., 1988. Métaconnaissance dans les tuteurs intelligents. In C. Frasson & G. Gauthier(eds.), *Proceedings of the first international conference on intelligent tutoring systems,* Montréal, Québec, 430-434.

Vivet, M., 1987. Systèmes experts pour enseigner: métaconnaissances et explications. In *Actes du congrès Cognitiva'87.*

Webster, J.G., 1994. Instructional objectives and bench examinations in circuits laboratories. *IEEE Transitions on Education,* Vol. 37, No.1, 111-113.

Wenger, E., 1987. *Artificial Intelligence and Tutoring Systems.* Los Altos, CA: Morgan Kaufmann.

Wescourt K. Beard, M. & Gould, L., 1995. Knowledge-based adaptive curriculum sequencing for CAI: application of a network representation. In *Proceedings of the ACM 1977 Annual Conference,* New York, NY, ACM. 234-240.

Woolf B.P., 1992a. Building knowledge based tutor. In Tomek, I. (ed.), *Computer Assisted Learning,* Berlin, Springer-Verlag, 46-60.

Woolf, B.P., 1992b. Towards a computational model of tutoring. In M. Jones & P.H. Winne (eds.), *Adaptive Learning Environments,* Springer-Verlag, Berlin, 209-231.

Woolf, B.P., 1989. Representing, acquiring, and reasoning about tutoring knowledge. In*Proceedings of the Second Intelligent Tutoring Systems Research Forum*, San Antonio, TX.

Woolf, B.P., 1988. Intelligent tutoring systems: a survey. In H. Schrobe & AAAI (eds.), *Exploring Artificial Intelligence.* Morgan Kaufmann.

Woolf, B.P., 1987. Theoretical frontiers in building a machine tutor. In Kearsley, G.P. (ed.),*Artificial Intelligence and Instruction,* 229-267, Addison-Wesley, Don Mills, ON.

Woolf B.P. & MacDonald, D., 1984. Building a computer tutor: design issues. *IEEE Computer,* Vol. 17, No. 9, 61-73.

Woods, W.A., 1970. Transition network grammars for natural language analysis. *CACM,* Vol. 3, No. 10, 591-606.

Zhou, G., Wang, J. T.-L. & Ng, P.A., 1996. Curriculum Knowledge Representation and Manipulation in Knowledge-Based Tutoring Systems. *IEEE Transaction on Knowledge and Data Engineering*, Vol. 8, No.5, 679-689.

Appendix A

Some Algorithms for Visualization

This appendix describes some algorithms for visual operations:

Algorithm 5.1: createCapabilityIcon

Algorithm 5.2 createTransitionNode

Algorithm 5.3 createLink

Algorithm 5.4 draggingAndMovingNode

Algorithm 5.5 deletingNode

Algorithm 5.6 deletingLink

Algorithm 5.7 insertingUnit.

A.1 Creating Capability Nodes

Algorithm 5.1 createCapabilityIcon

Input: startPoint (coming from a user's clicking on the screen),
TypeTemplate(coming from the user's selection on template panel),
Handling: create a capability icon without internal attributes
Output: display the icon on screen and store it to inner capability database.

```
Procedure: createCapabilityIcon
      Set default visual properties including label, levelNumber,
            fillColro, borderColor, pressedColor, selectedColor,
            currentFont, pressedState, selectedState;
      Compute cellStartPoint, cellWidth, cellheight,
      Initialize all inner attributes and exteranl relationships;
      Store the icon to inner database;

//draw the icon
if(pressedState[0]) fillColor[0]=pressendColor
      else if(selected[0]) fillColor[0]=selectedColor;
      drawRect(cellStartPoint[0], cellWidth[0], cellheight[0],
            fillColor[0]);
      drawBorder(cellStartPoint[0], cellWidth[0], cellHeight[0],
            border[0]);
      setFont(currentFont);
      drawLabel(label);

for(int I=1; I<=levelNumber, I++){
      if(pressedState[I]) fillColor[I]=pressedColor
      else if(selectedState[I]) fillColor[I]=selectedColor;
            draw levelShape(cellStartPoint[I], cellWidth[I],
```

```
                cellHeight[I], fillColor[I]);
            draw border(cellStartPoint[I], cellWidth[I], cellHeight[I],
                borderColor[I]);
}
//end procedure createCapabilityIcon
```

In algorithm 5.1, the overall cell of the capability icon is drawn first, and then all its level cells. The "draw level shape" statement in the algorithm will vary with the different template shape selected by the user. Some shape is simple such as a rectangle level and some others are complex such as a fan-shaped.

A.2 Create Transition Nodes

The Algorithm 5.2 describes the mechanism of creating transition nodes. The idea is similar to that in capability node, but the calculation about positions and sizes of visual cells is more complex than that in capability node.

```
Algorithm 5.2 createTransitionNode

Input: startPoint (coming from a user's clicking on the screen);
     TypeTemplate(coming from the user's selection on the screen);
Handling: creating a transition node icon with internal attributes;
Output: displaying the icon on screen and store it to inner transition
       node fdatabase.

Procedure: creatTransitionNode

      Set default visual properties including label, unitNumber,
       subUnitNumber, fillColor, borderColor, pressedColor,
       selectedColor, currentTransFont, currentUnitFont, pressedState,
       selectedState;
      Compute cellStartPoint, cellWidth, cellHeight,
      Initialize all inner attributes and external relationships;
      Store the attribute values to inner database;

//draw the icon
for(int I=0; I<unitNumber+2; I++){
      if(pressedState[I][0]) fillColor[I][0]=pressendColor
            else if(selected[I][0]) fillColor[I][0]=selectedColor;
            drawRect(cellStartPoint[I][0], cellWidth[I][0],
                  cellHeight[I][0], fillColor[I][0]);
            drawBorder(cellStartPoint[I][0], cellWidth[I][0],
            cellHeight[I][0], border[I][0]);
            if(I==0)setFont(currentTransFont)
            else setFont(currentUnitFont);
            drawLabel(label[I]);
      if(I>1){
            for(int j=1; j<=subUnitNumber[I], j++){
                  if(pressedState[I][j]) fillColor[I][j]=pressedColor
                  else if(selectedState[I][j])
                        fillColor[I][j]=selectedColor;
```

```
                        draw subUnitShape(cellStartPoint[I[j]],
                              cellWidth[I][j], cellHeight[I][j],
                                    fillColor[I][j]);
                        draw border(cellStartPoint[I][j],
                              cellWidth[I][j], cellHeight[I][j],
                                    borderColor[I][j]);
      }     //end if
}//end for
//end procedure createTransitionIcon
```

A.3 Creating Links

```
Algorithm 5.3 createLink

Input: sourceElement (from the user's clicking),
      DestinationElement( from the user's clicking)
Handling: create a link from sourceElement to destinationElement,
Output: display the link on screen, store the link icon to inner
      database.

Procedure: createLink{
      Set default visual properties including label, a=shape,
            headColor, tailColor, pressedColor, selectedColor,
             currentFont, labelColor, pressedState, selectedState;
      Compute startPoint, endPoint, arrowVector, manipulationArea;
      Initialize all inner attributes and external relationships;
      Store the attribute values to inner database;

//draw the icon
      if(pressedState) tailColor=pressedColor
      else if(selectedState)tailColor=selectedColor;
      drawTail(startPoint, endPoint, tailColor);
      drawHead(startPoint, endPoijt, arrowVector, headColor);
      drawLabel(startPoint, endPoint, manipulationArea, labelColor);
}//end procedure createLink
```

A.4 Laying Nodes out

The algorithm to drag and move nodes is as follows:

```
Algorithm 5.4 draggingAndMovingNode

Input: draggedIcon, currentMousePoint;
Handling: moving the draggedIcon to currentMousePoint;
Output: displaying the draggedIcon on currentMousePoint, storing the
      icon when mouse is released;

Procedure: dragginAndMovingNode{
     NotReleased=false;
     oldStartPoint←draggedIcon's startPoint;
     while(notReleased){
```

```
        get currentMousePoint;
        newStartPoint←currentMousePoint;
        compute Xincrease, Yincrease from newStartPoint to
            oldStartPoint;
        change all cellStartPoint's X coordination by Xincrease;
        change all cellStartPoint's Y coordination by Yincrease;
        erase draggedIcon;
        draw draggedIcon at newStartPoint;
        find all inLinks and outLinks of the draggedIcon;
        re-compute startPoint and endPoint of each inputLink, and each
            outputLink;
        erase all inputLinks and outputLinks;
        draw new inputLinks and outputLinks;
        oldStartPoint←newStartPoint;
      }//endWhile

      store the new attribute values of the icon to inner database;

}//end procedure draggingAndMovingNode
```

A.5 Deleting Nodes

The following steps are followed to delete nodes:

- erase the clicked icon from the screen;
- delete all input links and output links of the clicked icon from the screen and the inner database;
- delete the related output links of the source icon of each input link, and the related input links of the destination icon of each output link;
- delete the clicked icon from inner database.

The corresponding algorithm is:

```
Algorithm 5.5 deletingNode

Input: iconToDelete
Handling: delete the iconToDelete from screen and inner database,
      delete all input links and output links of the iconToDelete,
      removing the affection of these links to other icons
Output: the iconToDelete and its affection are selected from the screen
      and the inner database.

Procedure deletingNode{
     Erasing the iconToDelete fron the screen;
     inLinks←all input links of the iconToDelete;
     outLink←all outputLinks of the iconToDelete;
     deleting the iconToDelete from inner database;

     resoruceIcons←null;
```

```
    destinationIcons←null;
     if(!inLinks.isEmpty()){
            for(int I=0; I<linkLinks.length, I++){
            add the resourceElements of inLinks[I] to sourceIcons;}
     }//end if
     if(!outLinks.isEmpty()){
            for(int j=0; j<outlInks.length; j++){
            add the destinationElements of outlInks[j] to
                  destinationIcons;
            }
     }//end if
     deleting all the icons that are members of inLinks from the
         relations of each source in sourceIcons;
     deleting all the icons that ar ememebers of outLinks from the
         relations of each destination in destinationIcons;
    delete all icons in inLinks from the screen and the inner database;
    deleting all icons in outLinks from the screen and inner database;

}//end procedure deletingNode.
```

A.6 Visually Deleting Links

The steps to delete links contain:

- erase the clicked link from the screen;
- delete the clicked link from its source icon;
- delete the clicked lick from its destination icon;
- delete the clicked lick from the inner database;

The corresponding algorithm is as follows:

```
Algorithm 5.6 deletingLink

Input: clickedClinkto delete, and from the user's clicking);
Handling: delete the clickedLink from the screen and from the inner
      database, as well as removing its affection;
Output: the new screen display and the new inner database without the
      clickedLink.

Procedure: deletingLink{
     Erase the clickedLink from the screen;
     source←the source icon of the clickedLink;
     destination←the destination icon of the clickedLink;
     delete the clickedLink from the output relation of the source icon;
     delete the clickedLink from the input relationships of the
            destination icon;
     delete the clickedLink from the inner database;

}//end procedure deletingLink
```

A.7 Inserting Tutoring Unit in a Transition Node

The following algorithm 5.7 gives the steps to insert a tutoring unit. The steps to insert a cell are:

- Locate the cell to insert;
- Move all the successive cell forward a position;
- Initialize the newly inserted cell.

```
Algorithm 5.7 insertingUnit

Input: T: transition node to insert to, insertedPoint(an index of
      current unit, a new unit will be inserted in front of the index);
Handling: inserting a new unit to the related icon on the screen and in
      the inner database;
Output: new screen display and new inner database after inserting a new
      unit.

Procedure insertingUnit
     nmb←number of unit in T;
     while(insertedPoint>0 &&insertedPoint<the number of unit in T){
        expand the colume of all vectors of visual properties and inner
            attributes by 1;
        num++;
        for(int I=num; I>=insertedPoint; I--){
           copying the (I-1)th value of each visual properties vector to
                  the ith position;
           copying the (I-1)th value of each inner attribute vector to
                  the ith position;
        } //end for
        initialize the unit at the insertedPoint;
        re-paint the transition icon on screen;
        replace the old transition icon by the new transition icon in
            the inner database;
     }//end while

//end procedure insertingUnit.
```

Other algorithms such as deleting a unit, inserting a sub-unit and deleting a sub-unit are similar to this one.

Appendix B

Transition Nodes in the HTML Transition Network

In the example of capability transition network for teaching HTML, shown in Chapter 8, there are seventeen identified capability nodes and seventeen transition nodes. The following shows all transition nodes, tutoring units and sub-units.

Transition Node: Teach Stating Web
Prerequisite: N/A
Output Capability: Stating Web, 3 levels

Teach Stating Web				
scheme				
Teach Web Description				
Present introduction	Present description	Elicit description	Test description	
Teach Web Basic Features				
Introduce hypertext	Present multimedia	Present cross-platform	Elicit basic features	Test basic features
Teach Advanced Features				
Present distribution	Present dynamics	Present interaction	Elicit advanced features	Test advanced features

Transition Node: Teach Identifying Web Browsers
Prerequisite: Stating Web, minimum mastered level: 3rd
Output Capability: Identifying Web Browsers, 3 levels

Teach Identifying Web Browsers						
scheme						
Teach Web Browsers' Description						
Present description	Elicit description	Test description				
Teach Browsers' Job						
Present job description	Elicit browsers' job	Test Browsers' job				
Teach Popular Browsers						
Present introduction	Present Netscape	Present Mosaic	Present Lynx	Present Explorer	Elicit pop browsers	Test pop browsers

Transition Node: Teach classifying URLs
Prerequisite: Stating Web Browsers, minimum mastered level: 3rd
Output Capability: Classifying URLs, 3 levels

Teach Classifying URLs				
scheme				
Teach Definition of URL				
Present definition		Elicit definition		Test definition
Teach Structure of URL				
Present URL Structure	Present protocol	Present examples	Elicit URL structure	Test URL structure
Teach Popular URLs				
Present introduction	Present popular URLs		Elicit popular URLs	Test popular URLs

Transition Node: Teach Introduction to HTML
Prerequisite: Stating Web Browsers, minimum mastered level: 3rd
Output Capabilities: Stating Introduction to HTML, 3 levels;
Classifying URLs, 2 levels

Teach Introduction to HTML						
scheme						
Teach Definition of HTML						
Present definition			Elicit definition		Test definition	
Teach Basis of HTML						
Present tag	Present syntax	Present element	Present doc	Present definition	Elicit basis	Test basis
Teach Structure of HTML						
Present HTML structure	Present examples			Elicit HTML structure	Test HTML structure	

Transition Node: Teach Applying Basic Tags
Prerequisite: Stating Introduction to HTML, minimum mastered level: 3rd
Output Capability: Applying Basic Tags, 2 levels

Teach Applying Basic Tags					
scheme					
Teach Structure Tags					
Present html & head tags	Present Title tag	Present body tag	Present p tag	Elicit definition	Test definition
Teach Advanced Basic Tags					
Present Heading tags	Present comment tags		Elicit Basic tags		Test Basic tags

Transition Node: Teach Applying Links
Prerequisite: Classifying URLs, minimum mastered level: 3rd
Output Capability: Applying Links, 4 levels

Teach Applying Links				
scheme				
Teach Creating Links				
Present introduction	Present link tags	Present examples	Elicit definition	Test definition
Teach Linking Local Documents				
Present relative paths	Present absolute paths	How to Choose paths	Elicit linking local doc	Test linking local doc
Teach Linking Other Web Documents				
How to link remote docs	Present examples of remote docs		Elicit linking other docs	Test linking other docs
Teach Linking Specific Place within Document				
Creating links & anchors	Linking anchors within doc		Elicit linking within doc	Test linking within doc

Transition Node: Teach Applying Lists
Prerequisite: Applying Basic Tags, minimum mastered level: 2nd;
Applying Links, minimum mastered level: 3rd
Output Capability: Applying Lists, 3 levels

Teach Applying Lists				
scheme				
Teach Basic Tags of Lists				
Present introduction	Present basic tags	Elicit basic tags	Test basic tags	
Teach Typical Lists				
Present Numbered lists	Present Unordered lists	Present menu & directory lists	Elicit lists	Test lists
Teach Advanced Lists				
Present glossary lists	Present nested lists	Elicit advanced lists	Test advanced lists	

Transition Node: Teach Applying Tables
Prerequisite: Applying Basic Tags, minimum mastered level: 2nd;
Applying Links, minimum mastered level: 3rd
Output Capability: Applying Tables, 4 levels

Teach Applying Tables					
scheme					
Teach Basic Tags of Tables					
Present introduction	Present basic tags	Present examples	Use emptycells & caption	Elicit basic tags	Test basic tags
Teach Alignment					
Present alignment tags	Present examples	Present more examples		Elicit alignment	Test alignment
Teach Spanning Rows & Columns					
Present spanning tags	Present examples	Elicit spanning		Test spanning	
Teach Netscap Extension					
Present width of tab, column & border; Present cell spacing and cell padding			Elicit extension	Test extension	

Transition Node: Teach Applying Other Tags
Prerequisite: Applying Basic Tags, minimum mastered level: 2nd;
Applying Links, minimum mastered level: 1st
Output Capability: Applying Other Tags, 1 level

Teach Applying Other Tags								
scheme								
Teach Additional Functions								
Present char formatting	Present horizon rules	Present BR tag	Use black quoted tag	Present address	Present font sizes	Present background	Elicit other tags	Test other tags

Transition Node: Teach Applying Multimedia
Prerequisite: Applying Basic Tags, minimum mastered level: 2nd;
Applying Links, minimum mastered level: 4th
Output Capability: Applying Multimedia, 4 levels

Teach Applying Multimedia						
scheme						
Teach Using Images						
Present formats	Use image tags	Use image & text	Use Image & links	Know transparency	Elicit using images	Test using images
Teach Using Extended Images						
Wrapping text	Use alignment option	Modifying space	Present dim, scaling & border	Elicit Extended images	Test extended images	
Making Image Better						
Present rules for better images	Use alternatives	Elicit making image better	Test making image better			
Teach Using External Media						
Use external images	Use sound	Use video	Elicit using external media	Test using external media		

Transition Node: Teach Organizing Web
Prerequisite: Applying Basic Tags, minimum mastered level: 2nd;
Applying Links, minimum mastered level: 4th
Output Capability: Organizing Web, 5 levels

Teach Organizing Web						
scheme						
Teach Introduction to Organization						
Introduce Web pages & home pages	Elicit introduction to organization	Test introduction to organization				
Teach Setting Goals						
Present Published information types	Use rules for setting goals	Elicit setting goals	Test setting goals			
Teach Getting Main Topics						
Introduce breaking up text	Present examples	Elicit getting main goals	Test getting main goals			
Teach Organizing and Navigating						
Present Intro to organization	Use hierarchies	Use linear org	Combine linear & hierarchies	Organize Web	Elicit org	Test org
Teach Story Boarding						
What is story boarding?	Use Rules for story boarding	Elicit story boarding	Test story boarding			

Transition Node: Teach Applying Web Servers
Prerequisite: Applying Basic Tags, minimum mastered level: 2nd;
Applying Links, minimum mastered level: 4th
Output Capability: Applying Web Servers, 5 levels

Teach Applying Web Servers						
scheme						
Teach Introduction to Servers						
What is a server?	Introduce job of servers	Elicit introduction to servers	Test introduction to servers			
How to find a server?						
Present ways to find a server		Elicit finding a server		Test finding a server		
Install Server Software						
Present servers for Windows	Present servers for Unix	Present servers for Mac	Elicit installing server software	Test installing server software		
Organize & Install Your HTML Files						
Present rules for installing	Present File types	How to Install?	Move files between systems	What is your URL?	Elicit org	Test org
Better Server Administration & Design						
Use rules for better server administration		Elicit better server administration & design		Test better server Administration & design		

Transition Node: Teach Applying CGI
Prerequisite: Applying Web Servers, minimum mastered level: 4th
Output Capability: Applying CGI, 6 levels

Teach Applying CGI				
scheme				
Teach What CGI is				
Present definition of CGI	Introduce how CGI work	Present examples	Elicit definition of CGI	Test definition of CGI
Teach Use Conditions of CGI				
Present three conditions to use CGI		Elicit use conditions		Test use conditions
Teach Setting CGI on Servers				
Present CGI on CERN	Present CGI on NCSA	Elicit setting CGI on servers		Test setting CGI on servers
Teach CGI Script Behavior				
Present output header	Present output data	Elicit CGI script behavior		Test CGI script behavior
Teach Interactive Search				
Use rules	Present <ISINDEX> tag	Present sealing string &script URL	Elicit interactive search	Test interactive search
Teach Special Script Output				
Present responding	Introduce noResponse	Elicit special script output		Test special script output

Transition Node: Teach Applying Forms
Prerequisite: Applying CGI, minimum mastered level: 4th;
Output Capability: Applying Forms, 6 levels

Teach Applying Forms					
scheme					
Teach Anatomy of Forms					
Present form input tag	Introduce Script to process form	Elicit anatomy of forms	Test anatomy of forms		
Teach Simple Form Layout					
Present submission button	Elicit simple form layout	Test simple form layout			
Teach Text Input Fields					
Present radio button	Present check boxes	set up default values	Elicit text input fields	Test text input fields	
Teach Advanced Form Layout					
Present selections	Present text areas	Present hidden fields	Elicit advanced form layout	Test advanced form layout	
Teach Image Map					
Introduce definition of image map	Elicit image map	Test image map			
Teach Creating Image Map					
Get images	Create map files	Install map & program	How to link all together	Elicit creating image map	Test creating image map

Transition Node: Teach Simple Design
Prerequisite: Applying Basic Tags, minimum mastered level: 2nd;
Applying Links, minimum mastered level: 2nd
Output Capability: Designing Simple Web, 2 levels

Teach Simple Design					
scheme					
Teach Design Rules					
Write for online	Design & layout	Use links	Good habits & hints	Elicit design rules	Test design rules
Teach Simple Uses (Basic Tags & Links)					
Present example: P53	Present example: P66	Present example: 74-75	Present example: 85-88	Test simple uses	

Transition Node: Teach General Design
Prerequisite: Applying Lists, minimum mastered level: 2nd
Applying Tables, minimum mastered level: 3rd;
Applying Other Tags, minimum mastered level: 1st;
Applying Web Servers, minimum mastered level: 3rd.
Output Capability: Designing General Web, 4 levels

Teach General Design						
scheme						
Teach Design Rules						
Write for online	Design & layout	Use links	Good habits & hints	Elicit design rules	Test design rules	
Teach Simple Uses (Basic Tags & Links)						
Present example: P53	Present example: P66	Present example: 74-75	Present example: 85-88	Test simple uses		
Teach Using Lists, Tables, Servers & Other Tags						
Present exp:106	Present exp:107	Present exp:108	Present exp:110	Present exp:112	Present exp:201-222	Test uses
Teach Complicated Uses						
Present exp:119-124	Present exp:125-127	Present exp:134-137	Present exp:server	Test general uses		

Transition Node: Teach Business Design
Prerequisite: Applying Lists, minimum mastered level: 2nd;
Applying Tables, minimum mastered level: 3rd;
Applying Other Tags, minimum mastered level: 1st;
Applying Web Servers, minimum mastered level: 3rd;
Organizing Web, minimum mastered level: 4th;
Applying Forms, minimum mastered level: 5th
Output Capability: Designing Business Web, 4 levels

Teach Business Design					
scheme					
Teach Design Rules					
Write for online	Design & layout	Use links	Good habits & hints	Elicit design rules	Test design rules
Teach Simple Uses					
Present example: Bookworm		Present example: Vegetable Planning		Present example Papillion	Test simple uses
Teach Business Uses					
Present example of company		Present example of shopping			Test Business uses
Teach Complicated Uses					
Present example: Online Books		Present example: Encyc			Test complicated uses

Appendix C

Algorithms for Creating

Multiple Alternative Courses

This appendix presents the algorithms for creating multiple alternative courses corresponding to the content in Chapter 6.

Algorithm 6.1 getAllCapPreTransV

```
Input: allCapV: all capability nodes in Tnet,
   allTransV: all transition nodes in Tnet, and
   allLinkV: all links in Tnet.
Handling: create relations of all predecessor transition nodes of all
   capability nodes.
Output: allCapPreTransV.

Procedure: getAllCapPreTransV{
01    allCapPreTransV=null;
02        for all Ci ∈ allCapV{
03              theCapPreTransV=null;
04              inLinkV ← all input links of Ci;
05              for all lj ∈ inLinkV{
06                  theCapPreTransV.addElement(source element of lj);
07              }
08              allCapPreTransV.addElement(<ci, theCapPreTransV>)
09        }
      return allCapPreTransV;
}//end Procedure getAllCapPreTransV
```

Analysis of Algorithm 6.1;

Let the number of capability nodes in the vector allCapV be *Nc,* the total number of input links of all capabilities be *Nl,* and the average number of input links of each capability node be *Vl.* For each capability node, the times of inner loop (line 05 ~ line 07) are exactly the number of input links of the capability node. So the total time the algorithm takes is in

$$O(Nc, Nl) = O(Nc * Vl) = O(Nl)$$

Algorithm 6.2 getAllCapAncestCapV

```
Input: allCapV: all capability nodes,
      allCapPreCapV: the instant predecessor capabilities of each
      capability
Handling: finding all ancestor capability nodes of all capability nodes
   in Tnet.
```

```
Output: allCapAncestCapV: all ancestor capability nodes of all
   capability nodes.

Procedure: getAllCapAncestCapV{
   allCapAncestCapV=null;
   theCapAncestCapV;
   for all Ci∈ allCapV {
      theCapAncestCapV←getTheCapAncestCapV (Ci, allCapV)
            (see Algorithm 6.3);
      allCapAncestCapV.addElement(<Ci, theCapAncestCapV>);
   }
   return allCapAncestCapV;
}//end Procedure getAllCapAncestCapV.
```

Analysis of the Algorithm 6.2:

Let the number of capability nodes be *Nc*. In the worse case, i.e., all capabilities are connected into a link, the root node will call getTheCapAncestorCapV exactly (*Nc-1*) times; the direct predecessor of the root node will call getTheCapAncestorCapV exactly (*Nc-2*) times, ... getTheCapAncestorCapV takes a time in *O* (*Nc*) in the worse case (see the analysis of the algorithm 6.3). So the algorithm will take a time in the worse case in

$$T(Nc)=C*((Nc-1)+(Nc-2)+...+2+1) = C * (Nc*(Nc-1)/2).$$

That is $O(Nc^2)$.

```
Algorithm 6.3: getTheCapAncestCapV

Input: theCap: a capability node.
   allCapPreCapV: all predecessor capability nodes of all capability
   nodes.
Handling: finding all ancestor capability nodes of theCap.
Output: a vector containing all ancestor capabilities of theCap.

Procedure: getTheCapAncestCapV{
   theCapAncestCapV=null;
   theCapPreCapV←Cj, where  ∃<theCap, Cj> ∈allCapPreCapV;
   theCapAncestCapV.addElement(∀Ck∈theCapPreCapV);
   for all Ck∈ theCapAncestCapV{
      newAncestCapV=null;
      newAncestCapV←getTheCapAncestCapV(Ck, allCapPreCapV);
      theCapAncestCapV.addElement(∀Cw∈newAncestCapV);
   }
   return theCapAncestCapV;
}//end Procedure getTheCapAncestCapV.
```

Analysis of the Algorithm 6.3:

This algorithm recurrently calls itself. Let the total number of capability nodes is *Nc*; in the worse case, i.e., all capability nodes except for *theCap* are ancestors of *theCap*, the algorithm takes a time in *O* (*Nc*) because each capability node is visited exactly once.

Algorithm 6.4 Removing Redundancy and Invalid Elements

Input: *Tnet*, K: known capabilities, and G: goal capabilities.
Handling: finding and removing all redundant and invalid elements.
Output: *Cnet*1:the remaining parts of *Tnet* after the removing operation.

Procedure: RemovingRedundancyAndInvalidElements

```
C← all capability nodes in Tnet
T← all transition nodes in Tnet
L← all links in Tnet
Boolean changed = true;
While(changed){
    changed←false;
    for(int i=0, i<C.length; I++){
          if(C[i].state == CAP_FULL_MASTERED){
                remove C[i] from C;
                changed = true;
          }
          else if(C[i] has successive transition nodes and
                C[i]∉G and C[i] ∈K and the maximal mastered
                      level of C[i] is grater than all
                            levels of C[i]'s output links){
                      remove C[I] from C;
                      changed = true;
          }
          else if(C[i] has successive transition nodes,
                C[i] ∈G, the state of C[i]'s goal level is
                LEVEL_MASTERED, the maximal mastered level
                of C[i] is greater than all levels of its
                output links){ remove C[i] from C;
                changed = true;
          }
          else if(C[i] is not a goal, C[i] has no
                successive transition node, the state of
                C[i] is CAP_GOAL_MASTERED){
                remove C[i] from C;
                changed = true;
          }
          else if (C[i] has no input link in L, C[i] has
                no output link in L){
                remove C[i] from C;
                changed =true;
          }
          else if { C[i] has no output link, C[i] is not a
                goal){
                remove C[i] from C;
                changed = true;
          }
    }//end for loop
    for(int j=0; j<T.length, j++){
       if(T[j].state ==TRANS>PASSED){
                remove T[j] from T;
                changed = true;
       }
       else if (T[j] has no output capability in C){
```

```
            remove T[j] from T;
            changed = true;
        }
    }//end for (j..)
    for(int k=0; k<L.length; k++){
        if (L[k] has no output capability in C){
            remove L[k] from L;
            changed = true;
        }
        else if(L[k] has input capability in C){
            remove L[k] from L;
            changed = true;
        }
        else if(L[k] 's destination is a goal, level of L[k]
            is less than goal level of C){
            remove L[k] from L;
            changed = true;
        }
    }//end for(k..)
}//end while
Cnet1← C ∪ T ∪ L
return Cnet1;
} //End Procedure: removing Redundancy and Invalid Elements
```

Analysis of Algorithm 6.4:

This algorithm includes three *for* loops. Obviously, the algorithm takes a time in

$$T(C, T, L) = k * (C.length + T.length + L.length) = O(max\{C.length, T.length, L.length\})$$

Where *C*.length is the number of capability nodes,

T.length is the number of transition nodes,

L.length is the number of links, and

k is a constant.

Algorithm 6.5 getMaxGoalCap

```
Input: allCapV: all capability nodes in Tnet,
    allCapAncestCapV: all ancestor capability nodes of all capabilities,
    and
    allGoalV: all goal capabilities.
Handling: finding a goal capability whose ancestor (predecessor)
capabilities contain the most other goals in allGoalV.
Output: a maximal goal capability.

Procedure: getMaxGoalCap{
    maxGoal=null;
    maxGoalNmb=0;
    containedGoalNmb=0;
01    for(int i=0; i<allGoalV.length; i++){
```

```
02        ancestCapV←finding all ancestor capabilities of allGoalV[i]
03        containedGoalNmb←counting the goal contained in
                        ancestCapV;
04        if(containedGoalNmb > maxGoalNmb){
05            maxGoalNmb←allGoalV[i];
06            maxGoalNmb←containedGoalNmb;
07        }
08    }//end for loop
      return maxGoal;
}//end Procedure getMaxGoalCap.
```

Analysis of Algorithm 6.5:

Let allGoalV.length = *Ng* and the number of capability nodes in *Tnet* be *Nc*. Obviously, the loop times of *for* block (from line 01 to line 08) is *Ng*. The line 02 is to find all ancestors of the i-*th* goal. The time the line 02 takes is *O* (*Nc*) in the worse case. Each of other statements takes a constant time. So in the worse case (i.e. all capability nodes are goals and all capability nodes are connected into a link)), the algorithm takes a time in *O* (Nc^2).

Algorithm 6.6 getAllHeadCapV(get all head capabilities)

Innput: allCapV, allCapAncestCapV, and allGoalV.
Heandling: finding all head capabilities in allGoalV, these heads consticute a division of allGoalV.
Output: a set of all head capabilities in allGoalV.

Procedure: getAllHeadCapV{

```
      allHeadCapV=null;
      remainedGoalV=allGoalV;
01    while(remainedGoalV.isEmpty()){
02        candidateHead←getMaxGoal(allCapV,
             allCapAncestCapV, remainedGoalV);
03        allHeadCapV.addElement(candidatehead);
04        ancestCapV←finding all ancestor capabilities of
                  candidateHead in allCapHeadCapV;
05        for all ancestor ai∈ancestCapV{
06            if ai∈remainedGoalV{
                  remainedGoalV.removeElement(ai);
07            }
08        }//end for
09    } //end while
      return allHeadCapV;
}//end Procedure getAllHeadCapV.
```

Analysis of Algorithm 6.6:

In the worse case (i.e. all capabilities are goals and they are connected into a link), the statement in line 03 takes a time in *O* (Nc^2). In this case, the *while* loop in line 01 takes a constant time because the *for* loop removes all ancestors of the root node. So in this case the algorithm takes a time in *O* (Nc^2). In another case that every goal is not an ancestor of the other goals, the algorithm takes a time in *O* (*Ng*(Nc-Ng)*).

```
Algorithm 6.7 getTheheadSubNetsV

Input: theHead: a head capability, allCapV, allCapPreTransV,
      allTransV, and allTransPreCapV,
Handling: finding all possiblle sub-networks of theHead.
Output: a vector, theHeadSubNetsV, containing all
      sub-networks of theHead; each sub-network consists of
      all linked elements in the sub-network.

Procedure getTheHeadSubNetsV{
      TheHeadSubNetsV=null;
      preTransV←finding all predecessor transition nodes of theHead
      from all CapPreTransV;
      visitedPreTransV=null;
      if(preTRansV.siEmpty(){
            for(all t∈preTransV{
                  if (visitedPreTransV.nonContain(t)){
                        visitedPreTransV.addElement(t);
                        iniNet←link:<t, theHead>;
                        theHeadSubNetsV.addElement(iniNet);
                        candPreTransPreCapV←all predecessor
                              capability nodes of t from
                                    allTransPreCapV;
                        forming all links from each predecessor
                              capability nodes of t to the
                              transition nodes t and add them to
                              the network of t in
                              theHeadSubNetsV;
                        for all c∈candPreTransPreCapV{
                              thePreCapSubNetsV←recur:
                                    getTheHeadSubNetsV(c, allCapV,
                                    allCapPreTransV, allTransV,
                                    allTransPreCapV);
                              previousNet←the network where the
                                    t resides in theHeadSubNetsV;
                              neTnetsV←combining each subnetwork
                                    in thePreCapSubNetsV with the
                                    previousNet;
                              replace the previousNet in
                                    theHeadSubNetsV with all
                                    networks in newNetsV;
                        } //end for all c…
                  }//end if
            }//end for
      }end if
}//end Procedure getTheHeadSubNetsV
```

Analysis of Algorithm 6.7:

Let the number of transition nodes be *Nt*. If *Tnet* is a binary tree and all leaf nodes are transition nodes, the algorithm is a breadth-first search. Each intermediate transition node has two branches; i.e. each

intermediate transition nodes will create two sub-networks. So in this case the algorithm takes a time in $O(2^{(Nt/2)})$.

Algorithm 6.8 getTheHeadCapOneNetCompleteNetsV

```
Input: theHead, theSubGoalsV: all sub-goals of theHead,
      theNet: the sub-network to check.
Handling: generating all complete sub-networks of theNet.
Output: a vector, theHeadCapOneNetCompleteNetsV, containing
      all complete sub-networks of theNet.

Procedure: getTheHeadCapOneNetCompleteNetsV{
      theHeadCapOneNetCompleteNetsV=null;
      lostGoalV←find all goals that are in theSubGoalV and
            not in theNet;
      if(lostGoalV.isEmpty()) return theNet;
      if(lostGoalV.isNonEmpty()){
            //find all sub-head capabilities in lostGoalV;
            allSubGoalV←getAllHeadCapV(lostGoalV);
            allSubHeadCapSubNetsLinksV←getAllHeadCapSubNetsV(
                  allHeadCapV);
            for all nextSubHead∈allSubHeadCapV and all
                  netSubHeadSubNetsV∈allSubHeadSubNetsV{
                  theSubHeadCompleteSubNetsV=null;
                  for all nextSubNet∈nextSubHeadSubNetsV{
                        theSubHeadSubGoalV←get all sub goals
                        of the nextSubHead;
                        theSubHeadNexTnetCompleteNetsV←
                              getTheHeadCapOneNetCompleteNetsV(
                              nextSubHead, nextSubNet,
                              theSubHeadSubGoalV);
                  }// end for all nextSubNet..
                  theSubHeadCompleteSubNetsV.addElement(
                        theSubHeadNex Tnet CompleteNetsV);
            } //end for all nextSubHead..

            //get all combination of allSubHeads' complete
            //nets; each combination is an alternative
            //sub-case of the nextSubHead; then merge each
            //combination to theNet.
            allSubHeadCompleteNetsCombV←
                  getAllHeadNetsVCombinationV(
                        theSubHeadCompleteSubNetsV);
            for allNextCombination∈
                  allSubHeadCompleteNetsCombV{
                  nextNewNet←theNet.expandElement(
                              nextCombination);
                  theHeadCapOneNetCompleteNetsV.addElement(
                        nextNewNet);
            }//end for
      }//end if(lostGoalV.isEmpty())
      return theHeadCapOneNetCompleteNetsV;
}//end Procedure: getTheHeadCapOneNetCompleteNetsV.
```

Analysis of Algorithm 6.8:

In this algorithm, the sub-procedure: *getAllHeadNetsVCombinationV* attempts to get all combinations of all sub-networks in *theSubHeadCompleteSubNetsV*. Each combination will cover all sub-goals lost in theNet. The sub-procedure: *getTheHeadCapAllSubNetsVCombinationV* is to find all combinations of all complete sub-networks of theHead, and the sub-procedure: *getAllHeadCapAllSubNetsCombinationV* to get all combinations of all complete sub-networks of all head capabilities. In fact, each of combination of all complete sub-networks of all head capabilities is a course we want to create. The time the algorithm takes mainly depends on the recurrent statement *getTheHeadCapOneNetCompleteNetV* that takes a time in $O(2^{(Nt/2)})$ in the similar case to the algorithm 6.7. So the algorithm takes a time in $O(2^{(Nt/2)})$ in the similar case to that in the algorithm 6.7.

Algorithm 6.9 creating*Cnet*

```
Input: Tnet: a capability transition network,
       K: set of known capabilities, and
       G: set of learning goals (capabilities).
Handling: Creating multiple alternative courses that satisfy the
        transition of all capabilities in G based on Tnet.
Output: A vector, Cnet, containing all created courses that
       satisfy the needs of all capabilities in G; each course
       is a set of links.

Procedure: creatingCnet{
       Cnet←Tnet;
       Cnet←removing all redundant elements and invalid
             elements in Cnet;
       Generating basic relations in Cnet including
             AllCappreTransV, allTransPreCapV, allCapPreCapV,
             allTransPeTransV, allCapAncestCapV,
             allTransAncestTransV, allCapAncestTransV, and
             allTransAncestCapV;
       allHeadCapV←finding all head capabilities in Tnet;
       allHeadCapSubNetsV←getAllHeadCapSubNetsV(allHeadCapV);
       allHeadCapCompleteSubNetsV←getAllHeadCapCompleteNetsV(
             allheadCapV, allHeadCapSubNetsV);
       Cnet←getAllHeadCapCompleteSubNetsVCombinationV(
             allHeadCapV, allHeadCapCompleteSubNetsV);
}//end Procedure creatingCnet.
```

Analysis of Algorithm 6.9:

The time the algorithm takes mainly depends on the statement *getAllHeadCapSubNetsV* that takes a time in $O(2^{(Nt/2)})$ according to the analysis in the algorithm 6.8. So in the similar case as analyzed in the algorithm 6.8, this algorithm takes a time in $O(2^{(Nt/2)})$. Though we cannot reduce the complexity of the algorithm for creating alternative courses, the *Tnet* structure simplifies the overall transition network. So the algorithm can be still used for creating alternative courses in practice.

Appendix D

Algorithms for Computing *Utilities* and Review of Baysian Rule

This appendix describes algorithms for computing *utilities* defined in Chapter 7 and some relevant probability formulae used for identifying necessary tutoring units in a course (Chapter 7).

D.1 Algorithm for Computing *Utilities*

Algorithm 7.1 ComputeCourseUtility

```
Input: resType: vector including all resource types,
       MediaType: vector including all media types,
       StrategyType: vector including all strategy types,
       PreferredResTypeDegree: vector including preferred degree of all
            resource types,
       PreferredMediaTypeDegree: vector including preferred degree of
all
            media types;
       PreferredStrategyDegree: vector including preferred degree of all
            strategy type;
       DificultyDegree: vector, all resources' difficult degree
       ImportanceDegree: vector, all resources' importance degree.
       Course: vector including all elements in the selected course

Handling: compute the utility of the course

Output: U(Course): the utility of the Course

Procedure ComputingCourseUtility
       NecessaryTutoringUnit←compute necessary tutoring units in Course;
       resTypeDistribution←compute the resource type distribution in
       NecessaryTutoringUnit;
       mediaTypeDistribution←compute the media type distribution in
       NecessaryTutoringUnit;
       strategyTypeDistribution←compute the strategy type distribution
       in NecessaryTutoringUnit;
       U(Course)=null;
       For all Ai∈NecessaryTutoringUnit{
            U(Ai)=0;
            For all resource groups in Ai, Gj
                 ResTypeUtility(Ai, Gj) ←
                      preferredResTypeDegree*ResTypeDistribution(Ai);
                 mediaTypeUtility(Ai, Gj) ←
                      preferredMediaTypeDegree*MediaTypeDistribution(
                      Ai);
                 StrategyTypeUtility(Ai, Gj) ←
```

```
                    preferredStrategyTypeDegree*StrategyTypeDistribution(
          Ai);
      }   //end for all resource groups
```

$$U(Ai) = \max_{resGroup}\{\frac{(resTyprUtility * mediaTypeUtility * strategyTypeUtility) * impor\tan ceDegree}{\sum(dificultyDegree * sizeOf\ \mathrm{Re}source)}\}$$

```
      U(Course) ←U(Course)+U(Ai);
      }//end for all Ai…
      return U(Course)
//end Procedure computingCourseUtility
```

D.2. Reviews of Basic Probability Formulae

Let *A, B, E, H, …* stand for events, and *P(A)* stand for the probability of event *A*, some basic probability formulae related to our approaches are reviewed as follows:

- Three basic axioms:

  $0 \le P(A) \le 1$ (D.1)

  $P(Sure\ Proposition) = 1$ (D.2)

  $P(A\ or\ B) = P(A) + P(B)$ if *A* and *B* mutually exclusive (D.3)

- Conditional probability of joint events

$$P(A|B) = \frac{P(A,B)}{P(B)} \quad (D.4)$$

- $$P(A) = \sum_i P(A,B_i) = \sum_i P(A|B_i)P(B_i) \quad (D.5)$$

  where B_i is a set of exhaustive and mutually exclusive proposition.

- $$P(A|K) = \sum_i P(A|B_i,K)P(B_i|K) \quad (D.6)$$

  where B_i is the same as that in Eq. (D.5)

- Chain rule:

$$P(E_1,E_2,..,E_n) = P(E_n|E_{n-1},..,E_2,E_1)P(E_{n-1}|E_{n-2},..,E_2,E_1)...P(E_2|E_1)P(E_1) \quad (D.7)$$

- Basic Bayes rule

$$P(H|e) = \frac{P(e|H)P(H)}{P(e)} \tag{D.8}$$

where H: a hypothesis,

e: an observation or evidence,

$P(e|H)$: likelihood, the probability of evidence e in the context of the hypothesis H being true ,

$P(H|e)$: the posterior probability,

$P(H)$: the prior probability of the hypothesis H, and

$P(e)$: a normalizing constant, $P(e)=P(e \mid H)P(H)+P(e \mid \neg H)$.

Eq.(D.8) states that the belief we accord a hypothesis H upon obtaining evidence e can be computed by multiplying our previous belief $P(H)$ by the likelihood $P(e|H)$ that e will materialize if H is true.

D.3. Posterior Odds

By Bayesian rule, we have

$$\frac{P(H|e)}{P(\neg H|e)} = \frac{P(e|H)P(H)}{P(e)} \cdot \frac{P(e)}{P(e|\neg H)P(\neg H)}$$

$$= \frac{P(e|H)}{P(e|\neg H)} \cdot \frac{P(H)}{P(\neg H)}. \tag{D.9}$$

Defining prior odds on H as

$$O(H) = \frac{P(H)}{P(\neg H)} = \frac{P(H)}{1-P(H)}, \tag{D.10}$$

and the likelihood ratio as

$$L(e|H) = \frac{P(e|H)}{P(e|\neg H)}. \tag{D.11}$$

the posterior odds

$$O(H|e) = \frac{P(H|e)}{P(\neg H|e)} \tag{D.12}$$

are given by the product

$$O(H|e) = L(e|H)O(H). \tag{D.13}$$

D.4 Algorithms for Identifying Necessary Tutoring Units

Algorithm 7.2 Identifying MaxPassedTutoringUnit

Input: conditional probability matrix *M* (in figure 7.4),
T={A1, A2, …, An>, where *Ai* is a tutoring unit in *T*;
C={C1, C2, …, Cj, …, Cm}: where *Cj* is an output capability of *T*,
And *C1k1, C2k2, …, Cjkj, …, Cmkm:* indicated levels of all output capabilities.
Handling: finding the maximal tutoring unit in *T*, which corresponds to the acquirement of *C1k1, …, Cjkj, …, Cmkm*.
Output: maximal passed tutoring unit.

Procedure: IdentifyingCorrespondingTutoringUnit

lastConditionalProbVector←find the vector in the *n-th* row and the *m-th* column in figure 7.4;
conditionalProbMatrix←compute the conditional probability matrix based on lastConditionalProbVector by linear imitation method;
we get:

$$cond\,\mathrm{Pr}obMatrix = \begin{bmatrix} P(C1k1 \mid A1) & P(C2k2 \mid A1) & \ldots & P(Cmkm \mid A1) \\ P(C1k1 \mid A2) & P(C2k2 \mid A2) & \ldots & P(Cmkm \mid A2) \\ \ldots & \ldots & \ldots & \ldots \\ P(C1k1 \mid An) & P(C2k2 \mid An) & \ldots & P(Cmkm \mid An) \end{bmatrix}$$

Computing

$$MValue = \begin{bmatrix} \prod_{j=1}^{m} P(Cjkj \mid A1) \\ \prod_{j=1}^{m} P(Cjkj \mid A2) \\ \ldots \\ \prod_{j=1}^{m} P(Cjkj \mid Ai) \\ \ldots \\ \prod_{j=1}^{m} P(Cjkj \mid An) \end{bmatrix}$$

$$\max = indexOfMax\left\{\prod_{j=1}^{m} P(Cjkj \mid A1), \prod_{j=1}^{m} P(Cjkj \mid A2), \ldots, \prod_{j=1}^{m} P(Cjkj \mid An)\right\}$$
in *Mvalue*.

return *max;*
//end procedure identifyingCorrespondingTutoringUnit

The analysis of this algorithm is easy because the time it takes depends just on the number of tutoring units and the number of output capability levels. The following algorithm 7.3 is used for identifying all necessary tutoring units to meet certain learning goals.

Algorithm 7.3 IdentifyingNecessaryTutoringUnits

Input: conditional probability matrix *M* (as shown in figure 7.4);

```
      T=<A1, A2, …, An>, Ai is a tutoring unit in T,
      C={C1, C2, …, Cj, …, Cm}, Cj is an output capability,
      Ck={C1k1, C2k2, …, Cjkj, …, Cmkm}, Cjkj is the known level of Cj,
            and
      Cg={C1g1, C2g2, …, Cjgj, …, Cmgm}, Cjgj is the goal level of Cj.
Handling: finding necessary beginning tutoring unit and ending tutoring
      unit for acquiring Cg.
Output: a vector N = <begininingUnit, endingUnit>

Procedure: IdentifyingNecessaryTutoringUnits
      beginningUnit←identifyingMaxPassedTutoringUnit(M, T, Ck);
      endingUnit←identifyingMaxPassedTutoringUnit(M, T, Cg);
      return vector <beginningUnit, endingUnit>;
//end Procedure IdentifyingNecessaryTutoringUnits.
```

Appendix E

Glossary

Aggregated capability: a capability, which integrates several simple capabilities to form an incremental capability sequence.

Anet: an activity network, which is the currently tutoring activity sequence recommended by the system.

Authoring state: the signs representing the completeness of node information, which is used for visual navigation by a curriculum author.

Capability: a term from Gagné, which refers to teaching outcomes or human knowledge

Cnet: course networks, which is the output of the *VCOURSE* model and includes multiple alternative paths to support the achievement of a group of learning goals.

Course: a course is a sub-network of the capability transition network for a given domain, which covers just the necessary nodes and links for a particular learner's learning goals

Dynamic resource: a simulation, diagnosis or guidance program

Head capability (node): same as a head goal

Head goal: a selected learning goal by a learner, whose successive capabilities do not include any learning goal

Implied goal capabilities: in the *VCOURSE* model, when a student selects a capability as his/her learning goal, this system considers the capability's prerequisite capabilities that are not known or implied known capabilities as implied learning goals

Implied known capabilities: when a student selects a capability as his/her known capability, the system considers the prerequisite capabilities of the selected capability as implied known capabilities

Inner attributes: necessary information in a node for building capability transition networks, for instance the type of a capability, a description of the capability, and the levels involved in the capability node.

Learning state: the dynamic navigation mechanism for learning processes; for instance a capability node has several states that correspond to the current learner's states: full-mastered, partly-mastered, etc. A transition node has also several states related to the current context and the current student, for example recommended, enabled, disabled, etc.

Maximal head goal: a head goal whose predecessor capabilities include more learning goals than any other head goal in a given capability transition network.

Multiple level capability: a kind of teaching outcome, which can be identified as several incremental capability levels; this system provides a mechanism to organize tutoring activities for each capability level.

Resource group: a set of resources that are organized based on certain teaching strategies.

Scheme: a visual cell in a transition node in which a curriculum author can define the frame of the transition node for example how many tutoring units are needed, what resource types are needed, and what is the success criterion, etc.

State-driven reasoning: based on a group of rules the system can automatically infer the states of all visual nodes and cells when some node or cell state changes.

Static resource: a piece of media such as text, picture, image, audio data, etc.

Sub-unit: an instructional event that contains one or more resource groups; with the help of running programs a sub-unit can interactively carry out teaching activities.

Tnet: a capability transition network for a given domain, which is the central structure in our visual curriculum model, and is an AND/OR graph consisting of capability nodes, transition nodes, prerequisite links and output links.

Transition node: a node that represents a sequence of two-level tutoring activities (tutoring units and sub-units); activating a transition node requires that its prerequisite capabilities be acquired at certain levels, and the output of a transition node is some capabilities.

Tutoring unit: a member in a transition node, which contains a sequence of instructional events and can deliver certain capability levels

Utility: a proposed measure that is used for evaluating course adaptability to a particular student

VACT: the Visual Activity sequence, a sub-model in the ***VITCAM***, with which the system can dynamically recommend the current course and tutoring activities for a particular learner

VCOURSE: the Visual Course model, a sub-model in the ***VITCAM***, with which the system can create multiple alternative courses for a particular learner

Visual cell: any element in a visual composite icon for representing either the overall view of a capability or a capability level or the overall view of a transition node or a tutoring unit or a sub-unit; a user can click the visual cell to view or edit its inner attributes or set it as a learning goal or carry out interactive actions

Visual properties: necessary information in a node for drawing the node, for example, node shapes, sizes, labels, colors, position, etc.

VITCAM: the Visual Interactive Transition, Course and Activity Manager; it is the proposed visual curriculum model and consists of three sub-models: *VTRANS*, *VCOURSE* and *VACT*.

VTRANS: Visual Transition model, the backbone sub-model in the *VITCAM* model, with which a curriculum author can organize transition nodes to build capability transition networks.