CLIENT-SERVER SYSTEM FOR WEB-BASED VISUALIZATION AND ANIMATION OF LEARNING CONTENT

Vom Fachbereich Informatik der

Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades

Doktor-Ingenieur (Dr.-Ing.)

von

Zheng LU, M.Sc

geboren in Peking

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Referenten:	Prof. Dr. Andreas Koch
	Ass. Prof. Dr. Abdulhadi Shoufan
Tag der Einreichung:	09. Januar 2018
Tag der mündlichen Prüfung:	05. Februar 2018

Darmstadt, January 2018

D17

LU, Zheng : Client-Server System for Web-based Visualization and Animation of Learning Content
Darmstadt, Technische Universität Darmstadt,
Jahr der Veröffentlichung der Dissertation auf TUprints: 2018
Tag der mündlichen Prüfung: 05.02.2018

DECLARATION

Hiermit versichere ich, die vorliegende Dissertation ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, January 2018

Lu, Zheng

ABSTRACT

E-learning is an effective means for academic and continuous education nowadays. An important feature of E-learning is using animation to demonstrate the teaching contents. With the increasing popularity of multimedia technology, instructors started to demonstrate the complicated computer algorithms and abstract data structure using multimedia animation. As a result, Algorithm Visualization (AV) is one of the most frequently used tools in E-learning. Numerous researches showed that applying animations to illustrate complex algorithms enables students to understand these algorithms better and faster. But there are also many teachers who oppose the usage of AV in the teaching process. In order to make AV help students better in understanding the topics they need to learn, certain guidelines need to be considered while designing AV animations.

After reading and analysing a large number of related papers, four pedagogic theories were selected. These are **Epistemic Fidelity**, **Cognitive Constructivism**, **Dual Coding** and **Individual Differences**. It is believed that if developers observe these four learning theories when developing AV animation, the quality of AV can be effectively improved. An on-line platform named DLD-VISU for learning and teaching Digital Logic Design was developed based on these four theories. Specific implementation concepts and methods were developed in order to support these theories in DLD-VISU using advanced software engineering and programming techniques. DLD-VISU uses the MVC architectural pattern to achieve the separation of **Model**, **Controller** and **View** and uses the Dependency Injection (DI) method to achieve Inversion of Control (IoC). This can decouple classes and greatly reduce the difficulty of program maintenance and update. In addition, we use the Object Relational Mapping (ORM) technology to replace the traditional JDBC to access database.

DLD-VISU is not only able to display animation on web pages, but also serves as an AV development platform. It provides a powerful framework and a rich library of graphical functions. This framework encapsulates many powerful functions such as IoC, ORM and **Action dispatcher**. Developers can use it to quickly build an AV animation based on the MVC architectural pattern. An extensive library of graphical functions can help developers to draw the front-end animations fast. In order to present the ease-of-use of DLD-VISU, we use it to develop AV animation for different algorithms of binary decision trees as an important class of machine learning classifiers.

Another objective of DLD-VISU is to construct unified AV management platform. All the animations on this platform employ the

same style and GUI, which can spare the time that students spend to adapt to different AV programs. Furthermore, the platform provides a great amount of management modules that make it very convenient for teachers to publish and manage their own AV programs and perform real-time observation and analysis of student's learning using AV.

We conducted a series of tests on this platform, including dividing the students into two groups — one utilized the DLD-VISU platform and the other employed traditional methods, and testing both groups of students with same questions. The results show that DLD-VISU is helpful for improving students' achievements.

ZUSAMMENFASSUNG

E-Learning ist heutzutage ein wirksames Mittel für die akademische und kontinuierliche Bildung. Ein wichtiges Merkmal von E-Learning ist die Verwendung von Animationen, um die Lehrinhalte zu demonstrieren. Mit der zunehmenden Popularität der Multimediatechnologie begannen die Ausbilder, die komplizierten Computeralgorithmen und die abstrakte Datenstruktur mit der Hilfe von Multimedia-Animationen zu demonstrieren. Daher ist die Algorithmenvisualisierung (AV) eines häufige verwendeten Werkzeuge im E-Learning. Zahlreiche Forschungen haben gezeigt, dass die Anwendung von den Animationen zur Veranschaulichung komplexer Algorithmen es den Schülern ermöglicht, diese Algorithmen besser und schneller zu verstehen. Aber es gibt auch viele Lehrer, die sich gegen den Einsatz von AV im Unterricht einsetzen. Um AV wirklich die Studenten zu helfen, müssen bestimmte Richtlinien beim Entwerfen von AV-Animationen berücksichtigt werden.

Nach dem Lesen und Analysieren einer großen Anzahl verwandter Arbeiten wurden vier pädagogische Theorien ausgewählt. Dies sind **Epistemic Fidelity**, **Kognitiver Konstruktivismus**, **Dual Coding** und **Individual Differences**. Es wird angenommen, wenn Entwickler diese vier Lerntheorien bei der Entwicklung von AV - Animationen beobachten, die Qualität von AV effektiv verbessert werden kann. Basierend auf diesen vier Theorien wurde eine Online-Plattform namens DLD-VISU zum Lernen und Lehren von Digital Logic Design entwickelt. Um diese Theorien in DLD-VISU unter Verwendung fortgeschrittener Software-Engineering- und Programmiertechniken zu unterstützen, wurden spezifische Implementierungskonzepte und -methoden entwickelt. DLD-VISU verwendet das MVC - Architekturmuster, um die Trennung von **Model**, **Controller** und **View** zu erreichen, und verwendet die DI, um IoC zu erreichen. Dies kann die Klassen entkoppeln und die Schwierigkeit der Programmwartung und -aktualisierung stark reduzieren. Darüber hinaus verwenden wir die

ORM Technologie, um die traditionelle JDBC-Datenbank für den Zugriff zu ersetzen.

DLD-VISU ist nicht nur in der Lage, Animationen auf Webseiten darzustellen, sondern dient auch als AV-Entwicklungsplattform. Es bietet ein leistungsfähiges Framework und eine umfangreiche grafische Bibliothek. Diese Framework enthält viele leistungsstarke Funktionen wie IoC, ORM und **Action-Dispatcher**. Entwickler können damit schnell eine AV-Animation basierend auf dem MVC - Architekturmuster erstellen. Eine umfangreiche grafische Bibliothek kann die Entwicklern helfen, die Front-End-Animationen schnell zu zeichnen. Um die Benutzerfreundlichkeit von DLD-VISU zu demonstrieren, verwenden wir es, um eine AV-Animation für verschiedene Algorithmen von binären Entscheidungsbäumen als eine wichtige Klasse von maschinellen Lernklassifizierern zu entwickeln.

Ein weiteres Ziel von DLD-VISU ist der Aufbau einer einheitlichen AV-Management-Plattform. Alle Animationen auf dieser Plattform verwenden einen einheitlichen Stil und eine Benutzeroberfläche, wodurch die Zeit gespart werden kann, die Schüler für die Anpassung an verschiedene AV-Programme ausgeben. Darüber hinaus bietet die Plattform eine große Anzahl von Management-Modulen, die es Lehrern sehr erleichtern, ihre eigenen AV-Programme zu veröffentlichen und zu verwalten und Echtzeit-Beobachtungen der Nutzung des AV durch die Schüler durchzuführen.

Wir führten einige Tests auf dieser Plattform durch, einschließlich der Aufteilung der Studenten in zwei Gruppen - die eine nutzte die DLD-VISU-Plattform und die andere verwendete traditionelle Methoden. Beide Gruppen hat die gleiche Fragen. Die Ergebnisse zeigen, dass DLD-VISU hilfreich ist, um die Leistungen der Schüler zu verbessern.

PUBLICATIONS

1. A platform for visualizing digital circuit synthesis with VHDL. Proceedings of the 15th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2010

2. A Platform for VHDL Visualization. 8th European Workshop on Microelectronics Education 2010

3. A Web-Based Visualization and Animation Platform for Digital Logic Design. IEEE Transactions on Learning Technologies 2015 (Volume: 8, Issue: 2, April-June 1 2015)

ACKNOWLEDGMENTS

I would like to express my gratitude to all those who helped me during the writing of this thesis. My deepest gratitude goes first and foremost to Professor Abdulhadi Shoufan, for his constant encouragement and guidance. He has walked me through all the stages of the writing of this thesis. Without his consistent and illuminating instruction, this thesis could not have reached its present form.

Second, I would like to express my heartfelt gratitude to Professor Andreas Koch, who has offered me valuable suggestions in the academic studies. I am also greatly indebted to the professors and teachers at the TU Darmstadt: Professor Sorin A. Huss and Professor Max Mühlhäuser, who have instructed and helped me a lot. Especially, Professor Sorin A. Huss, who was my first supervisor, provided me much help and support, which I will never forget.

Last my thanks would go to my beloved family for their loving considerations and great confidence in me all through these years.

My sincere gratitude to my friends and my fellow classmates who gave me their help and time in listening to me and helping me work out my problems during the difficult course of the thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

ACS Access Control List

AJAX Asynchronous JavaScript and XML

ARFF Attribute-Relation File Format

AV Algorithm visualization

B/S Browser/Server

CC Cognitive Constructivism

CF Canonical Form

CNF Conjunctive Normal Form

CSS Cascading Style Sheets

C/S Client/Server

DAO Data Access Object

DI Dependency Injection

DLD Digital logic design

DNF Disjunctive Normal Form

DOM Document Object Model

DVEC DLD VISU Electronic Component

DVNL DLD VISU Node and Line

EF Epistemic Fidelity

FPGA Field-Programmable Gate Arrays

FSM Finite State Machine

GUI Graphic User Interface

HDL Hardware Description Languages

IoC Inversion of Control

JSON JavaScriot Object Notation

ORM Object Relational Mapping

PHP Hypertext Preprocessor

QMC Quine-McCluskey

SDLDS System for Digital Logic Design and Simulation

SVG Scalable Vector Graphics

VML Vector Markup Language

W3C World Wide Web Consortium

PO Persistent Object

ID3 Iterative Dichotomiser 3

1 INTRODUCTION

1.1 MOTIVATION

The continuous development of E-Learning helps to provide a powerful mechanism to improve learning efficiency. The biggest difference between E-Learning and traditional lecture methods is that E-Learning supplies an active learning method [1, 2, 6, 26, 38, 39, 75]. In traditional lecture methods, students can only obtain knowledge from teachers passively, teachers talk and students listen in a passive way. Students are assumed to enter the course with minds like empty vessels or sponges to be filled with knowledge.[69]. When students use an E-Learning software for self-study, they need to observe and communicate with this software actively so that they can catch up with the learning process.

E-Learning conditions are supported learning environments, in which learning processes of human individuals are supported by using digital technologies to record, store and transfer, handling and processing application and presentation of information [107]. E-Learning will prevail only if it brings proven educational and economic advantages over existing pre-digital learning conditions. The experience shows that E-Learning has provided a number of such advantages for learners as well as learning providers [107]. The features of E-Learning are:

1) Interactive and multimedia design of the learning content.

2) Performing learning processes over digital networks (Internet or Intranet).

Compared with traditional teaching, E-Learning has the following advantages [86]:

1) **Classroom Discussions:** Students talk at least as much as or more than teachers.

2) **Learning Process:** Most learning processes are carried out by individuals or in groups.

3) **Subject Matter:** Students also play a role in determining the subject matter.

4) **Emphases in the Learning Process:** Students learn more "how" and less "what".

5) **Motivation:** Students learn actively.

Although E-Learning has these advantages, it does not completely replace the traditional teaching. Some teachers even suspect the efficiency of E-Learning. The controversy over E-Learning mainly focuses on the following two aspects:

1) E-Learning applies animation, pictures and other means. Although it can be more intuitive and easy to show the operation of a complex algorithm to help students understand the efficiency of a new algorithm, rich animation can also distract students from studying.

2) Traditional teaching has more advantages in the interpretation of a concept or theorem, because teachers repeatedly stress key topics based on their teaching experience. The schedules of E-Learning are mainly controlled by the students, they are likely to miss the key topics.

Blended-Learning is a combination of traditional teaching and E-Learning. It can effectively avoid the lack of pure E-Learning mentioned above. Blended-Learning is a learning paradigm that attempts to optimize the advantages, potentials, and benefits of both traditional learning and E-Learning by eliminating their shortages and facing their challenges. Compared with the traditional learning paradigm, Blended-Learning is found to be consistent with the values of traditional learning paradigm adopted in almost all higher education institutions for decades, and has the proven potential to enhance the effectiveness and efficiency of meaningful learning experiences [1, 2, 6, 26, 38, 39, 75, 86, 107]. Blended-Learning is defined as a learning way that combines instruction-lead learning and on-line active learning, leading to reduced contact hours in classroom. It has the potential to improve the learning effect of students compared to equivalent fully on-line courses [16]. Table 1.1 presents comparison data showing success rates over two years of on-line offerings at University of Central Florida.

	Spring 2001	Summe 2001	Fall 2001	Spring 2002	Summe 2002	Fall 2002	Spring 2003
Traditional learning	91	93	91	90	94	91	91
Blended-Learning	91	97	94	91	97	92	91
Fully On-line learning	89	93	90	92	92	92	91

Table 1.1: Percentages of Students Succeeding(Grades of A,B, or C) in Traditional, Blended, and Fully Online Courses at UCF [16].

This thesis introduces an on-line platform where learners participate in learning process effectively, and teachers play an guiding role, instead of transferring information in materials to learners directly. The focus of this platform is to present the complete working process of an algorithm or a design process through interactive animations. Nowadays, there is a lot of such animation software or on-line tools. The biggest difference between these software and our platform is that the development of our platform is based on pedagogical theories. Some tests were performed by letting students work on this platform and the testing results were collected. The results can prove that these theories do improve the efficiency and effectiveness of E-Learning. In this way, further thoughts and theories could be provided for the development of E-Learning in the future. This platform is a Blended-Learning system, so this platform does not supply introductions of algorithms directly. That part is the content of classroom teaching. We chose digital logic design as the first course on this platform, and we named this web-platform "DLD-VISU".

So far, there has been a great variety of AV applications which can present just one or a few algorithms. That means, only one AV application is typically not able to cover all the topics of a course. As shown in Figure 2.2 (a), such a related AV application can only be used to illustrate the algorithm of AVL-Tree in the course "Data Structure". Students must seek for further AV applications, if they wish to learn more algorithms, such as sorting, searching, etc., resulting in two problems:

1) The way how the AV programs demonstrate the algorithms and their GUI could be significantly different. This is because the designers have distinct design concepts, different design objectives as well as various applied technologies. In this way, it would cost students much time to get familiar with these AV applications as well as adapt themselves to the operations and demonstrations of each AV application before they start to utilize them. This would reduce students' self-learning efficiency through these applications and waste their study time, eventually the teaching effects of AV applications may be influenced negatively.

2) Even though students can employ various AV applications for all subjects of a course, it is tough for them to manage the learning progresses uniformly. Additionally, it is also very cumbersome for teachers to obtain relevant feedback regarding students' learning conditions.

DLD-VISU can solve the above mentioned problems fundamentally. As a management platform of AV animations, DLD-VISU employs a unified way to present all provided animations and GUI. Students only need to be guided once when they register for using DLD-VISU.

Afterwards, they can use this platform during their college time very conveniently. Also, teachers can observe and manage students' learning - process. All learning data are stored in the database while students apply DLD-VISU to study. Students can retrieve and review their previous learning activities, grades, etc. at any time. Teachers may employ the ACS subsystem to publish and manage AV animations and use Self-Assessment subsystem to observe to what extent students already master the material. These two subsystems will be introduced in detail in Chapter 3.

Digital logic design (DLD) is a core course in several undergraduate majors including electrical engineering, computer engineering, and computer science. DLD is usually taught in the first or second year in colleges. This poses special difficulties to students for four reasons. Firstly, DLD is a comprehensive course with diverse topics that must be covered to enable students to attend advanced courses such as computer architecture and embedded systems. Secondly, a DLD course is rich in new concepts, theories, and approaches, which are not part of school education, as a rule. Thus, students face these topics without any or with very limited background. Thirdly, solving DLD problems manually is error-prone because of working with large numbers of 1's and 0's. For instance, the state table used to design a small finite state machine with four binary-coded states, four input signals, and two output signals, has 64 lines and 10 columns with a total of 640 zeros and ones. Flipping any of these bits by mistake may lead to a FSM circuit, that doesn't meet the specification. Finding this kind of error is tedious and its correction may require a new start from the beginning. This doesn't only cause frustration but also deters many students from trying to solve advanced problems for practising. Fourthly, an essential aspect in digital logic design is to learn how different design alternatives result in different non- functional properties of digital circuits. For instance, to investigate the effect of the FSM type, the state code, or the flip-flop type on the performance or on the gate usage of a finite state machine, different design alternatives for the same specification should be generated and compared. Given the complexity of generating one design alternative, it is obvious that a comprehensive evaluation of design alternatives is almost impossible.

DLD has been addressed in education literature since a long time. In several papers can be found that some related topics are mainly focused on, such as DLD course construction [5, 10, 104], the usage of commercial tools and Hardware Description Languages (HDL) for learning DLD [4, 83], and employing programmable logic to enhance the effectiveness of DLD learning process [110, 111]. Using HDL and commercial design tools as learning technologies is very useful. However, these tools operate on two ends of the design process and hide internal design steps which form the core learning outcomes in a typ-

ical DLD course. For instance, a commercial synthesis program can read an FSM specification in form of a state diagram or HDL code and generate the corresponding circuit. The question "How does this generation work", which is in the center of a DLD course, is not answered by these tools. Individual contributions on the visualization and animation for digital logic design can be found in the literature. The related work is discussed in Section 1.2.

DLD-VISU was designed and developed to visualize and animate different DLD topics. The main contribution of this tool to academic learning consists in a consequent exploitation of relevant pedagogical theories in conceiving the graphical animation process. These theories known as **Epistemic Fidelity**, **Cognitive Constructivism**, **Dual Coding**, and **Individual Differences**, specify the fundamental requirements for a successful animation solution in the field of education.

Currently, DLD-VISU supports the design of combinatorial circuits using logic gates, multiplexers, decoders, and lookup tables. The input function can be entered as a Boolean function or as a truth table and can be minimized using K-Map or Quine-McCluskey algorithm. Additionally, finite state machines can be synthesized step by step starting from a state diagram. All design steps are performed under students' interaction, so that intermediate values can be verified. Different FSM configurations regarding the machine type, the state code, and the flip-flop type can be selected to test various design alternatives. DLD-VISU enables instructors to set access rules that define when students can access which topic. This is important for instructors who use graded homework problems and assignments as assessment tools.

1.2 RELATED WORK

In this section we review the related work on the visualization and animation of digital logic design in the order of their relevance to our work. Then we describe the innovation of DLD-VISU and compare it with the related work.

Stanisavljevic et al. presented a *SDLDS* [98]. The system consists of three modules for design, simulation and evaluation. The design can be carried out either starting from a formal description or by instantiating and connecting library modules. Combinatorial circuits as well as finite state machines of both Moore and Mealy types are supported. A Boolean function is entered as a truth table. The system then creates the canonical forms of the function and draws the corresponding circuits using AND and OR gates as well as inverters. Additionally, the K-Map approach is used to create minimized functions and draw the optimized circuits. A finite state machine can be designed starting from state table that is entered by the user. Figure 1.1 shows an example of SDLDS.

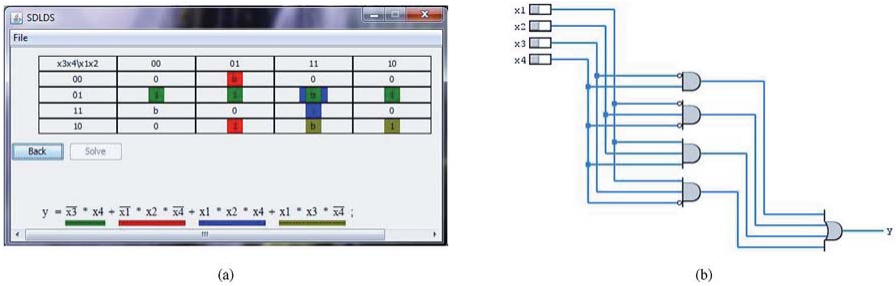


Figure 1.1: The K-Map and circuit example of SDLDS. [98]

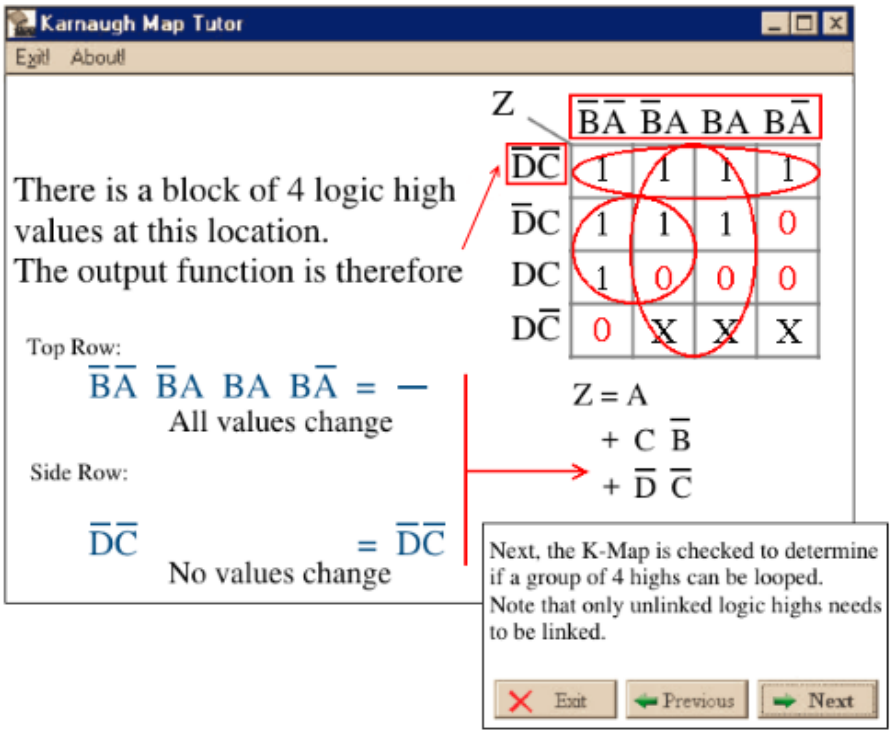


Figure 1.2: A K-Map example of WinLogilab. [36]

In [36] a Windows-based tools suite, denoted as WinLogiLab, is presented and supports interactive learning of the design of combinatorial and sequential logic circuits. The suite includes tools for number presentation and conversion, the design of combinatorial circuits with logical gates, Boolean function minimization using K-Map and Quine-McCluskey approaches, and the specification and simulation of general purpose finite state machines. Figure 1.2 shows a K-Map example of WinLogiLab.

DDMVL [30] is a mobile virtual laboratory funded by the Ministry of Science, Education and Sports of the Republic of Croatia. They provide a server to run the digital design experiments(combinatorial logic) and display the results in different terminal devices such as PCs, PDAs and mobile phones. Students can experiment at any moment and regardless of her/his whereabouts. This server can synthe-

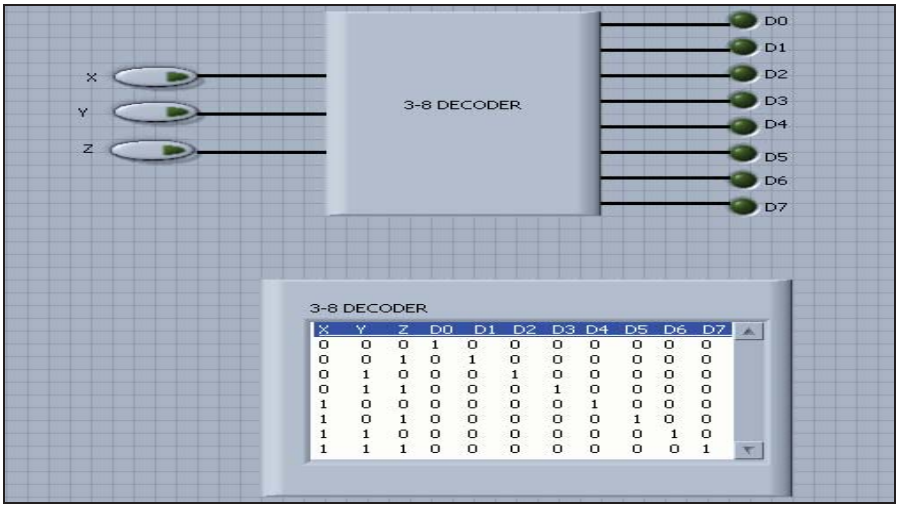


Figure 1.3: 3-8 Decoder of ODLDL [9].

size the combinatorial logic design and save the learning process with m(mobile)-learning server. The server includes users management, learning process management, learning process and digital model simulation support. The focus of DDMVL is to reduce unnecessary interactive operations so that students can get the same user experience on mobile devices as on PCs.

HADESis a versatile simulation and visualization platform for computer architecture based on Java applets [41]. HADES offers interactive simulations on the gate level and it includes a state diagram editor. The user can select the machine type, specify the inputs and outputs of the machine, and draw the state diagram accordingly.

In [109], the authors use the Flash technology to generate what they call Flash notes for DLD topics such as the basic logic gates, simplifying logical circuits, flip-flops.

Several simulators were presented for educational purposes. For instance, ODLDL [9] is a web-based system aiming at simulating logic design experiments. The experiments are implemented and published on the web. These experiments are digital logic gates, combinational logic circuits, seven segment display, sequential logic, and counters. After they have been published, the experiments can be accessed and controlled remotely via the internet. In order to view and control the experiment, the students must download and install a freely available web browser plug-in. Students can control the value of input signal and observe the output. In Figure 1.3 showed an example of a 3-8 decoder.

LOG is a digital simulator for UNIX, which was originally developed at UC Berkeley in the 1980s as a tool for teaching logic design. LOG is available as part of the CHIPMUNK package [18]. Logisim is a Java-based simulator of digital systems consisting of gates and flip-flops [14]. LoGen uses dynamic HTML and PHP to generate and

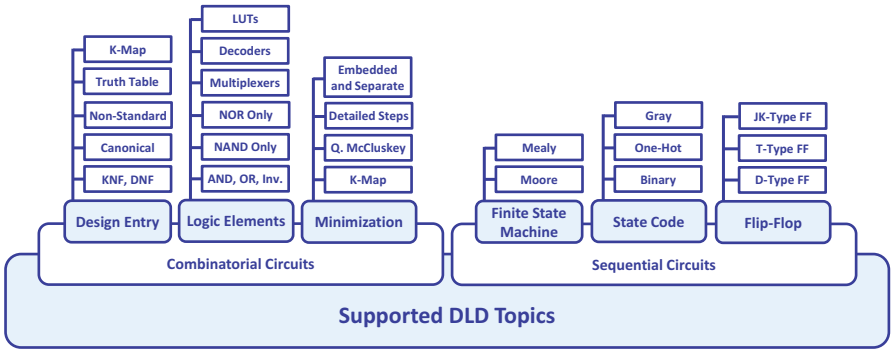


Figure 1.4: DLD-VISU Covered Topics.

simulate logic circuits on the gate level [55]. Other simulators available on-line include Smartsim [95] and Easysim [92].

DLD-VISU is a tool for the visualization and animation of digital logic design. Its main contribution consists in the definition of a theoretical learning framework and the application of this framework to a wide range of DLD topics, as depicted in Figure 1.4.The framework essentially relies on the findings in the field of algorithm animation to generate graphical processes for learning DLD. The motivation to this approach is that most DLD topics can be described in an algorithmic way. This is not only valid to the main design processes for combinatorial and sequential logic, but also to several intermediate steps such as function minimization using the K-Map or the Quine-McCluskey scheme, Boolean function implementation using multiplexers, decoders, NAND gates, or NOR gates. Even individual actions within these intermediate steps can be sophisticated enough for the computer, so that an algorithmic description is justified. One example for that is finding all prime implicants in a K-Map. The biggest challenge for any DLD visualization tool is to find an appropriate abstraction level for the graphical processes that contributes to the learning process effectively. This strongly relies on the instructor's experience in teaching DLD topics and her or his awareness of the level of details that are most appropriate for topics presentation.

This complies with the Epistemic Fidelity theory as one of the four fundamental theories that we adopted to build the DLD-VISU theoretical framework. This framework will be detailed in the chapter 3. We are not aware of any learning technology that uses a similar approach to produce graphics or graphical processes for DLD visualization. On the topical level, DLD-VISU focuses on the synthesis of digital logic rather than on simulation. Thus, it is mostly related to SDLDS [98] and WinLogiLab [36]. DLD-VISU supports various design aspects that are not covered by these two solutions as can be seen in the comparison given in Table 1.2. However, we should emphasize that the main strength of DLD-VISU is not its topic coverage but the way these topics are presented based on the proposed frame-

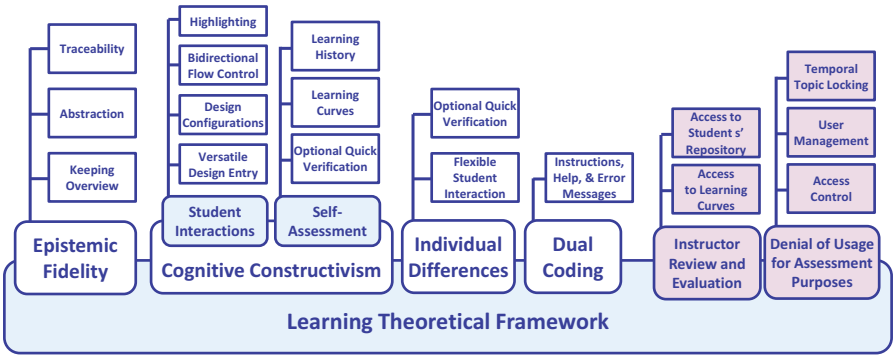


Figure 1.5: DLD-VISU Theoretical Framework.

work. For instance, DLD-VISU is the only solution that addresses the K-Map minimization in the methodical way that starts with determining all prime implicants, identifying the core implicants, and writing the minimized function as a sum of all core implicants and as many prime implicants as necessary. Especially if the minimized function is not unique, we will show all minimized functions for the students. These steps are addressed explicitly and with user interaction and self-assessment. Another example is the implementation of Boolean functions using NAND gates. DLD-VISU is the only tool that supports all possible ways to enter the function, to minimize it, and to implement it using AND and OR gates. Then, the AND and OR gates are replaced by their NAND equivalent circuits and the redundant inverters are removed to obtain the final circuit.

Furthermore, as a on-line solution DLD-VISU provides two functions that are of high relevance to the educational process and they are detailed in the next chapter, see Figure 1.5. Firstly, instructors can lock some topics for periods where students need to deliver related homework solutions. For that a user management system was introduced. It enables instructors to register their students and to define appropriate access rules. Secondly, instructors can access students' learning curves and history. Accessing students' learning curves helps instructors evaluate the learning level of students. The students' history can be used as a repository by instructors for the purpose of review and assessment.

1.3 STRUCTURE OF THE WORK

The remainder of the thesis is structured as follows. Chapter 2 describes the design concepts behind DLD-VISU. Chapter 3 details the different learning topics in DLD-VISU with examples. Chapter 4 describes the business logic, i.e. all algorithms in DLD-VISU. Chapter 5 explains implementation of DLD-VISU. As a common Blended-Learning platform, The DLD-VISU framework can not only be em-

Topic	WinLogiLab	SDLDS	DLD-VISU
Formal entry of Boolean function	Yes	No	Yes
Specifying the function form (DNF, CNF, non-standard, canonical)	No	No	Yes
Design entry as a truth table	Yes	Yes	Yes
Design entry as a K-Map	Yes	No	Yes
Minimization using K-Map	Yes	Yes	Yes
Minimization using Quine-McCluskey	Yes	No	Yes
Implementation using AND, OR gates and inverters	Yes	Yes	Yes
Methodical implementation using NAND gates	No	No	Yes
Methodical implementation using NOR gates	No	No	Yes
Methodical implementation using multiplexers	No	No	Yes
Implementation using decoders	No	No	Yes
Implementation using LUTs	No	No	Yes
Design of Moore and Mealy FSMs	No	Yes	Yes
FSM entry as a state diagram	Yes	No	Yes
Selecting or editing binary state code	No	Yes	Yes
Selecting or editing one-hot state code	No	Yes	Yes
Selecting or editing gray state code	No	Yes	Yes
FSM design using D-type flip-flops	No	Yes	Yes
FSM design using T-type flip-flops	No	Yes	Yes
FSM design using JK-type flip-flops	No	Yes	Yes
Minimizing the state and the output equations	No	Yes	Yes
Drawing the FSM state diagram	No	Yes	Yes

Table 1.2: Comparing DLD-VISU with SDLDS and WinLogiLab.

ployed for DLD courses, but also for other computer courses. Chapter 6 describes how to use DLD-VISU to quickly design and develop some algorithms for machine learning. Chapter 7 details the evaluation of DLD-VISU and Chapter 8 concludes the thesis.

2

DESIGN CONCEPT OF DLD-VISU

With a history which can be traced back to 1960s [11], E-Learning is enriched with the constant upgrade of network technology. E-Learning provides diversified learning methods that make use of multimedia technology. As mentioned in the previous chapter, Blended-Learning has been well recognized by teachers because of its good combination of traditional teaching and E-Learning. With the popularity of this technology, educators started to demonstrate complicated computer algorithms and abstract data structure using multimedia animation. As a result, AV became one of the most commonly used tools in Blended-Learning. Many researchers showed that the animation of complicated algorithms enables students to understand these algorithms better and faster. Some colleges and universities started to teach algorithms using AV as early as 1970s. However, the education circle has not reached an agreement on the effect of this approach. Many studies and experiments have been performed to identify the factors that determine the effectiveness of AV software [60, 101]. Through a comprehensive meta study, Christopher D. Hundhausen developed four theories that specify the effectiveness of AV software [20].

DLD-VISU was developed on the basis of these four theories to improve students' learning of the subject of digital logic design. This chapter focuses on the design concept of DLD-VISU which relies on the theoretical foundations of AV. In particular, the function and the scope of AV will be introduced in section 2.1; section 2.2 and section 2.3 details the pedagogical and web design concept of DLD-VISU; Section 2.4 specifies the application scope of DLD-VISU; and section 2.5 is the summary.

2.1 INTRODUCTION TO AV

AV generally means displaying computer algorithms using graphics or animation to help students understand the algorithms quickly and thoroughly.

The educators supporting the usage of AV software for teaching believe AV software is featured by the following advantages:

1) Compared with static objectives, mobile or flashing one could be caught faster. It could also be remembered longer by people [109].

2) If learning with AV software, students would interact with AV. Compared with the traditional passive learning mode, this active one can better arouse students' interests and attentions

3) AV software can spare instructors' time to demonstrate algorithms in class. In the traditional teaching mode, instructors have to spend a lot of time drawing on the blackboard to demonstrate one algorithm example.

AV was applied to the teaching of computer algorithms already in the 1970s and has been evolving since then. Initial AV software was merely in form of simple batch processing programs. Teachers created animations using simple scripting languages or batch processing commands [13]. More importance is attached to the interaction with the AV tools. Some AV software allows students to configure algorithm parameters for a dynamic display of the results. Other solutions enable students to create the animation required through interactive programming environment. For instance, in the ANIMAL system developed by Guido Rössling, students can edit an animation script using a special computer script language called ANIMALSCRIPT. ANIMAL main program can load the script and play the animation in the display window of ANIMAL [89].

Available AV solutions differ in functionality considerably. The reason for this is that the development of AV software is strongly targeted: AV software is generally developed by professors or instructors who teach computer science and develop AV solutions to display a specific kind of algorithms [20]. For instance, ANIMAL was created for the animation of sort and search algorithms. Due to this purpose-specific development, AV software lacks a standard for functionality and presentation.

AV software can be used for different purposes in the teaching and learning process as depicted in Figure 2.1. Seven conditions can be identified [88]:

1) **Lectures**: Instructors teach computer algorithms using graphic representations in class. Gurka and Citrin pointed out that AV in lectures is essentially "an extension of the blackboard, but with more capabilities available" [100, 109].

2) **Assignments**: Instructors can assign homework through AV software which can also be utilized by students to finish the homework. After the submission by students, the instructors can evaluate the learning quality of students using AV software [33] [34].

3) **Class discussion**: After generating animation using AV software, students can present it in class and discuss it with instructors and classmates. AV software does not only improve students' participation in class but also enhances the interaction between instructors and students.

4) **Labs**: Students can use animation tools in the laboratory to understand the operation principles of algorithms. In contrast to assignments, the experiments are performed in specific locations such as PC pool, and within a given time frame.

5) **Study**: By using AV software, students can learn algorithms any time and anywhere. Moreover, according to their own preferences and learning level, they can design the visualizations to learn algorithms by themselves. For example students with good performance can examine hard copies of visualizations constructed by others (professors or book authors) [34].

6) **Office hours**: Instructors can use AV solutions during office hours to support students individually and according to their level in the course and their performance [33].

7) **Test**: AV software can be employed to generate test problems by instructors. Also, students can be confronted by animation and asked to identify the corresponding algorithm as reported by Brown [13].

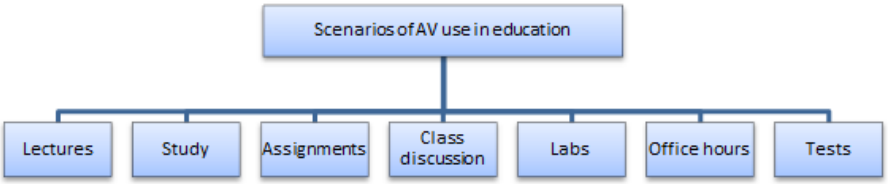


Figure 2.1: Possible conditions for applying algorithm visualization in education.

AV software is mostly applied in only one or two conditions according to the design targets followed by their developers. For instance, Figure 2.2 shows two screen shots of two AV programs that visualize the insertion and deletion algorithms for AVL Tree [3, 12]. Instructors can utilize these two programs to demonstrate the algorithms and students can use it for self-study. However, the software in Figure 2.2 (a) does not allow students to generate AVL Tree by inputting data on their own. Thus, it is obvious that this software is not suitable for the conditions of assignments, labs, and office hours mentioned above. The software in Figure 2.2 (b) allows students enter their own data to generate an AVL tree, so it cannot be used in the assignments and tests, because the students may use it to cheat in their homework by entering data and letting the software generate the result.

Single or limited functionality had a negative impact on the acceptance of AV software by many instructors [94]. Some instructors point out that the functionality of AV software is very limited and cannot cover all teaching requirements of teachers well. Besides, it takes students much time and effort to learn how to use AV software, which is

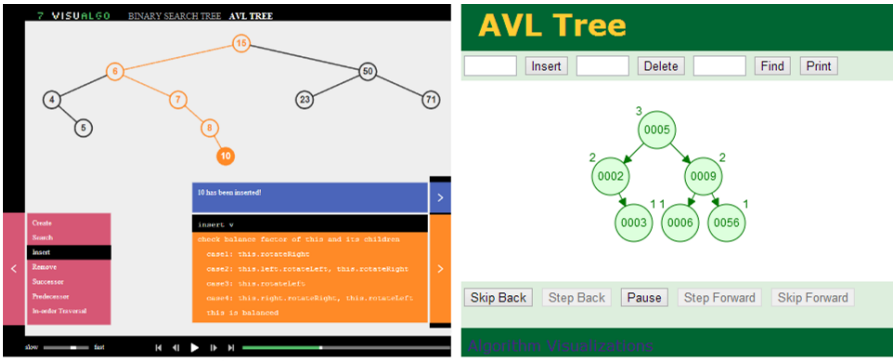


Figure 2.2: Two AV Softwares for AVL-Tree [3, 12].

already beyond the benefits brought by AV software. During the design of DLD-VISU we took this concern into consideration and tried to add functionalities and features that make this solution applicable in all the condition listed above, as will be explained in the following sections.

Many educators doubt the effect of AV software in teaching and some even strongly oppose using AV software in class. The educators against the usage of AV software in class put forward the following points [20]:

1) No time to learn the technique relating to AV;

2) Class hours are largely occupied due to utilization of AV software, which makes it impossible to carry out other class activities;

3) Too much time and energies would be consumed to create an AV animation that fits the teaching goal;

4) Compared with traditional teaching method, AV software would distract students' concentration instead of helping them to better understand algorithms.

Although AV has been widely used in E-Learning's field, but the effect of AV has not been unanimously approved. The impact of AV software has been analysed in many research reports in the recent decade. Diversified study results have been presented. According to some reports, the teaching quality can be greatly improved by AV software [15, 60]. While other reports pointed out that no remarkable assistance was made by AV software [82] [102]. Some reports even claimed that AV software would result in negative effect on teaching [74].

Christopher D. Hundhausen, Sarah A. Douglas and John T. Stasko proposed four pedagogical theories that affect the effect of AV software after doing statistical analyses on the experiment data from a

great deal of research reports about the effect of AV software in teaching [20]. All the four theories are targeted at both the dependent and independent variables. The four theories were discussed and studied together with some rich experienced teachers. Through the discussion, it is concluded by these teachers that all these theories were effective in improving the efficiency of AV software.

2.2 PEDAGOGICAL DESIGN CONCEPT

DLD-VISU relies on the **Epistemic Fidelity (EF)** theory and **Cognitive Constructivism (CC)** theory as the pedagogical framework for effective animation and visualization. The core idea of EF theory is using visual representation to ensure the correctness of knowledge transmission, and the CC theory holds knowledge must be individually constructed or reconstructed by the learner. In the process of knowledge transmission, the correctness of transmission is affected by some factors. Through the research of these factors, the researchers explored many different versions of EF theory. These versions only deal with one or more factors, so they are called **Weak EF Theory** [42]. Except EF and CC theories, the following two weak EF theories are used for DLD-VISU too.

1) Dual Coding
2) Individual Differences

2.2.1 *Epistemic Fidelity*

The EF theory forms form 4 assumptions.

1) **The Knowledge Representation Assumption:** The knowledge exists independently of humans, but can be instantiated as symbolic structures in humans' heads. [79]
2) **The Knowledge Flow Assumption:** The knowledge can be encoded in an AV by experts and decoded by learners. In other words, the knowledge is seen to flow from experts to AV, then to learners through animations. Therefore, if a learner can't obtain the knowledge transferred by an AV, the reason can only be "some flaw or inadequacy in the medium [24]"
3) **The Graphical Medium Effectiveness Assumption:** The graphical representations(figure or animation) can give an excellent ability to support representations that closely match an expert's mental model of an algorithm: that is, the way in which an expert conceives of the algorithm and how it works [42].
4) **The Epistemic Fidelity Assumption:** Follows from the previous 3 assumptions, we can conclude that a close denotational match

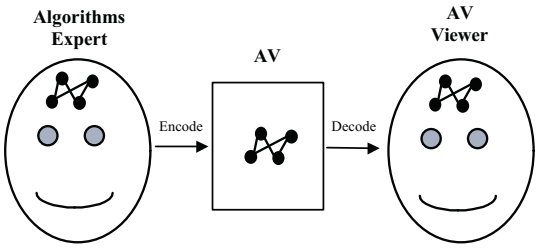


Figure 2.3: A Schematic Diagram of Knowledge Flow According to EF Theory.[20]

between an expert's mental model of the knowledges and an AV leads to efficient learning of those knowledge by the learners.

According to the EF theory, people always establish models for the natural world in their brains. All the behaviours and activities of people are based on such models. As models differ, people have different ideas and behaviours [80]. The following notions can be obtained by applying the theory in pedagogy: Teaching is a process of transferring the instructors' understanding model of knowledge to students. This assumption is graphically depicted in Figure 2.3. An instructor should first build a model for the knowledge in brain (expert's mental model) and then "encode" this model. Encoding means that the instructor transfers his expert's mental model through different methods such as textbooks, teaching materials, photos, or AV software. Students try to construct the model of knowledge by reading textbooks or teaching materials, or by viewing photos or AV animation. When it comes to algorithm visualization, the process of encoding the expert's mental model corresponds the process of designing and developing the AV software. Differing from other encoding approaches such as textbooks, the benefit of using animation is the ability to reflect the dynamic behaviour of the algorithm. The better the animation and the expert's mental model are matched, the easier the students decode the AV animation to their knowledge model and the higher the effectiveness of the AV solution will be. To enable instructors' expert's mental model to be transferred to AV animation better, AV developers should have rich teaching experience or communicate with well-experienced instructors before developing AV animation.

How to design an AV which closely match an expert's mental model? It relies heavily on the developer's rich teaching experience. Designing a sequential or combinatorial circuit is a sophisticated and dynamic process with various steps that operate on different data. Our expert's mental model for understanding this design process relies on three principles: **Keeping overview**, **Abstraction**, and **Traceability**. Keeping overview is essential to find oneself in the whole design process. Abstraction is a key principle to focus on the current design step and not to be distracted by other unnecessary details.

Figure 2.4: FSM Process Chart.

Traceability helps students understand the transition from one step to another and identify the origin of each boolean term or value and each logic component generated through the design process.

2.2.1.1 *Keeping overview*

Lawrence, Andrea Williams found that animations in which the conceptual steps of algorithms are redundantly labeled do lead to higher post-test performance. [61] According this conclusion we designed **Keeping overview** principles.

Designing a sequential or a combinatorial circuit is a complex process. For instance, ten steps are required to design a FSM: drawing a state diagram, encoding the states, generating a state table based on the state code and the state diagram, selecting the flip-flop type, generating the corresponding state equations and output equations according to the state table and the selected flip-flop type, minimizing the equations, and drawing a circuit diagram based on the minimized equations. There is an old saying that goes: "The endless attention to trees at the expense of forests". A beginners usually focus on solving the specific steps and tend to lose track of the entire process. In other words, the question "how to perform the individual design steps?" is frequently paid more attention than the question "why is some steps is being executed?". A low understanding of the target of individual steps can lead to incorrect, incomplete or suboptimal solutions. For example, when students encode the states without keeping in mind that this coding will affect the number of the flip-flops and the complexity of the input and output circuits, they may make an arbiter selection which may even lead to an unmanageable size of the state table. In order to help students understand the whole design process, animation tools should provide students with information about the general flow of this process in addition to a dynamic display of the current design step.

DLD-VISU enables students to keep track of the learned topic by displaying a progress chart for the design process at the top of the animation window. The process chart changes with the students' operation. As shown in Figure 2.4, the blue part in the progress chart indicates the steps completed or under processing, while the gray steps are uncompleted. By this means, the whole process is fully clear to students at any time.

Some steps in the progress chart are complex and consist themselves of several sub-steps. If all these sub-steps are displayed with

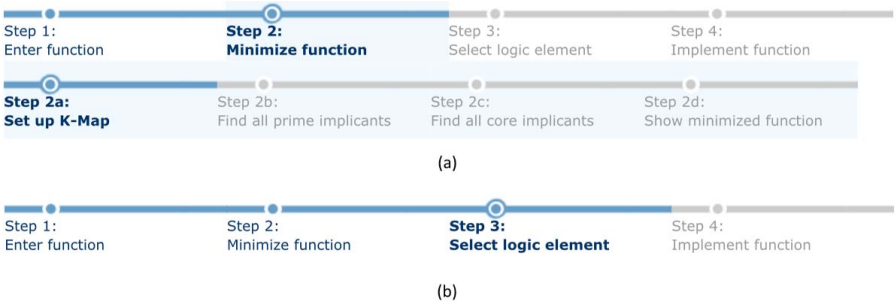


Figure 2.5: Progress Sub-chart Example.

the parent-steps at the same time, this may not only causes page jam but also confuses students due to too many steps, and this cannot represent the hierarchically arranged layers clearly. Therefore, a progress sub-chart appears under the main chart upon reaching the corresponding step. An example is given in Figure 2.5 Four main steps are required to implement one combinatorial circuit. After students select one minimizing algorithm in step 2, one progress sub-chart will be extended below step 2 of the progress chart to display all the steps required by the selected minimizing algorithm. The progress sub-chart disappears upon leaving step 2 to step 3. In this way, students will not be confused by the details of step 2.

2.2.1.2 *Abstraction*

DLD-VISU supports abstraction by selecting an appropriate granularity level for the animation steps on the one hand, and by presenting only necessary data in each step, on the other.

a) Selecting a proper length of the animation step is essential because too fine-grained steps would require students to click more. This may cause inconvenience. In contrast, too coarse-grained steps would demand that more data is presented in the animation window. This may lead to crowded pages and confusion.

b) By displaying only necessary information in each animation step, students can focus on this information and are not distracted by irrelevant data. Some instructors may claim that it is important for the student to be able to identify the relevant information out of a set of information, but Lawrence, Andrea Williams found that animations in which data elements are redundantly labelled do not lead to better post-test performance [61].

The step granularity was paid a special attention during the development of DLD-VISU and we made efforts to find the most appropriate

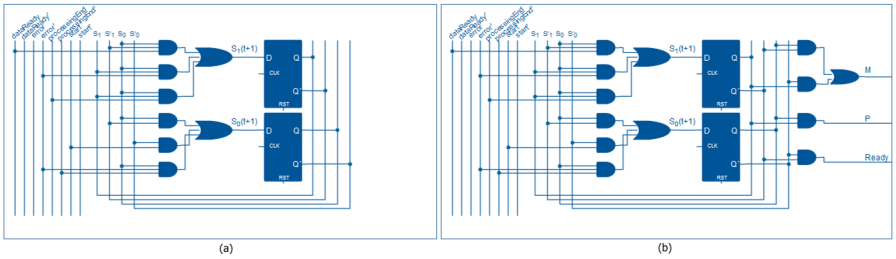


Figure 2.6: Input and Output Circuit.

length for each animation step. For instance, the process of designing a finite state machine is divided into ten steps: This division was mainly based on kind of activity required in the different step. So we have one step to edit the state diagram, one step to encode the states, one step to set up the stare table, one step to design the state memory, two steps to set up input and output equations, two steps to minimize input and output equations, and two steps to draw input and output circuits For instance, step 7 is to draw the input circuit diagram based on minimized state equations, see Figure 2.6 (a). Step 10 is to draw the output circuit diagram based on minimized output equations, see Figure 2.6 (b). While both steps can be combined as usually done in textbooks, we believe that this separation is important because students can better distinguish which part of circuits is constructed through input expressions and which part through output expressions.

As specified above, the contents displayed in each animation step are selected very carefully. Each animation step only contains the necessary information that helps students to understand the current step. For instance, in step 3 of the previous example, the state diagram is shown at the left side of the page while the right side is the state table, as shown in Figure 2.7 (a). In this step, students need to build the state table by observing the state diagram. In a next step, as shown in Figure 2.7 (b), the state diagram does not appear any more, because writing the state equations only requires the state table and not the state diagram.

2.2.1.3 *Traceability*

Traceability is highly important for learning digital logic design. Remember that solving DLD problems is error-prone as students have to deal with a large number of boolean terms and values. DLD-VISU enables students to identify the source of each term or value either by a stepwise generation of these terms and values, by highlighting using colors, or both. For example, when the *prime implicants* are derived from the K-Map depicted in Figure 2.8, students can trace how each implicant is determined not only with the aid of colors but also

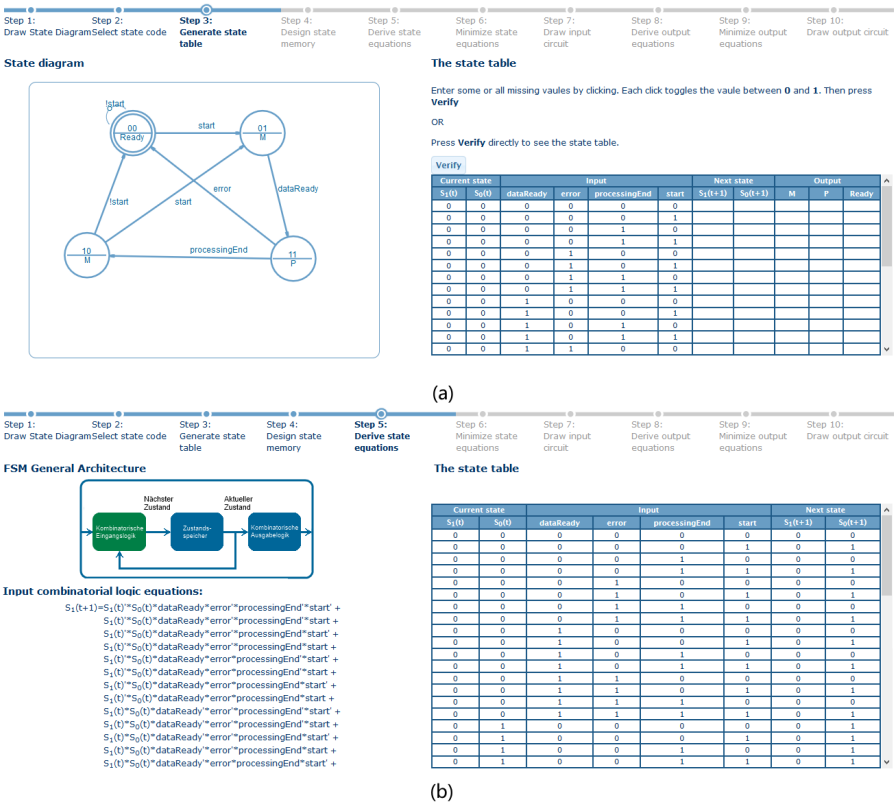


Figure 2.7: Step 3 and Step 5 of FSM.

by a gradual display of the minterm groupings in the map and the related implicants

2.2.2 *Cognitive Constructivism*

As described in the previous section, EF theory regards knowledge as representations of an objective reality that people carry around in their heads. Unlike the EF theory, the CC theory believes that the learner are the most important factor in the teaching process. The knowledge must be reconstructed by the learner.

Inspired by Piaget's research into childhood development [32], cognitive constructivism encompasses both an epistemological framework and pedagogy theory. The key assumption of CC theory is **learners *actively* construct their own knowledge**. But how can learners construct a new knowledge by themselves? According to CC theory, learners can construct a new knowledge actively by analyzing new experiences within the context of what they already know [29].

According to discussing about the assumption of CC theory, we can get some implications for pedagogy. First, instructors should not use AV technology to show a demonstration in which students are only the passive viewer. Passive watching of AV animation reduces the benefits of visualization tools regardless of the quality of these tools

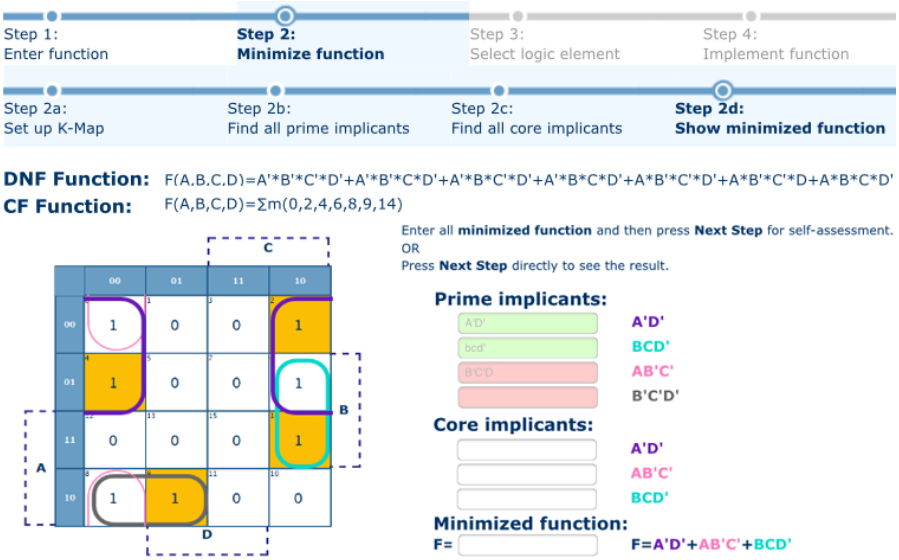


Figure 2.8: K-Map Minimization Using Prime and Essential Prime Implicants.

and the strength of the underlying expert's mental model. Indeed, the experimental results raise the possibility that an AV's epistemic fidelity matters far less than what the learners do with the AV [42]. Secondly, in order to grasp the essence of an algorithm from an AV, learners should construct the AV for themselves, in other words, active learners can choose their own problems based on their interests or abilities.

Many experimental results have shown that active learning can significantly improve the efficiency of self-learning. Table 2.1 shown 4 different experiment from Lawrence, Byrneet et al., and Kann et al.[50, 61, 103].

To comply with the theory of cognitive constructivism, DLD-VISU adopts the following two methods to improve students' interaction with the tools:

2.2.2.1 *Interactive animation*

The design of sequential and combinatorial circuits in DLD-VISU is accomplished through interactive animations. The tools dynamically generate different animations in accordance with the students' requirements. It is mainly represented in the following four ways:

a) When designing a combinatorial circuit, students can select different ways to start the design process. For example, students can enter the function in form of CNF or DNF, by selecting the corresponding rows in a truth table template, or by selecting the corresponding squares in a given K-Map template. When designing a sequential circuit, students can start with drawing the state diagram after selecting the machine type.

Study	Pedagogical Treatments	Dependent Measures	Key Results
Lawrence, 1993, ch. 6	1)Study text+passively view animation 2)Study text+actively view animation (by constructing own input data sets)	1) Post-test accuracy 2) Time to take post-test	Participants who actively viewed animation scored significantly higher than students who passively viewed animation
Lawrence, 1993, ch. 9	1) Lecture-only 2) Lecture+passively view animation 3) Lecture+actively view animation (by constructing own input data sets)	1) Free-response post-test accuracy 2) Multiple choice true-false post-test accuracy	On free-response post test, participants who heard lecture and actively viewed animation significantly outperformed students who only heard lecture
Byrne et al. 1996, ch. 2	1) Study text only 2) Study text+make predictions 3) Study text+view animation 4 Study text+view animation + make predictions	1) Post-test accuracy 2) Prediction accuracy	Participants who viewed animation and/or made predictions scored significantly higher on hard questions than participants who did neither
Kann et al. 1997	1) Program algorithm 2) Program algorithm +construct animation 3) Program algorithm +view animation 4) Program algorithm +view animation +construct animation	1) Programming accuracy 2) Post-test accuracy	Participants who viewed animation scored significantly higher on post-test than participants who did not view animation

Table 2.1: Summary of 4 AV Effectiveness Experiments. [35]

b) Students can control the whole design process by selecting different setting parameters. When designing a combinatorial circuit, students can choose:

   - Different design entry methods.
   - Different logic components, such as logic gates, multiplexers, decoders or the combination of lookup table and multiplexers. When students choose logic gates, they can further select the gate type (combination of and-gates and or-gates, nand-gates only, or nor-gate only).
   - The algorithm that should be used to minimize boolean functions.

   When designing a sequential circuit, students can choose a variety of parameters such as the type of the FSM, the state code, and the type of Flip-Flop.

c) Students may control the progress of animation by clicking on forward and backward buttons.

d) When students use mouse to point to an element on the screen, the element and the related content are highlighted.

2.2.2.2 *Self-Assessment System*

Students can participate in the design of sequential or combinatorial circuits through the interactions introduced above, instead of viewing the process passively. Additionally, students can participate in the design process actively through the self-assessment system. This system allows students to enter solution proposals for specific design steps and verify them. In this way, students can identify gaps and weaknesses on time through such immediate feedback. Some educators believe students can acquire new knowledge better through solving many exercises with high difficulties. With the self-assessment system of DLD-VISU, students can create a number of problems conveniently and quickly and make progress by solving these questions.

DLD-VISU self-assessment system is applied to the following topics: the minimization process of boolean functions using the K-Map algorithm and the Quine-McCluskey algorithm, encoding the states in the FSM design process, as well as the creation of the state table. How to use DLD-VISU self-assessment system is introduced below by taking the creation of a state table as an example. The remaining three topics will be introduced in Chapter 3 in detail.

The state table of FSM is used to specify the next state and the output functions. DLD-VISU does not display the complete state table to students directly but shows a table with the corresponding columns blank. Students can fill in these columns and verify their entries by

The state table

Enter some or all missing vaules by clicking. Each click toggles the vaule between **0** and **1**. Then press **Verity** OR
Press **Verity** directly to see the state table.

Verity

Current state		Input			Next state		Output
$S_1(t)$	$S_0(t)$	bn10	bn5	reset	$S_1(t+1)$	$S_0(t+1)$	open
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	1	0	0	1	0
0	0	0	1	1	0	1	0
0	0	1	0	0	1	0	0
0	0	1	0	1	1	0	0
0	0	1	1	0	1	0	0
0	0	1	1	1	1	0	0

Figure 2.9: Self-assessment while Creating the State Table.

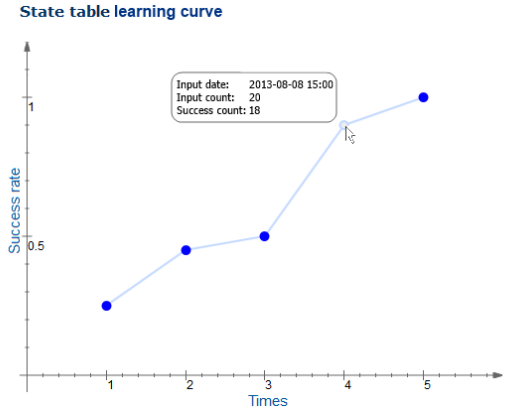


Figure 2.10: Learning Curve for State Table Setup.

clicking on the "Verify" button. DLD-VISU provides feedback according to students' entries, as shown in Figure 2.9. Green and red background colors indicate correct or incorrect entries, respectively. Note that self-assessment actions are optional. So, DLD-VISU provides a complete table even if students did not enter every value. Values that were not entered by students are returned without background color.

To allow students to keep track of their learning progress, learning curves and a learning history are integrated into the self-assessment system.

a) **Learning curves**

Every time students carry out self-assessment, the information about the rate of correct entries is saved in the system database automatically, as shown in Figure 2.10 students and instructors can observe the curves at any time to track the learning progress. Based on the curve data, students can adjust their study plans and instructors can adjust their teaching approaches for the corresponding topics.

b) **My History**

My saved animations

Date	Design process name	Show Animation	Show w/o Animation	Delete
2013-08-08 15:00:00	fsm_moore_binaryCode_Dflipflop			
2013-08-12 10:59:50	fsm_mealy_onehotCode_Tflipflop			

Figure 2.11: My Animations History Example.

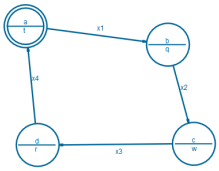
Upon completing an animation process, the system saves the output of all animation steps along with all set parameters in a database automatically; as shown in Figure 2.11. Students can view the animation later, which has several advantages:

- There is no need for students to re-enter input data or re-set the design parameters.. This saves time and allows students to reload older designs efficiently.
- Instructors can prepare some animation before the class time and reload the process in the classroom quickly.
- Students can generate animation by setting different parameters and investigate different design alternatives. For instance, students can design the same FSM using different types of flip-flops such as D-type or JK-type flip-flops. By observing the differences between the generated circuits, students can better understand the features of such flip-flops. The example in Figure 2.11 shows that a student has saved two FSM animations: a Moore machine with binary coding and D-type flip-flops; as well as a Mealy machine with one-hot state code and T-type flip-flops.
- Students could view previous animations in two modes, as shown in Figure 2.11. In the "Show Animation" mode, the system reloads the data and the student can play the animation step by step without entering any value or setting any parameter. In the "Show without Animation" mode, the system returns the animation results as a PDF document which can be observed, printed or downloaded by students. An example of such document is shown in Figure 2.12 that shows the design of a finite state machine.

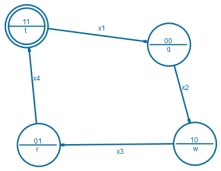
2.2.3 *Dual Coding*

Paivo proposed that it is more helpful for students to acquire knowledge through teaching materials that combine pictures and words rather than incorporating only one of these forms [21]. According to the CC theory, the form of knowledge encoding determines how difficult for students to master the topics. The dual coding theory demands that instructors encode the experts' mental models to literal

Step 1: Draw state diagram



Step 2: Select state code: One-hot Code



Step 3: State Table

Current state		Input				Next state		Output			
$S_1(t)$	$S_0(t)$	x1	x2	x3	x4	$S_1(t+1)$	$S_0(t+1)$	q	r	t	w
0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0	1	0	0	0
0	0	0	0	1	0	0	0	1	0	0	0
0	0	0	0	1	1	0	0	1	0	0	0
0	0	0	1	0	0	1	0	1	0	0	0
0	0	0	1	0	1	1	0	1	0	0	0
0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	1	1	1	1	0	1	0	0	0
0	0	1	0	0	0	0	0	1	0	0	0
0	0	1	0	0	1	0	0	1	0	0	0
0	0	1	0	1	0	0	0	1	0	0	0
0	0	1	0	1	1	0	0	1	0	0	0
0	0	1	1	0	0	1	0	1	0	0	0
0	0	1	1	0	1	1	0	1	0	0	0
0	1	0	1	0	0	0	1	0	1	0	0
0	1	0	1	0	1	1	1	0	1	0	0
0	1	0	1	1	0	0	1	0	1	0	0

Step 4: Select flip-flop type: D-flip-flop

Step 5: State equations:

$S_1(t+1)=S_1(t)'*S_0(t)'*x1'*x2'*x3'*x4' + S_1(t)'*S_0(t)'*x1'*x2'*x3'*x4 + S_1(t)'*S_0(t)'*x1'*x2*x3'*x4' + S_1(t)'*S_0(t)'*x1'*x2*x3'*x4 + S_1(t)'*S_0(t)'*x1*x2'*x3'*x4' + S_1(t)'*S_0(t)'*x1*x2'*x3'*x4 + S_1(t)'*S_0(t)'*x1*x2*x3'*x4' + S_1(t)*S_0(t)'*x1'*x2'*x3'*x4' + S_1(t)*S_0(t)'*x1'*x2'*x3'*x4 + S_1(t)*S_0(t)'*x1'*x2*x3'*x4' + S_1(t)*S_0(t)'*x1'*x2*x3'*x4 + S_1(t)*S_0(t)'*x1*x2'*x3'*x4' + S_1(t)*S_0(t)'*x1*x2'*x3'*x4 + S_1(t)*S_0(t)'*x1*x2*x3'*x4' + S_1(t)*S_0(t)'*x1*x2*x3'*x4 + S_1(t)*S_0(t)*x1'*x2'*x3'*x4' + S_1(t)*S_0(t)*x1'*x2'*x3'*x4 + S_1(t)*S_0(t)*x1'*x2*x3'*x4' + S_1(t)*S_0(t)*x1'*x2*x3'*x4 + S_1(t)*S_0(t)*x1*x2'*x3'*x4' + S_1(t)*S_0(t)*x1*x2'*x3'*x4 + S_1(t)*S_0(t)*x1*x2*x3'*x4' + S_1(t)*S_0(t)*x1*x2*x3'*x4$

$S_0(t+1)=S_1(t)'*S_0(t)'*x1'*x2'*x3'*x4' + S_1(t)'*S_0(t)'*x1'*x2'*x3'*x4 + S_1(t)'*S_0(t)'*x1'*x2'*x3*x4' + S_1(t)'*S_0(t)'*x1'*x2'*x3*x4 + S_1(t)'*S_0(t)'*x1'*x2*x3'*x4' + S_1(t)'*S_0(t)'*x1'*x2*x3'*x4 + S_1(t)'*S_0(t)'*x1*x2'*x3'*x4' + S_1(t)'*S_0(t)'*x1*x2'*x3'*x4 + S_1(t)'*S_0(t)'*x1*x2*x3'*x4' + S_1(t)'*S_0(t)'*x1*x2*x3'*x4 + S_1(t)*S_0(t)'*x1'*x2'*x3'*x4' + S_1(t)*S_0(t)'*x1'*x2'*x3'*x4 + S_1(t)*S_0(t)'*x1'*x2*x3'*x4' + S_1(t)*S_0(t)'*x1'*x2*x3'*x4 + S_1(t)*S_0(t)'*x1*x2'*x3'*x4' + S_1(t)*S_0(t)'*x1*x2'*x3'*x4 + S_1(t)*S_0(t)'*x1*x2*x3'*x4' + S_1(t)*S_0(t)'*x1*x2*x3'*x4 + S_1(t)*S_0(t)*x1'*x2'*x3'*x4' + S_1(t)*S_0(t)*x1'*x2'*x3'*x4 + S_1(t)*S_0(t)*x1'*x2*x3'*x4' + S_1(t)*S_0(t)*x1'*x2*x3'*x4 + S_1(t)*S_0(t)*x1*x2'*x3'*x4' + S_1(t)*S_0(t)*x1*x2'*x3'*x4 + S_1(t)*S_0(t)*x1*x2*x3'*x4' + S_1(t)*S_0(t)*x1*x2*x3'*x4$

Step 6: Minimized state equations:

$S_1(t+1)=S_0(t)'*S_1(t)'*x2 + S_0(t)'*S_1(t)'*x4 + S_0(t)'*S_1(t)*x3' + S_0(t)'*S_1(t)*x1'$

$S_0(t+1)=S_0(t)'*S_1(t)' + S_0(t)'*S_1(t)*x3 + S_0(t)'*x1'$

Step 7: Draw input circuit: see step 10

Step 8: Output equations:

$q=S_1(t)'*S_0(t)'$

$r=S_1(t)'*S_0(t)$

$t=S_1(t)*S_0(t)$

$w=S_1(t)*S_0(t)'$

Step 9: Minimized output equations:

$q=S_0(t)'*S_1(t)'$

$r=S_0(t)'*S_1(t)'$

$t=S_0(t)'*S_1(t)$

$w=S_0(t)'*S_1(t)$

Step 10: Draw circuit diagram

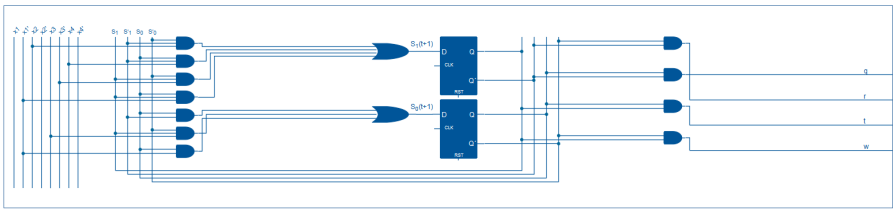


Figure 2.12: An Example for export an animation to a PDF document.

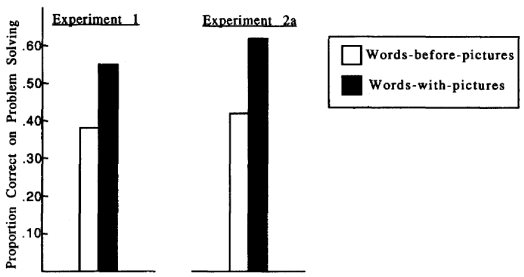


Figure 2.13: Proportion of correct responses on a problem-solving experiment for words-before-pictures and words-with-pictures groups in Experiments 1 and 2a [67].

descriptions and pictures or animation. The combination of these two kinds of code makes the encoding results closer to the experts' mental models.

The correctness of this theory was verified by Richard E.Mayer and Richard B. Anderson through two experiments [67] with 48 students at the University of California. 30 students took part in experiment 1 and experiment 2a and 48 students made experiment 2b.

First of all, 30 students were divided into two groups and each group contained 15 students. The students with zero experience of using or repairing bicycle pump learnt the working principle of bicycle pump by using different teaching materials. The teaching materials for one group were prepared with words and pictures separated. These students listened to the principle descriptions of pump first and then saw eight pictures that demonstrated the principle. The teaching materials for the other group combined words and pictures. Each picture was attached with descriptive words. Both groups could use the teaching materials for three times. Then, they needed to conduct two experiments. For Experiment 1, the students needed to answer four questions about the principle of pump. In Experiment 2a, the students needed to repeat the working principle of pump by words. See Figure 2.13 presenting the experiment results. The experiments proved that the teaching effect with the teaching materials combining words and pictures was superior to those with separate words and pictures.

Experiment 2b was used to test which way is best for learning:

- with pictures only;
- with words only;
- with the combination of pictures and words.

The experiment steps were the same as Experiment 1 and 2a. The students were divided into four groups randomly. Group 1 read the literal descriptions, group 2 listened to a recorded clip, group 3 watched pictures only, and group 4 watched the same pictures but the pictures were attached with the corresponding literal descriptions.

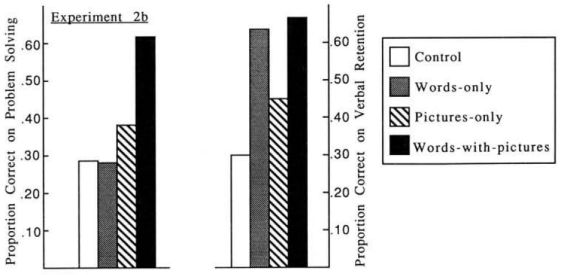


Figure 2.14: Proportion of correct responses on problem-solving and verbal recall experiments for four groups in Experiment 2b [67].

The experiment results are shown in Figure 2.14. The performance of group 4 was much better than the other groups.

The two groups of experiments above demonstrated that dual coding could enhance the teaching quality effectively. During the design of DLD-VISU, each animation or interactive operation was provided with literal descriptions so that students can be sure about the content of the current step. The following examples suggest this aspect. The literal descriptions in interaction and animation are displayed in the following two examples.

In the first step of designing a combinatorial circuit as shown in Figure 2.15 (a), the system firstly displays the prompt "How would you like to enter your function?" with two options "Boolean function" and "Truth table". If students choose boolean function, the system displays the prompt of next step "In which form should the boolean function be represented?" (Figure 2.15 (b)). According to the students' choice, the system guides students until they complete the function entry (Figure 2.15 (c), (d)). With this gradual prompt, students can deepen their understanding of the different function forms such as the CNF and DNF and the mutual conversion between the standard form and canonical form.

DLD-VISU also represents corresponding literal descriptions while displaying animations. As shown in Figure 2.16 (a), when the illustrating animation of K-Map algorithm enters Step 3, the animation on how to find all *prime implicants* in K-Map is displayed on the left side of the screen and the system accordingly prints the literal descriptions on the right upper side of the screen. Upon clicking on the next-step button, the animation of finding the *core implicants* is played continuously on the left side of the screen while the literal descriptions on the right side is upgraded accordingly, as shown in Figure 2.16 (b). Students can acquire a better understanding of the current step by reading its literal description.

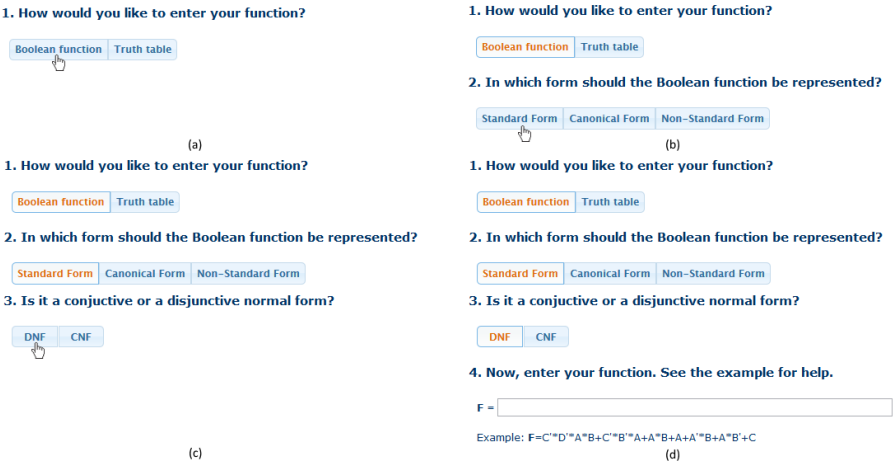


Figure 2.15: The "enter function" step of designing a combinatorial circuit using Dual Coding.

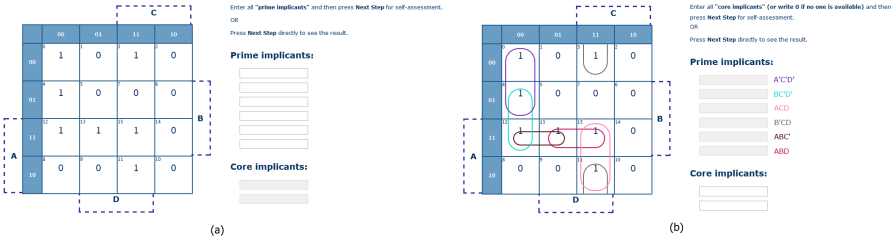


Figure 2.16: The K-Map animation using Dual Coding.

2.2.4 *Individual Differences*

Individual differences refers to the extent and type of distinctions among individuals on some of the significant psychological traits, personal characteristics, cognitive and emotional components. [85]. It is manifested in both quality and quantity. Qualitative differences relate to mental and physical features as well as to the behaviours. In contrast, quantitative differences refer to the development level and speed. The study of individual differences in pedagogy mainly aims at finding approaches to improve education and teaching. In the teaching activities where a class is treated as a unit, students have many differences, which affects the teaching process and results directly or indirectly. **Individual differences** were highlighted by educators earlier. Individual teaching modes were proposed already in the 1950s and 1960s. The "*Personalized System of Instruction*" [56], for instance, refers to an individual teaching scheme developed and experimented by Keller at the University of California. This system emphasizes the students' self-control of learning progress to adapt to the individual differences. Individually Prescribed Instruction is another individual teaching mode proposed by Bowen et al. at the University of Pittsburgh [91]. According to the individual teaching mode, different students should be provided with different teaching

methods and plans, which may cause a large cost and overhead, thus, an increasement in the school fees. In the last thirty years, some researches about the individual differences and teaching learners have been transferred from the development of individual teaching strategies to the discussion of the relationship among the learners' individual differences and teaching effect. However, no unified and convincing conclusion has been made on such kind of researches.

Individual differences were also taken into account during the design of DLD-VISU. For instance, when learning K-map algorithm, students can use function with three or four variables depending on their learning or understanding levels. For the design of sequential circuits, students can design FSMs with different complexities in terms of state diagram, number of states, state code, or the type of flip-flops. In addition, the speed of the design process via DLD-VISU can be controlled by the users to fit her or his learning level. The entire design process can be reviewed by calling the history functions mentioned above.

2.3 WEB DESIGN CONCEPT OF DLD-VISU

With the increasing network speed and decreasing access fees in the recent years, diverse convenient and low-price or free internet applications are becoming a part of people's lives. Web 2.0 also becomes a mature service platform. When it was firstly proposed in 1999, web 2.0 referred to the internet mode where users can generate contents [23]. In the contrary, the content of traditional websites was generated by companies. Facebook is a typical web 2.0 product.

Similar to the concept of web 2.0, E-Learning also enters its 2.0 era. From the perspective of E-Learning 2.0, the traditional E-Learning system only applies Internet technology to transfer the fixed learning contents to students. However, the new E-Learning environment pays more attention to interaction, customization, and socialization [53].

The concept of E-Learning 2.0 is reflected in the design idea of DLD-VISU. Differing from traditional E-learning software, DLD-VISU integrates the usability, interaction, flexibility and sociability into its design. The specific descriptions of these features are presented below.

2.3.1 *Usability*

Simplicity and convenience are of great importance for learning software. If students need to take much effort to install and configure learning software or when this software runs only on specific systems, it is obvious that students would not be patient about such a solution. To enable a convenient usage, DLD-VISU was designed as a web application which has the following advantages:

1) **Cross-platform:** The operation environment of pure web page applications is browser-related and independent of the users' operating system or hardware. DLD-VISU runs on any PC, tablet PC or smart phone running Windows, Linux, Mac OS, or Android, as long as these platform provide a browser such as Internet Explorer, Safari, Firefox, or Chrome. In contrast to common network applications, "pure" web page applications contain no plug-ins. The diversified plug-ins can be embedded in the web pages of the common network applications to enrich the browser with additional functions such as playing video, displaying Flash animation, performing bank transactions, etc.. However, embedded plug-ins also brings certain inconvenience to users. Firstly, all the embedded plug-ins should be installed before users put them to use but the users might fail to do so under some circumstances. For instance, some anti-virus software causes conflicts with some plug-ins and even forbids installing plug-ins as they are recognised as virus. Also, the right of installing plug-ins on some public computers (such as in school computer rooms, or in library and students' dormitory) is not provided to users, as a rule. If plug-ins cannot be installed correctly, users will be unable to get access to the relevant functions. Secondly, if users' browsers are upgraded and the new version is no longer compatible with the plug-in installed previously, the user has to upgrade or reinstall the plug-in manually or even buy the new one provided by the plug-in provider. Thirdly, different browsers or operating systems have different extents of supporting the plug-ins. For example, the Flash plug-in can only be installed on desktop systems but not on Apple IOS systems. For these reasons, DLD-VISU uses only *Cascading Style Sheets (CSS)* and JavaScript, which are supported by all browsers as the network standard of *World Wide Web Consortium (W3C)* certificate.

2) **Maintainability:** The Browser/Server (B/S) model software using pure web page is more advantageous in the aspects of maintenance and upgrade than the desktop software, Client/Server (C/S) software or B/S software with plug-ins. Developers can upgrade and update the system whenever it is possible to upload the new code to server instead of sending a new version of the program to users or upgrading patch. Users get access to the upgraded system directly without any additional operation. The upgrade function in DLD-VISU is of highest importance because its concept relies on feedback provided by students and instructors. In the following we give one example for an upgrade that has been accomplished during the development and test of DLD-VISU.

As mentioned previously, the learning efficiency and outcomes can be improved by motivating students to work actively through enhancing their participation. Therefore, many interactive operations are embedded to DLD-VISU. However, the grade of interaction should be chosen carefully, as too many interactive operations may make students feel tired and lose interest. For example, during editing the state diagram in the first version, all the states were placed in one row and students had to replace them appropriately. During the test, students and lecturers replied that this approach process was tedious. Therefore, in the new version, the system places the state upon clicking on a desired point within the state diagram editor.

2.3.2 *Flexibility*

Since DLD-VISU almost has no limit on operating environment, students can formulate a suitable learning plan. The learning record is stored on the server instead of local devices, which allows students to continue learning even with a different device and at different time. Some lecturers refuse using AV software since students may use it to complete homework simply by entering data and letting the software generate the solutions. In order to avoid such misusage, a special **Access Control List (ACS)** was designed and embedded into DLD-VISU. The ACS manages an Access Control List, which includes all teaching themes supported in DLD-VISU. The lecturers can lock any theme for a specific period of time. In the example shown in Figure 3.28, the lecturers locked Multiplexers and Decoders for the week (May 1-8, 2013), so that students could not use DLD-VISU to design combinatorial circuits with these components in this period. Note, however, that this lock does not prevent students from recalling previous animations of these themes from the history database.

2.3.3 *Sociability*

DLD-VISU could not only display animation but also save the process of animation operated by students. So students could also demonstrate their animation for other students or compare their animation mutually. Through discussions with other students, students can deepen their understanding of knowledge.

2.4 DLD-VISU APPLICATION SCOPE

DLD-VISU is not only AV software but also a network education platform that could be applied to the whole teaching process. Seven application fields of AV software have been introduced in chapter 2.1.

These seven fields have been totally covered by DLD-VISU. Table 2.2 shows how each of them is supported by DLD-VISU.

2.5 SUMMARY

DLD-VISU was developed based on four pedagogical theories. In this chapter we summarize the theories and show the role of each of them played in DLD-VISU.

1) Epistemic Fidelity: Using graphs can make the transfer of experts' mental models to viewers robust and efficient, because graphics have brilliant capabilities to encode experts' mental models.

2) Cognitive Constructivism: Building someone's own understanding can be achieved by participating in AV actively.

3) Dual-coding: Representing Knowledge that is dually coded (e.g. graphics and text) boosts the transfer of knowledge most robustly and efficiently.

4) Individual Differences: Some people are able to gain more benefits from AV than others due to the differences between human cognitive abilities and learning styles.

Epistemic Fidelity and **Dual-coding** are used for Encoding of knowledge. **Cognitive Constructivism** is used for constructing the resource for knowledge. The role of **Individual Differences** is unspecified, but we take this theory into account in the complete design process.

Conditions	Examples
Lectures	Instructors could use DLD-VISU to demonstrate the complete process of how design a circuit according to a FSM.
Study	Students could use DLD-VISU any time and anywhere.
Assignments	Students could finish their homework by using self-assessment system. For example, instructors gave a question of using the K-Map algorithm to simplify an equation to students. Students could finish it by inputting such equation to the DLD-VISU, and system will draw one blank K-Map for students. Students then should find all *prime implicants*, *core implicants* and finally simplified result and fill them in the position designated by the system. The system will then judge and save the result. Instructor could check the results to see how students finish their homework.
Class discussion	Using self-assessment system could DLD-VISU also be used in class discussion. For example, in the step "Select state code", when students select a coding, DLD-VISU does not directly show the code, so instructors can first let the students calculate the correct coding by themselves, and discuss whether the answer of students is right.
Labs	DLD-VISU has strong traceability, instructors could prepare some special examples - for example,an equation without *core implicants* or has more than one results- so that students could experiment in class and see the result. This could deepen students' understandings on the difficult points.
Office hours	Instructors can help students individually out of classroom according to their learning history.
Tests	Using access control system, instructor could prepare exam questions, so students could never acquire the answers to these questions by DLD-VISU

Table 2.2: Examples for seven conditions supporting by DLD-VISU.

3

VISUALIZATION AND ANIMATION IN DLD-VISU

This chapter explains the visualization and animation processes supported in the DLD-VISU which includes 2 main parts: the animation functions and the learning management functions. The Animation function includes the design of the combinatorial logic and sequential logic. Boolean function minimization using K-Map or Quine-McCluskey algorithm is a part of the combinatorial logic design but can also be invoked separately. At the end of this chapter, we introduce the learning management functions, this includes **Self-Assessment System**, **Access Control System** and the corresponding user register system.

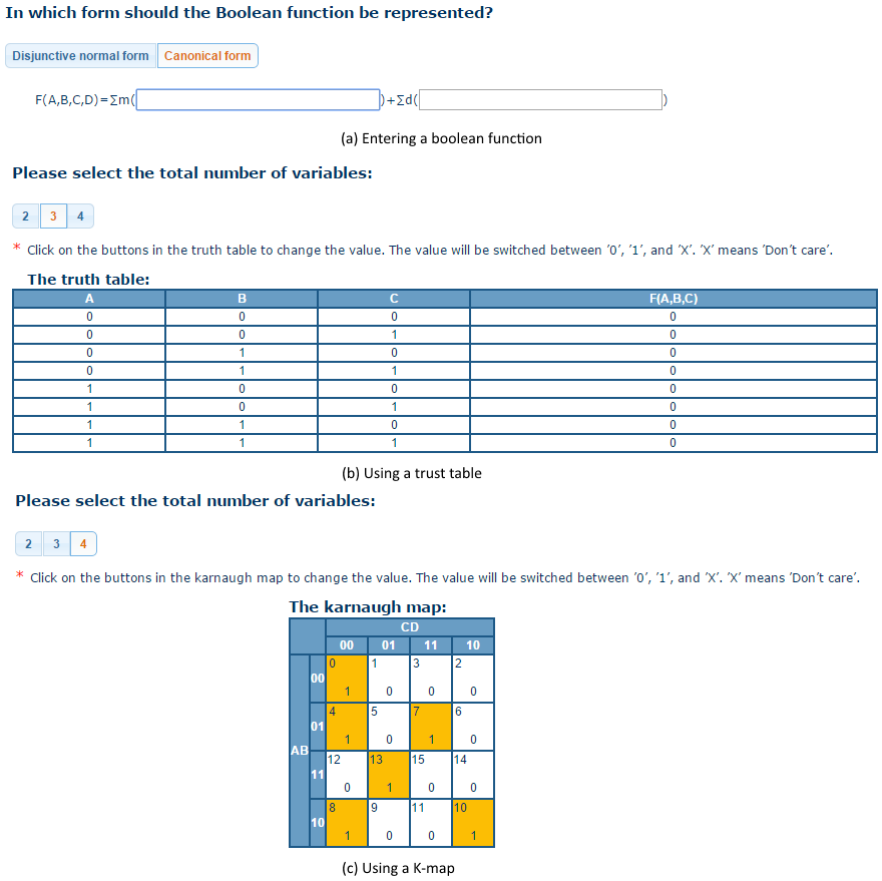
3.1 AV OF COMBINATORIAL LOGIC DESIGN PROCESS

Before we introduce AV of combinatorial logic design process, we first introduce two Minimization algorithms: K-Map Minimization Algorithm and Quine-McCluskey Minimization Algorithm. Students can either study these two Minimization algorithms alone, or use these two algorithms in the combinatorial logic design process to see their effect.

3.1.1 *K-Map Minimization Algorithm*

DLD-VISU visualizes the design of K-Map Minimization Algorithm in 5 steps. In the first step of K-Map minimization, the system provides three ways for users to choose, including: entering a boolean function with DNF Form; using Trust-Table or using K-Map Figure 3.1. The system implements different GUIs in the second step according to users' choice in the first step. As presented in Figure 3.2, if users input boolean function manually in the first step, the system displays

Figure 3.1: The First step of K-Map minimization.



(a) Entering a boolean function

(b) Using a trust table

(c) Using a K-map

Figure 3.2: Three different presents of second step of K-Map minimization.

a Text-box for users to input. Users can also select distinct ways to input according to their own habits, e.g. using DNF Form or Canonical Form. Note that users are not allowed to employ CNF Form to input in the course of demonstrating K-Map Minimization Algorithm. Because K- map correlates with DNF Form. If users input CNF Form, it needs to be transformed to be DNF Form before minimization. While this transformation does not have direct help for users to understand K-Map Minimization Algorithm. In accordance with the **Abstraction** principle, we eliminate the support to CNF Form during designing the system. If users choose Truth-Table to create K-Map, the system creates a Truth-Table according to the variable number set by users. After that users are able to pick different rows of the Truth-Table to generate corresponding boolean functions. If users choose to generate K-Map directly, the system creates a blank K-Map in accordance with the variable number set by users and users may directly fill content in this K-Map. The reason why the system supplies three different methods for users is to hope they are able to create K-Map according to their habits so as to reduce the time spent by users to accommodate themselves to the system, making users more concentrated on algorithms. If users select Truth-Table or K-Map as input methods,

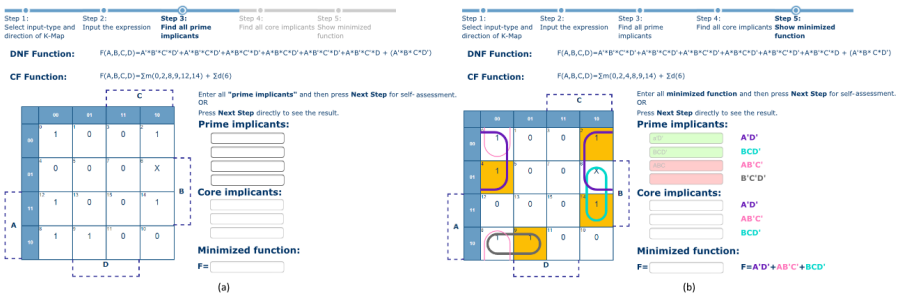


Figure 3.3: K-Map Minimization

the system demands users to input the number of variables first. We make use of **Individual Differences** theory here and users may select different amount of variables. DLD-VISU offers the demonstrations of minimizing distinct K-Maps with 2 to 4 variables. If the number of variables is greater than 4, the corresponding K-Map is getting very big. For example, the K-Map with 5 variables consists of 32 cells. Too complicated K-Maps do not help users to understand K-Map better but make them spend time in finding *implicants*. In order to deepen users' understanding to the relationship between K-Map and DNF Form, we apply **Dual Coding** theory. If users fill values in K-Map or Truth-Table, the system is automatically generating the corresponding boolean functions, making users able to intuitively observe the correlations between boolean functions, K-Map and Truth-Table.

The K-Map is set up with the function *minterms* as depicted in Figure 3.3 (a). In the following, all *prime implicants* are determined. For self-assessment, students are able to edit the *implicants* and verify their entries. The same steps are applied to determining the *core implicant*.

Figure 3.3 (b) shows the animation window after completing all minimization steps. This figure depicts that students have entered three out of four *prime implicants* in step 3, whereas the third entry is wrong. In the steps 4 and 5 students did not give any entries. A *prime implicant* as a boolean term is displayed in the same color as the outline color of the corresponding grouping on the K-Map. The squares with orange background in the map highlight the *minterms* that are covered by only one *prime implicant*, thus, justifying why this *implicant* is a *core implicant*. For instance, the *minterm* AB'C'D is only covered by the *prime implicant* AB'C'. This is why AB'C' is a *core implicant* which is highlighted on the map by a bold outline. *Prime implicants* stay with a thin outline such as B'C'D'. Note that the final result may not be unique, and DLD-VISU will list all the best results, but students do not need to enter all the correct results, just one correct answer is enough.

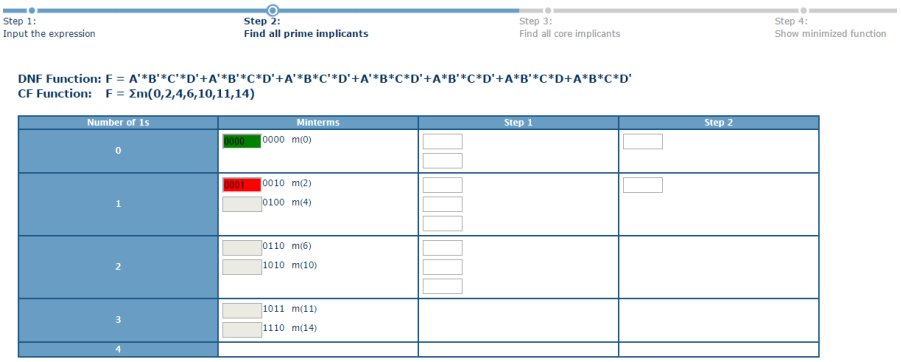


Figure 3.4: Show all minterms in the "Minterms" column.

3.1.2 *Quine-McCluskey Minimization Algorithm*

For the functions of more than 4 variables, identifying the *implicants* and the *core implicant* using K-Map can get tedious and error-prone. In the contrary, the Quine-McCluskey (QMC) algorithms is a tabular search-based approach which makes it more suitable for minimizing large functions. QMC proceeds as follows. The first step consists in finding all *implicants* and is itself a multi-step procedure as depicted in the example of Figure 3.4. Students can fill corresponding *minterms* in each empty cell. The system checks the answers and presents feedback to students by changing background color of the textbox. Green means correct and red means incorrect. Students may also click "next" button to skip this process and directly see the result. In the example displayed in Figure 3.4, students filled two *minterms* (mo and m2) of which mo is correct and m2 is incorrect. The background color of the other unfilled textboxes are grey.

In an initialization step, the function *minterms* are grouped according to the number of 1's and listed in blue color in the "minterms" column of the table. This grouping according to the number of 1's accelerates the search for the adjacent *minterms* that follow in a successive way. The adjacent *minterms* found in the "minterms" column are listed one by one in a new column (Step 1), where the different bit is replaced by "-" as depicted in Figure 3.5(a).

In order to make students better and more clear observe this changing process, the system highlights relevant *minterms* with red color. In the example demonstrated in Figure 3.5(a), m(0, 1) in the step 1 column is produced by m(0) and m(1) in the "Minterms" column. Therefore, mo and m1 are highlighted when the system creates m(0, 1).

Any *minterm*, for which an adjacent *minterm* is found, is marked by a gray color and in the crossed as depicted in Figure 3.5(b). Upon completing the list in Step 1, this list is searched for adjacent *minterms*, which are inserted in a new column (Step 2). For that the sign "-" is treated as a third bit value. This procedure is repeated iteratively until

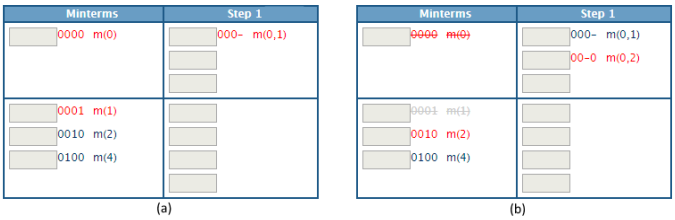


Figure 3.5: The Animation for finding all prime implicants.

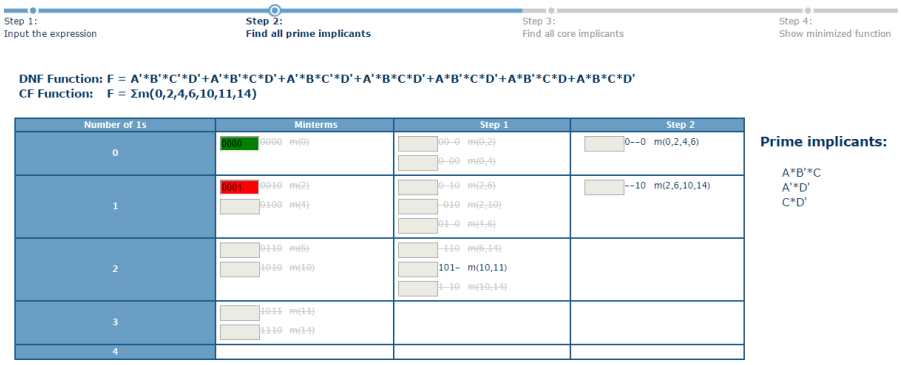


Figure 3.6: Show the prime implicants

no adjacent *minterms* are available. The *implicants* correspond to the remaining entries with blue color are depicted in Figure 3.6.

In the second step the *core implicant* are determined. This is done using a matrix that shows which *minterms* are covered by each *implicant*. In Figure 3.7, for instance, the *minterms* m(0) and m(4) are only covered by the *implicant* A'D'. That is why this *implicant* is a *core implicant*. This is indicated by highlighting the check sign "X" in the corresponding columns. In the last step the minimized function is set up as the sum of all *core implicant* and as many *implicants* as necessary to cover all the *minterms* of the function. If several *implicants* are possible, the one with the least number of variables is taken, i.e., the one that covers as many *minterms* as possible.

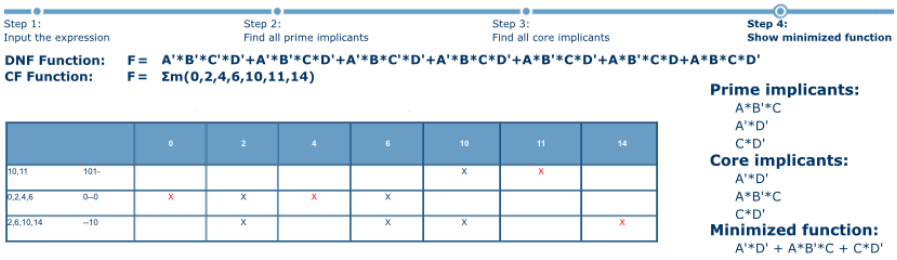


Figure 3.7: Finding Core Prime Implicants.

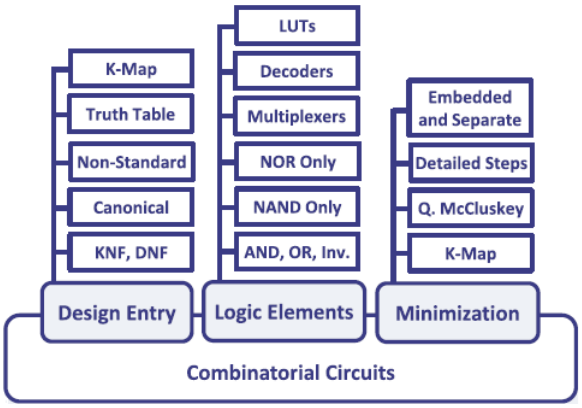


Figure 3.8: All covered topics of Combinatorial logic design process.

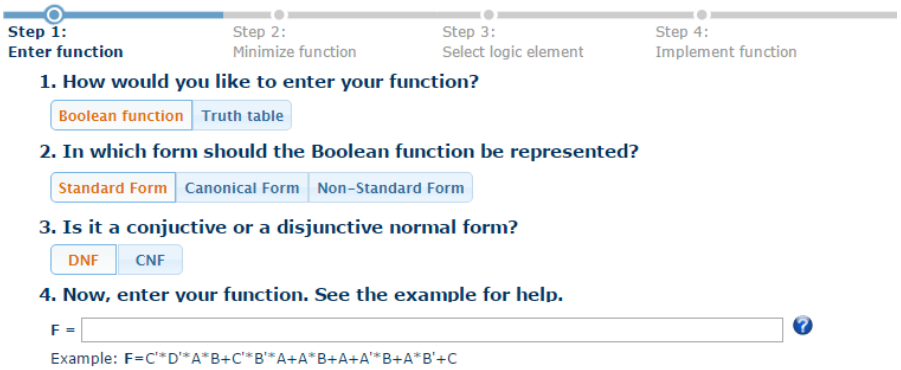


Figure 3.9: Design Entry as a Boolean Function.

3.1.3 *Combinatorial logic design process*

A combinatorial logic circuit may be implemented using gates, multiplexers, decoders, or look-up tables. DLD-VISU is not only valid to the main design processes for combinatorial logic, but also to several intermediate steps such as function minimization using the K-Map or the Quine-McCluskey scheme, boolean function implementation using multiplexers, decoders, NAND gates, or NOR gates. Figure 3.8 lists all knowledges about combinatorial logic circuit that students can learn through DLD-VISU.

DLD-VISU partitions the design process of combinatorial circuits into four steps as detailed in the following, see the progress chart at the top of Figure 3.9.

1) Function Entry: Students can enter the function either in a boolean form or as a truth table. When a boolean function is selected, students are requested to choose the standard form, the canonical form, or the non-standard form. If a standard form is selected, students can select the DNF or the CNF. Then the boolean function can be edited using the same format given in

the last line as an example, see Figure 3.9. Similarly students can select the canonical form (sum of *minterms*) and enter the function as a list of *minterm* numbers such as F = ∑m(0,1,4,9). The non-standard form can include sums of products as well as products of sums. Note that Questions 2, 3 and 4 in Figure 3.9 appear dynamically depending on the answer given to each previous question. This helps students' focus on one question at one time, which is in line with the abstraction concept of DLD-VISU. Furthermore, note that requesting students to answer the above questions aims at making them familiar with the terminology of boolean algebra. For instance, they can easily identify that a boolean function comes in one of three forms, and that the standard form may be a CNF or a DNF. If a truth table is chosen to enter the function, then students are requested to enter the number of variables. Then, a corresponding truth table is displayed in which all output values are 0 by default. The student can click on the desired table line to switch the corresponding output value between 0, 1, and X (Don't care). While editing the truth table the corresponding boolean function represented in the disjunctive normal form and in the compact canonical form appears under the table automatically, see Figure 3.2.

2) Function Minimization: The minimization of logical functions is a main learning outcome in any DLD course. Students should be kept aware that a circuit, which implements a function with less components, is more economic, more reliable, shows better performance, and consumes less power. Minimizing logical functions using boolean algebra is only practical for small functions with limited number of variables. The K-Map approach is popular for functions of up to 4 variables. The Quine McCluskey algorithm is more appropriate for functions of more variables. DLD-VISU supports both approaches as will be detailed in the following. In the minimization step students are asked to select a minimization method. Depending on this selection a progress sub-chart is shown under the minimization step of the main progress chart, see Figure 2.5. Note. As mentioned previously, a progress sub-chart is a progress chart that represents a step of the main chart in expanded form. The sub-chart has a two-level numbering. The first number is the same number of the expanded step in the main progress chart. The second letter is a sequential letter starting with a. In this paper, sometimes we just show the sub-chart, for space reasons.

3) Selecting Implementation Style: DLD-VISU supports the design of combinatorial circuits with gates, multiplexers, decoders, as well as with look-up tables plus multiplexers which are typical in modern FPGAs. Students can select one of these alternatives

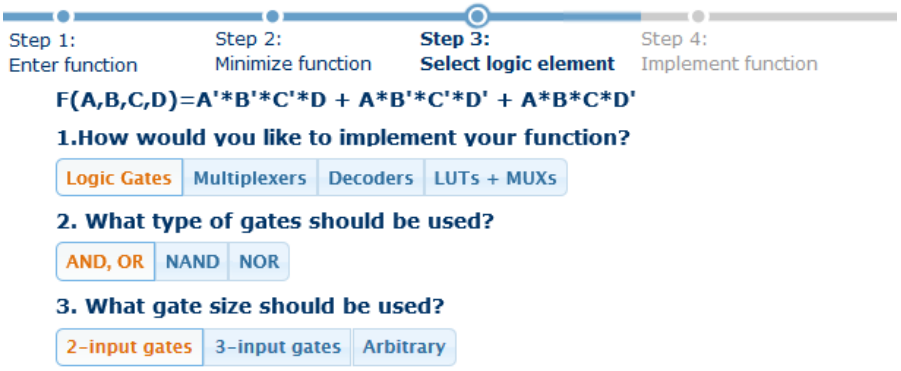


Figure 3.10: Selecting the Implementation logic elements.

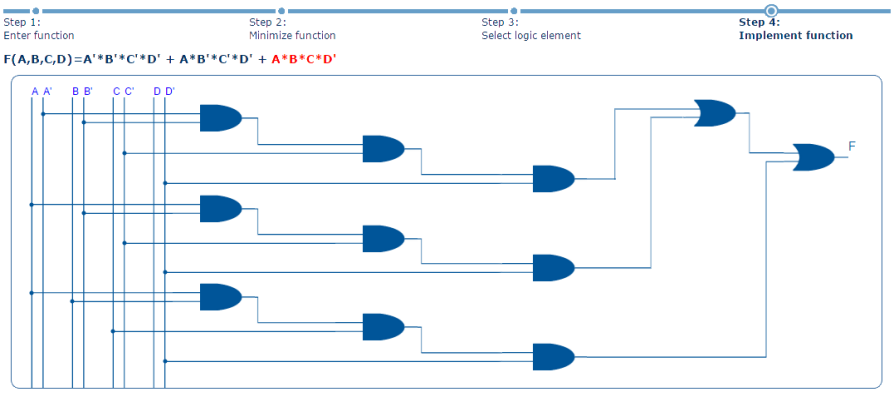


Figure 3.11: Implementing a Combinatorial Circuit with AND and OR.

as illustrated in Figure 3.10. When logic gates are selected, students can also specify the type and the size of the gates that are employed.

4) Implement function: In this step, the minimized function is mapped to the combinatorial circuit using the selected components.

   a) Implementation with Gates: The implementation with AND and OR gates is straightforward. Figure 3.11 shows the implementation of the function F(A,B,C,D) = A'B'C'D + AB'C'D' + ABCD' with 2- input AND and OR gates. The implementation with NAND or NOR gates is more complex and accomplished as a four-step process. The following points explain this process for the same function above assuming that it should be implemented using 2-input NAND gates.

      - First, the circuit is implemented with AND and OR gates only. The resulting circuit will be the same as the one given in Figure 3.11 .

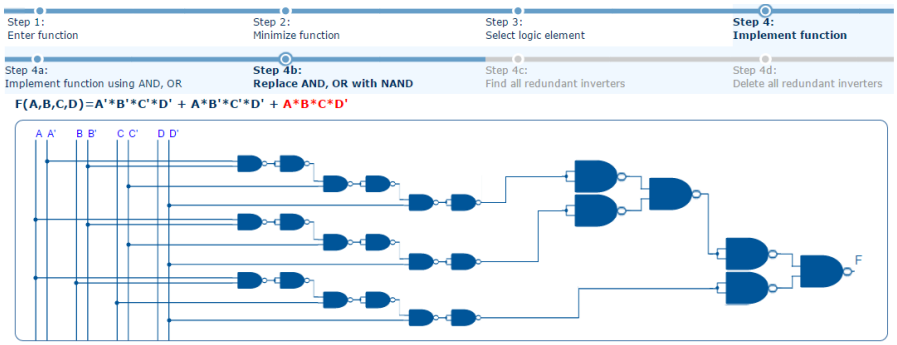


Figure 3.12: Replacing AND and OR Gates by their NAND Equivalents.

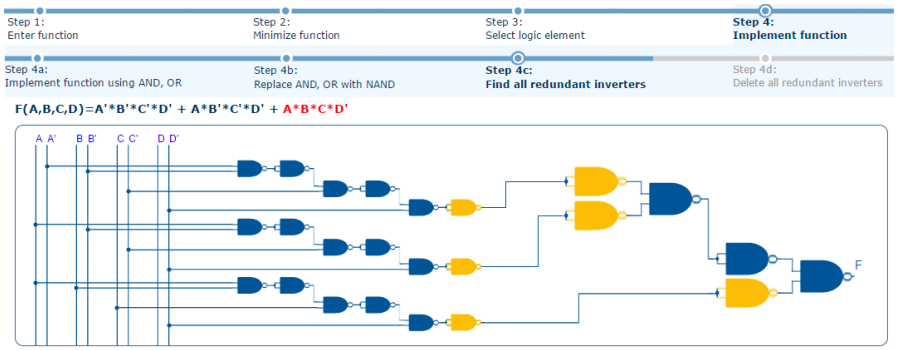


Figure 3.13: Identifying Redundant Inverters.

- Each AND gate is replaced by its equivalent NAND circuit, which consists of two NAND gates. The first NAND gate takes the inputs. Its output is connected to both inputs of the second gate that realizes an inverter. Similarly, each OR gate is replaced by its equivalent NAND circuit which consists of three NAND gates. Figure 3.12 shows the result of this step.
- In the next step, all the redundant inverter pairs are found and marked as presented in Figure 3.13. A redundant inverter pair consists of two consecutive inverters.
- Finally, the marked redundant inverter pairs are deleted as depicted in Figure 3.14.

b) Implementation with Multiplexers If multiplexers are selected to implement the function, the boolean function is firstly expanded according to Shannon's scheme. The expansion relies on using the function variables as control signals for the MUXs. The MUXs' data inputs either stem from the outputs of previous MUXs or are connected to the constant values 1 or 0. Currently only 2-1 MUXs are supported by DLD-VISU. According to Shannon, the function

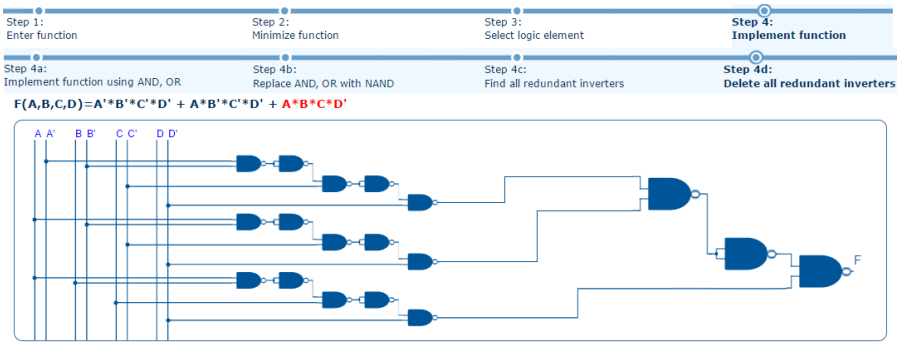


Figure 3.14: Deleting Redundant Inverters.

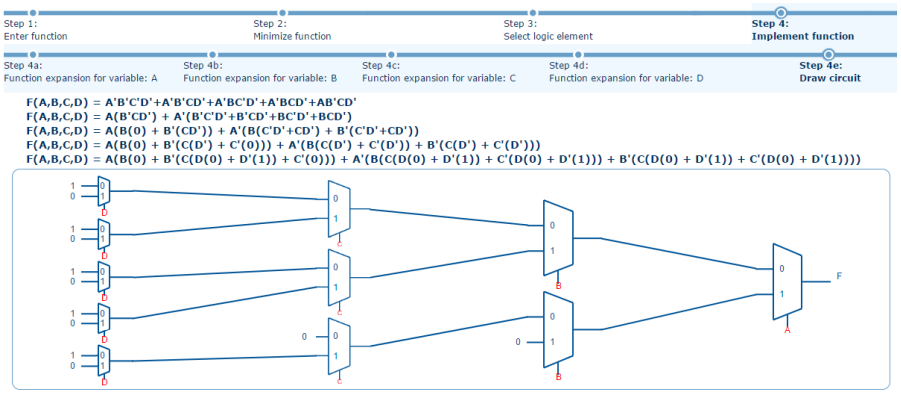


Figure 3.15: Implementing a Combinatorial Circuit with Multiplexers.

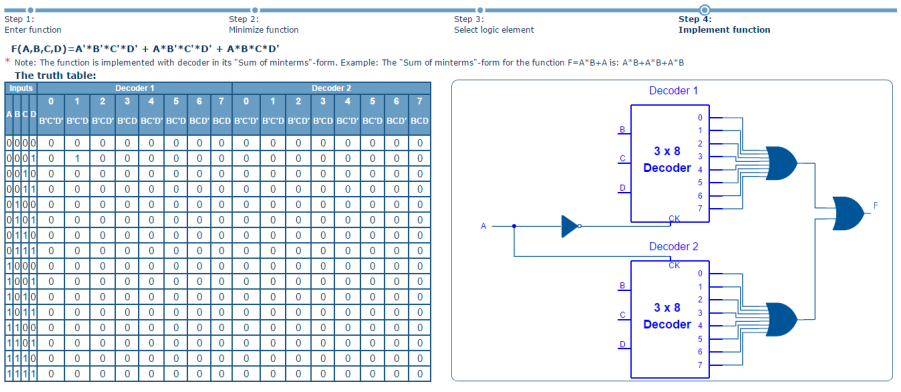


Figure 3.16: Implementing a Combinatorial Circuit with Decoders.

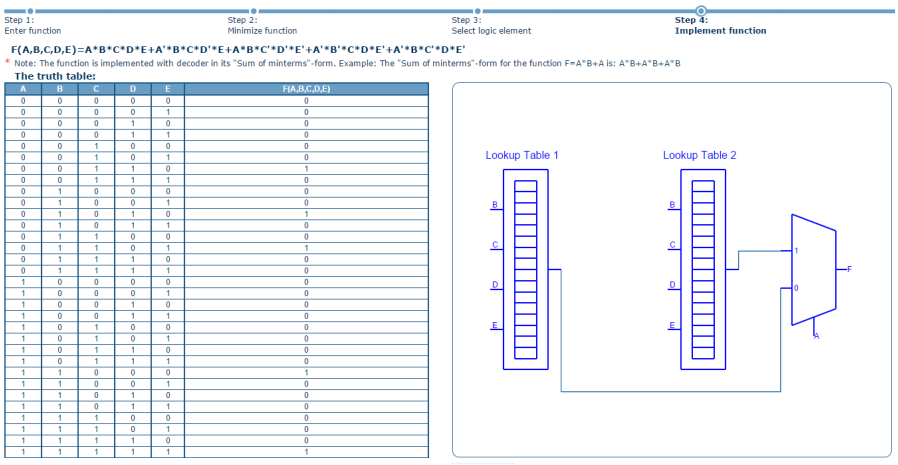


Figure 3.17: Implementing a Combinatorial Circuit with LUTs + MUXs.

expansion is displayed step by step followed by drawing the entire circuit as seen in Figure 3.15.

c) Implementation with Decoders DLD-VISU supports the implementations with 3x8 and 4x16 decoders. If the function has 2 or 3 variables, the circuit is built up using one 3x8 decoder. If the function includes 4 variables, the circuit can be implemented either using one 4x16 decoder or two 3x8 decoders. The functions with 5 variables are implemented using two 4x16 decoders. Figure 3.16 shows how DLD-VISU generates a decoder circuit for a 4-variable function using two 3x8 decoders.

d) Implementation with Look-up Tables and Multiplexer DLD-VISU supports the implementation with 16-bit lookup tables. Additionally, the 5-variable functions can be implemented using two look-up tables and one multiplexer. The goal of this implementation form is to introduce students to SRAM-based Field-Programmable Gate Arrays (FPGA), see Figure 3.17.

3.2 AV OF SEQUENTIAL LOGIC DESIGN PROCESS

DLD-VISU visualizes the design of finite state machines in ten steps as shown in the progress chart in Figure 2.4. Both Moore and Mealy machines are supported with binary, Gray, and one-hot state code as well as D-type, T-type, and JK-type flip-flops. Thus, a total of 18 design alternatives can be visualized, as demonstrated in Figure 3.18.

The design steps described below takes only one of these design alternatives into consideration. This design alternative is a Moore machine, with binary-coded states and D-type flip-flops.

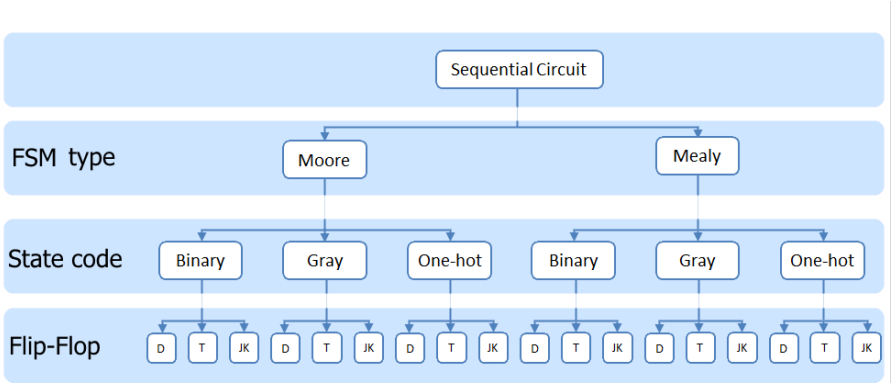


Figure 3.18: All design alternatives of Sequential Circuit of DLD-VISU.

1) State Diagram Entry: After selecting the FSM type, students can specify the machine behavior as a state diagram. Figure 3.19 shows an example for an edited state diagram of a simple Moore machine. Four buttons are available to enter the state diagram. The properties of a state or a transition can be edited in the boxes right to the state diagram. State and transition properties differ depending on the machine type. In a Moore machine, for instance, the state has two properties: the state name and the output signals that should be active in that state. Several output signals can be active in a state. For that, the signal names must be separated by the space character. Additionally, one state can be marked as the start state, which the machine returns to after reset. A transition has just one property, which is the boolean expression that represents the condition for that transition. The boolean expression can be the name of a single input signal or a complex expression with several input signals. To build such an expression in DLD-VISU, the characters "!", "*", and "+" are used for inverting, AND, and OR operation, respectively. Students may deepen their understanding to FSM by drawing FSM diagram manually. In particular, by employing moore and mealy FSMs respectively to realize a same requirement and observing the differences between two FSM diagrams can make students really grasp the working principle of FSM.

   In the course of design and development, we also considered to set "select state code" or "construct state table" to be the first step of the whole sequential logic design process or allow students to import FSM diagram drawn by other tools/softwares instead of drawing on their own, which saves the time in drawing FSM diagram for students. Because beginners are not familiar with FSM. Various mistakes may be made by students in the course of drawing FSM diagram, such as some state do not have connections with others, forgetting to mark transition condition, or making transition direction opposite, etc. It may take students

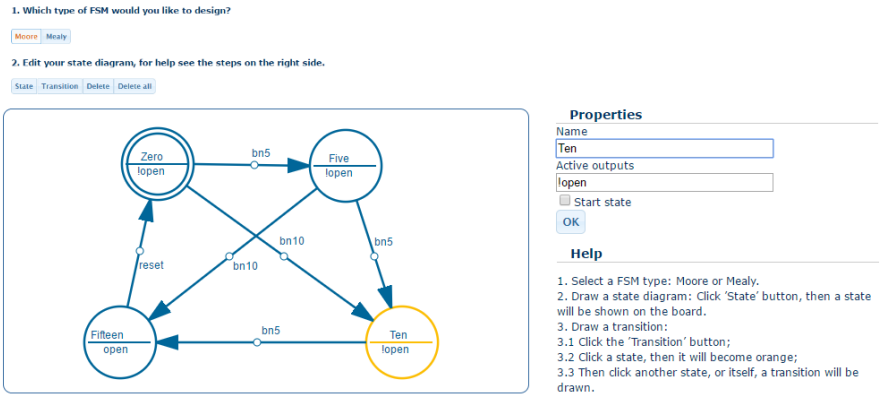


Figure 3.19: Editing state diagram for FSM design.

plenty of time to correct these mistakes, leading to their reluctance to continue using DLD-VISU. However, after care consideration, we finally determined to set "drawing FSM diagram" as the first step of the whole design process. Because according to Cognitive Constructivism theory, the learning initiative of students may be enhanced by drawing FSM diagram all by themselves and continuously correcting errors of FSM diagram happens to be active exploration, discovery and construction of knowledge. In addition, students can decide the complexity of FSM diagram by themselves, which is the reflection of Individual Differences theory in DLD-VISU.

2) State Code Selection: The state code has an important effect on the behaviour and the resource usage of a FSM. One-hot code, for instance, enables a relatively simple input and output logic. The result is a higher-performance FSM, however, at the cost of flip-flop usage. With DLD-VISU students can select and enter the state code. Consequently, the state diagram is regenerated, where the state names are replaced by the state code, see 3.20. The correct state code table is also displayed so that the student can map the state code to the state names. With the aid of background color, the student can assess which entries are correct or wrong. As mentioned before, Self-Assessment is also a significant method to guide students to learn actively.

3) State Table Creation: From the state diagram with encoded states, the state table can then be created as displayed in Figure 3.21. If the state table is too large to fit in the animation window, DLD-VISU adds scrollbars as shown in the example. This figure also shows that students have entered some values for the next state and the output signals. Correct and wrong entries are marked by a green or red background color, respectively. For the traceability, the state diagram is also displayed

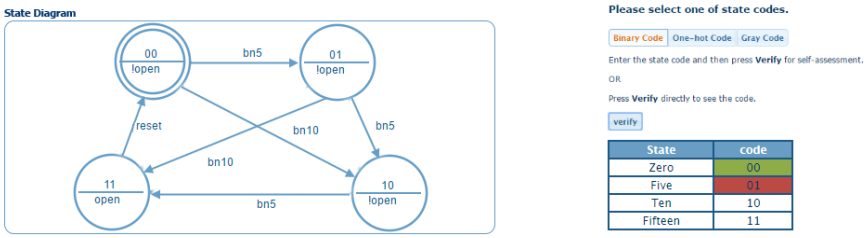


Figure 3.20: State coding.

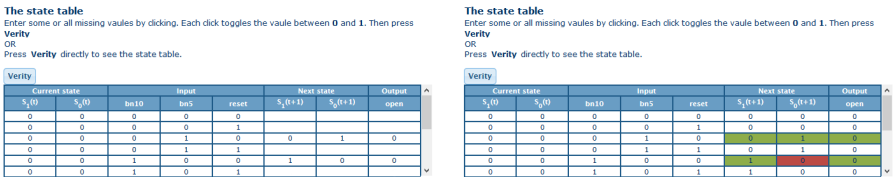


Figure 3.21: State table creation.

in this step. In Figure 3.21, however, we cut out the state diagram, for space reasons. From the following step on, the state diagram will not be displayed anymore because the next steps are accomplished based on the state table only. This complies with the abstraction principle on which DLD-VISU is built.

In the State table produced by DLD-VISU, "Next state" and "output" columns are empty. Such design avoids to present answers directly so that students is able to own the opportunities to think. As shown in Figure 3.21(a), students can fill their own answers in some empty cells. The other unfilled cells are filled by DLD-VISU. DLD-VISU does not force students to fill all empty cells. If students already master how to create state table, they can directly click "Verify" button to skip the filling step.

4) State Memory Design: This step simply consists in selecting the type of the flip-flops that should be used to implement the state memory. The number of generated FFs depends on the number of states and the state code. Figure 3.22 shows how DLD-VISU generates two D-type flip-flops, JK flip-flops or T flip-flop for the example FSM. The FSM general architecture shown in this figure is displayed to keep overview of the current design step by highlighting the corresponding component. In the following steps, this architecture diagram will not be shown again, for space and focus reasons.

5) Design of Combinatorial Input Logic: The design of the input logic circuit is accomplished in three steps (steps 5, 6, and 7 in the FSM progress chart, see Figure 2.4): In the first step the state equations are derived from the state table. A state equation

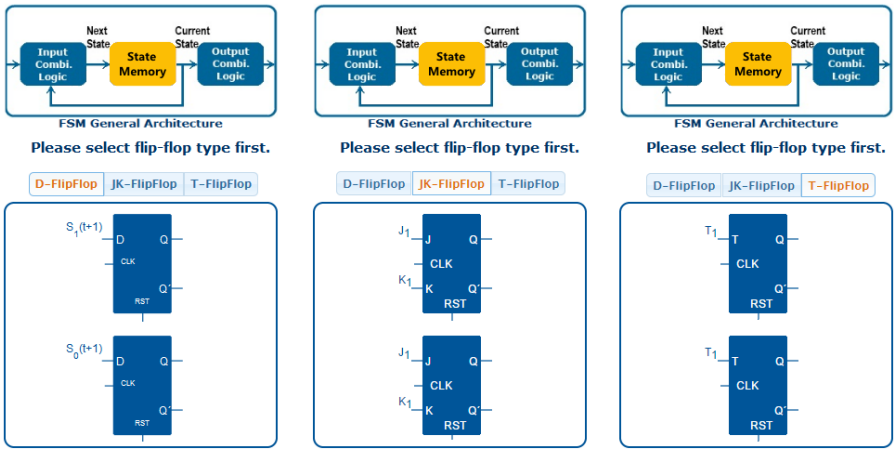


Figure 3.22: State memory design.

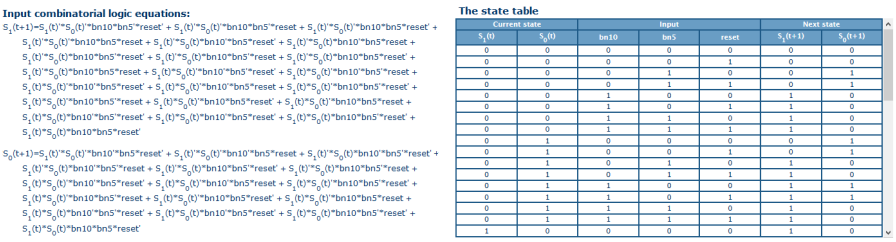


Figure 3.23: Generation of State Equations.

describes the next state as a function of the current state and the input signals, see Figure 3.23. Using the mouse cursor, a table line and the corresponding term in the equations can be highlighted for the sake of traceability. Note that the column related to the output signal is omitted in the table of Figure 3.23. This is because output signals do not affect the input logic. DLD-VISU hides output signal columns in this step for the purpose of abstraction.

In the next step the state equations are minimized, see Figure 3.24. The minimization is performed using the Quine McCluskey algorithm implicitly, i.e., without animation. FSM design is an advanced topic in digital logic design and it is assumed that students at this stage are familiar with the design of combinatorial logic including minimization.

Lastly, the combinatorial input circuit corresponding to the minimized function is displayed gate by gate by highlighting the corre-

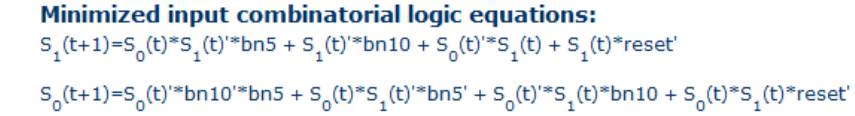


Figure 3.24: Minimization of State Equations.

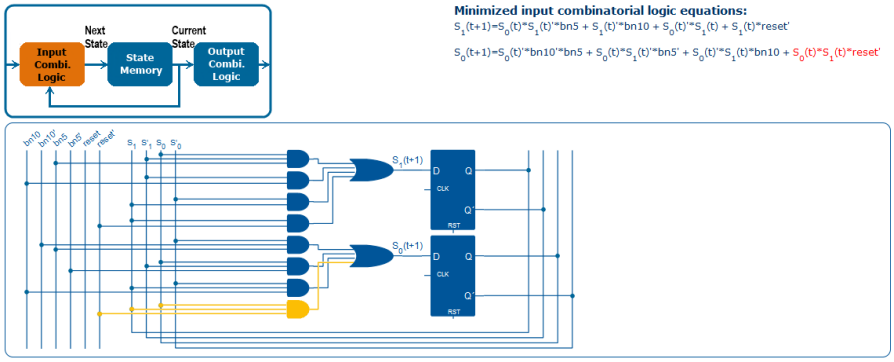


Figure 3.25: Combinatorial Input Circuit.

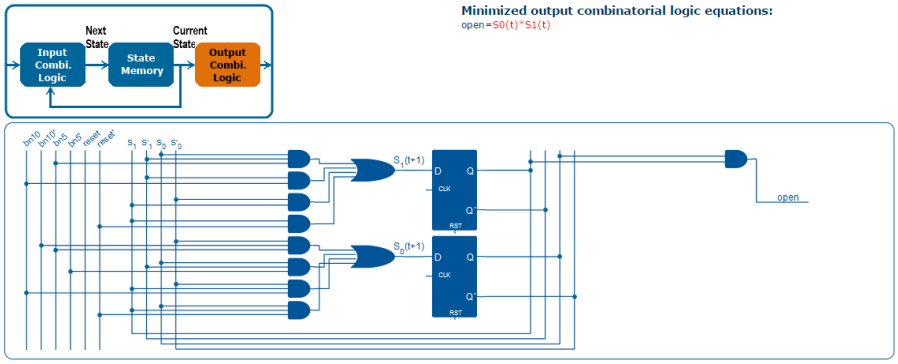


Figure 3.26: Complete Circuit.

sponding term in the minimized state equations for the traceability, see Figure 3.25.

6) Design of Combinatorial Output Logic: The design of combinatorial output logic is visualized in three steps of the progress chart (steps 8, 9, and 10). These steps are highly similar to the construction of the input logic. Therefore, only the final circuit is shown here for brevity, see Figure 3.26 In the first step, the output equations are derived from the state table. An output equation describes an output signal as a boolean function of the current state in Moore machines. In Mealy machines, an output signal is a function of both the current state and the input signals. During deriving the output equations, therefore, DLD-VISU hides the next-state columns in the state table for Mealy machines as well as the input signal columns for Moore machines. In the next step the output equations are minimized. In the last step, the combinatorial output circuit corresponding to the minimized function is displayed gate by gate to obtain the overall FSM circuit.

After students finish running the whole sequential logic design process, DLD-VISU uploads automatically this process to Server through the "history" function introduced in the chapter two. Students can re-

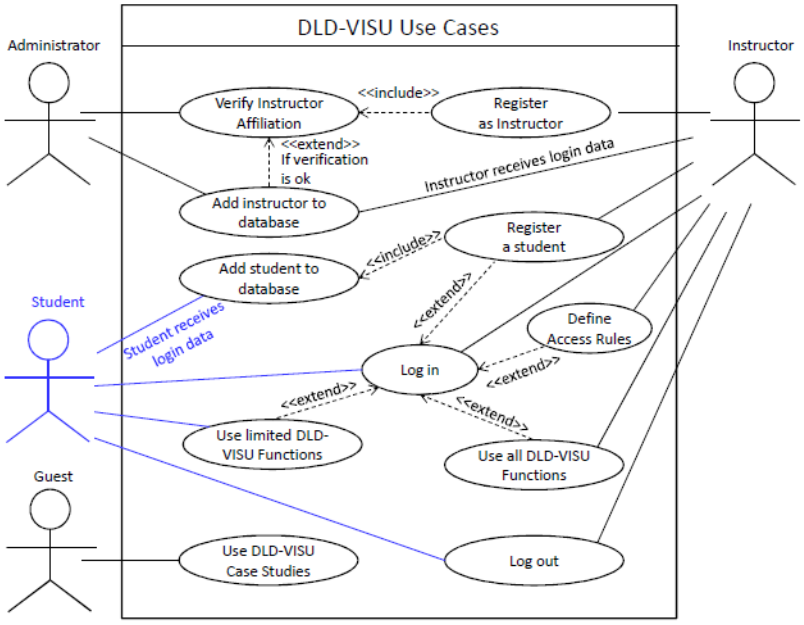


Figure 3.27: DLD-VISU Use-Case Diagram.

peat running this process by clicking "History" menu or download a 10-step animation in PDF format.

3.3 LEARNING MANAGEMENT FUNCTIONS

DLD-VISU is intended to accompany students taking a DLD course. We are aware that some instructors do not welcome a learning technology that enables students to generate solutions directly. Students may use such a technology to solve homework problems and assignments. To tackle this problem, DLD-VISU is enhanced with a special **Access Control System** that differentiates users who are divided into three types: Instructor, Student, and Guest. The use case diagram given in Figure 3.27 illustrates the actions and interactions of these actors as detailed in the following.

An instructor uses the web to register. Besides some personal data, an instructor must enter his professional email address and website. The system administrator verifies the data usually through on-line search and sends a confirmation along with log-in data to the instructor. An instructor can then log in and perform one of three actions:

1) An instructor can register one or more students that attend her or his course using the web. For this purpose the student name and email address must be entered. Upon receiving the request, the system automatically adds the students to the database related to the corresponding instructor and sends log-in data to them.

Topic		Lock Period	
		From	To
Combinatorial Logic	Logic gates		
	Multiplexers	2013-05-01	2013-05-08
	Decoders	2013-05-09	2013-05-12
	Look up tables		
	LUT s+ MUXs		
	K-map minimization		
	Quine-McCluskey minimization		
Sequential Logic	Moore FSM		
	Mealy FSM		
	Binary state coding		
	Gray state coding		
	One-hot state coding		
	D-Flipflop		
	T-Flipflop		
	JK-Flipflop		

Figure 3.28: Access Control Rules Example.

2) To assure that students cannot use DLD-VISU to generate solutions for graded homework problems or assignments, the instructor can lock different topics using a list that we call the Access Control List, see Figure 3.28. The upper entry in the figure, for instance, shows that students cannot use DLD-VISU to design combinatorial logic using multiplexers in the time period from the first to the 8th of May 2013.

A student willing to use DLD-VISU has to send an inquiry to her or his instructor per email. After getting log-in data, students can log in and use the animation tools according to the access control rules defined by the instructor. This indirect registration approach aims at reducing the administrative work which would be needed to verify that some student really attends the course of a specific instructor. Users who wish to use DLD-VISU without registration or log-in can access the tools as guests. A guest, however, can only run the built-in case studies.

4 DLD-VISU BUSINESS LOGIC

In this chapter, the implementations of algorithms in DLD-VISU are illustrated in details. The algorithms are divided into two categories. The first category are the algorithms relating to DLD courses, such as K-Map algorithm, Quine-McCluskey algorithm, and state code algorithm in FSM, etc. They are implemented through Java and run on server side. The other category of algorithms is employed to show animations, such as animation of diverse circuits, animation of the K-Map algorithm, etc. Such algorithms are realized with JavaScript and CSS and run on client side, which means on browser. Except for these two categories of algorithms mentioned above, JavasScript is also used to develop some validation algorithms on front end. They are applied to validate user input and report errors as well as in Self-Assessment system. In 4.1, the whole process of producing DLD-VISU animations is introduced. The algorithms run on server side and on browser side are demonstrated in 4.2 and 4.3, respectively. The validation algorithms are illustrated in 4.4.

4.1 THE PROCESS OF PRODUCING DLD-VISU ANIMATIONS

In this subsection, a concrete example is given to illustrate the process of creating DLD-VISU animations. The Figure 4.1 presents the workflow diagram of the K-Map animation algorithm. The whole process is divided into three steps:

1) The browser sends the validated data to server side.

2) Once server gets the data, it calculates the corresponding results according to the K-Map algorithm as well as all intermediate data. At last, all data are sent back to browser. Note that, not only the resulting data are sent back to browser in this step, but also the intermediate data which are needed for minimization. These data are applied to generate the animation on browser to present the whole process of minimizing the K-Map algorithm. Using animations to show the important intermediate steps of algorithms is the biggest difference between DLD-VISU and other learning software, also embodying the EF theory - transfer experts' mental models to students as good as possible - introduced in Chapter 2.

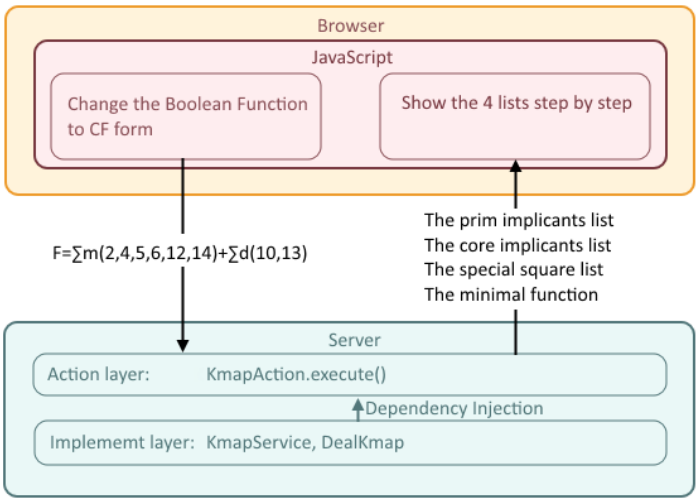


Figure 4.1: The flow diagram of K-Map Animation

3) After the browser obtain the data from server, they produce the corresponding animations in terms of these data and display them to students.

It is worth to mention that the animation of K-Map algorithm is played in the browser with four steps (see the concrete steps in Chapter 3). The synchronous transmission is not adopted here, which means the browser sends a request to server to get demanded data for the current step, once students take a step. Server returns the four groups of data required by the browser at one time. The reasons for this are as follows:

1) The data amount transmitted each time from server to client is pretty small, e.g. a list of prime implicate. Therefore, delivering the results generated through 4 times at one time cannot impact transmitting speed obviously.

2) If the results are transmitted 4 times, server is required to run the same program 4 times, which increases the burden of server. An example is presented here to explain this case. The Browser only need to show all *core implicants* in the second step. However, in order to gain *core implicants*, server needs to run the program of the first step again so that all *prime implicants* can be obtained, because *core implicants* are picked from these *prime implicants*. In this way, the program of the first step is run twice on the server side. Moreover, intermediate results can be also stored in static variables or in databases. But the load of server is enlarged accordingly.

3) Decreasing the frequency of communication between server and browser from 4 times to 1 time is able to reduce the waiting time of students during their operations.

In the following, the specific work flow of each step in details is described.

1) On the browser side: Three distinct means are provided for students to input boolean function on the browser side. When students select K-Map and Truth-Table to input boolean function, the JavaScript programs on the browser side are going to generate a CF format according to students' operation and send it to server subsequently. When students input boolean function in DNF format manually, the JavaScript programs on the browser side are going to verify whether this boolean function conforms to the DNF specification. If not, the system gives students a hint to check their input.

2) On the server side:
   - After struts 2 intercepts the requests from client side, it searches for the specific class — *KmapAction* — within the mapping table of action in *struts.xml*. This class is designed to handle such requests. In DLD-VISU, action and its corresponding class file is usually given a same name. Note that, *kmapAction* objects can be created under normal circumstances. However, Spring is employed to manage the actions, which are created by Struts 2. Therefore, Struts 2 is no longer responsible for the creation of concrete objects rather hands over this job to Spring. In this way, the word "kmapAction" here is not the corresponding class name but the bean id of Spring.
   - Struts 2 delivers the subsequent jobs to Spring. Then, Spring searches for the item whose bean ID is *kmapAction* in its configuration file *applicationContext.xml* and finds the class demanded for creating *kmapAction* objects according to this item. In addition, Spring also injects the objects required by *kmapAction* objects -for example *kmapService* and *dealKmap* -into *kmapAction* objects via DI. The reasons why to use Spring to manage Struts 2 and the concrete implementing methods are presented in Chapter 5.
   - The methods, such as *initKmap*, *findPrimeImplicants*, *findCoreImplicants*, etc., of *kmapService* objects are utilised in the *execute* method of *kmapAction*. The algorithms are explained in next section.
   - According to the results from searching in *struts.xml*, struts 2 sends the minimal function and other intermediate parameters, such as *prime implicants* list, *core implicants* list, and special square list, back to a designated page *kmap_animation.jsp*.

3) On the browser side:

- The JavaScript programs on the browser side get four group data returned by server: *prime implicants* list, *core implicants* list, special square list and minimal function. The special square is the square in the map which contains only one circle. It is the decision basis whether a *prime implicant* is a *core implicant*.
- In accordance with the number of variables in boolean function, a proper K-Map (4x4, 4x2 or 2x2) is drawn in the browser.
- Using JavaScript and CSS to shows the animation of K-Map, the details about how the animation is showed are presented in next section.

The principles of producing animations in DLD-VISU are explained above by demonstrating the animation production course of K-Map. In the following chapter, the specific implementations of the algorithms are introduced.

4.2 THE ALGORITHMS ON SERVER SIDE

The main algorithms on server side are K-Map algorithm, Quine-McCluskey algorithm, state coding algorithm, state table setup algorithm as well as circuit drawing algorithm, etc. The state coding algorithm encodes states based on the encoding chosen by the user. The state table setup algorithm is used to generate the corresponding state table based on the state code and the state diagram. Although this two algorithms are our original algorithm, there is no specific introduction in this chapter because they are relatively simply. The realization process of this 2 algorithms meet the comment of program code . Among these algorithms, K-Map algorithm and circuit drawing algorithm are picked and described in detail. The K-Map algorithm is improved so that it is more suitable for students to learn by themselves. The circuit drawing algorithm is our original algorithm and complicated.

4.2.1 *K-Map Algorithm*

So far, a lot of on-line [52] [51] or off-line [36] programs relating to the K-Map algorithm can be found in internet. However, most of them are only able to show the minimal results rather than the minimizing processes. In this way, students cannot understand and learn the K-Map algorithm in a better way. Besides, there also exist two further questions:

1) As presented in Figure 4.3, 4.4, and 4.5, most programs only provide one minimal result. But the minimal result may not be

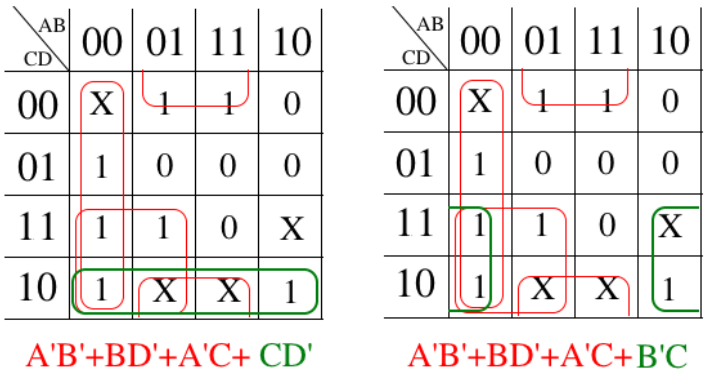


Figure 4.2: An Example for 2 different minimal results of the same function.

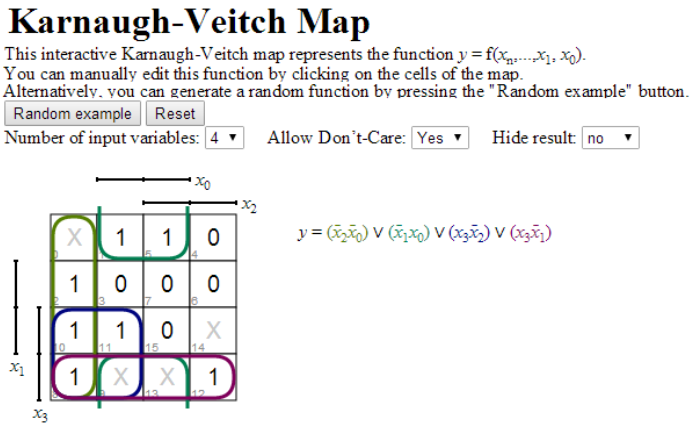


Figure 4.3: The screenshot of online K-Map tool "Karnaugh-Veitch Map".[52]

unique. Figure 4.2 shows an example, the results are A'B'+ BD' + A'C + CD' or A'B'+ BD' + A'C + B'C. If students perform their own minimization and get a different result from that obtained by using on-line programs, they could be confused or even acquire an incorrect understanding of the K-Map algorithm. As a matter of fact, both two results might be correct.

2) Some programs might present wrong results because of design flaws in algorithms. As shown in Figure 4.4, **Karnaugh-Map explorer 2.0** could lead to wrong answers (the *minterm* with red border is redundant), when multiple 2x2, 4x1, and 1x4 circles are contained in a K-Map. Because the sequence of drawing circles by these programs is not flawless.

The K-Map algorithm is a method of simplifying boolean algebra functions. The required boolean functions are transferred from DNF form or canonical form onto a two-dimensional grid, each cell position represents one combination of input conditions, while each cell value represents the corresponding output value. The number of rows

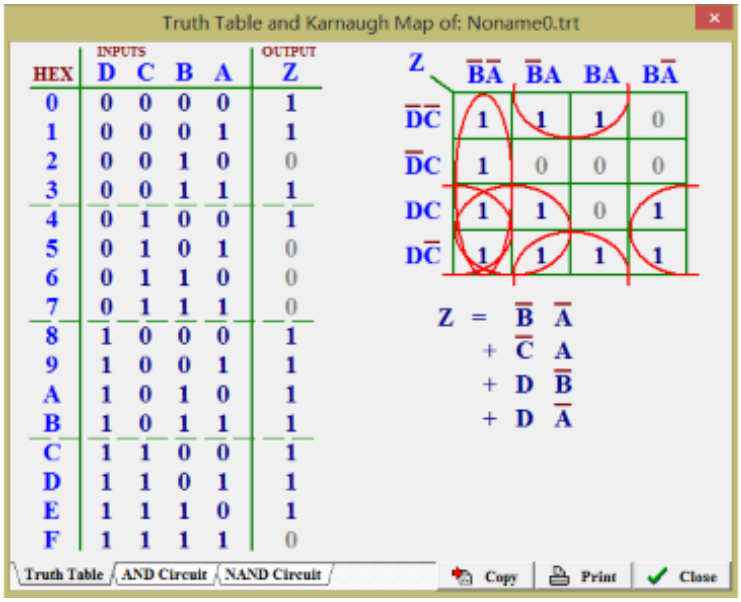


Figure 4.4: The screenshot of offline K-Map tool "WinLogiLab".[36]

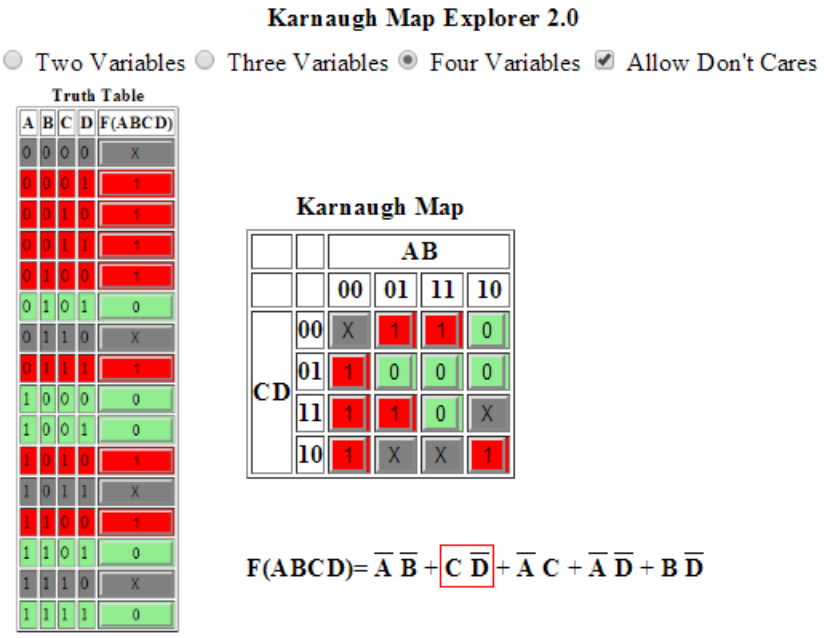


Figure 4.5: The screenshot of online K-Map tool "Karnaugh-Map Explorer 2.0".[51]

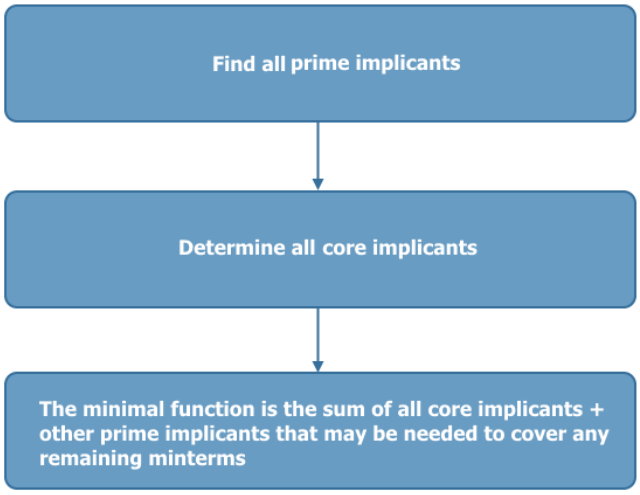


Figure 4.6: K-map Minimization Procedure.

and columns in the K-Map is determined by the variables of the function. There are 2^n cells in a K-map, n is the number of variables of the function. The *minterms* (minimal terms) for the final expression are found by encircling groups of 1s in the map. The Algorithm have following rules:

1) *Minterm* groups must be rectangular.

2) *Minterm*groups must have an area that is a power of 2.

3) *Minterm* groups rectangles should be as large as possible without containing any 0s.

4) A *minterm* can be covered by more than one rectangular.

Before introducing the process of K-Map algorithm, some definitions are explained first.

Prime implicant: A product term resulting from combining the maximum possible number of adjacent squares.

Core implicant: A prime implicant covering a *minterm*, which is not covered by any other *prime implicant*.

Figure 4.6 shows the procedure of K-Map:

The first step of the K-Map algorithm is finding all *prime implicants*, in other words, find all biggest rectangles. The so-called biggest rectangles refers to the rectangles which owns the largest number of adjacent *minterms*. The algorithm of drawing rectangles by *Karnaugh-Map explorer* is demonstrated in Code 4.1. It can be perceived by observing the code that Karnaugh-Map explorer searches for the rectangles including 16, 8, 4, and 2 adjacent *minterms* in a specific order 4x4 $\Rightarrow$ 4x2 $\Rightarrow$ 2x4 $\Rightarrow$ 1x4 $\Rightarrow$ 4x1 $\Rightarrow$ 2x2. Yet the problem is that the number of adjacent items in both 4x2 and 2x4 rectangles are 8. Additionally,

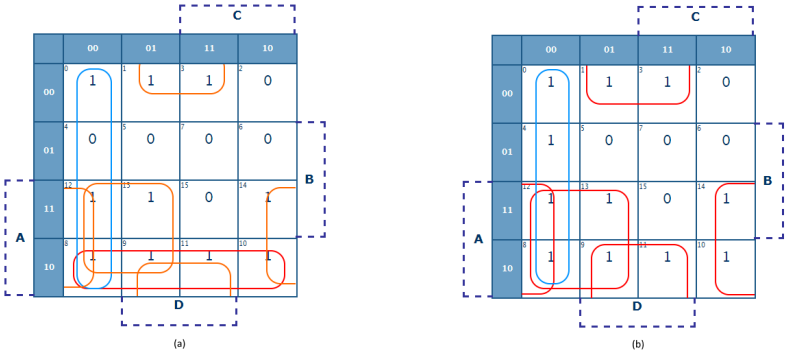


Figure 4.7: An Example of finding different prime implicants according to different order.

the count of adjacent entries in the rectangles 1x4, 4x1, and 2x2 is also same, i.e. 4. The results are usually different according to the sequences in which programs deal with these rectangles of distinct kinds. More specifically, handling 1x4 rectangles before 2x2 rectangles leads to a different result from that by handling 2x2 rectangles before 1x4 rectangles. Figure 4.7 presents a relevant example. In 4.7 (a), the program searches for adjacent *implicants* in the order 4x1 ⇒ 1x4 ⇒ 2x2, finding 5 *implicants* in this graph. While in 4.7 (b) 2x2 rectangles are considered first. After that, the four *minterms* in the bottom row are already visited. They are not checked and taken into consideration for 4x1 rectangles any more, which results in one implicant less.

```
SearchRect(4, 4, true, Rects, true);
SearchRect(4, 2, true, Rects, true);
SearchRect(2, 4, true, Rects, true);
SearchRect(1, 4, true, Rects, true);
SearchRect(4, 1, true, Rects, true);
SearchRect(2, 2, true, Rects, true);
```

Code 4.1: A part of the algorithm of "Karnaugh-Map explorer" [51].

In the above described example, **Karnaugh-Map explorer** generates the result directly in terms of *prime implicants* rather than *core implicants*. What if *core implicants* are applied? Is it possible to avoid the errors mentioned above? In Figure 4.7 , it is easy to tell even though *core implicants* are determined, the sequence of choosing *prime implicants* does influence the result. Because there still exist some *minterms* that are not covered by *core implicants*. The first conclusion is drawn through these two examples — **the order of checking rectangles that own the same number of adjacent minterms does have impact on results.**

In the example shown in Figure 4.8 , one K-Map generates two distinct results, both of which are correct. When students carry out

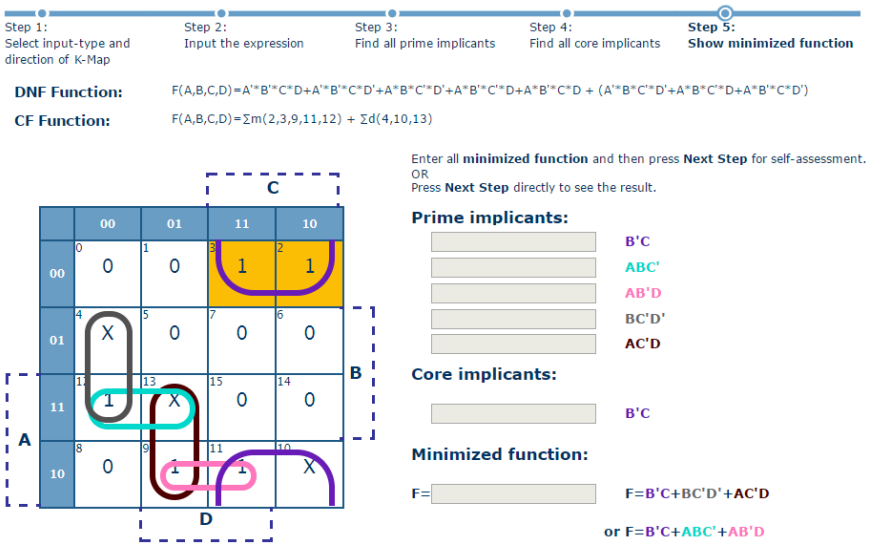


Figure 4.8: A K-Map example for 2 results.

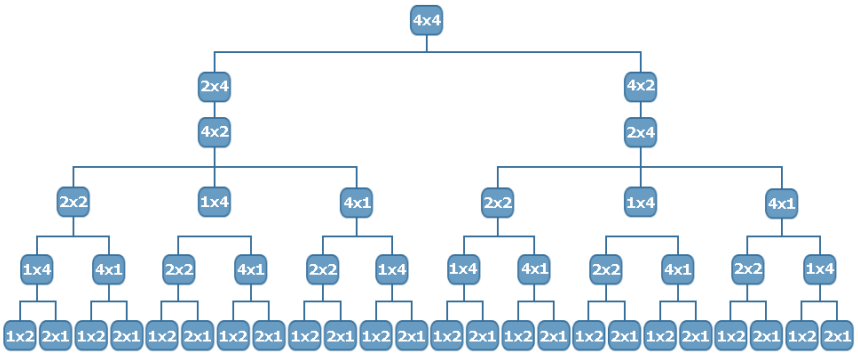


Figure 4.9: The 24 orders of selecting the prime implicants.

Self-Assessment, each answer is right. Now we are able to reach the second conclusion that **programs must get all possible and correct answers through calculation.**

In accordance with the two conclusions above, we optimize the K-Map algorithm. After all *core implicants* are found, we put the remaining *prime implicants* into mini-function in different orders. As illustrated in Figure 4.9 , there are 24 distinct sequences. However, after observing it turns out that the sequences of 2x4 and 4x2 have no impact on results. Thus, only 12 orders needs to be considered, which means at most 12 answers can be gained. Subsequently, we compare the number of *minterms* in each answer and pick the answers that have the least *minterms*. Then, the quantity of variables are compared in these selected results further. Finally, the result that has the least *minterms* and the least variables is the correct answer. If the count of such results is more than one, all of them are treated as correct. The Pseudo Code is demonstrated in Code 4.2.

```
Input: The set of 1s oneArray; The set of don't cases xArray;
    The number of variables n;
Output: The mimimal function F.
1. Initializing a K-Map map[i][j] according to the number of
    variables
     if n = 2
       i = 1, j = 1;
     else if n = 3
       i = 2, j = 1;
     else if n = 4
       i = 2, j = 2;
2. Inserting the list of 1s and don't cases into K-map, 2 denote
    don't care.
3. Finding all prime implicants P according to following order.
  finding the rectangles r, which the size is 4*4; P.push(r);
  finding the rectangles r, which the size is 4*2; P.push(r);
  finding the rectangles r, which the size is 2*4; P.push(r);
  finding the rectangles r, which the size is 1*4; P.push(r);
  finding the rectangles r, which the size is 4*1; P.push(r);
  finding the rectangles r, which the size is 2*2; P.push(r);
  finding the rectangles r, which the size is 2*1; P.push(r);
  finding the rectangles r, which the size is 1*2; P.push(r);
  finding the rectangles r, which the size is 1*1; P.push(r);

4. Finding all core implicants.
  for each minterm m in map do
    if m is covered only 1 rectangle r.
    F.push(r);
    P.pop(r);
  end for

5. Finding minimal function
  5.1 Finding rest prime implicants RP that may be needed to
      cover any remaining minterms.
  orderList = {{22,14,41,21,12},
         {22,14,41,12,21},
         {22,41,14,21,12},
         {22,41,14,12,21},
         {14,22,41,21,12},
         {14,22,41,12,21},
         {14,41,22,21,12},
         {14,41,22,12,21},
         {41,22,14,21,12},
         {41,22,14,12,21},
         {41,14,22,21,12},
         {41,14,22,12,21}
        };
  for i = 0 to orderList.length do
    for j = 0 to orderList[i].length do
      finding the rectangles r in P, which the size is
          orderList[i][j];
      RP[i].push(r);
```

```
48      end for
49    end for
50
51    for i = 0 to RP.length do
52      for j = 0 to RP[i].length do
53        calculate the number of rectangles nr[i] in RP[i], and
              calculate the number of variables $nv[j] for each
              rectangles.
54        if $nr[i] <= $nr[i - 1] and nr[j] <=nr[j - 1]
55          result = RP[i];
56      end for
57    end for
58    F.push(result);
59    return F;
```

Code 4.2: The pseudo-code for K-Map algorithm.

4.2.2 *Quine–McCluskey algorithm*

For the boolean functions with few variables, employing K-Map to perform minimization visually is very intuitive and convenient. However, K-Map becomes very large when the number of variables is more than 5, increasing the difficulty for users to find out *prime implicants*. Thus, the advantage - intuition - does not exist any more. Besides, K-Map algorithm is a "visual" algorithm for which it is inappropriate to make use of computers to carry out auxiliary calculations. Because such a "visualization" just provides the convenience for users to observe with naked eyes (put the *minterms* whose difference is 1 in a table one by one). But computers do not need such a sorting during computing. Hence, there is not any commercial software that adopts K-Map to minimize boolean functions. The Quine-McCluskey algorithm was developed by W.V. Quine and extended by Edward J. McCluskey. It is functionally identical to K-Map, but the tabular form makes it more efficient to be applied in computer algorithms.

Quine–McCluskey algorithm is divided into two steps:

1) Finding all *prime implicants* of the function using a *minterm* table.

2) Use those *prime implicants* in a *prime implicant* chart to find the *core implicants* of the function, as well as other *prime implicants* that are necessary to cover the function.

The introduction of the algorithm is shown as follows.

1) All *minterms* that evaluate to one are first placed in a *minterm* table. Don't care terms are also added into this table, so they can be combined with *minterms*.

2) One can start combining *minterms* with other *minterms*. If two terms vary by changing only a single digit, that digit can be replaced with a dash indicating that the digit doesn't matter. Terms that can't be combined any more are marked with an asterisk (*). For instance, -110 and -100 can be combined, as well as -110 and -11-, but -110 and 011- cannot.

3) A *core implicant* table is constructed. Along the side, the *prime implicants* that have just been generated by step 3 are filled, and there are the *minterms* specified earlier on the top. The terms which do not much matter are not placed on top because they are not necessary inputs.

4) We run along the top row and have to look for columns with only one "X". If a column has only one "X", this *prime implicant* is a *core implicant*.

5) If the *core implicants* do not cover all *minterms*, other *prime implicants* may be needed to cover any remaining *minterms* in this case.

4.2.3 *FSM diagram algorithm*

A JSON object is generated according to a FSM diagram drawn by students.

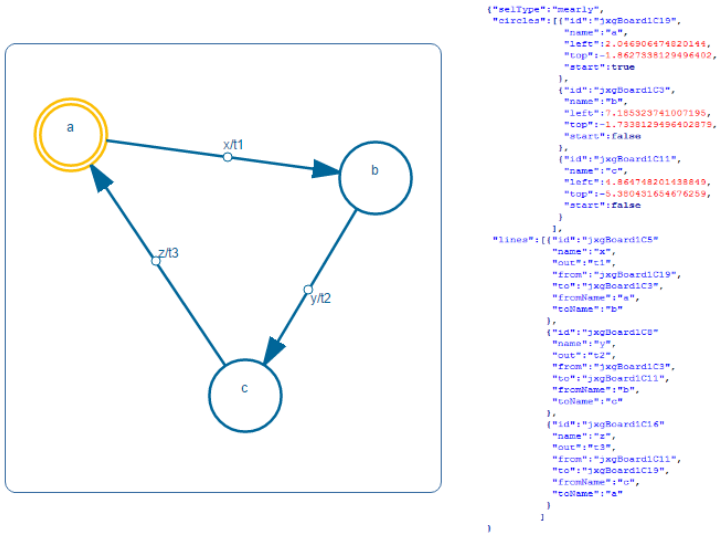
JSON is a simple data format which is lighter than XML. Douglas Crockford first specified and popularized the JSON format [22]. It is built on two structures [43]:

1) A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.

2) An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

Code 4.3 shows the details of FSM diagram.

```
FSM:{
  "selType":String, //FSM type: moore or mearly
   "circles":Array, //states
   "lines":Array //translates
   }
Circles:{
  "id":String,
  "name":String,
  "left":Float,  //the x-coordinate of this state
  "top":Float,  //the y-coordinate of this state
  "start": Boolean //start state or not
  }
Lines{
```




```
{"selType":"mearly",
 "circles":[{"id":"jxgBoard1C19",
             "name":"a",
             "left":2.046906474820144,
             "top":-1.8627338129496402,
             "start":true
            },
            {"id":"jxgBoard1C3",
             "name":"b",
             "left":7.165323741007195,
             "top":-1.7338129496402878,
             "start":false
            },
            {"id":"jxgBoard1C11",
             "name":"c",
             "left":4.864748201438849,
             "top":-5.380431654676259,
             "start":false
            }
           ],
 "lines":[{"id":"jxgBoard1C5"
           "name":"x",
           "out":"t1",
           "from":"jxgBoard1C19",
           "to":"jxgBoard1C3",
           "fromName":"a",
           "toName":"b"
          },
          {"id":"jxgBoard1C8"
           "name":"y",
           "out":"t2",
           "from":"jxgBoard1C3",
           "to":"jxgBoard1C11",
           "fromName":"b",
           "toName":"c"
          },
          {"id":"jxgBoard1C16"
           "name":"z",
           "out":"t3",
           "from":"jxgBoard1C11",
           "to":"jxgBoard1C19",
           "fromName":"c",
           "toName":"a"
          }
         ]
}
```


Figure 4.10: A FSM diagram and corresponding JSON data.

```
14  "id":String,
15  "name":String,
16  "out":String,
17  "from":String, //the id of source state.
18  "to":String,  //the id of target state.
19  "fromName":String, //the name of source state.
20  "toName":String //the name of target state.
21  }
```

Code 4.3: The JSON code for FSM diagram.

The JSON object is encoded into a JSON string and the string is delivered to server. Figure 4.10 shows an example that a JSON string is generated based on the left FSM diagram.

The algorithms of state code and state table of FSM are pretty simple. Therefore, no further explanation is performed here. The concrete implementation is already presented in program documentation.

4.2.4 *The circuit diagram algorithm*

The circuit diagram algorithm is mainly used to calculate the size and location of the components and the wires in a circuit diagram. In order to make it convenient for students to learn, we apply a 2-layer structure which is easy to be understood to design circuits, i.e. each *minterm* of combinatorial logic equations is achieved through an AND-gate and the *minterms* are combined via a OR-gate. The number of input of each AND-gate is equal to the number of variables of its *minterm*. The number of input of each OR-gate is equal to the number of *minterms* of a combinatorial logic equation. In this way, the input signal of AND-gates and OR-gates is not fixed, which brings the

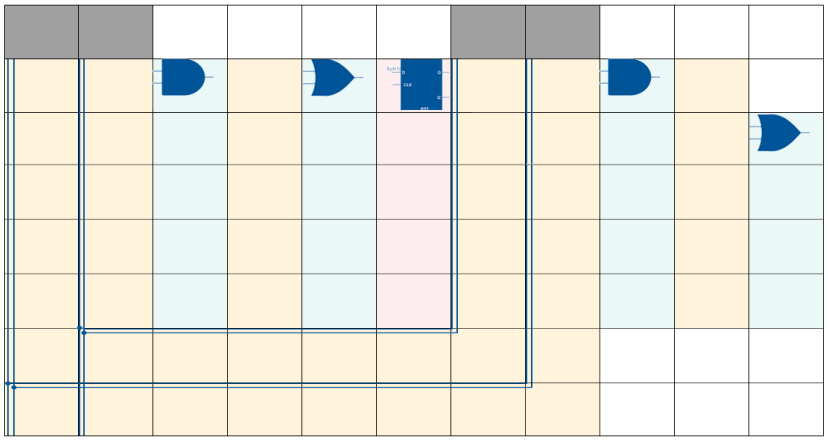


Figure 4.11: The layout of 2-layer circuit.

benefit that the number of logic layers of input/output circuits is set to be two and the convenience for students to observe the correlation between circuit diagrams combinatorial logic equations.

The 2-layer structure is adopted to design circuits. Hence, the basic structure of circuits is fixed, no matter how complex the circuits are. So as to compute the concrete position of each component, the layout of whole circuit diagrams is designed firstly. The *DIV* element in which a circuit diagram exists is partitioned into 8 rows and 10 columns. The specific layout is presented in Figure 4.11. The light yellow areas are for wiring. AND/OR-gates are placed in the baby blue areas. Flip-flop is put in the light pink areas. The grey areas on the top of the figure are utilized to display wire names, such as so, so', reset, etc. The concrete illustration is displayed in Chapter 3.

The algorithm computes the quantity of AND/OR-gate as well as flip-flop that are needed in circuit diagrams firstly. The number of flip-flops is equal to the number of equations. The number of AND-gates is equal to the number of *minterms* in all equations. The number of OR-gates is equal to the number of equations, which include more than 1 *minterms*.

After that, the heights of AND/OR-gate as well as flip-flop are calculated according to the computed quantity, making sure that all AND-gates, OR-gates, and flip-flops are able to be uniformly shown in the defined areas. Because the ratio of width and height of these electrical components is settled, we can calculate their widths after acquiring their heights. In order to avoid the case that electrical components are excessively big when circuit diagrams are way too simple, we set a maximum height for each component. When the calculated height is bigger than the maximum value defined by the system, DLD-VISU takes the system default maximum value as the height of the current component. The system default maximum values are always calculated on the basis of the resolution of users' current screen, which guarantees that DLD-VISU is able to display complete circuit

diagrams for different screen resolutions. code 4.4 demonstrates the formula of computing the height and width of flip-flop, where *ffSpaceY* indicates the distances between 2 flip-flops, *pageHeight* represents the height of page in the browser, and *ffCount* represents the amount of flip-flop in circuit diagrams.

```
1 double ffHeight = ffDefaultHeight > (0.625 * pageHeight /
      ffCount - ffSpaceY)
2             ? ffDefaultHeight : (0.625 * pageHeight / ffCount -
                  ffSpaceY);
3 double ffWidth = ffHeight * 0.75;
4 double ffTop = 0;
5 double ffLeft = 6 - 1.25 * ffWidth; // 6 is the x-coordinate of
      the terminal point of the output line,
6                             // the length of input and output
                                  line are 0.125 * ffWidth.
```

Code 4.4: computing the position of electronic components.

After the calculation of the position and size of all components is accomplished, server sends the coordinates of all components in string format back to client at one time. "@" is applied in this string which consists of coordinates to separate them. Each *AndOrInfo* is on behalf of the information about the AND/OR-gate as well as wires of a *minterm*. For example, a function includes four *minterms* and then server returns four *AndOrInfo*. In accordance with the distinct components that are contained, the formats of *AndOrInfo* are also different. There are five formats in total. Table 4.1 introduces the specific content of every group of information.

4.3 THE ALGORITHMS ON BROWSER SIDE

Such algorithms are mainly employed to display animations and interact with users. A great amount of AJAX techniques are employed in these algorithms in order to improve user experience. A detailed explanation to AJAX technique is given in 5.5.

4.3.1 *The animation algorithm for K-Map*

In the research relating to E-Learning theories introduced in Chapter 2, we find that studying actively can improve the quality of students' self-learning to a great extent. In order to provide students opportunities to think independently, we do not show minimizing processes and results to students at a time rather in multiple steps. In this way, students can do their own calculation before they see the results directly, which advances students' participation. The browser used by students receive the four groups of parameters from server in terms

Parameter name	Description	Content
gateSize	The size of AND/OR gate	andWidth # andHeight # orWidth # orHeight @
ffInfo	The count, position and size of flip-flops	ffType # ffCount # $ffTop_1$ % ... % $ffTop_n$ # ffLeft # ffWidth # ffHeight @
ffWireInfo	The position of the extension wires of all flip-flops	ffWireTop # ffWireLeft # $ffWireHeight_1$ # $ffWireHeight_2$ # ffWireWidth # ffWireSpaceX # ffWireSpaceY@
inputWireInfo	The position and the name of the extension wires of all input signals	$inputWireName_1$ % ... % $inputWireName_n$ # inputWireTop # inputWireLeft # $inputWireHeight_1$ # $inputWireHeight_2$ # inputWireWidth # inputWireSpaceX # inputWireSpaceY @
AndOrInfo	An AND-gate, an OR-gate, and a wire between them	'O' orTop # orBeginX # orBeginY # orEndX # orEndY # 'A' andTop # $andBeginX_1$ % ... % $andBeginX_n$ # $andBeginY_1$ % ... % $andBeginY_n$ # andEndX # orIndex@
AndOrInfo	An AND-gate and the output wire of this AND-gate.	orBeginX # orBeginY # orEndX # orEndY # 'A' andTop # $andBeginX_1$ % ... % $andBeginX_n$ # $andBeginY_1$ % ... % $andBeginY_n$ # andEndX # orIndex @
AndOrInfo	An OR-gate and the input wire of this OR-gate.	'O' orTop # orBeginX # orBeginY # orEndX # orEndY # andBeginX # andBeginY # andEndX # orIndex @
AndOrInfo	A wire, connect to 2 existed component	orBeginX # orBeginY # orEndX # orEndY # andBeginX # andBeginY # andEndX # orIndex@

Table 4.1: The form of every group of component.

of which the browser creates the corresponding elements and hide them. When students trigger a relevant step, the animation elements of this step are just shown.

We sketch the creating process of animation in the step - show *prime implicants*. The same manner is applied for the steps: a) show *core implicants* and b) show minimal function.

1) The list of *prime implicants* includes an array of rectangles and each rectangle contains four values which indicate the coordinates, the length and the width of the top left point. JavaScript programs invoke the *drawPrimImplicants* method to draw a rectangle in the corresponding position. DLD-VISU utilizes *DIV* elements with transparent background and colorized border to represent the rectangles. We do not use Scalable Vector Graphics (SVG) elements here, because the visualizing effect using *DIV* elements is not worse than using SVG elements, since CSS 3.0 is introduced in DLD-VISU. Besides, the consumption of browser resources by *DIV* elements is way less than SVG elements. Furthermore, *DIV* elements are the standard elements of HTML DOM, which means it is more convenient to set size and position for them than for SVG elements.

2) Invoking the *getEquation* method and getting the corresponding *prime implicant* of each rectangle through the array of rectangles. Putting each *prime implicant* into a *DIV* and setting the color of this *DIV* equal to the color of corresponding rectangles.

3) Hiding the *DIV* objects created above. When students click the button "Show Prime Implicants", the *showPrimImplicants* method is triggered. Then, each *prime implicant* as well as its corresponding rectangle are displayed on the screen in turn, which is convenient for students to observe. This embodies the principle Traceability which is described in chapter 2.

4) When students click the "Show Prime Implicants" button, the presentation of *prime implicants* and their rectangle is divided into multiple steps separated by 1 second and animated. Thus, there is no need for students to click "Next Step" continuously so that they can actually concentrate on the algorithms.

4.3.2 *The animation algorithms for FSM*

In the whole process of presenting the FSM algorithm, the AJAX technique is used for the 9 steps (from step 2 to step 10) to update the content on one page asynchronously. In each step, users send one or multiple requests to server and server sends the results back to the callback functions of AJAX on the browser. These functions are respon-

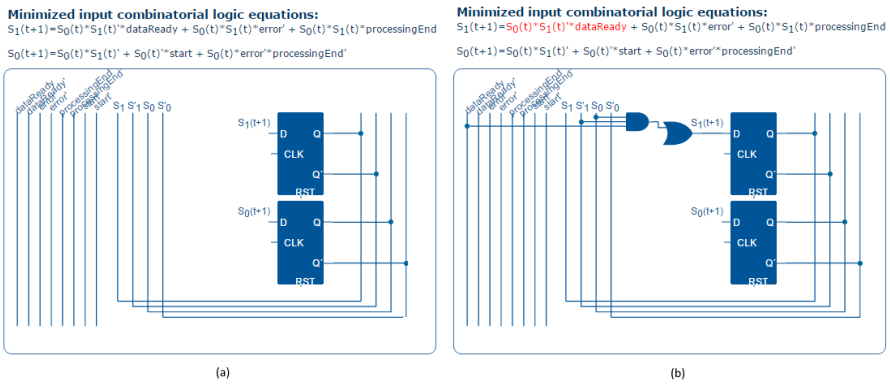


Figure 4.12: An example of the animation of generating a circuit.

sible for displaying the returned results in a proper position on the current page without refreshing the page.

In contrast to other E-Learning software of circuit syntheses, DLD-VISU does not present whole circuit diagrams for students at a time. At the beginning, only flip-flops and their extension wires on output side as well as the vertical extension wires of input signals are shown on a screen, as presented in Figure 4.12 (a). The elements are painted in the board according to 4 sets of data (gateSize, ffInfo, ffWireInfo and inputWireInfo).

Every time students click the "Next" button, the *minterms* of combinatorial logic equations on the top of the screen are highlighted in turn. At the same time, the components and their wires that are corresponding to the highlighted *minterms* are added in the circuit diagram below, as illustrated in Figure 4.12 (b).

4.4 THE VALIDATION ALGORITHMS ON FRONT-END

Such algorithms are divided into two categories. One category is to validate the initial parameters input by students, e.g. to validate whether the boolean function input by students conforms to the DNF format. For this purpose, we apply JavaScript regular expressions on front-end. The regular expressions utilized to validate DNF and CNF are listed in Code 4.5. The other category is to check if the students' answers are correct while performing self-assessment. The algorithms in this category mainly include two functions. On one hand, they format students input. Then, they compare students input with the correct answers applied by the system and store the comparing results in database as well as show them to students.

```
1 function checkDNormalInput(dNormalForm){
2   dNormalForm = dNormalForm.toLowerCase();
```

```
  var pattern =
      /^[A-Z]'?(\*[A-Z]'?)*(\+[A-Z]'?(\*[A-Z]'?)*)*$/gi;
  return pattern.test(dNormalForm);

}
function checkKNormalInput(kNormalForm){
  kNormalForm = kNormalForm.toLowerCase();
  var pattern =
      /^\([A-Z]'?(\+[A-Z]'?)*\)(\([A-Z]'?(\+[A-Z]'?)*\))*$/gi;
  return = pattern.test(kNormalForm);
}
function checkNNormalInput(nNormalForm){
  nNormalForm = nNormalForm.toLowerCase();
  var pattern =
      /^([A-Z]'?(\*[A-Z]'?)*(\+[A-Z]'?(\*[A-Z]'?)*)*\+)*(([A-Z]'
  ?\*)+\(([A-Z]'?(\+[A-Z]'?)*)\)(\*[A-Z]'?)*|([A-Z]'?\*)*\((
  [A-Z]'?(\+[A-Z]'?)*)\)(\*[A-Z]'?)+)(\+[A-Z]'?(\*[A-Z]'?)*(\
  +[A-Z]'?(\*[A-Z]'?)*)*)*$/gi;
  return = pattern.test(nNormalForm);
}
```

Code 4.5: Regular grammar of validating the DNF and the CNF input.

5 IMPLEMENTATION OF DLD-VISU

In the process of developing DLD-VISU, different programming languages and technologies was used. On the server side, Java is utilized to implement the diverse algorithms in the DLD courses and JSP technology is employed to interact with the client. On the page side, *SVG* and *Vector Markup Language (VML)* is adopted to generate the required graphs as well as JavaScript and *AJAX* techniques is used to control these generated graphs in order to create animations and realize the interaction with users. In this chapter, these techniques are introduced and the specific implementation process of DLD-VISU are described. Section 4.1 demonstrates the applied programming languages. Section 4.2 presents the frame structure of the system. The composite framework SSH2 and its sub-frameworks are introduced in section 4.3 and its concrete application in the project DLD-VISU is explained in section 4.4. The front-end animation technology used in the system is shown in section 4.5 and section 4.6 specifies the realization methods of the animation algorithms adopted in the DLD-VISU.

5.1 SELECTION OF THE SYSTEM DEVELOPMENT LANGUAGES AND DATABASE

So far, the most common web development technologies include the following: ASP.net, PHP, JSP, Perl, Python, etc.. ASP.net is a web application framework which is developed on the basis of the .net framework of Microsoft and provides users with the function of building powerful web applications or services for enterprises. Nowadays, the development languages of ASP.net are only C# and BASIC.net. Although ASP.net itself is free of charge, the servers on which ASP.net is deployed are not cheap, leading to pretty high development costs.

Due to the flexibility and excessively redundant syntax of the scripting languages like Perl and Python, the code written in such languages is elusive and difficult to be maintained. A Perl 5 tutorial book written by Randal L. Schwartz [93], in the first chapter of which he states: "Yes, sometimes Perl looks like line noise to the uninitiated, but to the seasoned Perl programmer, it looks like check summed line noise with a mission in life." Considering that DLD-VISU platform needs to be upgraded continuously, we decided not to use Perl and Python. Because of the above factors, we only put PHP and JSP as the candidate programming languages for DLD-VISU.

	1000×1000	10000×10000	MySQL	Oracle
PHP	5m	86m	85ms	69s
JSP	13ms	1.34m	260ms	13s

Table 5.1: Performance comparison between PHP and JSP.

PHP is a server scripting language embedded in HTML. It massively cites the syntax of C and Perl and combines its own characteristics, making web developers able to create dynamic web sites very quickly. PHP is open-source. Working together with Apache and MySQL which are also free can rapidly build dynamic websites, e.g. the famous Wikipedia which is just developed through PHP and MySQL. The current host systems leased by most network providers are equipped with free Apache + PHP + MySQL.

Though JSP is also a scripting language, it has essential differences with PHP and ASP. Both PHP and ASP are interpreted and executed by the language engine, while JSP is compiled into Servlet first and then interpreted and executed by Java Virtual Machine. The compilation is only performed at the first time users request the JSP page. In this way, it is widely believed that the execution efficiency of JSP is higher than both PHP and ASP.

To compare the execution efficiency of PHP and JSP, PHP and JSP are tested respectively on a single laptop. The test consists of 4 parts: 1000×1000 times arithmetic operation, 10000×10000 times arithmetic operation, 100 times database access (MySQL) and 100 times database read/write (Oracle). The corresponding results are presented in Table 5.1.

In the aspects of both code execution speed and database read / write speed, JSP is better than PHP. Only in the test aiming at MySQL database, JSP falls behind PHP. The reason for this consequence is that PHP optimizes the operation on MySQL database and provides a set of functions dedicated to access MySQL database. Whereas JSP makes use of JDBC to access database and has not done any optimization for the operation on MySQL.

Other than the fast execution velocity, the biggest advantage of JSP is the high development efficiency. Users can not only embed code into web pages directly but also invoke the Servlet running on server. However, such web architectures are not easy to maintain and cannot meet the requirements of large-scale applications because of putting the code of business logic and page display together. Instead of them, the web architecture based on MVC is put to use. MVC is a architectural patten, the main idea of MVC is to divide an application into three parts: model, view, and controller. The web architectures that take advantage of MVC can loose the coupling among different parts and separate business logic and page as well as data. In this way,

when one module changes, the others can still run. Therefore, the web architectures on the basis of MVC fit better in the trend of web application development.

PHP and JSP can be achieved MVC by using the third-party framework. But the flexibility or scalability of the third-party MVC framework of PHP is not as good as the framework of JSP. For example, Laravel is a free, open-source PHP web framework, created by Taylor Otwell and intended for the development of web applications following the MVC architectural patten. But the installation of Laravel is cumbersome and unfriendly, and the speed of development is slow.

Due to the very good support for MVC architectural pattern and faster execution velocity than PHP, JSP is selected to develop websites finally.

MySQL MySQL has become the most popular open source database because of high performance, low cost and good reliability.

MySQL is chosen to use for DLD-VISU because of it's high performance and it is cost free. It is an open-source relational database management system, it was created by a Swedish company, MySQL AB, founded by David Axmark, Allan Larsson and Michael Monty Widenius. MySQL is the most popular open source database at present. Figure 5.1 shows all MySQL tables of DLD-VISU. The fields with a golden key indicate the primary keys of the tables and the fields with a red rhombus represent the foreign keys. Table 5.2 introduces the role of each table. The detail of the fields and the constraints is specified in the appendix A.

5.2 MVC ARCHITECTURAL PATTERN

Trygve Reenskaug introduced MVC first in the 1970s [87]. Glenn E. Krasner and Stephen T. Pope express MVC as a general concept in 1988 [54]. MVC is an architectural design pattern which does not introduce new functionality. It just guides developers to improve the architecture of applications so that model and view of applications are able to be separated, leading to better efficiency of development and maintenance.

In MVC architectural pattern, an application is divided into *Model*, *View*, and *Controller*. The *Model* part comprises business logic and business data. The *View* part encapsulates input/output format of applications, which is also known as GUI. The *Controller* part is in charge of coordinating *Model* and *View*. According to users' request, it decides which the *Model* to handle business and the *View* as the response to users.

The duties of these three parts of MVC architectural pattern are very specific. The *Model* is responsible for output content and the *View* for output format. Usually, there is no logic realization in the *View* and the *Model* does not rely on the *View*. A certain *Model* may

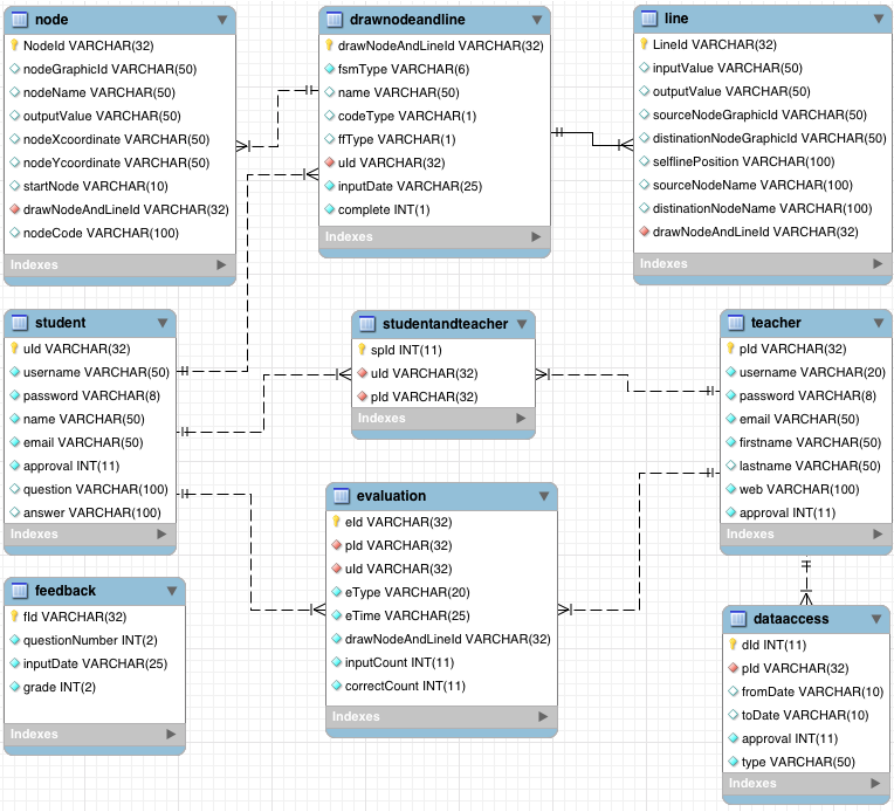


Figure 5.1: The database of DLD-VISU.

be displayed in different ways, i.e. in the different *Views*. Meanwhile, one *View* can also present multiple *Models*.

Because of mutual separation of 3 parts, each part can change independently without bringing influence on the others, which greatly raise the flexibility and re-usability of applications.

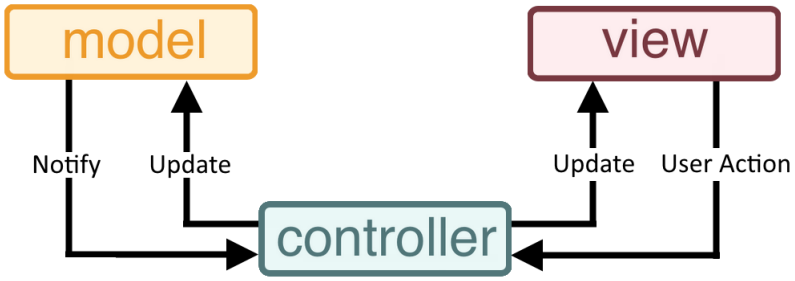


Figure 5.2: Web MVC architectural pattern.

Figure 5.2 shows the interactions between different parts of MVC architecture pattern. In the following, the interaction process is explained according its working sequence.

1) Firstly, a *View* is displayed to users so that users can perform operations on it and fill in some data.

The name of tables	Explanation
drawnodeandline	This table stores all informations of the FSMs, such as the type of a FSM, the type of the state code, the type of flip-flop, etc..
node	This table stores the information of all states in the FSMs, such as the name, the position, the id of the FSM of these states.
line	This table stores the information of all transition in the FSMs, such as the target, the source, the id of the FSM of these states.
teacher	This table stores the information of all instructors.
student	This table stores the information of all students.
studentandteacher	The relationship between table student and table teacher is n:m. So studentandteacher table is created, in order to break down the relationship between table student and table teacher from n:m to n:1 and m:1.
evaluation	This table stores the score of self-assessment of the students.
dataaccess	This table stores the the opening and closing time of all knowledge topics.
feedback	This table stores the score of feedback from student. The feedback is anonymous, so this table have no connection with table student and teacher.

Table 5.2: The introduction of the database

2) Then, users send out a request.

3) The request of user sent by the *View* reaches the *Controller*. Such a request contains the business functionalities that users want to accomplish and the related data.

4) The *Controller* handles this user request. It encapsulates the data in the request and then invokes a proper *Model* to update.

5) The *Model* deals with the business functionalities requested by users, meanwhile updates and maintains the *Model* status.

6) When the *Model* status changes, the *Model* is going to inform the *Controller*.

7) The *Controller* obtains the data which needs to be shown and selects a proper *View* to display these data. After that, it just waits for the users' next operation and repeats the whole process from 1) to 7).

5.3 THE BENEFIT OF MVC

In the earlier development, some programmers did not realize the advantages brought by MVC and disobeyed it during development, re-

sulting in unclear structural division of applications and dysfunction of each part. When business functionalities change, no matter business logic or page display is modified, it causes considerable modification to the code. A slight move in one part may affect the situation as a whole. This gives rise to low efficiency, many mistakes during software development and the difficulties in maintenance.

Following MVC architectural pattern to develop systems can greatly avoid the above mentioned problems. The core concept of MVC is to loose coupling. MVC pattern divides an application into *Model*, *View*, and *Controller* by carefully analysing and defining functionalities. Then it tightly controls the communication among these three parts in order to gain a reusable, extendable, and maintainable application that has clear structure and rationally distributed functionalities.

Hence, taking advantage of MVC architectural pattern can bring the following benefits:

1) **Loose Coupling**: In MVC architectural pattern, *Model* and *View* are decoupled.

2) **Lower Development Costs and Higher Cooperate**: MVC architectural pattern helps men designate the responsibilities for each part, which makes programmers perform their own roles. Java programmers only need to care the realization of business logic which is the *Model* part. While page designers just think how to display pages which is the *View* part.

3) **Better Maintainability**: When software requirements vary, the three parts of MVC pattern are able to change independently without mutual influence to each other, which makes applications easier to maintain and extend.

5.4 SSH2 FRAMEWORKS

In last chapter, the core concept of MVC architectural pattern and the advantages of using it to perform development is introduced. As a matter of fact, MVC architectural pattern already becomes the main trend of web application development today. So as to distinguish from earlier JSP development pattern, the JSP web applications built on the basis of MVC architectural pattern are called to be JSP Model 2. For example, Servlet+JSP+JavaBean is a typical Model 2. JavaBean acts as the *Model*, JSP as the *View*, and Servlet as the *Controller*.

Except Servlet+JSP+JavaBean and EJB pattern provided by Java itself, there also exist many third-party web application frameworks based on MVC architectural pattern, such as Struts1, Struts2, Spring-MVC, etc. DLD-VISU adopts Struts2 to realize MVC pattern.

Other than Struts2 framework, Spring and Hibernate frameworks are also employed in DLD-VISU. Hibernate is responsible for the implementation of *ORM*. Spring acts as a light-weighted *IoC* container. On one hand, it is in charge of search, location, creation and management of objects as well as the dependencies between them. On the other hand, it can make Struts and Hibernate work better. The combination of these three frameworks is named as SSH. According to the different versions of Struts, SSH is categorized into SSH and SSH2. SSH2 indicates those SSH group frameworks which apply Struts 2.0 or later version. The reason why to differentiate Struts 1.0 from Struts 2.0 or later version is that big differences exist between the working principles of Struts 1.0 and Struts 2.0.

The primary business flow of the systems built by SSH2 is showed in Figure 5.3:

1) In UI layer, JSP is responsible for sending request and receiving response, which realizes the interaction with users. Struts 2 designates the received request to a proper action according to the configuration file.

2) In business layer, the Spring IoC container which manages service components provides Action with Model component as well as its collaborative *Data Access Object (DAO)* component. In addition, it also supplies some container components like event handler, buffer pool, etc.. to boost system performance and ensure the integrity of data.

3) In persistence layer, the ORM of Hibernate is the key to interact with database, process the data requested by DAO component, and return the processing results.

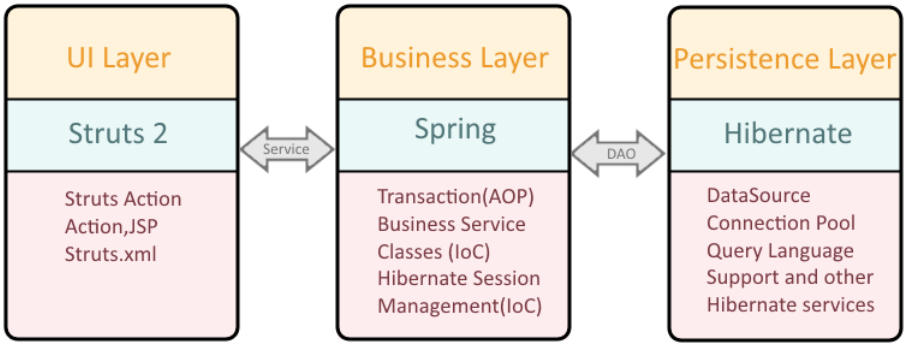


Figure 5.3: Business flow chart for SSH2.

Applying the above development model, not only are *Model*, *View*, and *Controller* totally separated in terms of structure, but also in the aspect of work-flow, the isolation of business layer and persistence layer is accomplished. Thus, no matter how front-end varies, only a

slight modification is performed to the *Model* layer. Meanwhile, the change of database does not influence front-end. In this way, the reusability of systems is immensely increased. Besides, due to the loose coupling between layers, it is convenient for teams to work concurrently, which greatly improve the development efficiency.

In the following, we explain first how Struts 2, Spring, and Hibernate work respectively, and then describes how to integrate these three frameworks, and how DLD-VISU specifically uses these three frameworks.

5.4.1 *Struts2*

Apache Struts is an open-source web application framework for developing Java EE web applications. It uses and extends the Java Servlet API to encourage developers to adopt a MVC architecture pattern [57]. Struts 2 is the next generation of Struts 1 and a brand new framework built by combining Struts 1 and WebWork technique. This new framework takes WebWork as its core and adopts the dispatcher mechanism to process user requests, which makes business logic controller completely separate itself from Servlet API. Therefore, Struts 2 can also be seen as the update product of WebWork.

As mentioned above, the main functionality of Struts 2 is to implement the partition of systems into 3 parts. Subsequently, we specify the working principle of Struts 2.

Struts 2 framework can be roughly divided into three parts: *FilterDispatcher*, *Action*, and enterprise business logic components implemented by users.

Thereinto, *FileDispatcher* is the base of Struts 2 framework, including the control flows and processing mechanism inside the framework. *Action* and business logic components are realized by users themselves. During the development of *Action* and business logic components, users need to write the relevant configuration informations so that *FilterDispatcher* may use them. The basic operating principle of Struts 2 is as follows:

1) The client browser sends a *HttpServletRequest* request indicating Servlet containers to the server (e.g. Tomcat).

2) According to the configuration informations in *web.xml*, Tomcat submits the request to *FilterDispatcher* of Struts 2.

3) *FilterDispatcher* inquires *ActionMapper* whether to invoke one *action* to process this request. If *ActionMapper* decides to invoke one certain *action*, *FilterDispathcer* leaves the request to *ActionProxy* to process.

4) *ActionProxy* accesses the configuration file *Struts.xml* through *ConfigurationManager* in order to find out the *action* class that needs to be called.

5) *ActionProxy* creates an *ActionInvocation* instance. In the mean time, *ActionInvocation* invokes *Action* through proxy pattern.

6) Once the execution of *Action* is finished, *ActionInvocation* is in charge of searching for the corresponding returning result in accordance with *Struts.xml*. The returning result is usually a JSP page.

7) The result (JSP page) is returned to the client browser.

An example is presented in the following to clarify how Struts 2 works.

Firstly, the configuration file of Tomcat *web.xml* needs to be modified. The controller of Struts 2 is added into this file. The concrete operation is demonstrated in Code 5.1 lines 9-16. Through this configuration, Tomcat is going to deliver all requests coming from the client browser to the core controller of Struts 2 – *FilterDispatcher* – to process. Tomcat does no deal with these request any more.

```
1  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns="http://java.sun.com/xml/ns/javaee"
       xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
       xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
       http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
       version="2.5">
2    <welcome-file-list>
3      <welcome-file>index.jsp</welcome-file>
4    </welcome-file-list>
5    <context-param>
6      <param-name>contextConfigLocation</param-name>
7      <param-value>/WEB-INF/applicationContext.xml</param-value>
8    </context-param>
9    <filter>
10     <filter-name>struts2</filter-name>
11     <filter-class>
12       org.apache.struts2.dispatcher.FilterDispatcher
13     </filter-class>
14   </filter>
15   <filter-mapping>
16     <filter-name>struts2</filter-name>
17     <url-pattern>/*</url-pattern>
18   </filter-mapping>
19   <listener>
20     <listener-class>
21       org.springframework.web.context.ContextLoaderListener
22     </listener-class>
23   </listener>
24 </web-app>
```

Code 5.1: Loading Sturts 2 into Tomcat server.

When users send a request, it is intercepted by *FilterDispatcher* which searches for the enterprise business logic component relating to this request in *Struts.xml*. In this example(code 5.2 line 9), when users click the "Binary Code" button in browser, this browser is going to send a request named as *getBinaryCode* to server. *FilterDispatcher* intercepts this request, searches the *Struts.xml* file to seek the *action* tag that has the same name with the request, and finally obtains *stateCodeAction* class and *getBinaryCode* method required by the request through the class and method attributes of the *action* tag. Struts 2 instantiates *stateCodeAction* class via *ActionInvocation* object, executes *getBinaryCode* method, and returns the result presented in Code 5.3 line 2 to an assigned page which is indicated in the sub-tag result of the action tag. In this example, such a page is info_center.jsp.

```
1  function getCodeTable(){
2    var url;
3    var val = $("input:radio[name='code']:checked").val();
4    if(val=="0"){
5      code = "binaryCode";
6      url="getBinaryCode.action";
7    }
8    var pars={drawNodeAndLineId:$("#drawNodeAndLineId").val()};
9    $.post(url,pars,setStateCodeTable);
10 }
```

Code 5.2: Submit an action to server.

```
1  <action name="getBinaryCode" class="stateCodeAction"
       method="getBinaryCode">
2    <result name="success">info_center.jsp</result>
3  </action>
```

Code 5.3: Action information in struts.xml.

Through the above specified example, we can very clearly observe that *FilterDispatcher* is the core of MVC controller layer. By means of *FilterDispatcher*, Struts 2 assigns the requests coming from client to the corresponding business logic component to deal with and returns the result to client. Thus, the isolation between *View* layer and *Model* layer is accomplished.

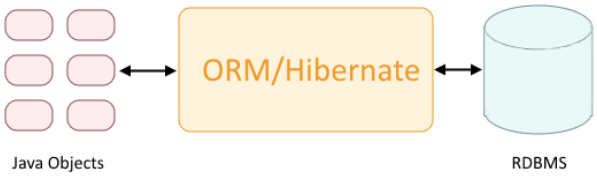


Figure 5.4: Relationship diagram of Hibernate.

5.4.2 *Hibernate*

Hibernate is an open-source object-relational mapping framework and was created by Gavin King in 2001. It is a bridge between traditional Java objects and database servers and used to deal with O/R mapping.

Hibernate is a light-weighted framework and does not need any support of server. Hence, it can be applied in any scenarios where JDBC is employed. That is, Hibernate can be utilized in both Java client applications and Servlet/JSP web applications. We explain in this section the design philosophy and working principle of Hibernate in details.

5.4.2.1 *The design philosophy of Hibernate*

In the development of J2EE applications, the data in the persistence layer is usually stored in relational databases. Java supplies a set of Java APIs named as JDBC which is used to access relational databases. These Java APIs make Java applications able to execute SQL statements and communicate with any relational databases which confirm to SQL specification. Though JDBC is able to meet the requirement of performing operations on databases, making use of JDBC demands programmers to insert SQL statements in the code in application layer, causing bad reflection on the object-oriented programming idea and plenty of redundant code.

As previously explained, a problem of mismatching object model and relational database exist in the object-oriented application development. Object-oriented programming is developed on the basic rules of software engineering, such as low coupling, high cohesion, encapsulation, etc., while relational databases derive from mathematics theories. There exist evident differences. Relational databases use tables to store data, whereas an object-oriented programming language like Java employs objects to store data. During operating on relational databases, four problems relating to mismatch come along and they are displayed in Table 5.3. ORM is just the measure to solve all these mismatch issues.

ORM stands for Object-Relational Mapping which is a technology transforming data between relational database and object-oriented

Mismatch	Description
Granularity	The attribute number of object is much more than the column number of database table.
Inheritance	RDBMSs do not support any inheritance which is inherent in object-oriented programming languages.
Relationship	Object-oriented programming languages use reference to express relationship, while RDBMS applies foreign key to present the relationship between objects.
Navigation	The way how to access objects in Java and RDBMS is completely different.

Table 5.3: Different between Objects and RDBMS.

programming languages, such as Java, C#, etc. Compared with JDBC, an ORM system has the following advantages:

1) Using code to access objects rather than database tables.
2) Hiding the details of SQL query in object-oriented logic.
3) Unnecessary to handle the implementation of database.
4) Entity is not database structure but a concept based on business.
5) Auto-generation of event management and key.
6) Quick development of applications.

5.4.2.2 *The operating principle of Hibernate*

The core components of Hibernate are:

1) **Configuration class** is used to read Hibernate configuration file and create *SessionFactory* object.
2) **SessionFactory interface** is responsible for the creation of Session instance.
3) **Session interface** is employed to operate *Persistent Object (PO)*. It contains the get, load, save, update, delete methods which are applied to load, save, update and delete PO. This is the core interface of Hibernate.
4) **Query interface** is utilized to perform Query operations on PO. It can be created by the *createQuery* method of Session.
5) **Transaction interface** manages Hibernate events. It mainly includes *commit* and *rollback* methods and can be generated by the *beginTransaction* method of Session.

State	Managed by Session	Existing in database
Transient	No	No
Persistent	Yes	Yes
Detached	No	Yes

Table 5.4: The distinction of these three states of PO.

Persistent Object

Persistent Object can be general JavaBeans. Only one point that they correlate with Session is special. There are three states of JavaBeans in Hibernate:

1) Transient. When a JavaBean object exists independently and does not have any relationship with the data in database, this JavaBean object is called as Transient Object.

2) Persistent. When a JavaBean object correlates with a Session, it becomes a Persistent Object.

3) Detached. At the moment the Session is closed, this object is also out of the persistent status and becomes a Detached Object which can be freely used by applications in any layer.

The distinction of these three states is shown in Table 5.4.

5.4.2.3 *The work-flow of Hibernate*

We demonstrate the working principle of Hibernate through a specific example. The following is a fragment of code for register of new teachers in DLD-VISU.

```
1 public void addTeacher(Teacher teacher) throws
      HibernateException{
2   sessionFactory sf = new
        Configuration().configure().buildSessionFactory();
3   Session s = sf.openSession();
4   Transaction transaction = s.beginTransaction();
5   s.save(teacher);
6   transaction.commit();
7   s.close();
8 }
```

Code 5.4: The code for add a teacher.

1) First of all, we configure the file *hibernate.cfg.xml*, the concrete steps are displayed in Code 5.5.

   a) Configuring the database type to be MySQL (line 2).

b) adding the user name and password used to access database (line 4).

c) adding database mapping file. (**.hbm.xml* line 18-30)

2) Reading and parsing the configuration file *hibernate.cfg.xml* through *Configuration().configure* method.

3) Reading and parsing the mapping information from `<mappingre-source = "com/org/model/Teacher.hbm.xml"/>` in the file *hibernate.cfg.xml*. In this example, Hibernate parses the file *Teacher.hbm.xml*, which is presented in Code 5.6. In the file *Teacher.hbm.xml*, we map com.org.model.Teacher Class and the "teacher" table in database as well as indicate the mapping relationship between the attributes of Teacher class and the columns of the "teacher" table.

4) Creating SessionFactory through *config.buildSessionFactory* method.

5) Opening Sesssion using *sessionFactory.openSession* method.

6) Creating Transaction via *session.beginTransaction* method.

7) Performing the persistence operation *s.save(teacher)*.

8) Committing the transaction by *transaction().commit* method.

9) Closing Session.

10) Closing SessionFactory.

```
1  <bean id="dataSource"
       class="org.apache.commons.dbcp.BasicDataSource">
2    <property name="driverClassName"
         value="com.mysql.jdbc.Driver"/>
3    <property name="defaultAutoCommit" value="false"/>
4    <property name="url"
         value="jdbc:mysql://localhost:3306/c63_tgdi"/>
5    <property name="username" value="root"/>
6    <property name="password" value="Che1uZ0j"/>
7  </bean>
8  <bean id="sessionFactory"
9  class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
10   <property name="dataSource">
11     <ref bean="dataSource"/>
12   </property>
13   <property name="hibernateProperties">
14     <props>
15       <prop key="hibernate.dialect">
16         org.hibernate.dialect.MySQLDialect
17       </prop>
18       <prop key="hibernate.show_sql">
```

```
        true
      </prop>
    </props>
  </property>
  <property name="mappingResources">
    <list>
      <value>com/org/model/Line.hbm.xml</value>
      <value>com/org/model/Drawnodeandline.hbm.xml</value>
      <value>com/org/model/Node.hbm.xml</value>
      <value>com/org/model/Student.hbm.xml</value>
      <value>com/org/model/DataAccess.hbm.xml</value>
      <value>com/org/model/Teacher.hbm.xml</value>
      <value>com/org/model/StudentAndTeacher.hbm.xml</value>
      <value>com/org/model/Evaluation.hbm.xml</value>
      <value>com/org/model/Feedback.hbm.xml</value>
    </list>
  </property>
</bean>
```

Code 5.5: Configuration of hibernate.

```
<hibernate-mapping>
  <class name="com.org.model.Teacher" table="teacher"
      catalog="c63_tgdi">
    <id name="pId" type="java.lang.String">
      <column name="pId" length="32"/>
      <generator class="uuid.hex"/>
    </id>
    <property name="username" type="java.lang.String">
      <column name="username" length="20"/>
    </property>
    <property name="password" type="java.lang.String">
      <column name="password" length="8"/>
    </property>
    <property name="firstname" type="java.lang.String">
      <column name="firstname" length="50"/>
    </property>
    <property name="lastname" type="java.lang.String">
      <column name="lastname" length="50"/>
    </property>
    <property name="email" type="java.lang.String">
      <column name="email" length="50"/>
    </property>
    <property name="web" type="java.lang.String">
      <column name="web" length="100"/>
    </property>
    <property name="approval" type="java.lang.Integer">
      <column name="approval" length="11"/>
    </property>
  </class>
</hibernate-mapping>
```

Code 5.6: Teacher.hbm.xml.

The sequence diagram of the whole process is presented in Figure 5.5. Thereinto, the part highlighted by the red frame is the work-flow of Hibernate.

From the above introduced example, we can see that no SQL statement is inserted in Java-end. All operations on databases are encapsulated by Hibernate into the operations to Java objects, which brings the following benefits:

1) Encapsulating the code using JDBC to access database reduces the repeated code in data access layer.

2) Hibernate utilizes XML files to handle the mapping between Java classes and database tables. It is unnecessary to write any code.

3) If a database table varies (e.g. add, delete, and update columns), it only needs to change the corresponding attribute tags in the XML file.

4) Abstracting unfamiliar SQL types, providing programmers with familiar Java objects, and greatly simplifying the programming in DAO layer.

5) Due to the light weight of Hibernate, its performance is very good and it does not bring additional burden to server.

6) Hibernate supports all kinds of relational databases and multiple complicated relationship, e.g. one-to-one, many-to-many, etc.

7) Because of taking advantage of Hibernate, the code does not refer specific JDBC statements, which tremendously increases the portability of code.

5.4.3 *Spring*

Spring is a light-weighted container framework. The first version of it was written by Rod Johnson, who released the framework with the publication of his book in October 2002 [48]. The core features of the framework can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform [49] . Although the framework does not impose any specific programming model, it has become popular in the Java community. By far, Spring includes dozens of various components which are used

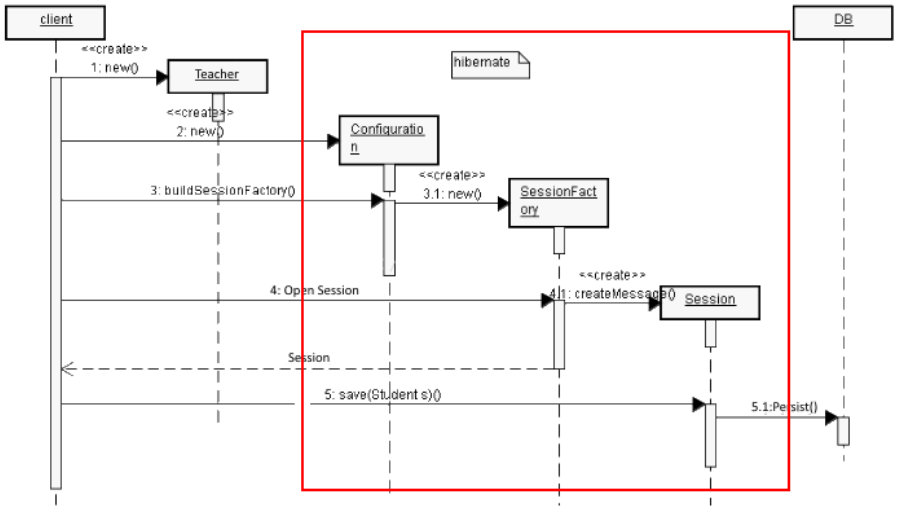


Figure 5.5: Sequence chart for add a teacher to database

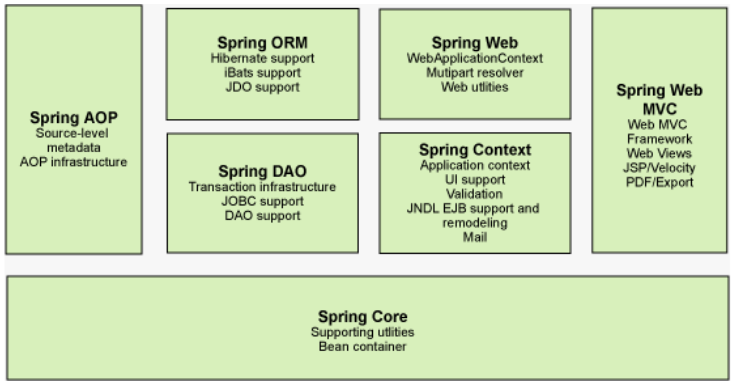


Figure 5.6: Some components of Spring [96].

to realize distinct functionalities. Figure 5.6 [96] presents some components of Spring. DLD-VISU employs IoC components which is also the core components of Spring. It becomes more convenient to configure and manage Struts 2 and Hibernate by taking advantage of IoC of Spring. Before introducing Spring, we explain the design philosophy of IoC first.

5.4.3.1 *The design philosophy of IoC*

It is well know that in the software systems which are designed using object-oriented methods, their underlying implementation comprises N objects and all objects cooperate with each other to realize the business logic of systems.

Figure 5.7 illustrates a gear set. It owns multiple independent gears that mesh with each other, work cooperatively, and accomplish tasks together. In such a gear set, if there is something wrong with one gear, the whole gear set can be affected and cannot function normally.

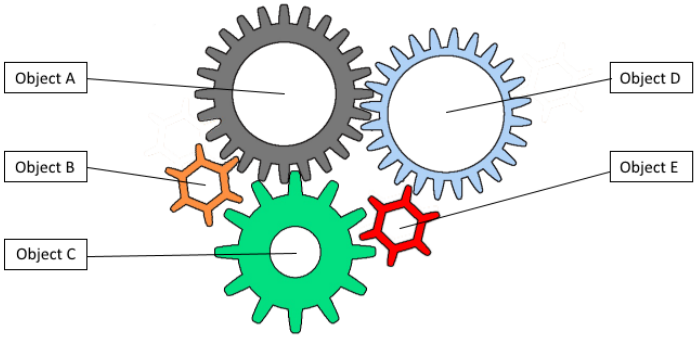


Figure 5.7: The coupling of objects in a sofeware

The meshing of gears in one gear set is similar with the coupling relationship among the objects of software systems. The coupling relationship of objects is inevitable and necessary. It is the foundation of cooperative work. As the scale of industrial applications grows bigger and bigger nowadays, the dependencies among objects become more complicated accordingly. There often exist multiple dependencies between objects. Therefore, programmers are facing greater challenges in the aspect of analysing and designing systems. Tight coupling between objects can result in the domino effect.

How to lower the coupling degree between systems, modules, and objects is always one of the goals that are pursued in software engineering. To solve this problem – high coupling between objects, Michael Mattson came up with the IoC theory to carry out the decoupling between objects [65] . IoC is the abbreviation of Inversion of Control. In 1996, Michael Mattson brought up the IoC concept in an article exploring object-oriented frameworks for the first time. The basic idea of IoC is to accomplish decoupling among those mutually dependent objects with the help of "third-party". As shown in Figure 5.8, because of introducing the "third-party" in the middle which just is the IoC container, the dependencies among the objects A, B, C, and D are decoupled and the drive of gears totally rely on the "third-party" which means the control of all objects is handed over to the "third-party" IoC container. In this way, IoC container becomes the key of the whole system and acts as an adhesive to bind all objects in the system together. If the adhesive loses, the objects also lose their connections wit each other.

Let us take away the IoC container in Figure 5.8 first and then observe the system. The objects A, B, C, and D do not have coupling now and also have no contact with each other. Thus, programmers do not need to consider B, C, and D in the course of implementing A. The dependencies among objects are decreased to the lowest level. So if a system utilizes IoC container in the development, the complexity of designing the system can be immensely reduced.

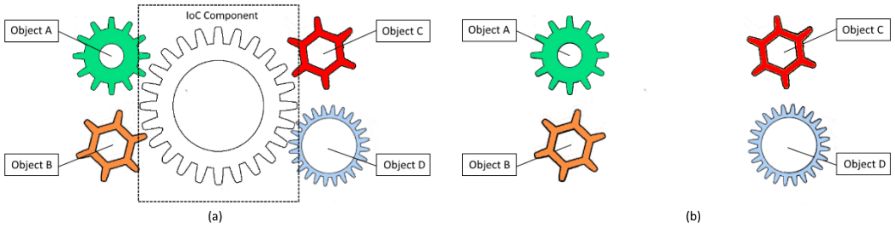


Figure 5.8: Decoupling with IoC

Before software systems introduce IoC container, as presented in Figure 5.7, the object A relies on the object B. When the object A is initialized or runs to some points, it must actively create the object B by itself or use the already existing one. That is, no matter the control or utilization of the object B is in A's hands.

After software systems introduce IoC container, the situation changes fully. As displayed in Figure 5.8, the object A and the object B lose their direct connection because of the addition of IoC container. Thus, when the object A needs the object B, the IoC container is responsible for creating the object B which is injected in the place where the object A demands it.

From the comparison of the situations with and without IoC container, it is easily to tell that the process in which the object A obtains the dependent object B becomes a pro-activeness from a passive action. The control is inverted. This is where the name "Inversion of Control" comes.

5.4.3.2 *The design philosophy of Dependency Injection*

The design philosophy of IoC is explained above. There are two ways to implement IoC: DI (Dependency Injection) and Service Locator. The implementation way adopted by Spring is Dependency Injection that is clarified subsequently.

DI was brought forward by Martin Fowler in 2004 for the first time [27]. Martin Fowler explored in his thesis which controls are reversed in the process of implementing IoC. After detailed analyses and argumentations, he acquired the conclusion that the course in which dependent objects are obtained is reversed. When the control is reversed, the process in which dependent objects are gained becomes proactive injection by IoC from proactive creation by a caller-object. In fact, his conclusion just indicates the method to realize IoC – injection. The so-called injection is to dynamically inject some dependencies into objects during IoC container runs.

In the traditional realization, the relationship between components is controlled by the program code. We usually use the keyword new to perform the combination of two components, which makes the components coupled. IoC solves this problem well. It brings the correlation of components from inside program to outside container,

which means some dependencies are dynamically injected into components by the container in the course of running.

5.4.3.3 *Advantages and disadvantages of IoC*

Table 5.5 describes the advantages and disadvantages of IoC framework in details.

5.4.3.4 *The implementation principle of IoC*

The basic technology of IoC is *Reflection Programming*. Until now, C#.net, Java, and PHP5 all support *Reflection Programming*. The so-called reflection is to dynamically create objects according to given class names (in String format). Such a programming method is able to decide which kind of objects are created at the moment of creation.

As a matter of fact, the reflection came along very early. But it was always ignored and did not gained further utilization. Only because reflection programming at that time was ten times slower than the normal way to generate objects. Today, reflection technology has already been improved and mature. The distinction between their speeds is not that big anymore. Reflection programming is just once or twice slower than the normal way.

We can treat the pattern of IoC container as the improvement of the factory pattern. The IoC container is seen as a factory. The objects that are produced in this factory are defined in the configuration file. Then, taking advantage of reflection programming and according to the class names given by the configuration file to generate corresponding objects. In terms of implementation, object creation in the factory pattern is written in code, while IoC creates objects by defining them in the configuration file. Consequently, factory and object creation are isolated so that the flexibility and maintainability can be boosted.

Subsequently, we demonstrate a DLD-VISU example to show how Spring realizes IoC.

1. First of all, we configure the file *web.xml* by inserting the listener of Spring and designate the configuration file used by Spring *applicationContext.xml*.

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```

Code 5.7: Loading Spring into Tomcat server.

Advantages	Disadvantages
Good maintainability: after using IoC, the coupling between classes does not exist anymore. It is very convenient to perform unit tests and debug as well as troubleshoot. Every single class in code can be tested separately. There is no mutual impact and it only needs to guarantee the functionalities of the class itself do not go wrong. This is the benefit brought by loose or no coupling among components.	Due to introducing the third-party IoC container into software systems, the procedure to create objects becomes sophisticated. The thing is originally between two ends, now more steps are required to finish the thing. Therefore, we may feel not intuitive at the beginning we use IoC framework.
The independence of development: each development team member only needs to care about his own business logic and does not need to consider the others' work at all. Because his task totally has nothing to do with the others' and can be tested lonely without depending upon the others'. Hence, it is very easy for programmers to partition a large task into some small parts in the process of developing a complex project. Accordingly, the development efficiency and the product quality must also be greatly improved.	Introducing a brand new framework is going to increase the cost. Team members have to study it and in the future maintenance, new comers also have to master the same knowledge.
Good reusability: although reusability is a main characteristic of object-oriented programming, it is not easy to reuse code because of the complicated dependencies among objects. IoC carries out this principle better and increases reusability.	Owing to that IoC generates objects by reflection, the running efficiency must be damaged to some extent. If programmers would like to pursue the running efficiency, they must judge and weigh.
As same as the peripheral equipment like USB, modules own the hot plugin characteristic. IoC uses the external way to generate objects, which is to define the object creation in the configuration file. Thus, it is easy to change an implementation subclass, we only need to modify the configuration file.	For the products of IoC framework, e.g. Spring, it is necessary to do plenty of configurations that are sophisticated. To some small projects, the cost is pretty high.

Table 5.5: The advantages and disadvantages of IoC framework

Attribute	Explanation
id	The identifier of bean. It cannot be repeated
class	The corresponding class of this bean
property	The dependency of this class
ref	Which bean is referred

Table 5.6: The advantages and disadvantages of IoC framework

2. Spring manages the objects that are needed to be injected and their dependencies by configuring the <*Bean*> tag in the file *applicationContext.xml*. Spring encapsulates Java Classes as beans first. Each bean is mapping to a class. The dependencies between classes are set through attribute *"property"*. Table 5.6 lists several common attributes of Bean and their explanations.

After the initialization of Spring, all beans declared in the file *applicationContext.xml* are managed by Bean Factory which is the IoC container that we mentioned previously. When running system requires an object, Spring automatically instantiates the required object and injects it into the assigned object. In the example displayed in Code 5.8, we can see there are three beans in *applicationContext.xml*. They respectively correspond to *KmapAction*, *KmapService* and *DealKmap* classes. The execution of *kmapAction* depends on *kmapService*, and the execution of *kmapService* depends on *DealKmap*.

```
<bean id="kmapAction" class="com.org.action.KmapAction"
    scope="prototype">
  <property name="kmapService">
    <ref bean="kmapService"/>
  </property>
</bean>
<bean id="kmapService" class="com.org.service.impl.KmapService"
    scope="prototype">
  <property name="dealKmap">
    <ref bean="dealKmap"/>
  </property>
</bean>
<bean id="dealKmap" class="com.org.algo.kmap.DealKmap"
    scope="prototype">
</bean>
```

Code 5.8: An example of the dependency between 3 beans

As shown in Code 5.9, an object can be used directly in a method without instantiating *kmap-Service* as usual by using `KmapService kmapService = new KmapService()`. That's because the dependencies between *kmapAction* and *kmapService* in *applicationContext.xml* have been already configured. When *kmapAction* is executed, Spring au-

tomatically generate an instance of *kmapService* object and inject it into *kmapAction* through *setKmapService* method. The whole process is started and maintained by Spring. When the instance of *kmapService* object is not demanded any more, Spring deletes it. Note that Spring applies Java standard constructors or setter methods to inject instances, not causing any intrusion to systems. *KmapAction* class does not need to inherit any Spring superclass or implement any Spring interface. As long as it contains a setter method owned by dependent objects, such as *setKmapService* in this example, Spring is going to invoke this method to inject the created instance into *KmapAction*. Such a characteristic – no intrusion – is just an advantage of Spring being as a light-weighted framework. Figure 5.9 show the sequence process of above example.

```
public class KmapAction extends ActionSupport{
  IKmapService kmapService;
  public String execute() throws Exception {
    ......
    String oneArray[] = oneList.split(",");
    String[] variables = variableNames.split(",");
    ......
    kmapService.initKmap(oneArray, xArray, variables.length);
    kmapService.findPrimImplicants();
    kmapService.findCoreImplicants();
    ......
  }
  public void setKmapService(IKmapService kmapService) {
    this.kmapService = kmapService;
  }
}
```

Code 5.9: An example of Dependency Injection

5.4.4 *The integration of SSH2 framework*

As described above, Struts 2 is decided to employ to realize MVC architecture pattern and Hibernate to encapsulate the operations on database in the course of designing DLD-VISU. Although the isolation between MVC parts and ORM are achieved after introducing Struts 2 and Hibernate, the work on maintaining different configuration files is accordingly increased. Meanwhile, the complexity of the system is also raised. With the increase of the system complexity, the dependencies among classes are getting more complicated, which is against the maintenance and upgrade of the system. To solve this issue, we introduce Spring framework. Taking advantage of Spring to manage Struts 2 and Hibernate brings four benefits:

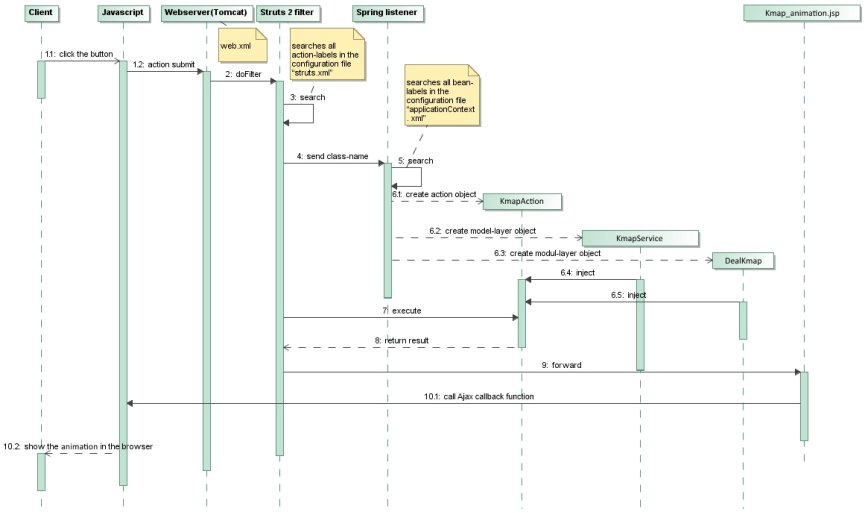


Figure 5.9: The sequence diagram of the cooperation between Spring and Struts2

1) **Aggregating configuration files:** We can just utilize one configuration file *ApplicationContext.xml* to configure Hibernate and Struts2. It is unnecessary to use multiple configuration files.

2) **Unifying framework management:** The instances needed by Struts 2 and Hibernate are all created and maintained by Spring, including the Action object of Struts 2 and the Session object of Hibernate. Due to the characteristics of Spring IoC, the dependencies among classes are eliminated, reducing the system complexity and the difficulty of maintaining as well as upgrading the system.

3) **Declarative services:** It is more convenient to implement the rollback of transactions by applying the AOP function of Spring which is declarative and only needs to be simply configured rather than programming.

4) **Light-weighted framework and no intrusion:** Spring is a light-weighted framework which is distinguishing from Struts 2. There is no need for the system to add extra code because of the insertion of Spring framework, leading to better portability and debugging.

5.4.4.1 *The steps of integrating SSH2*

The steps of integrating SSH2

1) Inserting Hibernate framework: Importing relevant Hibernate jar files into the lib directory.

2) Inserting Struts 2 framework:

a) Importing relevant Struts 2 jar files into the lib directory.

b) Configuring *web.xml* by adding Struts 2 filters in <*web-app*> tags. The concrete code is presented in Code 5.1.

c) Configuring *struts.xml*. Creating <*action*> tags required by the system. Because the Action instances of Struts 2 are generated by Spring uniformly. The class attribute of the <*action*> tags here is no more specific class but the corresponding bean id of Spring.

3) Inserting Spring framework:

a) Importing relevant Spring jar files into the lib directory.

b) Importing the plugin jar files which are relevant for combining struts and spring.

c) Configuring *web.xml* by inserting Spring listener and indicating the location of Spring configuration file. The code is shown in Code 5.7.

d) Configuring *applicationContect.xml* by adding the configuration information of Hibernate.

- Using `<bean id="sessionFactory">` to configure sessionFactory
- Using `<bean id="transactionManager">` to configure transaction manager.
- Using <*tx:*> tags to configure the dispatching characteristics of transactions. Setting read-only='true' for all operations except for addition, deletion, modification, and select so that the performance can be improved.
- Using <*aop:*> tags to configure which methods of which classes participate in transactions.
- Configuring the beans that utilize Hibernate by adding <*sessionFactory*> tag for them.

DLD-VISU utilized the Spring AOP framework to realize the transaction management of databases, such as database validation, failure rollback, etc. Hibernate also supplies the functionality of transaction management, while it is much simpler and more convenient in Spring. Programmers just need to write relevant rules in the configuration file. Spring may automatically accomplish transaction management, resulting in greatly simplifying development.

As displayed in Code 5.10, Spring makes use of <*tx:*> tag to declare transaction management. In DLD-VISU, we bind all classes in the *com.org.service.impl* package with transaction management and set the type of the "add", "delete", "modify", and "select" transactions to be "REQUIRED" which means, if the system has no transaction now, then it creates one; if a transaction is already opened, the system just

uses it and does not generate a new one, i.e. the system currently owns only one transaction. Because all "add", "delete", "modify", and "select" operations are incorporated in one transaction. Once the rollback of this transaction is executed, all these operations are cancelled.

```
<tx:advice id="txAdvice"
    transaction-manager="transactionManager">
  <tx:attributes>
    <tx:method name="recover*" propagation="REQUIRED"/>
    <tx:method name="remove*" propagation="REQUIRED"/>
    <tx:method name="add*" propagation="REQUIRED"/>
    <tx:method name="delete*" propagation="REQUIRED"/>
    <tx:method name="update*" propagation="REQUIRED"/>
    <tx:method name="*" read-only="true"/>
  </tx:attributes>
</tx:advice>
<aop:config>
  <aop:pointcut id="allManagerMethod" expression="execution(*
      com.org.service.impl.*.*(..))"/>
  <aop:advisor advice-ref="txAdvice"
      pointcut-ref="allManagerMethod"/>
</aop:config>
```

Code 5.10: The transaction management Code in applicationContext.xml

5.4.4.2 *The advantages of SSH2*

The working principles of Struts, Spring, and Hibernate are already demonstrated respectively. Table 5.7 summarizes the advantages of SSH2 framework.

5.5 THE ARCHITECTURE OF OF DLD-VISU FRAMEWORK

DLD-VISU adopts SSH2 framework to carry out development, further divides the system into 5-layer architecture on the basis of the 3 layers of MVC. From top to bottom, the constituent parts of the system are respectively: Action layer, Service layer, DAO layer, Model layer, and Persistence layer (database). Thereinto, the interface-oriented programming concept is utilized in the Service layer which is more specificity partitioned into Interface layer and Implementation layer. Subsequently, the functionalities of each layer are explained in details.

Action layer is in charge of receiving clients' requests, invokes the corresponding objects in the Service layer in accordance with these requests, and returns the corresponding View (JSP) to clients according to the responses from the Service layer. Service layer is employed to deal with concrete business logic. If business logic demands persis-

Struts 2	Spring	Hibernate
Achieves MVC pattern, leads to clear architecture, makes programmers only focus on the implementation of business logic.	No intrusion (The existence of Spring framework in the business logic code cannot be felt.)	ORM. Only needs to operate objects, making development more object-oriented.
Abundant tag library, immensely enhancing the development efficiency.	The couplings between components are very loose.	No intrusion.
Provides diverse implementations of interceptors.	There is no need for programmers to implement the singleton pattern on their own.	No need for server support.
Can master the relationship among all system parts through configuration file.	Can achieve transaction management through AOP.	
	Integrates other frameworks.	

Table 5.7: The advantages of SSH2

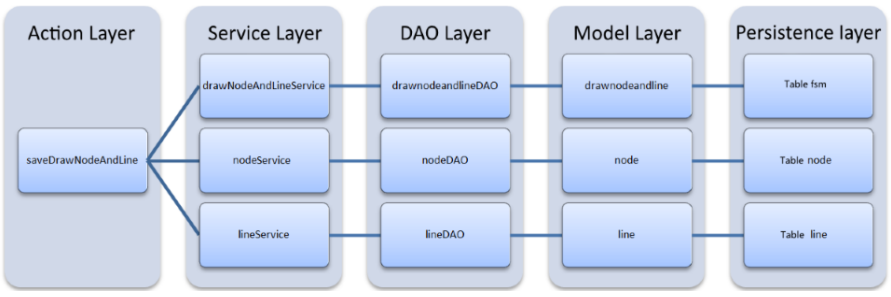


Figure 5.10: An example for five layer of DLD-VISU system

tence, such as reading/writing databases, this layer is going to designate the DAO layer to finish relevant work.

DAO layer stores Model into the databases. The reason why to use DAO layer is to separate the Model layer and the database so as to realize the function of running across data platforms, i.e. switching between different databases. If the DAO layer is not put to use, the operations on databases must be written in the Model layer. Once the database is changed, all code in the Model layer needs to be modified. It brings bad influence not only on the reusability and maintainability of code but also on the code of other layers, e.g. the operations on databases in the Service layer. By contrast, the abstraction-oriented programming can be adopted to build distinct classes for distinct databases after the DAO layer is introduced. These classes all inherit one same interface. Hence, even if the system changes databases, it is unnecessary to do any modification to the Service layer.

Model layer is responsible for supplying the objects that match with databases to the Service and DAO layers in order to realize ORM. During designing the Service and DAO layers for DLD-VISU, we adopted the following measures:

1) Inheriting the HibernateDaoSupport class in the DAO layer rather than in the Service layer. This eliminates the intrusion of Spring to the Service layer and boosts the readability and reusability of code in the Service layer.

2) Setting transactions on the Service layer. It brings the advantage: when an object of the Service layer needs to invoke multiple objects of the DAO layer to complete a transaction (Such as, before registering new teacher, checking if this teacher is already in the system must be performed; while saving a FSM diagram, *lineDAO*, *nodeDAO*, and *fsmDAO* are invoked respectively to store FSM diagram, state and transition), these objects of the DAO layer are placed in one transaction. Once any operation to any one DAO object fails, it makes sure that the whole transaction is totally rolled back.

Here have a example to demonstrate the specific operation of the system. Figure 5.10 shows the process after users utilize DLD-VISU to store FSM diagram.

1) After users finish drawing a FSM diagram and click next step, browser sends a request named as *saveDrawNodeAndLine.action* to server.

2) Struts 2 already registers a Filter on Tomcat server to intercept all requests. So Tomcat hands over this request to Struts 2 to deal with.

3) Struts2 searches its own configuration file to find the <*action*> tag with the name *saveDrawNodeAndLine* as well as its class attribute. Because DLD-VISU applies Spring to manage Struts and Hibernate uniformly. The value of the class attribute *drawNodeAndLineAction* here is the bean id in the Spring configuration file.

4) Spring inspects that it is demanded to inject an action object into Struts. It searches its own configuration file to find the bean with id named *drawNodeAndLineAction*, runs the class corresponding to the found bean – the *saveDrawNodeAndLine* method of *com.org.action.DrawNodeAndLineAction*, and creates instances of *drawNodeAndLineSerivce*, *nodeService*, and *lineService* according to the dependencies of the configuration file as well as injects them into the *DrawNodeAndLineAction* class. As specified above, the action layer invokes the service layer to accomplish concrete business logic. In this example, *drawNodeAndLineAction* respectively invokes three classes of the Service layer – *drawNodeAndLineSerivce*, *nodeService*, and *lineService* – to finish the function of storing FSM diagram, node, line into databases. A fragment of the relevant code is demonstrated as Code 5.11.

5) The *drawNodeAndLineSerivce*, *nodeService*, and *lineService* classes call *drawnodeandlineDAO*, *nodeDAO*, and *lineDAO* respectively to perform the operation of storing into databases. At this time, the *drawnodeandlineDAO*, *nodeDAO*, and *lineDAO* instances are created and accordingly injected into three Service classes by Spring (the *NodeServiceImpl* class just declares a private variable *NodeDAO* but does not instantiate it).

6) *drawnodeandlineDAO*, *nodeDAO*, and *lineDAO* utilize Hibernate to store relevant data into databases through Spring. As shown in Code 5.12, *NodeDAO* inherits a superclass *HibernateDaoSupport* that is an API connecting to Hibernate supplied by Spring. By inheriting this superclass, *NodeDAO* can directly apply the *getHibernateTemplate* method encapsulated by it. The *getHibernateTemplate* method provides the methods, such as save, update, delete, etc., to perform the operations to databases.

7) *DrawNodeAndLineAction* decides to return which page to browser according the response from the Service layer. In this example, after the Service layer correctly stores the related information into the database, a constant "SUCCESS" is returned. Struts 2 makes browser jump to fsm.JSP in accordance with the configuration file.

8) When the whole process is finished, browser jumps to fsm.JSP and users enter the second step of the whole FSM study flow.

```
public String saveDrawNodeAndLine(){
  Node node;
  Line line;
  ......
  drawnodeandline=new Drawnodeandline();
  drawnodeandline.setDrawNodeAndLineId(UUID.randomUUID().toString());
  drawnodeandline.setFsmType(drawRootStr.substring(8));
  drawnodeandline.setName("");
  drawnodeandline.setuId(uId);
  ......
  this.drawNodeAndLineSerivce.addDrawNodeAndLine(drawnodeandline);
  for(int i=0;i<listJson.size();i++){
    node = new Node();
    ......
    node.setDrawnodeandline(drawnodeandline);
    nodeService.addNode(node);
  }
  for(int i=0;i<lineArray.length;i++){
    line = new Line();
    String[] tempArray = lineArray[i].substring(1).split(",");
    line.setSourceNodeName(tempArray[4].substring(tempArray[4]
    .lastIndexOf(":")+1,tempArray[4].length()));
    ......
    lineService.addLine(line);
  }
  ......
  return SUCCESS;
```

Code 5.11: A fragment of code for saveDrawNodeAndLine method

```
public class NodeDAO extends HibernateDaoSupport {
  public void addNode(Node node){
    this.getHibernateTemplate().save(node);
  }

  public void updateNode(Node node){
    this.getHibernateTemplate().update(node);
  }

  public void deleteNode(Node node){
    this.getHibernateTemplate().delete(node);
  }

  public List<Node> getNodeList(String drawlinenodeId){
    Criteria criteria =
        this.getSession().createCriteria(Node.class);
    if(!"".equals(drawlinenodeId)){
      criteria.createCriteria("drawnodeandline").
      add(Restrictions.eq("drawNodeAndLineId", drawlinenodeId));
    }
    return (List<Node>)criteria.list();
```

```
21    }
22 }
```

Code 5.12: A fragment of code for NodeDAO class

The architecture and work-flow of DLD-VISU is described above with the focus on the server side. In the following sections, how to design the animation and interaction of DLD-VISU is introduced.

5.6 THE ANIMATION SYSTEM OF DLD-VISU

As an on-line animation learning system, the design of animation is a focus of DLD-VISU. The main development manners of on-line animation in the present are two manners. One of them is to make use of the products like Flash, Flex, etc. of Adobe Systems to create embedded web animation. The other is utilizing JavaScript to control web vector graphics(VG) to produce animation. The reason of why DLD-VISU selects the JS+VG see the table 5.8:

Although Flash is way better than JavaScript in the aspects of embedding multimedia and producing vector animation, such advantages are disappearing with the gradual popularization of HTML 5. For example, we have to rely on Flash to insert a video in a web page earlier, while it may be realized through the new tag <*video*> of HTML 5 today.

In summary, Flash is indeed good at creating complex vector animation. However, considering that DLD-VISU only needs to draw some simple vector shapes, such as circuit diagrams, FSM Diagram, etc., the combination of JavaScript and web vector graphics is able to meet the requirement. In addition, DLD-VISU has a high demand of users' interaction with the system, in which JavaScript is doing better obviously.

The basic idea of creating animation of DLD-VISU is to use SVG or VML to draw required vector graphics on web pages, then make use of JavaScript to control these graphics to realize the animated effects, such as moving, hiding, etc. In the following section, we accordingly explain how to draw vector graphics and employ JavaScript to control them.

5.6.1 *SVG and VML*

So far, there mainly exist three methods to show vector graphics on web pages directly.

1) VML is an XML-based file format for two-dimensional vector graphics and developed by Microsoft. VML is easy to learn and supplies lavish vector graphic elements and attributes. It is very

	Flash	JavaScript+VG
Open	Flash uses the ActionScript language to develop animation. The code must be complied into swf file which is then played by the swf player – Flash player - embedded in web pages. However, what we hope is to create animation at real-time according to the demands of students.	JavaScript is a web front-end development language. It can interact and work cooperatively with back-end applications very well.
Costs	The costs of developing Flash are very high. Because the development environments like FlashIDE, etc. are charged.	Completely free.
Ease of use	Due to a Flashplayer proxy layer in the middle, many auxiliary functions cannot be employed by Flash flexibly.	JavaScript applications own better ease of use.
Easy to develop	Programmers can only apply FlashIDE to perform development.	There are many open-source JavaScript function packages that encapsulate the operations to JavaScript animation, e.g. jQuery. Programmers may directly use these function libraries, greatly boosting the development efficiency.

Table 5.8: The Comparison Flash and Javascript+vector graphices

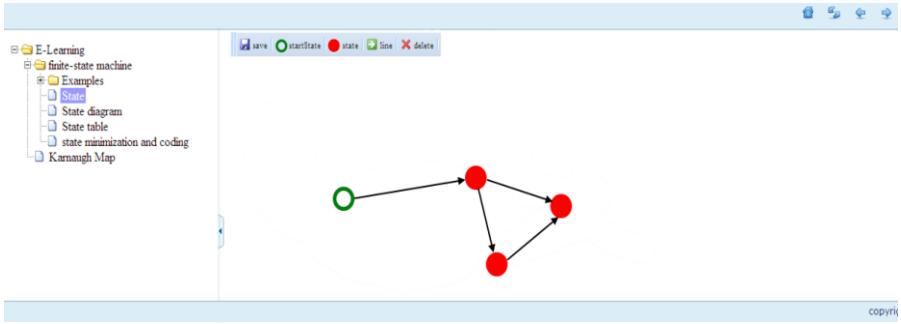


Figure 5.11: The 1st version of DLD-VISU

convenient for users to employ these elements to insert various vector graphics on web pages. An additional namespace "V" is before VML elements compared with usual HTML elements. Beyond that, there is no distinction between VML and HTML elements, which means users are able to apply JavaScript and CSS to operate VML elements like HTML elements. This just is the powerfulness of VML. However, only IE browser can present VML elements, and The IE 9.0 and later versions no longer provide support for VML.

2) Inserting SVG in web pages. SVG is an XML-based vector image format for two-dimensional graphics with support for interactivity and animation. The SVG specification is an open standard developed by the W3C since 1999. In the present, most browsers are supporting SVG other than IE 8.0 or earlier.

3) Employing <*Canvas*> tag of HTML5.0. <*Canvas*> tag allows scripting languages to dynamically render bitmap images. <*Canvas*> tag was initially just utilized inside Apple company and added into HTML5 standard later, becoming one of the new characteristics of HTML5. Today, most browsers are supporting <*Canvas*> tag other than IE 8.0 or earlier.

We started designing and developing DLD-VISU in 2010. To develop a Alpha version as soon as possible, we were using VML to carry out the front-end animation development, seeing in Figure 5.11, which was a forced choice. Because IE 8.0 or earlier versions do not support SVG. IE is a browser that held the highest market share. Moreover, the sum of the market share of IE6.0, IE7.0, IE8.0 reached 84%. In the end of 2010, Microsoft released IE 9.0 which provides the support for SVG format. After 2010, the market share of other browsers were gradually growing. This change is more obvious in universities. In many PC pools of universities, Linux operating system is used and only Firefox or Chrome is installed in this system. In order to adjust to such a change, we started considering to employ SVG in DLD-VISU.

As shown in Figure 5.12 and 5.13, in terms of the data from Net Market Share, there are 11.15% of users who still make use of IE8.0

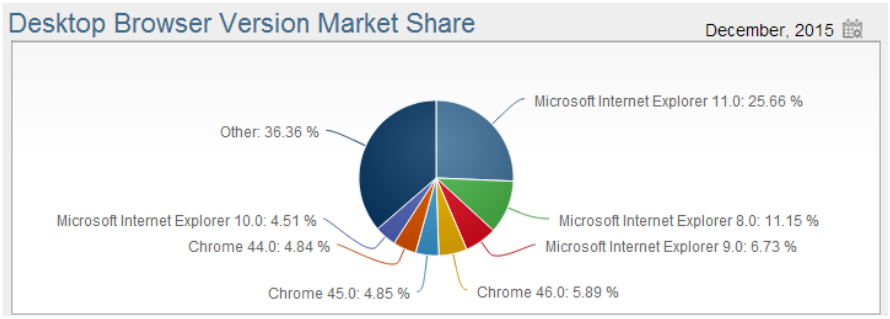


Figure 5.12: Desktop Browser Version Market Share 2015.[63]

by December 2015 and even after 2017 years, more than 3% percent of users are still using IE8, and the number of users using IE8 is still more than the total number of users using IE9 and IE10. This means these users cannot use DLD-VISU normally, if we apply SVG to make animation. Of course, we can force users to utilize one certain browser. However, it may make users so unaccustomed to or dislike this browser that they are reluctant to use it, bringing negative influence on learning effect. Besides, some new browsers cannot be installed in some old computers.

In order to avoid the above mentioned issues, we use JSXGraph library to develop DLD-VISU. JSXGraph has started as a seminar at the University of Bayreuth, there are four members in the developer team now[45]. Developer team describes JSXGraph as follows: "JSXGraph is implemented in pure JavaScript, does not rely on any other library, and uses SVG, VML, or canvas". The working principle of JSXGraph is very simple and supplies a set of JavaScript API which can be employed by programmers to create vector graphics. JSXGraph is able to translate the code written by programmers into the format supported by the current browser on the basis of the browser type and version adopted by users. For example, if JSXGraph detects users are making use of IE8.0, it applies VML elements to display vector graphics; if Chrome, it takes advantage of SVG. In the example presented in Figure 5.14, we use the board.create method provided by JSXGraph to draw a circle in browser. In fact, JSXGraph may choose to use VML or SVG to draw shapes all by itself, according to different browsers.

JSXGraph library is a pure JavaScript function library. When it is used, there is no need to make any configuration on server side. Only the file jsxgraphcore.js is imported into the jsp files that demand it. The earlier versions of JSXGraph run a little slowly while new version makes some improvement on performance. After repeatedly tests, adopting new version of JSXGraph brings no obvious impact on DLD-VISU.

Browser Version	
Chrome	56.65%
Firefox	15.56%
Internet Explorer 11	8.50%
Safari	3.43%
Edge	3.32%
Internet Explorer 8	3.06%
Internet Explorer 9	1.48%
Opera	1.41%
Sogou Explorer	1.33%
Internet Explorer 10	0.91%

Figure 5.13: Desktop Browser Version Market Share 2017.[63]

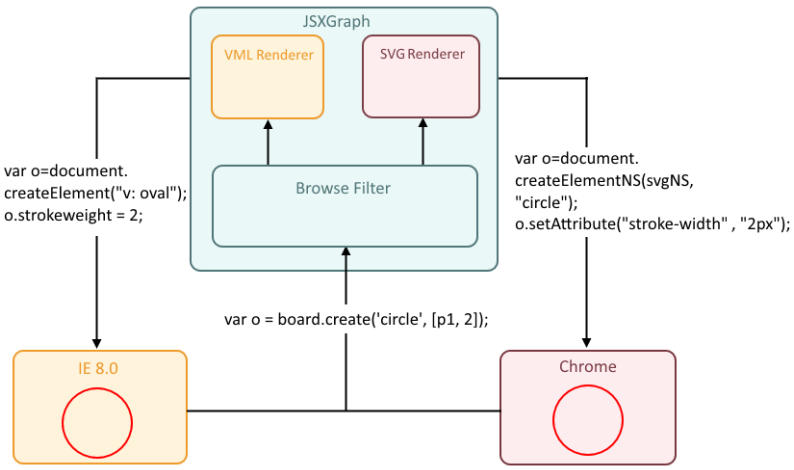


Figure 5.14: The working principle of JSXGraph.

Constructor parameters	Description
State(x)	x: The coordinate of the midpoint of state. The radius of State is the system default value.
State(x,y)	x: The abscissa of the midpoint of state. y: The ordinate of the midpoint of state. The radius of State is the system default value.
State(x,y,r)	r: The radius of State.
State(x,y,r,objName)	objName: display name on State.
State(x,y,r, objName, objActiveoutput)	objActiveoutput: when FSM is a moore FSM, Output is displayed on State.
State(x,y,r, objName, objActiveoutput, isStart)	When isStart=true, State is shown with two concentric circles. The default value of isStart is false.
State(x,y,r, objName, objActiveoutput, isStart,name)	name is the id of this graphic. Users can apply the statement getElementById of JavaScript to obtain this graphic object.

Table 5.9: The constructor list of state class.

5.6.2 *DVNL and DVEC library*

We make use of JSXGraph library to develop two graphics libraries for DLD-VISU. One of them is utilized to plot FSM Diagram and named as DLD VISU Node and Line (DVNL). The other is applied to draw circuit and named to be DLD VISU Electronic Component (DVEC). These two graphics libraries completely employ object-oriented development pattern. Each graphic Class not only contains drawing functionality, but also encapsulates some methods relating to this graphic. Furthermore, these two graphics libraries are pure JavaScript function library. They can be utilized in both DLD-VISU and other web applications. DVNL consists of two components – State and Connection – which are respectively used to draw State and Transition condition. Subsequently, it is described in details how State is used. The usage of Connection is specified in the appendix B.

State has 7 kinds of constructors which are accordingly applied to meet distinct requirements of users. The specification of these constructors are illustrated in table 5.9.

Except for supplying abundant constructors, we also bind mouse click events to State. Users can modify the method addMouseEvt to finish their own mouse events.

DVEC possesses 14 graphic components presented in Figure 5.15.

1) Logic gate: And; Or; Nand; Nor; Invert; Xor; Xnor
2) Mux
3) Decoder

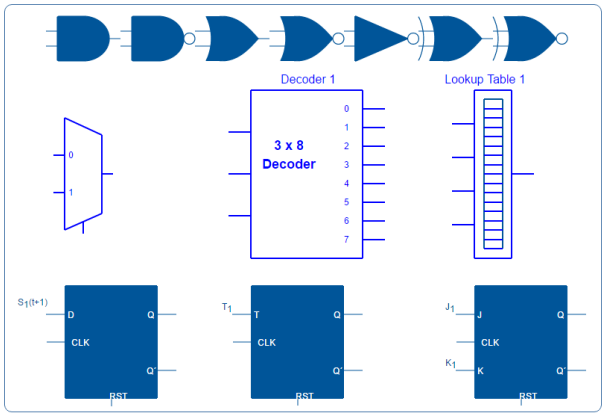


Figure 5.15: The Graphic components.

Constructor parameters	Description
AndGate ([x,y]|,Board)	x,y are the coordinate of the midpoint of the AND-gate. The length and the width adopt the system default values. Board is a drawing board object on which the AND-gate is drawn. If the Board is not designated, the AND-gate is going to be drawn on a default board.
AndGate [[x,y],width,height]|,Board)	x,y are the coordinate of the midpoint of the AND-gate. width, height are the width and height of AND-gate.
AndGate ([x1,y1,x2,y2]|,Board)	x1,y1 are the coordinate of the top left corner of the AND-gate. x2,y2 are the coordinate of the bottom right corner of the AND-gate

Table 5.10: The constructor list of And class.

4) Lookup-Table

5) Flip-Flop: D Flip-Flop; T Flip-Flop; JS Flip-Flop

6) Path

We demonstrate how to use DVEC to create a circuit graphic by introducing the constructors of AND-gate. Table 5.10 shows 3 constructors of AND-gate. DVEC encapsulates the complex process of drawing SVG. By using these constructor, users can easily customize the size and location of electronic components on the board.

Other than constructors, we also build some useful methods for circuit components. Table 5.12 lists some methods of multiplexer. DVEC is only used for displaying graphics. Therefore, we just encapsulate the methods relating to graphics display rather than the methods relating to functions, e.g. the logic function of components, which are

the tasks on the model layer. The detailed description of other components of DVNL and DVEC are presented in the appendix B.

Methods	Description
setMainAttribute (attributes)	Set the attributes of the component @param attributes {json} for example: color, highlight color...
setMuxText (text, xOffset, attributes)	Set 0,1 or 1,0 ports of multiplexer @param text {String[]} Set 0,1 text to the multiplexer. @param xOffset {int} Set the xoffset to the text. @param attributes {json} Set the color attributes to the text.
setInput (number, length, isAllHide, attributes)	Set input-line @param number {int} The number how many inputs there are. @param length {int} The length of input line. @param isAllHide {boolean} Whether hide the input line when they are created. @param attributes {json} Set the color attributes to input-line.
setOutput (length, isHide, attributes)	Set output-line @param length {int} The length of output line. @param isHide {boolean} Whether hide the output line when they are created. @param attributes {json} Set the color attributes to output-line.
setSelector (number, length, isAllHide, isAtBelow,attributes)	Set selector @param number {int} The number how many selectors there should be. @param length {int} The length of selector lines. @param isAllHide {boolean} Whether hide the selector line when they are created. @param isAtBelow {boolean} Set the selector lines upper of below. @param attributes {json} Set the color attributes to selector.
setInputText (no,text, xOffset, yOffset, attributes)	Set or create a text for a input-line @param no {int} the id number. @param text {String} The text to be created or set @param xOffset {int} Set the xoffset to the text. @param yOffset {int} Set the yoffset to the text. @param attributes {json} Set the color attributes to the text.
setOutputText (text, xOffset, yOffset, attributes)	Set or create a text for a output-line @param text {String} The text to be created or set @param xOffset {int} Set the xoffset to the text. @param yOffset {int} Set the yoffset to the text. @param attributes {json} Set the color attributes to the text.

Table 5.11: The methods list of Mux class.

Methods	Description
setSelectorText (no,text, xOffset, yOffset, attributes)	Set or create a text for a Selector @param no {int} the id number. @param text {String} The text to be created or set @param xOffset {int} Set the xoffset to the text. @param yOffset {int} Set the yoffset to the text. @param attributes {json} Set the color attributes to the text.
moveTo (position)	Move the component to a new position. @param position {array} x, y position
getInputPoint (no)	Get the outermost point of a input-line @param no {int} the id number.
getOutputPoint (no)	Get the outermost point of a output-line @param no {int} the id number.
getSelector (no)	Get the outermost point of a selector @param no {int} the id number.

Table 5.12: The methods list of Mux class.

It is very convenient to apply DVNL and DVEC to draw FSM diagram and sequential/combinational circuit diagrams on web pages. Besides, DVNL may also be utilized to plot a wide variety of directed graphs, such as the max flow/min cut graphs, the shortest path graphs in course data structure, the decision tree diagram in course machine learning etc. Figure 5.16 presents a simple example, one NAND-Gate as well as one Or-Gate are in this circuit diagram and they are connected by Path. Note, the genuine circuit diagram does not contain the dotted lines and red points in Figure 5.16. Code 5.13 shows all code that realizes this function.

First of all, we generate a drawing board in a div element whose id is *divDiagram* on web pages and set this drawing board to be default. From now on, all graphics we draw are displayed on the default drawing board. The main function of drawing boards is to set coordinate systems. DVEN applies the coordinate system of drawing boards rather than browsers to plot graphics, which makes it convenient for users to arrange the layout of circuit diagrams. The top left corner of the drawing board is the origin of the coordinate system. The abscissa ranges from 0 to n and the ordinate from 0 to n. An unit length of the DVEC default drawing board responds to 100 px in browsers. Users may also configure the mapping relationship between the coordinate systems of drawing boards and browsers on their own.

Subsequently, we draw a NAND-gate on the drawing board. In this example, the coordinates of top left corner and bottom right corner are employed to appoint the position of the NAND-gate. The coordinate system utilized here is of the drawing board. After that, we set two Inputs and color for this NAND-gate. Then, a NAND com-

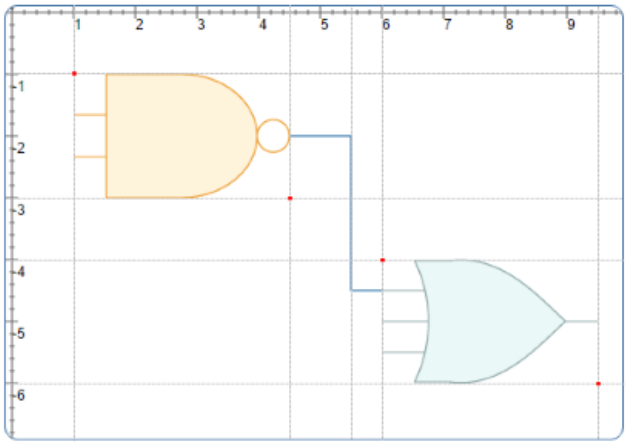


Figure 5.16: An example for DVEC

ponent is finished. From the code 5.13 we can see that DVEC does not directly create a NAND-gate components, but by first creating an AND-gate component, and then use the method *addNotCircle* to convert this AND-gate to a NAND-gate. We continue drawing a Or-Gate by using the same way and apply LogicPath to connect it with And-Gate. It can be seen from the code that we only require 3 statements to accomplish all tasks, if the default setting of DVEC (the default component is blue and includes two inputs) is employed. All components created by DVNL and DVEC are the standard DOM component, so users can use CSS and JavaScript to manipulate these components. For example, users can move a AND-gate 50 px to the right with *$("#and-gate").animate({left:'+50px'}, "slow")* statement.

```
1  //initialize a board for all graphic components.
2  SGH.b = JXG.JSXGraph.initBoard('divDiagram',
       {axis:true,originX:5,originY:5});
3  SGH.setDefaultBoardForGate(SGH.b);
4  //create a AND-gate using left-upper and right-bottom point
       coordinate.
5  var newAnd = SGH.AndGate([1,-1,4,5,-3]);
6  newAnd.setInputLine(2);//create 2 input line for this AND-gate.
7  //draw a small circel in front of the AND-gate, change this
       AND-gate to NAND-gate.
8  newAnd.addNotCircle();
9  //set thie color of the NAND-gate.
10 var gateAttr = {"color":"#FEF3DB","StrokeColor":"#EE9E2B"};
11 var lineAttr = {"color":"#EE9E2B"};
12 var circleAttr = {"StrokeColor":"#EE9E2B"};
13 newAnd.setAttr(gateAttr,lineAttr,circleAttr);
14 //create a OR-gate
15 var newOr = SGH.OrGate([6,-4,9.5,-6]);
16 newOr.setInputLine(3);
17 gateAttr = {"color":"#EAF8F7","StrokeColor":"#8EA7A8"};
18 lineAttr = {"color":"#8EA7A8"};
```

```
19 newOr.setAttr(gateAttr,lineAttr);
20 //connect the NAND-gate and the OR-gate, the path start at
       point1, it will take a vertical angle at the place with
       x_offset horizontal offest to point1 then it goes further to
       point2
21 SGH.LogicPath([4.5,-2],[6,-4.5],1);
```

Code 5.13: An Example of DVEC

5.6.3 *The introduction of jQuery and its usage in DLD-VISU*

In the last section, we have already explained how to create vector graphics on web pages. In this section, we are going to specify how to make use of JavaScript to control the movement of vector graphics, generating animated effects.

It is very convenient to apply JavaScript to control *DOM* elements on web pages. No matter a SVG graphic or a vector graphic produced by VML is indeed a DOM element. Hence, we only need to assign a unique id for each graphic element. In this way, the *getElementById* method of JavaScript can be used to obtain the control to the element with a specified id value. Then, this element can be utilized to produce animated effects by modifying its CSS properties like position, size, color, etc.

So as to reduce the complexity of code and boost the development efficiency, we take advantage of jQuery during developing animations for DLD-VISU. jQuery is an open-source JavaScript library and was established by John Resig [113]. Its design philosophy is "write less, do more". The definition of jQuery on its official website is as follows: "jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and AJAX much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript."

As shown in Figure 5.17, jQuery is the most widely used JavaScript library now (Monitor by W3Techs). The reason for its popularity is that jQuery is in possession of the following characteristics:

1) **Light weight:** jQuery has very small volume. Its compressed version is less than 80k (v1.12). Compared with other JavaScript libraries and frameworks (e.g. Extjs), jQuery is much more ligh-weighted. In this way, the loading speed of web pages may not be affected.

2) **powerful DOM selectors:** jQuery stands out from various JavaScript libraries and frameworks because of its very powerful selectors. The selectors of jQuery have the subsequent features:

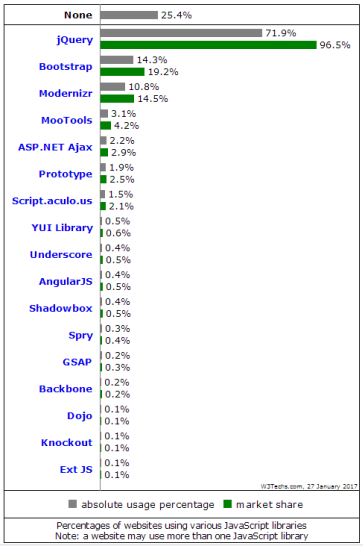


Figure 5.17: Usage of JavaScript libraries for websites [106]

a) **Concise code:** Code 5.14 presents the comparison of the code that respectively uses jQuery selectors and JavaScript native method to acquire DOM elements.

b) **Improving event-handling mechanisms:** While using JavaScript to get DOM objects, it is necessary to write an event-handling function for each operation that possibly goes wrong. For example, before obtaining a DOM object, it is needed to search HTML document and check whether this object is contained. If the above step is omitted and the DOM object does not exist, the browser is going to report an error and JavaScript code stops running. In contrast, there is no need to worry about such an issue while applying jQuery. Even though the DOM object does not exist, the browser is not going to report errors. Because jQuery already intercepts and handles this problem.

3) **Prominent encapsulation of DOM operations:** jQuery encapsulates the commonly used operations on DOM elements into functions. For example, users may apply append method to insert new objects into an appointed DOM object.

4) **Perfect encapsulation of AJAX:** jQuery performs 3-layered encapsulations to AJAX. Their complexity decreases layer by layer. The top encapsulation only demands one statement to accomplish AJAX operation. Users are able to choose a proper encapsulation to adopt based on their own requirements.

5 **Outstanding compatibility with browsers:** jQuery supports all browsers now.

6 **Linked operations:** jQuery applies the popular linked operations and supports to write multiple operations in one line of code, making the code more concise and elegant.

7 **Real separation of scripts and pages:** while using JavaScript, it is needed to insert event-response code into web pages, resulting in the mix of script code and page content and reducing the re-usability of pages. While using jQuery selectors, events can be dynamically bound to the DOM element after loading the page, completely isolating the script from the page. As displayed in Code 5.14, when traditional JavaScript statements are utilized to bind a click event to a button, the statement "onclick = functionName" must be inserted into the button tag. On the contrary, when jQuery is adopted, only the method *$("#id").click (function())* is employed to realize the binding. At the mean time, JavaScript code does not exist in the button tag any more.

```
1 $("#id");   //using JS: document.getElementById("id")
2 $("tagname");  //using JS: document.getElementsByTaName("id")
```

Code 5.14: Comparison the Selector of jQuery and JavaScript

In DLD-VISU, we use jQuery to accomplish three tasks:

1) Making use of the animation functions provided by jQuery to create animations.

2) Making use of the AJAX functions provided by jQuery to realize asynchronous interaction.

3) Separation of JavaScript code and HTML code.

The part about AJAX is introduced in next section. In the following, we specify how to utilize the animation functions of jQuery to develop animation for DLD-VISU.

In the second chapter, we formulate three criteria – Keeping overview, Abstraction, and Traceability – so as to achieve Epistemic fidelity theory. Thereinto, Keeping overview requires that we update progress chart at any time. In order to accomplish this function, we design three different CSS styles for each step of progress chart, respectively representing the finished step, the current step, and the unfinished step. When users click the "next" button, the appearance of steps varies by changing the CSS styles relating to steps. The specific code is displayed in Code 5.15 and the addClass and removeClass methods of jQuery are employed.

The Abstraction principle requires that we just keep the information that is relevant to the current step on pages and hide all irrelevant information. The hide and show method of jQuery are applied to

accomplish this job. Code 5.16 demonstrates when users click "next", state diagram is hidden, whereas FSM Geneal Architecture is shown.

```
function loadnext(divout,divin){
  ......
  $('#step' + divin).addClass('current');
  $('#step' + divout).removeClass('current');
}
```

Code 5.15: The function for changing the current step of the flow-chart

```
$('#delay').animate({top:"5px"},1000,function(){
  $('#divStateTable').slideUp(500,function(){
    $('#divStateTable').css('height','1%');
    $('#divStateTablePanel').hide();
    $('#divSelectFF').show();
    $('#divController4').show().animate({top:
        "120px",left:"5%"}, 1500,function(){
      $(".architecture-box").css("background-position","0px
          -150px");
      $('#n4').attr("disabled","");
    });
  });
});
```

Code 5.16: Hide and show some DOM elements using jQuery

According to the Traceability principle, if a mapping relationship between page contents exists, we need to supply methods so that students are able to observe this relationship conveniently. For example, in the sixth step of FSM flow, a circuit diagram is displayed in the middle position of browser and the corresponding equations are on the top. When students click any one And-Gate in the circuit diagram, the whole part relating to this And-Gate is highlighted with red.

In order to achieve the references among DOM elements, we design an algorithm: adding a ref property for elements and setting values of ref properties to be the IDs of referenced elements. In the above example, the ref value of each and-Gate is the id of its implicant. When students click and-gate, the system gains the ref value of the current element, finds the element with id that has the same value of ref, and highlights this element.

Through three examples above, we describe how to generate animation on browsers using jQuery. In next section, we explain how DLD-VISU implements the asynchronous interaction with users.

5.6.4 *Introduction of AJAX and its usage in DLD-VISU*

In the traditional page refresh mode, every time users click links or buttons, browsers reload the corresponding pages, which not only wastes web bandwidth but also causes the interruption of user experience. In a on-line teaching system, such an interruption may bring bad impact on the coherence of thinking or distract students' concentration. Meanwhile, delaying in each interaction between students and the system is also wasting time and diminishing students?interest to learn. So as to avoid such a case, we hope when only a part of web page content varies, it is unnecessary to reload the whole page but only refresh the necessary part. In the past, all we can do for this was resorting to Flash or Java applet, whereas people adopt the AJAX technology today.

AJAX is the abbreviation of Asynchronous JavaScript and XML. Jesse James Garrett mentioned this concept for the first time in his article "AJAX: A New Approach to Web Applications" in February 2005 [28]. AJAX is not a new technology rather a new method that is able to apply several existing technologies.

1) XHTML and CSS for presentation
2) The DOM for dynamic display of and interaction with data
3) HTML, JSON or XML for the interchange of data, and XSLT for its manipulation
4) The *XMLHttpRequest* object for asynchronous communication
5) JavaScript to bring these technologies together

As shown in Figure 5.18 a, in a traditional web application interaction, every time users trigger a HTTP request that is sent to server, clients can only wait idly when server processes this request. In addition, no matter simple data or a whole page requested by users, server is going to return a whole HTML page after processing. In this way, it takes time and bandwidth for browser on the client side to reload the whole page. On the contrary, AJAX applications can just fetch required data from server and process them on client side through JavaScript. Because server does not return whole pages any more and there is also no need for browsers on the client side to refresh the whole pages, the data switched between server and browser is greatly reduced. Thus, the responses can be obtained faster than before.

The working principle of AJAX: As presented in Figure 5.18 b, the working principle of AJAX is to insert a middle layer (AJAX engine) between users and server to asynchronism users' operations and responses of server. The core of AJAX comprises JavaScript, *XMLHTTPRequest*, and DOM object. Browsers send asynchronous requests

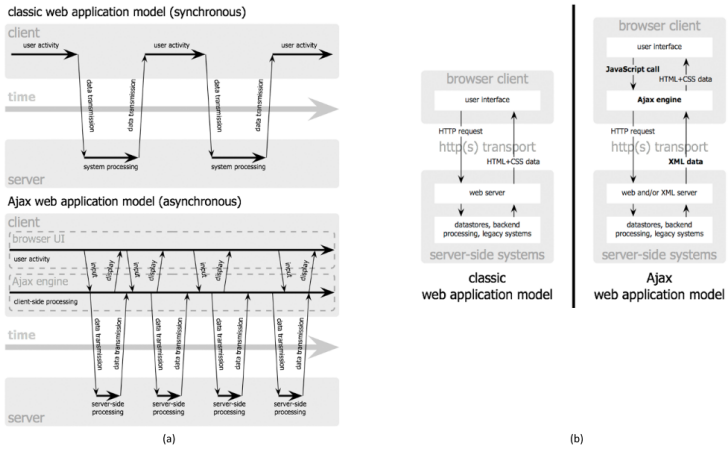


Figure 5.18: The synchronous and asynchronous pattern[28].

to server through *XmlHttpRequest* object and apply JavaScript to embed the data obtained from server in the DOM object relating to the current page.

It is worth mentioning that JSON is employed as the transmission format of data in practical work, although the X of AJAX represents XML. JSON is a native format of JavaScript. Therefore, JavaScript supplies the functions that make JSON data and JavaScript objects interchangeable. While using XML to transport data, users need to write their own functions to realize the functions of resolution and interchange.

During utilizing AJAX to transport data, *XMLHTTPRequest* object plays a key role. *XMLHTTPRequest* is an extension of JavaScript and was originally an *ActiveX* component of IE5. It is not a standard of W3C. Hence, distinct browsers have different manners to create *XMLHTTPRequest* objects. IE6 and earlier versions are to generate an *ActiveX* object. IE7 and later versions as well as other browsers are to create a local JavaScript object. To adapt to different browsers, checking the type and version of browsers during programming is a must. In DLD-VISU, the AJAX function provided by jQuery is employed to perform AJAX operations. Thus, it is not necessary to carry out the above mentioned check about browsers. Besides, the code is far less than that using the native AJAX function of JavaScript. Code 5.17 shows an example which utilizes the jQuery AJAX function to dynamically display state code. As presented in Figure 3.20, the page can show results without refreshing after students choose an encoding manner.

```
1 function getCodeTable(){
2   var url;
3   var val = $('input:radio[name="code"]:checked').val();
4   if(val=="0"){
5     code = "binaryCode";
6     url='getBinaryCode.action';
7   }else if(val=="1"){
```

```
8       code = "onehotCode";
9       url='getOneHotCode.action';
10    }else if(val=="2"){
11      code = "garyCode";
12      url='getGrayCode.action';
13    }
14    changeCodetoName();
15    $('#error').hide();
16    $('#divStateCode').hide();
17    $('#divStateCode1').hide();
18    var pars={drawNodeAndLineId:$('#drawNodeAndLineId').val()};
19    $.post(url,pars,setStateCodeTable);
20  }
21  function setStateCodeTable(data){
22    $('#divStateCode').html(data);
23    $('#stateTitle').html("<s:property
          value='getText(\"statediagram.state\")'/>");
24    $('#codeTitle').html("<s:property
          value='getText(\"code\")'/>");
25    $('.codeResult').hide();
26    $('#divStateCode').show(1000);
27
28  }
```

Code 5.17: Utilizes the jQuery AJAX function to dynamically display State code

Although using AJAX can enhance user experience, the asynchronous transmission way is utilized for interactions. Therefore, the order and mutual impact of the interactions must be considered while employing AJAX to interact on a same page. Then, programmers are demanded to be cautious of designing interaction flow during development so that conflicts can be avoided. For example, the FSM learning procedure of DLD-VISU is partitioned into 10 steps. Except for the first step, the others are implemented through AJAX technology on a same page. During developing this part, we spent plenty of time in testing all possible interactions.

Furthermore, in the development with AJAX, network delay – the time interval from the moment users send requests to the moment server responds – needs to be taken into consideration discretely. If no clear response is given to users or data is not read properly, it makes users feel delay, which is not the hope of users and they also cannot understand. The solution is to apply a visualizing component to inform users that the system is performing back-end operations.

At last, the advantages and disadvantages of AJAX are summarized as follows.

Advantages:

1) **Updating data without refresh.** The biggest advantage of AJAX is to communicate with server and maintain data under the con-

dition that pages are not refreshed, making web applications respond faster to users' requests, avoiding to transmit useless information in web, reducing waiting time of users, and bringing very good user experience.

2) **Asynchronously communicating with server.** AJAX employs an asynchronous way to communicate with server and does not interrupt the operations of users, resulting in the capability of faster response. It optimizes the communication between browser and server and reduces the transmission time of unnecessary data as well as lowers on-line data volume.

3) **Load balance between front- and back-end.** AJAX is able to transfer some jobs performed by server to client and make use of the idle resources of client to deal with them so that the burden and bandwidth of server can be reduced. The principle of AJAX is "fetch data based on needs", which may at the extreme diminish the burden of server caused by redundant requests and responses as well as boost performance of sites.

4) **The standard based by AJAX is widely supported.** All technologies applied by AJAX are standard. There is no need to download browser plugins or applets.

5) **Separating UI and applications.**

Disadvantages

1) **AJAX removes the functions of Back and History that ruin the mechanisms of browsers.** Users usually hope to cancel their previous operations by clicking back button. However, this function cannot be achieved in AJAX applications. This issue can be solved indeed (e.g. GMail adopts a hidden *iFrame* to deal with this issue). But it leads to very high development costs and deviation from rapid development required by AJAX. This is a severe problem brought by AJAX.

2) **Security of AJAX.** AJAX technology brings very good user experience and also new security threats to IT enterprises at the same time. AJAX technology serves to establish a direct channel to back-end data, which makes developers inadvertently expose more data and business logic than earlier.

3) **Deviating from the original intention of URL and locating resources.** When users utilize AJAX to communicate with server, URL in the address bar of browsers does not vary. Thus, when a user sends a URL to another user, they both might see different contents.

4) AJAX **does not have very good support for mobile devices.** Some portable devices (e.g. mobile phones, PDA, etc.) are still not supported very well.

6

ANIMATION DEVELOPMENT USING DLD-VISU

In the Chapter 2, some reasons for why some teachers hold the opposite idea to introduce AV tools were mentioned. One of them is that AV software only have very single function, making teaching requirements not able to be covered very well. For example, the two websites that demonstrate the AVL algorithm listed in the Chapter 2 only provide the animation of the AVL algorithm but no other algorithms which are required in the lecture "data structure", such as the different kinds of sorting algorithms and the shortest path algorithms. Of course, teachers can also find some other AV tools to animate these algorithms on Internet. However, different developers of AV tools have different objectives. So the GUI and visualisation of these AV software might be accordingly distinct, resulting in the increasing difficulty of students' self-study, distracting them, and lowering their initiatives of self-learning.

Some teachers tend to carry out further development of these existing AV tools or tailor them according to their own demands so that these AV software could meet their teaching requirements better. Meanwhile, the GUI would keep unchanged. However, it is hard to make this wish come true. First of all, some AV tools are not open source. Thus, it would be difficult for teachers to get the source code. Secondly, even though teachers are able to obtain the source code, the costs of secondary development would be very high. Because the development languages of these AV tools are very distinct. Yet DLD-VISU solves such problems very well. DLD-VISU is not only an E-Learning tool that supplies DLD related knowledge for students so that they can learn by themselves but also a development platform of creating AV animations. Teachers can apply all the library functions provided by DLD-VISU to conveniently build the teaching animations that can meet their own requirements.

DLD-VISU simplifies the AV animation development by teachers in two aspects to a great extent. In the framework aspect, developers can employ the SSH2 framework that is already deployed by DLD-VISU to construct the AV animations, which saves a lot of development time. The advantages of applying DLD-VISU to develop web animations are listed as follows.

1) It spares developers lots of time by directly utilising the MVC framework provided by DLD-VISU to develop and extend AV software instead of building their own MVC framework.

2) DLD-VISU supplies the ORM function, which makes it convenient for developers to perform various object-oriented operations to databases.

3) DLD-VISU furnishes the *Dependency Injection* function. Developers are able to conveniently manage the dependencies amongst classes through a single configuration file. In this way, the complicated dependencies between classes become clear, leading to a higher development efficiency and better readability of programs. Such points are significant for the projects in which many persons collaborate und that need to be extended very often.

In the aspect of creating front-end animations, developers may adopt the abundant library functions offered by DLD-VISU to build the front-end animations. A short explanation of the DLD-VISU library functions is presented subsequently. A concrete example is demonstrated in the section 6.3.

1) A set of methods to deal with the progress chart:
   a) The method *setChart* can be used to generate a process chart or a progress sub-chart. The user saves the name of each step to an array, *setChart* method according to this array to generate the corresponding chart.
   b) The method *setStep* is applied to manipulate the current step. The method highlights the text in current step and turns all steps before current step grayed out.
   c) The method *showSubChart* can be used to show a sub-chart.
   d) The method *hideSubChart* can be used to hide a sub-chart.

2) A set of methods visualise the data obtained from the back-end into tables and displays them in the front-end.
   a) The method *setTable* can be used to generate a table according to a two-dimensional array from server side.
   b) The method *mergeRowAndColumn* is applied to merge rows or columns.
   c) The method *getTable* can be used to get HTML code from a two-dimensional array.
   d) The method *highlightRow* can be used to set or remove highlight of a row.
   e) The method *highlightColumn* can be used to set or remove highlight of a column.

3) The graph theory is used a lot in many lectures in the field of computer science, such as the shortest path algorithms, the

minimum flow algorithms, and binary-tree algorithms of data structure as well as Petri-net of computer hardware basics, etc. Developers can use the DVNL function library provided by DLD-VISU to create the animation of directed graphs. In the section 6.3, a concrete example is shown to explain how DVNL works.

4) DLD-VISU affords the DVEC function library. That is put to use to draw diverse circuit components on websites. DVEC has so far already supported seven gate circuit components, Mux, Decoder, and Lookup-table.

5) DLD-VISU even encapsulates the Self-Assessment function. Thus, it is convenient for developers to accomplish the secondary development of the Self-Assessment function.

6) DLD-VISU provides spell checking for boolean functions.
   a) The method *validateDnf* is applied to check the boolean function in DNF form.
   b) The method *validateKnf* is applied to check the boolean function in CNF form.
   c) The method *validateCf* is applied to check the boolean function in Canonical Form (CF) form.
   e) The method *validateNsf* is applied to check a normal boolean function.

In this Chapter, it is demonstrated how to use DLD-VISU to rapidly develop the animation of the decision tree algorithm of machine learning. The decision tree algorithm is introduced in the section 6.1 and the pedagogical design concept relating to the decision tree development is explained in the section 6.2. In the section 6.3, it is highlighted how to apply the function library and frameworks provided by DLD-VISU to achieve the animation developments. At last, the presentation process of the whole animation is illustrated.

6.1 THE INTRODUCTION OF DECISION TREE

The decision tree algorithm is one of the classic algorithms in the field of machine learning, which is one of the basics to study machine learning. Through a series of rules, it uses a group of classified data to create a decision tree and then takes advantage of this created decision tree to proceed with the class of the remaining unclassified data. A data point contains multiple features, some of which play a key role for classification. The construction of decision trees is to find such features that have the decisive effect and build an upside-down tree according to their determination degree. Note that the feature which has the highest determination degree is put as the root. The

next step is to search for the feature with the second highest determination degree in the sub-datasets of the individual branches recursively until all data in the sub-datasets belong to a same class.

Therefore, the process of constructing decision tree is, in fact, a process of classifying datasets recursively in accordance with data features. The first problem needed to be solved is to determine which features in the current dataset have the highest decisive effect for classifying data. In order to find such decisive features to obtain the best results, each feature in the dataset must be evaluated to pick up the best feature for the data classification. After evaluation, the original dataset is divided into several sub-datasets which distribute in all the branches of the first determination node. If all data in a branch belong to a same class, then this branch is finished processing. Now this branch is a leaf node and its class is decided. If the data in a branch do not belong to a same class, the process of classifying sub-datasets is repeated continuously until all data that exist in this sub-dataset have a same class. The algorithm to classify sub-datasets is same as to classify the original dataset.

It can be seen from the above analysis that the key of the decision tree algorithm is to find the most important data features. As evaluation criteria vary, different decision tree algorithms come along. DLD-VISU has realised the animations of three of them - ID3, C4.5, and the Cart algorithm.

6.1.1 *ID3 algorithm*

Iterative Dichotomiser 3 (ID3) is an algorithm invented by Ross Quinlan[84] used to generate a decision tree from a dataset. ID3 algorithm utilises **information gain** as the standard to measure the determination degree of features. Before introducing information gain, a concept **information entropy** used commonly in information theory needs to be explained firstly. Information entropy is defined as the average amount of information produced by a stochastic source of data. The more uncertain a data is, the bigger its entropy becomes. The information entropy expression of a random variable x is as follows:

$$H(x) = -\sum_{i=1}^{n} p_i log_2 p_i$$

where n notates that x has n distinct values, p_i is the probability when x = i, and the basis of log is usually 2 or e.

After making the definition of information entropy clear, it is easy to get the entropy of a dataset S is:

$$Entropy(S) = -\sum_{i=1}^{n} p_i log_2 p_i$$

While using the feature A to classify the dataset S, the corresponding entropy(A) is:

$$Entropy(S, A) = -\sum_{i=v_1}^{v_n} \frac{|S_i|}{|S|} Entropy(S_j)$$

where |S| is the total number of dataset , v_1, ...,v_n are values of feature A, S_i is the number of dataset whose value is equal to v_i at feature A.

Thus, Gain(A) is:

$$Gain(S, A) = Entropy(S) - Entropy(S, A)$$

ID3 algorithm is calculating the gains of all features in a dataset and employing the feature with the biggest gain to classify the dataset. All the elements in each dataset S_i are checked and evaluated whether they are in a same class. If all the elements are in a same class, this sub-dataset is not divided again. Otherwise, the gains of all the features in this sub-data are calculated and use the feature with the biggest gain to classify this sub-dataset. The above steps are repeated until all the elements in each sub-dataset belong to a same class.

ID3 algorithm calculates the gains of all features in a dataset and selects the feature with the biggest gain to classify the dataset. All the elements in each dataset S_i are checked and evaluated whether they are in a same class. If the answer is a "yes", it is unnecessary to classify this sub-dataset further. Otherwise, the gains of all the features in this sub-dataset are calculated and the feature with the biggest gain is adopted to classify this sub-dataset. This *calculation-selection-classification* process (the above described steps) is repeated until all elements in each sub-dataset belong to a same class.

6.1.2 *C4.5 algorithm*

The principle of ID3 algorithm is pretty simple and it is easy to implement. But there also exist some flaws at the meantime. Such as:

1) Using gain as the standard to evaluate features makes ID3 algorithm more prone to select the features with more values. Gain reflexes how much the uncertainty decreases after a certain condition is given. It is very apparent the finer a feature is divided, the higher its certainty becomes accordingly, that is, the corresponding gain grows with the decrease of the information entropy. Therefore, the features with more values have bigger gains under some circumstances.

2) ID3 algorithm can only handle discrete features.

Given the above mentioned shortcomings of ID3, Ross Quinlan made some improvements to it. The improved algorithm is named

The ordered list of values of a feature	The candidate break point
3.2	-
6.7	4.95
8.9	7.8
11.4	10.15
16.0	13.7

Table 6.1: The List of candidate break points

as C4.5 which applies **Gain Ratio** as the criterion to evaluate features and conquers the flaws of ID3 algorithm - using gain to classify features results in the tendency to choose the features owning more values. The equation of **Gain Ratio** is:

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitInformation(S, A)}$$

$$SplitInformation(S, A) = -\sum_{i=v_1}^{v_n} \frac{|S_i|}{|S|} log_2 \frac{|S_i|}{|S|}$$

where |S| is the total number of dataset , v_1, ...,v_n are values of feature A, S_i is the number of dataset whose value is equal to v_i at feature A. On the contrary to ID3 algorithm which is only able to process discrete data, C4.5 can handle successive features. The concrete steps are:

1) Sorting samples or sub-datasets of samples in an ascending order according to the values of the successive variables

2) Assuming that a feature has *n* continuous values. Then there are *n-1* candidate break points. The value of each candidate break point is the midpoint of two continuous variables of the features sorted in the step 1, as presented in Table 6.1. For example, there are two values: 6.7 and 8.9. Their candidate break point is (6.7+8.9)/2 = 7.8 accordingly. In accordance with this break point, the original successive features can be divided into two groups — one including the features that are bigger than 7.8, the other having the left features which are smaller than or equal to 7.8. In this way, the discretization of the feature is realised. After that, the best break point is picked out by calculating the **Gain Ratio** of each break point.

6.1.3 *CART algorithm*

CART algorithm utilises **Gini Index** as the criterion to measure impurity. **Gini Index** is defined as:

$$Gini(S) = 1 - \sum_{i=1}^{n}(\frac{|C_i|}{|S|})^2$$

where **n** notates that the dataset **S** has n classes , C_i is the number of the data of class i.

If the Dataset S is divided into two parts, S_1 and S_2, according to a feature A, the corresponding Gini(S,A) is:

$$Gini(S, A) = \frac{|S_1|}{|S|}Gini(S_1) + \frac{|S_2|}{|S|}Gini(S_2)$$

Differing from ID3 and C4.5 algorithms, the CART algorithm considers all dichotomies of each feature. If a feature A has v possible values, there would be 2^v possible sub-datasets. For example, the feature *"income"* in Table 6.2 owns three possible values {low, medium, high}. The possible sub-datasets are correspondingly {low, medium, high}, {low, medium}, {low, high}, {medium, high}, {low}, {medium}, {high} and {}. Additionally, {} and {low, medium, high} among them are not taken into consideration because they do not produce division. Thus, 2^{v-2} possible dichotomies exist for the feature *"income"*. The CART algorithm needs to calculate the *Gini indices* of all dichotomies of each feature and choose the smallest one to be the *Gini index* of the feature. After determining the *Gini index* of all features, the CART algorithm employs the feature with the smallest *Gini index* as the root of decision tree and builds the left/right sub-tree by using the dichotomy of the feature whose *Gini index* is the smallest. For example, the *Gini index* of the feature *"income"* is the smallest compared with other features. Then the feature *"income"* is selected as the root. The *Gini indices* of its all sub-datasets are listed in Table 6.3, among which the *Gini indices* of the sub-datasets {medium, high} and {low} are the smallest($\frac{10}{14}(1 - (\frac{7}{10})^2 - (\frac{3}{10})^2) + \frac{4}{14}(1 - (\frac{2}{4})^2 - (\frac{2}{4})^2) = 0.443$). Therefore, the left and right sub-trees of decision tree contains the data from the dataset S whose *"income"* features have the value medium, high and low, respectively.

6.1.4 *summary*

ID3, C4.5 and CART are three of the most common decision tree algorithms. They corresponds to three different feature selection criteria individually - *Gain*, *Gain Ratio*, and *Gini index*. Aside from the distinct criteria, the other parts of these three algorithms are same. The pseudo code is presented in Code /refcodeDT.

```
Input: The dateset of training: D ={{x1,y1},{x2,y2}....{xn,yn}};
Input: The List of Features: A ={a1,a2...an};
Output: The decision tree T.
TreeGenerate(D,A):
generate node;
```

age	income	student	credit_rating	buys_computer
youth	high	no	fair	no
youth	high	no	excellent	no
middle	high	no	fair	yes
senior	medium	no	fair	yes
senior	low	yes	fair	yes
senior	low	yes	excellent	no
middle	low	yes	excellent	yes
youth	medium	no	fair	no
youth	high	yes	fair	yes
senior	medium	yes	fair	yes
youth	medium	yes	excellent	yes
middle	medium	no	excellent	yes
middle	high	yes	fair	yes
senior	medium	no	excellent	no

Table 6.2: An example of the training Set

Group 1	Group 2	Gini index
	low, medium, high	-
low	medium, high	0.450
medium	low, high	0.458
high	low, medium	0.443
low, medium	high	0.443
low, high	medium	0.458
medium, high	low	0.450
low, medium, high		-

Table 6.3: All dichotomies of a Feature

```
IF all date in D belong to the same class C
  mark node as a leaf with the class C
  return
END IF
IF A == null
  mark node as the class C1, which C1 is the class with the most
      members.
  return
END IF
pick out the best feature a* from A
FOR EACH value a in a*
  create a branch for node to notate Dv as a sub-dataset of D
      where value of a* is a
  IF Dv == null
    continue;
  ELSE
    A = A\{a*};
    TreeGenerate(Dv,A);
  END IF
END FOR
```

Code 6.1: The pseudo-code for decision tree algorithm.

6.2 PEDAGOGICAL DESIGN CONCEPT

DLD-VISU was developed based on four pedagogical theories. So it is the first thing to consider how to reflex these four theories during designing the animation for decision tree.

6.2.1 *Epistemic Fidelity*

Through analyses of the algorithm, it is found that decision tree is a pretty complex recursive algorithm. Meanwhile, some complicated calculations are also included. In order to make students master this complex process quickly, it is reckoned that the three key principles (*Keeping overview*, *Abstraction*, and *Traceability*) are suitable for the animation development of decision tree.

Keeping overview: A progress chart is placed on the top of the web page, as shown in Figure 6.1, students can keep track of how far the algorithm proceeds via this progress chart, which is especially important for students to understand recursive algorithms.

Abstraction: The step 3 is partitioned into n small steps according to the number of nodes in decision tree that is n, as illustrated in Figure 6.5, it utilises animations to demonstrate how to determine the current node in each step. In each sub-step, only the data and tables that are relevant for computing the current node are displayed on the page. For example, the Attribute Table on the upper right corner of

the page varies with different nodes, leading students to focus on the calculation of the current node.

Traceability: The objective of decision tree algorithm is to classify data. Under such circumstances, how to keep trace of and observe data conveniently becomes very significant. As the animation plays, the current node of decision tree on the bottom left of the page is highlighted with yellow. At the same time, the related data in the Training Set Table on the upper left are also stressed. In this way, it is pretty easy for students to watch the animations and observe the data.

6.2.2 *Cognitive Constructivism*

In order to make students learn actively, the Self-Assessment system is added during the animation development of decision trees. Calculating the evaluation standards - *Gain*, *Gain Ratio*, and *Gini index* - is one of the difficulties in these three decision tree algorithms. Students are able to master the calculation only by practising the calculation manually and repeatedly. As the animations play, the corresponding results are not shown directly. Students can fill their own calculating answers in the pages and click the "Verify" button to compare them with the correct answers.

6.2.3 *Dual Coding*

As displayed in Figure 6.6, animations are employed to present not only the process of constructing decision tree but also the formulas and computing processes. The formulas of computing *Gain*, *Gain Ratio*, and *Gini index* are sophisticated, which includes a number of logarithms and fractions. A better displaying form $\frac{n}{m}$ is adopted to express fractions instead of the common version on websites n/m so that students can better observe and understand the formulas.

6.2.4 *Individual Differences*

Considering the distinct requirements of students who own the knowledges of different levels, the diverse input/output manners are designed. Students can define the quantity of features and the size of datasets in terms of their own demands. The program has supported 2-4 features so far. Each discrete feature may support up to three distinct values. Additionally, successive features are also supported. It is reckoned that more features and values only increase the complexity of decision trees and can actually not bring more help for understanding the decision tree algorithm.

Furthermore, the program allows students to create decision trees via importation. So far, the ARFF format of files has been supported.

An ARFF file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the machine learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software. ARFF format is described in the data mining book written by Ian H. Witten and Eibe Frank [37]. Weka is one of the most widely applied software in the field machine learning. It is very convenient for students to search for the data in ARFF format, relieving them from boring input and making them focus on the decision tree algorithms themselves.

After an animation of decision tree algorithms is finished, the system stores the generated decision tree into an XML file, the format of which is shown in Figure 6.n. Students can download such XML files any time.

```
<root>
  <DecisionTree value="null">
    <outlook value="sunny">
      <humidity value="high">no</humidity>
      <humidity value="normal">yes</humidity>
    </outlook>
    <outlook value="overcast">yes</outlook>
    <outlook value="rainy"/>
      <windy value="FALSE">
        <temperature value="hot">yes</temperature>
        <temperature value="mild">yes</temperature>
        <temperature value="cool">yes</temperature>
      </windy>
      <windy value="TRUE">yes</windy>
  </DecisionTree>
</root>
```

Code 6.2: The XML export file of a decision tree.

6.3 IMPLEMENTATION OF THE E-LEARNING TOOLS OF DECISION TREE

As described previously, DLD-VISU is developed on the basis of MVC architecture pattern — *Model*, *Controller*, as well as *View*. The developments of these three parts are explained subsequently.

6.3.1 *Model*

Two tasks are performed in the Model layer: the concrete implementation of the decision tree algorithm and data persistence.

Three Java classes are used - ID3.java, C45.java and CART.java - to accomplish the implementation of these three decision tree algorithms. The pseudo code is presented in the last section. The basic

details of decision tree algorithms are not explained here. Because the focus of this section is to specify how to take advantage of the DLD-VISU framework.

Note that some additional methods are defined to store and transfer the intermediate steps so that they can be well played in the *View* layer, apart from the implementation of the algorithms themselves.

Except for the concrete decision tree algorithms, the students' input datasets also need to be saved. Since the configuration of Hibernate and Spring in DLD-VISU is already achieved. The development of this part becomes very easy. The steps are as follows:

1) Create a table *train* in the database to store the datasets of students' input or import.

2) Define *Train* Class and set variables according to the columns in the table *train*, i.e. there exist an one-to-one mapping between the variable names in *Train* Class and the columns in the table *train*. Moreover, create getter and setter methods for these variables.

3) Define *TrainDAO* Class and write the methods — add, update as well as delete — that perform actions to database in this class. As displayed in code 6.3, it is unnecessary to concretely define how to carry out operations to database. DLD-VISU encapsulates such functionality already. For example, one piece of record is able to be inserted through one statement *this.getHibernateTemplate().save(train)*.

4) Define *TrainServiceImpl* Class to encapsulate DAO class, see Code 6.4. The task of this class is to separate DAO layer and application layer. This step could be omitted for improving the development efficiency.

5) Insert the statements displayed in code 6.5 into the file applicationContext.xml.

```
1  public class TrainDAO extends HibernateDaoSupport{
2    public void addTrain(Train train){
3      this.getHibernateTemplate().save(train);
4    }
5
6    public void updateTrain(Train train){
7      this.getHibernateTemplate().update(train);
8    }
9
10   public void deleteTrain(Train train){
11     this.getHibernateTemplate().delete(train);
12   }
13
14
```

```
15    public Train getTrain(String trainId){
16      Criteria criteria =
            this.getSession().createCriteria(Train.class);
17      criteria.add(Restrictions.eq("trainId", trainId));
18      return (Train)criteria.uniqueResult();
19    }
20  }
```

Code 6.3: The code of TrainDAO.

```
1  public class TrainServiceImpl implements ITrainService {
2    TrainDAO trainDAO;
3    public void addTrain(Train t) {
4      this.trainDAO.addTrain(t);
5    }
6    public void updateTrain(Train t) {
7      this.trainDAO.updateTrain(t);
8    }
9    public void deleteTrain(Train t) {
10     this.trainDAO.deleteTrain(t);
11   }
12   public Train getTrain(String trainId) {
13     return this.trainDAO.getTrain(trainId);
14   }
15   public TrainDAO getTrainDAO() {
16     return trainDAO;
17   }
18   public void setTrainDAO(TrainDAO trainDAO) {
19     this.trainDAO = trainDAO;
20   }
21 }
```

Code 6.4: The code of TrainServiceImpl.

```
1  <property name="mappingResources">
2    <list>
3      <value>com/org/model/Train.hbm.xml</value>
4    </list>
5  </property>
6  <bean id="trainDAO" class="com.org.dao.TrainDAO">
7    <property name="sessionFactory">
8      <ref bean="sessionFactory"/>
9    </property>
10 </bean>
11 <bean id="trainService"
       class="com.org.service.impl.TrainServiceImpl">
12   <property name="trainDAO">
13     <ref bean="trainDAO"/>
14   </property>
15 </bean>
```

Code 6.5: The insertion of new statements into applicationContext.xml.

After the above steps are accomplished, *TrainServiceImpl* can be used to perform the actions, such as addition, deletion, modification, search, etc. to the table *train*. It can be seen that DB operation modules may be constructed very rapidly by taking advantage of the DLD-VISU framework. There is no doubt that the development time can be greatly reduced by using DLD-VISU, when the DB in a project owns many tables.

6.3.2 *Controller*

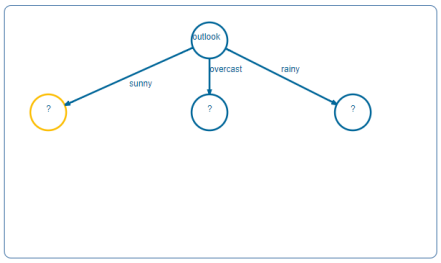
During using the DLD-VISU framework, it is only necessary to define an Action class to realize the communication between *Model* and *View* layers. The information on the *View* layer, such as the uploaded files by students, students' input data, etc., is transferred to the correlated variables in *TrainAction* class. Then, *TrainAction* invokes the decision tree algorithm classes for example ID3 on the *Model* layer to compute these data and return the results to *View*.

Transferring parameters between front- and back-end of web applications was a very complicated job formerly. A great amount of code for receive and send functions could not be avoided. However, such work can be taken over by DLD-VISU now. Developers only need to add the Action Class name written by themselves into the files *structs.XML* as well as *applicationContent.XML* and DLD-VISU handles the remaining work (The concrete tasks are realised by Struts and Spring. But developers do not need to understand the principles how Struts and Spring work).

6.3.3 *View*

In the *View* layer, developers can apply the JavaScript function library supplied by DLD-VISU to generate page elements and animations. In the animation development of decision trees, the JavaScript functions offered by DLD-VISU are put to use in the following scenarios:

1) Utilise the process chart function module to show process charts on pages. This module has two functions:
   a) As animations proceed, the texts and icons of the current step in the process chart are highlighted.
   b) Some sub-steps are shown/hidden in animations.
2) Apply the table module to generate tables in a unified style, e.g. the list of datasets presented in Figure 6.2 and attribute table demonstrated in Figure 6.5.




```
function drawDesicionTree(){
  SGH.isMoore = false;
  SGH.State(6.5,-1,0.5,"outlook");
  SGH.State(2,-3,0.5,"?");
  SGH.State(10.5,-3,0.5,"?");
  SGH.State(6.5,-3,0.5,"?");
  circle = SGH.getElementById(SGH.circles,"jxgBoard1C11");
  circle.setColor(HEIGHTLIGHT);
  SGH.Connection("jxgBoard1C3","jxgBoard1C11","sunny","","",0,0,false);
  SGH.Connection("jxgBoard1C3","jxgBoard1C27","overcast","","",0,0,false);;
  SGH.Connection("jxgBoard1C3","jxgBoard1C19","rainy","","",0,0,false);
}
```

Figure 6.1: An example of drawing decision tree

3) Employ the DVNL graphic function library to construct decision trees. Figure 6.1 presents an example of using DVNL functions to draw a decision tree on web pages. Developers may only use one statement to create an object of a node or line on pages through the DVNL function library. By executing the methods related to this object, developers are also able to carry out some operations to this object, such as show, hide, rename, etc., to realise animated effects.

4) Use the validation module to implement the Self-Assessment function.

5) Take advantage of validation module to check students' inputs.

More than 80% functions of decision tree on the *View* layer are achieved via the five function modules mentioned above. As shown in Figure 6.n, all elements on this page are built by means of DLD-VISU functions other than some texts, which means not only can developers build teaching animations with a uniform style rapidly but it also brings some convenience for students so that they can adapt themselves to use DLD-VISU faster.

6.4 VISUALIZATION AND ANIMATION OF DECISION TREE

6.4.1 *Dataset Entry*

Students can create datasets by manual input or importing a file in the ARFF format that is stored in database. Students can select a dataset generated previously at any time to run the decision tree animation, as displayed in Figure 6.2. Figure 6.3 presents how students add a dataset manually.

Students need to set type, name and candidate values for each feature and class firstly. As mentioned previously, the program has so far supported no more than 4 features and 2 classes as well as up to 3 distinct values for each discrete feature or class. The type contains

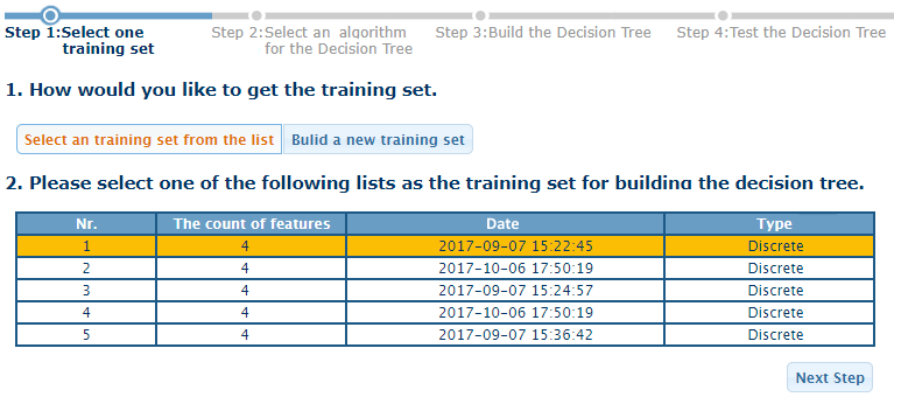


Figure 6.2: Import An ARFF data

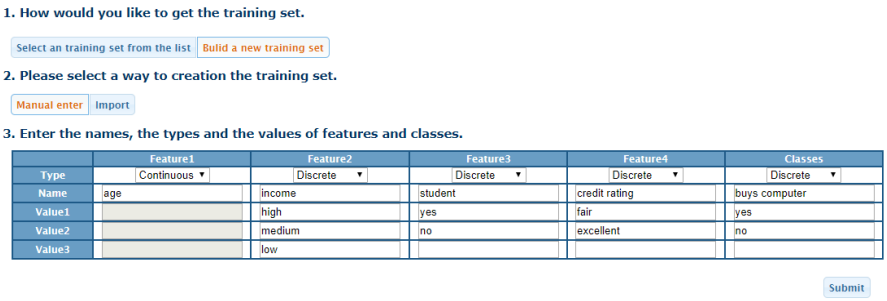


Figure 6.3: Creation a training set with manual

"discrete" and *"continuous"*. When students set a feature to be *"continuous"*, the text area of the candidate value of this feature becomes grey. In Figure 6.3 the type of "age" is *"continuous"*, students cannot input candidate values for it. When the number of features is smaller than 4 or the values of a feature are less than 3, the corresponding text areas are kept to be white/blank.

After setting all properties of features and class, click the "Submit" button, as shown in Figure 6.4. The system produces an empty "Training Set" table. Students can click "Add a new row" to generate a new piece of data, use the drop-down menu to choose a value for *"discrete"* type and use the text box to input the value for *"continuous"* type. After all values are set, students click the "next step" button to proceed.

The whole manual input process is divided into two parts in order to simplify the work that students input datasets. Because a dataset usually includes dozens of pieces of data.

Since students already input the candidate values for each feature at first, the program can take advantage of them to produce a drop-down menu, when students create datasets. In this way, students only need to select a value for each feature from the drop-down menu, avoiding that students input values for features manually. This design saves students' time and also prevents the occurrence of typos.

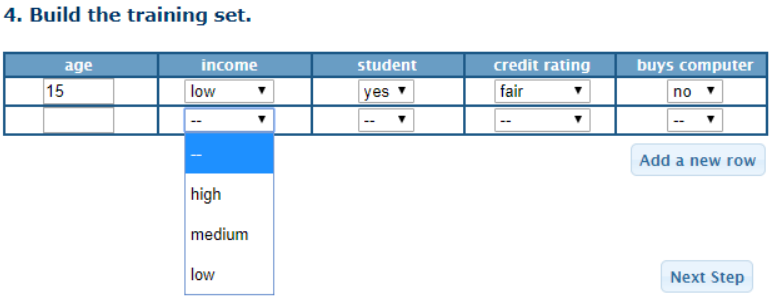


Figure 6.4: Creation a data using drop-down menu

6.4.2 *Select an algorithm for the decision tree*

In this step, students can choose a decision tree algorithms according to their own demands. Note that ID3 algorithm cannot be selected when the type of a feature is *"continuous"*. Because ID3 algorithm is only able to process the data whose type is *"discrete"*.

6.4.3 *Build the decision tree*

In this step, the program specifies the complete constructing process of decision trees. The whole page is partitioned into four parts. Training Set table is on the upper left, the decision tree that is going to be built on the bottom left, an auxiliary table on the upper right corner that helps students to calculate the measure standards of each feature in the current node, e.g. the gain in ID3 algorithm, as well as the computing process and results of the measure standards of each feature in the current node on the bottom right corner. Besides, this part also includes the Self-Assessment function.

Figure 6.5 shows the whole process how to use ID3 algorithm to classify data. Students need to calculate and choose the feature that can be treated as the root. The upper right table is a feature summarization created according to the datasets on the left side. During computing the entropy of each feature, students need to repeatedly count the number of data in the left table that satisfies the requirements. This action is not difficult but easy to go wrong. Once students make a mistake in this action, the subsequent calculating results may be influenced. In order to make students focus on understanding algorithms, the system provides students with the summarization table on the upper left order to help them to finish the calculations in the next step. At the meantime, students can also observe the table structure to deepen their understanding of algorithms.

Students can click "Verify" button to check their own inputs. Green indicates that their answers are correct and red suggests they are wrong. Besides, students are also able to skip manual input and click "Verify" button to watch the results directly. As presented in Figure

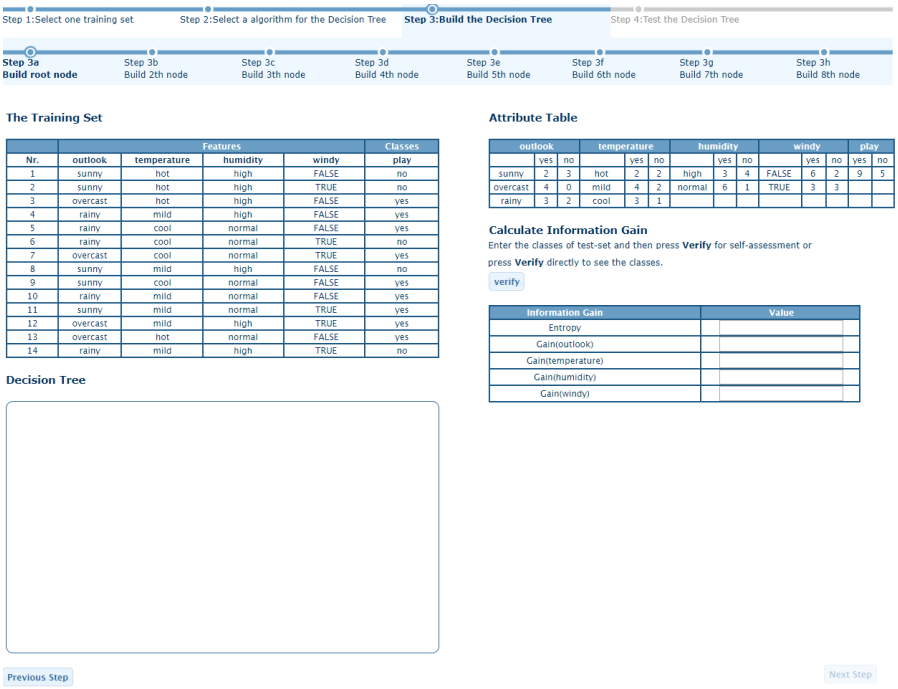


Figure 6.5: The layout of page of step 3

6.6, the system applies yellow as the background color to highlight the best feature. The system shows not only the correct answers but also the calculating process on the bottom right corner so that students can better understand algorithms.

When students click "next step" button, the system adds the best feature chosen in the last step as a node into the bottom left decision tree and highlights the node which needs to be calculated in the next step. At the mean time, the data in the training set on the upper left corner that is relevant for the calculation in the next step is also highlighted. Thus, students may better keep track of and observe algorithms. As presented in Figure 6.7, the auxiliary table on the upper right corner is updated accordingly.

The above steps are repeated until the class of all data is finished. The final results are displayed in Figure 6.8.

6.4.4 *Test the decision tree*

In order to deepen students' understanding of decision tree, the test function is furnished so that they can check the created decision trees. As shown in Figure 6.9, students may input a piece of unclassified data and click "Test" button. The system presents the class of the data. Meanwhile, the determining process is highlighted in the decision tree on the left side.

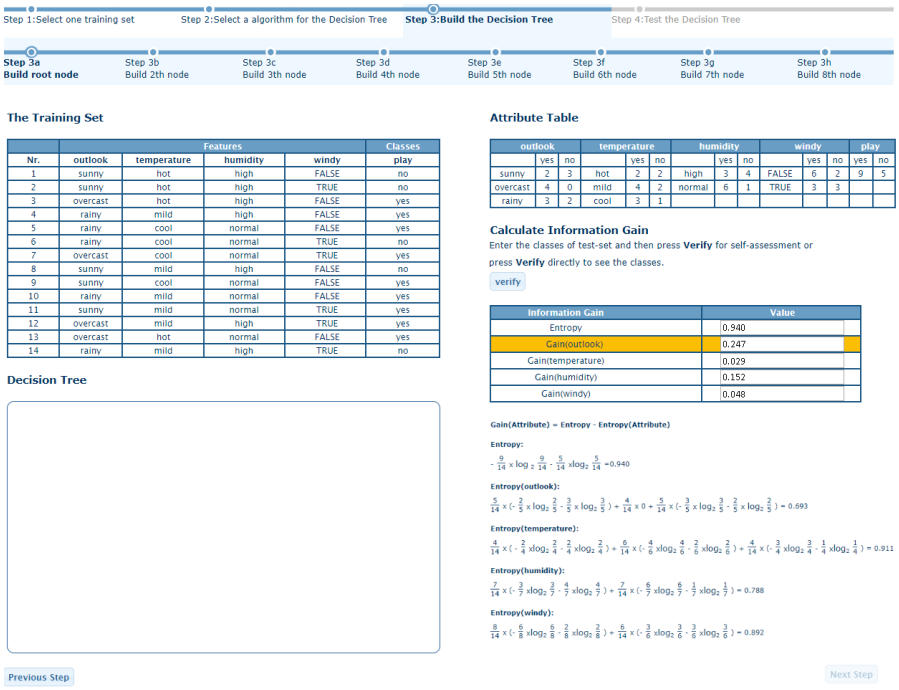


Figure 6.6: The Self-Assessment system of animation of decision tree

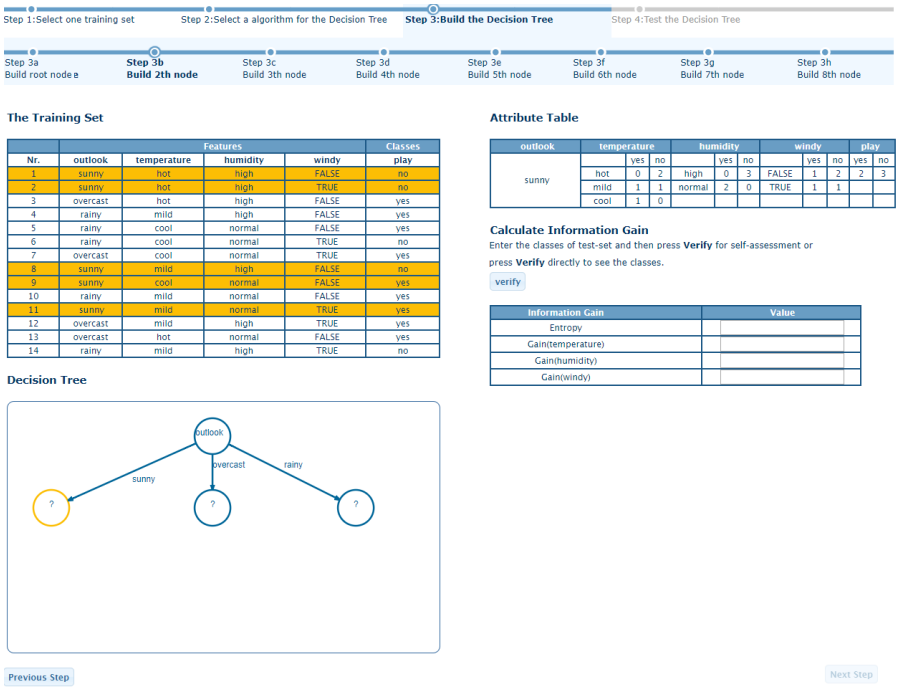


Figure 6.7: Building first node of the decision tree

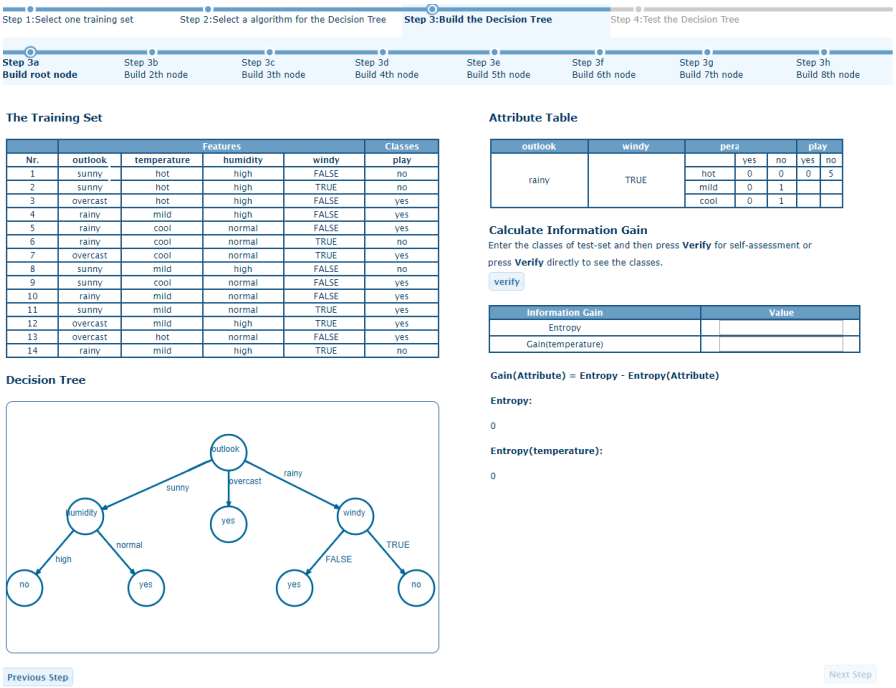


Figure 6.8: The final result of building a decision tree

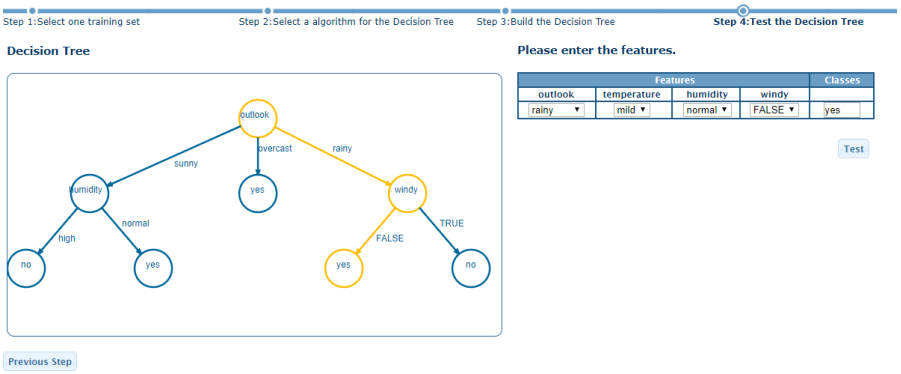


Figure 6.9: Test the decision tree

6.5 SUMMARY

In this chapter, we see the whole process of using DLD-VISU to develop a teaching animation. The entire development cycle are three months. However, the formulation and modification of the animation scheme already took two and a half months. During this time, the design plan was changed many time, so that it could meet the four teaching theories better. It is only spent more than a week for the real code. This fully proves that the using DLD-VISU to develop teaching animation can greatly enhance the development efficiency and reduce the development costs.

7 DLD-VISU EVALUATION

In this section two evaluation methods and related results are presented. The first method is to carry out surveys on students and the second method is to analyse the impact on students' performance brought by applying DLD-VISU.

7.1 STUDENTS' SURVEY

Two surveys are conducted in total. DLD-VISU contains only FSM and k-map functions in the first survey. The purpose of this survey is to prove that the idea of DLD-VISU is correct, that is, AV based on four pedagogical theories can improve the effect of students' self-study. After the first survey, the development of the combinatorial logic module continues. In order to test the effect of this part, the second survey is performed.

7.1.1 *First students' survey*

This section presents the results of the survey that was conducted by 179 first-year students in the Technical University of Darmstadt against the end of the semester, in which they took the DLD lecture.

The survey included 20 questions. The questions 1-3 asked students about the overall view of DLD-VISU, the questions 4-12 related to the pedagogical theories, the questions 13-14 were regarding the GUI of DLD-VISU, the questions 15-17 invited students to express their opinions towards on-line tools and the questions 18-20 inquired how students think about the extendibility of the DLD-VISU.

As explained previously, DLD-VISU was realized based on four pedagogical theories that describe the requirements for an effective animation solution. In the previous sections, it was highlighted how these theories are mapped to concrete aspects in the presented software solution. In order to see how far this mapping can serve its purpose, the students' feedback was collected by asking questions relating to their experience about DLD-VISU and then analysed. For this purpose, the questions 4-12 of the survey were most important.

For instance, question 1 in Figure 7.1 and 7.2related to the first impression which was highly important for the acceptance of any E-Learning tool. It is believed that the clear layout and the uncomplicated operation of DLD-VISU have also contributed to the positive and very positive impressions that 73% of students had, aside from

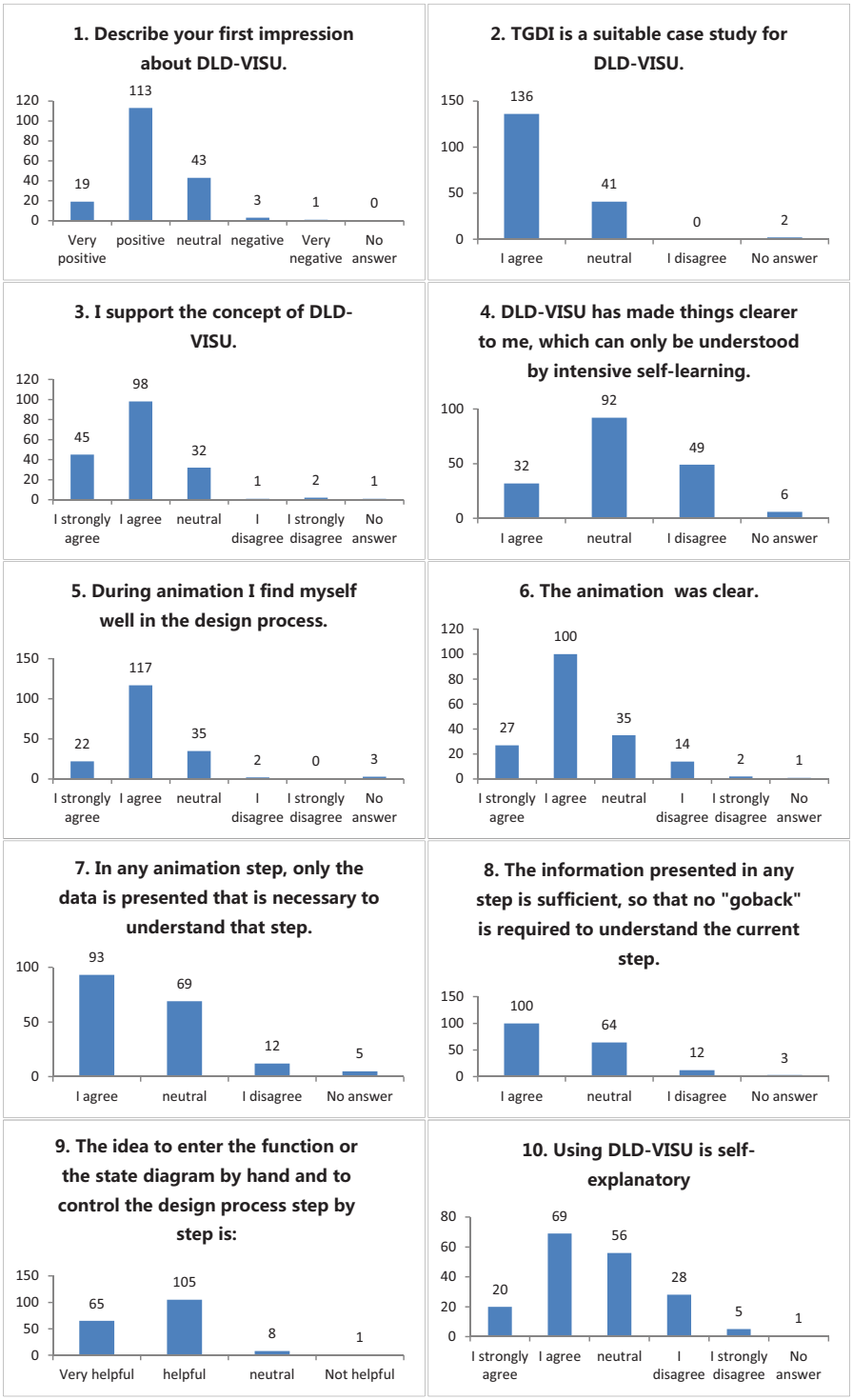


Figure 7.1: Students' Feedback Results part 1

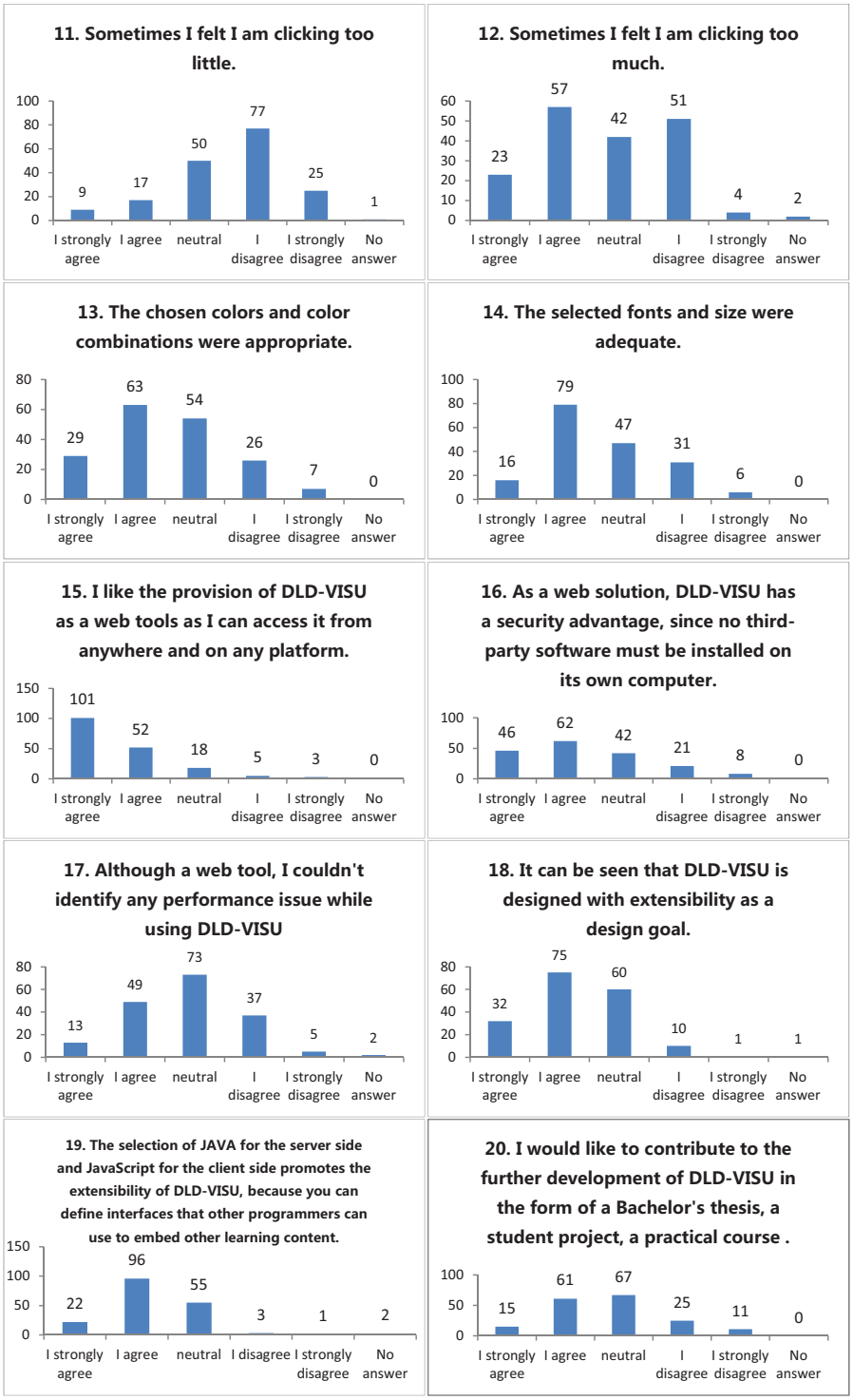


Figure 7.2: Students' Feedback Results part 2

its advantages in the learning process. Question 2 and 3 reflected the extent of students' support for DLD-VISU from another aspect. Almost all students believed that TGDI is a suitable case for DLD-VISU, which suggested that after the trial students think DLD-VISU is helpful for learning. 79% of students advocated the concept of DLD-VISU, indicating that the AV presentation of DLD-VISU had already earned students' support to a great extent.

It seemed that almost 78% of students had found their way through the animation steps according to the answers to question 5. This implied that DLD-VISU helped them keep the overview of the design process. As mentioned in the previous chapters, *Keeping Overview*, *Abstraction*, and *Traceability* are the three principles which reflect the instructional model.

Question 7 indirectly referred to the *Abstraction* concept in DLD-VISU. The answers to this question presented that 54% of students agreed that only the data that is necessary for helping understanding is displayed in the current animation step. Almost 39% gave, however, a neutral answer to the same question. This number was high enough to review the solution and then improve its *Abstraction* feature at several points, which is an essential objective of the feedback system.

The *Traceability* principle was addressed in question 8 and students gave similar answers to it.

In order to learn about if students could understand the mentor models very well in the teaching courses, we put up with two general questions, aside from the corresponding questions to *Keeping Overview*, *Abstraction*, and *Traceability*. Question 6 presented that almost 71% students considered the animation was clear, indicating the animation in DLD-VISU is easy to comprehend, which is obviously good for students to learn by themselves. On the contrary, the answers of question 4 were not that satisfying. Only 18% students thought DLD-VISU could facilitate self-learning, whereas 27% held an opposite opinion. This suggested there still was much space for DLD-VISU to make further progress, e.g. inserting more detailed text descriptions for algorithms, which just matches the answers of question 10. Almost half of the students found that DLD-VISU is self-explanatory according to question 10. This confirmed that the textual instructions and messages were sufficient for this part of students to use DLD-VISU without extra help. Adding text to the graphic is essential for effective animation according to the *Dual Coding* theory.

Question 9 implied that students must be active in the animation process by entering the design specification and controlling the design process. Recall that the *Cognitive Constructivism* theory presumes user interaction for an effective animation tool. What's interesting is that almost all the students found this kind of interaction as helpful or very helpful.

The *Individual Differences* assumes that students benefit from visualization differently depending on their cognitive abilities and learning styles. Again, DLD-VISU does not define user-specific actions but students, for instance, can try designs at different difficulty levels that fit their learning abilities. The granularity of the animation step (e.g., should the entries of a K-map displayed all at once, line by line, or square by square?) is an important parameter that may be set by the students according to their learning level. The need for such an improvement in DLD-VISU can be seen from the answers to question 11 and 12. While 45% felt that the animation steps were too fine-granular because of the need to click frequently, 30% had an opposite opinion.

The purpose of question 13 and 14 was to make a research on the font sizes and colors applied in DLD-VISU. Only half of the students were satisfied with the font sizes and colors used at that time. In order to improve the GUI of DLD-VISU, we discussed with some students about these two questions after this survey. At last, GUI was re-programmed up to 80%.

Question 15 and 16 showed that most students welcome on-line learning due to various advantages including the flexibility and the lack of the need to download and install software on the own computer. Only 37% of the responses experienced a performance issue in using DLD-VISU as a web solution according to question 17.

DLD-VISU is not only a software which illustrates the DLD algorithms, but also a developing platform of AV. Therefore, expandability is a significant characteristic of DLD-VISU, which is proven by the answers of question 18-20. Most students whose majors are computer science reckon DLD-VISU has very good expandability and 42% of them show their interest in participating in the development and expansion of DLD-VISU.

7.1.2 *Second students' survey*

At that time, DLD-VISU was evaluated by 15 first year students for 6 weeks (this is a "mentor-system" program in Technical University of Darmstadt). The evaluation was performed every week and the system was improved according to the students' suggestions and feedback. For example, some additional explanation was needed for Shannon algorithm in terms of students' opinions. Moreover, the most improvements proposed by students were usually about the layout and colors. For example, as presented in Figure 7.3, different colors were applied to illustrate circuits in the previous versions of DLD-VISU. According to some students, too many colors might result in chaos. Besides, the students who have achromatopsia and anomalous trichromatism are not able to distinguish colors. After the improvements, only two colors are employed to display the circuits: blue for general components and wires and yellow for the components and

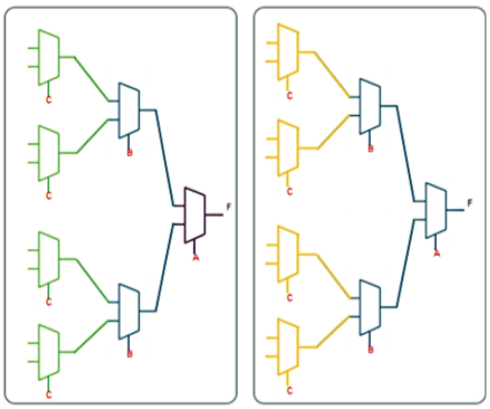


Figure 7.3: An example for the improvement of the GUI.

wires in a current and active step of algorithms. The blue and yellow were selected because the relevant research conduced by Kalloniatis M and Luu C.[62] pointed out that the proportion of blue-yellow color blindness is the smallest of all color blindness(equally rare for males and females: 0.01% for both). After implementing of the new customization and improvements, the new version was tested, before it was evaluated in the following week. The evaluation process ran in a loop. The test students had to complete a prepared questionnaire (see Figure 7.4) after they tried out the newest version of DLD-VISU in the last week. Their opinions to DLD-VISU were submitted anonymously and the people who participated in the testing were voluntary students, which led to open and honest opinions. This time 30 questions were designed and some of them were same as in the first survey, and some questions were specifically for new functional modules. As the Figure 7.4 shows, the questions were aiming at six aspects.

1) **First Impression:** 12 students voted for "very positive" and 3 students voted for "positive". This result is better than in the first survey, because they had been dealing with DLD-VISU for six weeks.

2) **Didactics:** Most of the students voted for "very positive". Some few students voted for "neutral". It can be clarified that some of the students are still in the first semester and do not know the stuff of DLD. They cannot say whether the DLD-VISU could support the teaching of DLD or not.

3) **Representation:** Most of the students had a very positive impression about the animation, the font, text size, colors and layout used in the application. This is not surprising because all colors and fonts are re-designed according to their opinions.

4) **Function input:** At this point, students were asked if they had problems with the input function or whether they needed additional external support. Most of the students coped with the

		very positive	positive	neutral	negative	very negative
First impression	My first impression of the project was	**12**	**3**			
		agree very	**agree**	**neutral**	**not agree**	**none agree too**
Didactics	TGDI is a perfect case study for the project.	11	2	2		
	Through this project, things have become clear to me that would otherwise be understood only through intense self-study.	6	4	3	1	
	In every animation step, sufficient information is presented so that no extra help is needed to understand each step.	8	4	2		
	In each animation step only informations are presented, which are necessary to understand this step. You can focus and you will not be distracted by irrelevant information.	8	6	1		
	The sequence of animation steps can I find reasonable	8	7			
Representation	The animation interface was clear.	6	8	1		
	The operation of the tool was self-explanatory.	6	8	1		
	The chosen colors and color combinations were appropriate.	9	6			
	The selected font and font size were appropriate.	9	5	1		
Function input (step 1)	The input for different functions DNF, KNF ... etc works without problems. The navigation between the different fields is easy.	8	6			
	The page allows correction of wrong input	6	8	1		
	The input fields are provided with examples and help-text, so that no extra help is needed.	7	6	2		
Function design (step 4)	The animation of the different functions with the following logic elements is understandable:					
	1. Gates	8	3	2		
	2. Decoder	3	2	3		
	3. Look-up Tables	3	3	2	1	
	4. Multiplexer	6	2	3	1	
	5. FPGA		3	4		
	Look And Feel: The colors and color combinations are appropriate. Graphics, media, elements, and content are clear and appealing:					
	1. Gates	9	6			
	2. Decoder	7	7	1		
	3. Look-up Tabeles	7	6			
	4. Multiplexer	8	5	1		
	5. FPGA	5	5	2		
Opinion to the project	The project supports the curriculum.	11	2			
	The project is meeting its objectives.	9	4			
	The project makes my learning effectively.	7	5	2		
	The project is usable without need of a manual or user help.	10	5			
	Hints/examples makes the project easy to use.	9	5	2		
Your recommendation: Please make a choice	□ This project would be useful for teaching, especially TGDI. I recommend to integrate the project.	15				
	□This project is beneficial, but I have serious reservations.					
	□ Project is not meeting its objectives and should not be integrated (Please give the reasons).					

Figure 7.4: Result of the students' evaluation.

function input. This is because additional information and hints for entering the function are available in the appropriate location of web pages. If users enter a faulty function, an error message is displayed next to the function input field.

5) **Function Design:** The implementation of the Boolean function can be performed step-by-step by users, which made a very positive impression on most students who participated in the test.

6) **Opinion to the Project:** Most of the students believed that the application met its objectives and helped them to learn effectively.

All students supported in integrating the DLD-VISU for learning of DLD. In the Figure 7.4, you can see some of the students' comments and suggestions regarding improving the old versions of the DLD-VISU. In general, the students were very enthusiastic and had left positive impression about the DLD-VISU.

7.2 IMPACT ON STUDENTS PERFORMANCE

The survey presented in the previous section gave an indirect evaluation of the effectiveness of DLD-VISU. At that time the self-assessment system had not been implemented yet. Currently, the learning curves that record the self-assessment results provide an efficient way to evaluate DLD-VISU.

7.2.1 *Evaluation in fall 2013*

In this semester, DLD-VISU was presented in the classroom to 38 electrical and computer engineering students taking the DLD course at the sophomore level. The students who had interest asked to request access data by sending an email to the instructor. Nineteen students registered. However, only seven out of these students accessed and used DLD-VISU. This low usage may be attributed to the fact that the tool was presented very late in the semester, specifically in the last week before the final exam. Each of the seven students completed the K-map minimization process for 4.8 functions in average. The overall success rate for all students and all trials was 79.74%. In the final exam all the students had to tackle a question about K-map minimization with 5 marks. It turned out that DLD-VISU users solved this final-exam question with an average of 4.49 marks. Yet, those non DLD-VISU users obtained 3.89 marks in average. Thus, DLD-VISU users accomplished this task 15.4% (100*(4.49-3.89)/3.89) better than the others. However, this improvement cannot be seen as a net gain: through the comparison of the overall students performance and it turned out that the DLD-VISU users had a higher average, however only by 5.1%.

7.2.2 *Evaluation in fall 2014*

In this semester, an experiment was conducted to gain more data on the effectiveness of DLD-VISU by answering the following question: Given a specific minimization task, what is the advantage of using DLD-VISU to perform this task over the traditional manual approach? This question is especially challenging: If a student starts solving a problem using DLD-VISU, then she or he would get immediate feedback and see the correct solution so that a succeeding manual solution of the same problem would not help in assessment. Even if the student starts with the manual solution, she or he would gain experience with the function so that solving it with DLD-VISU would be easier. For a valid and fair comparison, therefore, the case is completely avoided that a student solves the same problem twice. For an in-depth understanding of the effectiveness of DLD-VISU, furthermore, minimization problems of different difficulty levels were

provided. For that the concept of difficulty level is specified as follows.

Difficulty Level: The difficulty level of the function minimization problem using K-map increases with the following factors.

1) The number of prime implicants NPI in the K-map. When NPI increases, then the probability of overlooking one or more prime implicants is higher.

2) The difference between the number of prime implicants and the number of core implicants, which is notated as D_{PI-CI} . A higher D_{PI-CI} value indicates the availability of more overlapped prime implicants and, thus, the availability of more than one optimal function. In contrast, if D_{PI-CI} is zero, there is only one optimal function.

Based on this understanding, the difficulty level is specified using the heuristic formula:

$$DL = NPI + D_{PI-CI} or$$

$$DL = 2NPI + NCI$$

where NCI is the number of the core implicants.

Experiment execution: The experiment was conducted with 49 students who had already took or were taking the DLD course at the Technical University of Darmstadt and at Khalifa University, respectively. The experiment participants at both universities were familiar with the K-map minimization approach and had a good understanding of the related concepts minterm, implicant, prime implicant, and core implicant. The students were asked to bring their laptops to the class room and to make sure they have access to the internet. One day before the students were registered by admin in the DLD-VISU system and divided into two groups (Group A and Group B) with almost comparable average performance in the course. At the beginning of the experiment, the K-map approach was reviewed and DLD-VISU was demonstrated for the first time. Students were tutored step-by-step how to use DLD-VISU to minimize Boolean functions and instructors made sure that all the students run one example successfully, which was firstly explained on the white board. The K-map review and the DLD-VISU demo took almost 10 minutes. Each student had to minimize eight functions (named F1 to F8), whereas the index as the subscript in the function name corresponds to its difficulty level. These functions are:

1) F_1 = ∑m(0,3)+ ∑d(1,2)

2) F_2 = ∑m(0,1,2,3,13,14)+ ∑d(12,15)

3) F_3 = ∑m(0,1,2,3)+ ∑d(9,11)

	Approach	Count of Participants	Performance	Procession Time
Group A	Manual	25 Students	80.14	39.92 min
Group B	DLD-VISU	24 Students	93.79	32.13 min
Difference			13.65	7.79 min
Advantage of using DLD-VISU			17.03%	19.5%

Table 7.1: Performance and Processing Time for Groups A and B

4) $F_4 = \sum m(4,5,14,15) + \sum d(7,13)$

5) $F_5 = \sum m(4,5,7,13,14) + \sum d(9,11,15)$

6) $F_6 = \sum m(0,1,2,7,8,9,13) + \sum d(10,11)$

7) $F_7 = \sum m(0,2,5,10,12,15) + \sum d(8,13)$

8) $F_8 = \sum m(0,2,10,12,13,15) + \sum d(5,8)$

The 25 students of Group A had to minimize these functions manually. The 24 students of Group B used DLD-VISU to do the same. The functions had to be processed in the order of their indices. Manual solutions had to be delivered on paper. DLD-VISU solutions were evaluated automatically and grades (success rates) were registered in the learning curves, as it was detailed in Section 3. The time amount needed to complete the manual solution or the DLD-VISU solution by each student was recorded.

Results and analysis: Students had to determine all the prime implicants and core implicants before writing the minimized function. The manual solutions were graded using the same pattern used in DLD-VISU to determine the success rate. Table 7.1 shows the average performance of both groups as well as the average time that was spent to complete the respective task. The table shows that DLD-VISU users completed the task in 19.5% less time and performed almost 17.03% better than non DLD-VISU users. Given the sample size of 49 students and the constraints, under which the experiment was conducted, these values showed a clear advantage in using DLD-VISU. Remember that the participants had no previous experience with DLD-VISU in contrast to the manual solution. This doesn't only make the experiment results more significant, but it also shows the ease of operation of the proposed tools.

Another aspect that can be investigated is the student performance in relation to the difficulty level. Figure 7.5 shows the advantage of using DLD-VISU as a function of the difficulty level. The curve points in this figure were determined using the same formula used to calculate the advantage of using DLD-VISU in Table 7.1, however, for each question separately.

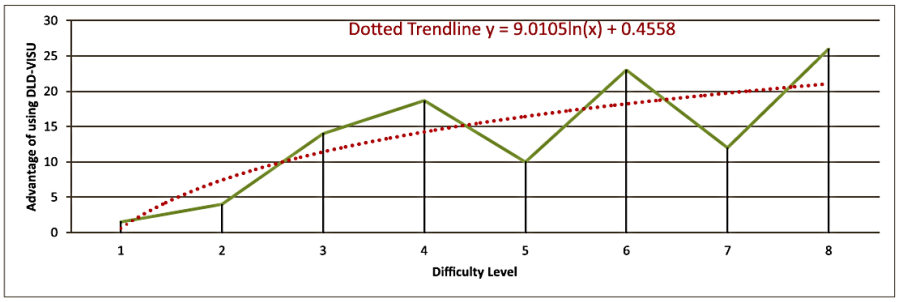


Figure 7.5: Advantage of using DLD-VISU against difficulty level.

For instance, the average performance of DLD-VISU users (Group B) in minimizing F3 was almost 14% better than the performance of Group A. In spite of two exceptions, the trend-line of this function showed that the advantage of using DLD-VISU increases with the difficulty level. The effect of the increasing difficulty level seems to be compensated by the improved learning level with every trial due to the immediate feedback that students get from DLD-VISU. It is believed that this feature is especially important for the acceptance of DLD-VISU. The two exceptions F5 and F7 may be associated with the way the experiment was executed and with the selection of these functions itself. As mentioned before, F5 was processed directly after F4. However, F5 had accidentally a large similarity with F4, so that both groups performed well in minimizing F5. However, it should not be excluded that these exceptions were "at least partially" caused by issues in the definition of the difficulty level itself. Evaluating this aspect needs more data, which will be a part of future work.

8

SUMMARY AND FUTURE WORK

DLD-VISU was built based on a long experience in teaching digital logic design and a critical view of effective methods to present its different topics. The proposed software solution is a reflection of this experience under consideration of important theories related to the effectiveness of algorithm animation. After a large number of investigations, four pedagogy theories were selected as the theoretical basis for improving AV efficiency. In order to apply these four theories of the development of AV specifically, we designed concrete realization methods for each theory.

We believe that this approach is of general value in the field of learning technology and can be applied beyond digital logic design. The presented results in terms of students' survey and experiments showed the effectiveness of the tools. Informal discussions with the students who tested the tools revealed strong enthusiasm for DLD-VISU. Also, several instructors whom we talk to welcomed the idea of DLD-VISU especially the detailed presentation of the FSM design process.

DLD-VISU is not just a on-line platform for DLD course, it is also a powerful development framework for the development of the E-Learning topics. In chapter 6 we have shown how DLD-VISU can be reconfigured to accommodate machine learning algorithms, and it can be applied to many other courses using its framework and JavaScript graphic library.

DLD-VISU is also an unified AV management platform. All the animations on this platform employ an uniform style and GUI, which can reduce the time that students spend to adapt to different AV programs. Furthermore, the platform provides some management subsystem that make it very convenient for teachers to publish and manage their own AV programs and perform real-time observation of students' usage.

DLD-VISU was developed since 2010, at that time the SSH2 combination framework is a very advanced technology. The SSH2 framework build a better structure for Java projects and the excellent structure is critical to a large project. But compared with some new technologies, SSH2 framework also shows some shortcomings. For example, the combination of three different frameworks has made it difficult to upgrade any of these frameworks, and the work efficiency of SSH2 is also lower than some new frameworks, e.g. SpringMVC. So transplanting the DLD-VISU from SSH2 framework to springMVC

framework before the next version has been decided. This has the following benefits:

1) Because the Spring and the SpringMVC belong to a same development team, the integration of these two frameworks does not require complex configuration, and the upgrade of them is also very convenient.

2) The design idea of SpringMVC is more advanced than Struts2, and SpringMVC is more flexible [47].

3) The implementation efficiency of SpringMVC is faster than SSH2 [8].

In addition to expanding its functionality to include simulation and further topics, the long-term plan is to expand DLD-VISU to a course-based animation platform for different courses.

APPENDIX I DATABASE

The database consists of 11 tables. In order to store Chinese characters or special characters in German, the collation of database is utf8_bin. The Type of the database is InnoDB.

A.1 THE TABLES FOR STORING FSM DIAGRAM

Three of these tables(drawnodeandline, line, node) are used to save the FSM diagrams. Table A.1 is used to store the general information of each FSM diagrams, such as name, type and so on. The position of each state is stored in table A.2 and the position of each transition is stored in table A.3. The relationship between drawnodeandline and line/node is 1:n.

Field	Type	Comments
drawNodeAndLine	varchar(32)	The primary key of this table.
fsmType	varchar(6)	The type of the FSM, e.g mearly or moore.
name	varchar(50)	The name of the FSM.
codeType	varchar(1)	The state code chosen by the user, e.g one-hot code, binary code or grey code.
ffType	varchar(1)	The type of flip-flop chosen by the user, e.g T,D or JK.
uId	varchar(32)	The ID of a student. This field is the foreign key of table student.
inputDate	varchar(25)	The time to create the FSM.
complete	int(1)	Whether the student has completed the whole process.

Table A.1: Table drawnodeandline

A.2 THE TABLES FOR STORING TRAINING SET

Two of these tables(train, trainattribute) are used to save the training set. Table A.4 is used to store the general information of each training set, such as the type of classes, the count of attributes and so on. The attributes of each training set is stored in table A.5. The relationship between train and trainattribute is 1:n.

Field	Type	Comments
NodeId	varchar(32)	The primary key of this table.
nodeGraphicId	varchar(50)	The ID of the state.
nodeName	varchar(50)	The name of the state.
outputValue	varchar(50)	The output signals of a state. (only for moore)
nodeXcoordinate	varchar(50)	The x-coordinate of the state.
nodeYcoordinate	varchar(50)	The y-coordinate of the state.
startNode	varchar(10)	whether the state is a start state.
drawnodeandlineid	varchar(32)	The ID of a FSM. This field is the foreign key of table drawNodeAndLine.
nodeCode	varchar(100)	The state code of the state.

Table A.2: Table node

Field	Type	Comments
LineId	varchar(32)	The primary key of this table.
drawnodeandlineid	varchar(32)	The ID of a FSM. This field is the foreign key of table drawNodeAndLine.
inputValue	varchar(50)	The input signals of a transition.
outputValue	varchar(50)	The output signals of a transition. (only for mearly)
sourceNodeGraphicId	varchar(50)	The ID of the source state. This field is the foreign key of table node.
distinationNodeGraphicId	varchar(50)	The ID of the target state. This field is the foreign key of table node.
selflinePosition	varchar(100)	The position of the self-transition. (x-coordinate,y-coordinate)
sourceNodeName	varchar(100)	The Name of the source state.
distinationNodeName	varchar(100)	The Name of the target state.

Table A.3: Table line

A.3 THE TABLES FOR REGISTER

Three of these tables(student, teacher, studentandteacher) are used to save the Information of register. A student can choose more than one teacher's course, a teacher can also teach more than one student. So the relationship between table A.6 and table A.7 is n: m. In order to maintain a 1:n relationship between the tables, a middle table A.8 is created.

A.4 TABLE DATAACCESS

The table A.9 is used to save the **Access Control System**.

Field	Type	Comments
trainId	varchar(32)	The primary key of this table.
classesType	varchar(15)	The type of the classes. e.g discrete or continuous.
attributeCount	int(11)	The number of the attribute in the training set.
trainName	varchar(50)	The name of the training set.

Table A.4: Table train

Field	Type	Comments
attributeId	varchar(32)	The primary key of this table.
attribute1	varchar(50)	The value or the name of attribute1.
attribute2	varchar(50)	The value or the name of attribute2.
attribute3	varchar(50)	The value or the name of attribute3.
attribute4	varchar(50)	The value or the name of attribute4.
attribute5	varchar(50)	The value or the name of classes.
attribute6	varchar(50)	The preparation field.
attribute7	varchar(50)	The preparation field.
attribute8	varchar(50)	The preparation field.
attribute8	varchar(50)	The preparation field.
attribute8	varchar(50)	vreparation field.
attribute8	varchar(50)	The preparation field.
trainId	varchar(32)	The ID of a training set. This field is the foreign key of table train.
attributeType	int(1)	whether this record is the name of attribute or the value of the attribute.

Table A.5: Table line

A.5 EVALUATION

The table A.10 is used to save the data of self-assessment system.

A.6 FEEDBACK

The table A.10 is used to save the feedback from user. We placed 10 questions on the web page to collect the user's satisfaction with the DLD-VISU. Users can score each question based on their satisfaction, scores from one star to five stars.

Field	Type	Comments
pId	varchar(32)	The primary key of this table.
username	varchar(20)	The user name of the teacher.
password	varchar(8)	The password of the teacher.
email	varchar(50)	The email of the teacher.
firstname	varchar(50)	The first name of the teacher.
lastname	varchar(50)	The last name of the teacher.
web	varchar(100)	The web page the teacher. Used to check whether the information submitted by the teacher is correct.
approval	int(11)	Is the account activated.

Table A.6: Table teacher

Field	Type	Comments
uId	varchar(32)	The primary key of this table.
username	varchar(50)	The user name of the student.
password	varchar(8)	The password of the student.
email	varchar(50)	The email of the student.
name	varchar(50)	The name of the student.
approval	int(100)	Is the account activated.
question	varchar(100)	The question when a student forgets the password.
answer	varchar(100)	The right answer of the question.

Table A.7: Table student

Field	Type	Comments
spId	int(11)	The primary key of this table, auto increment.
uId	varchar(32)	The ID of a student. This field is the foreign key of table student.
pId	varchar(32)	The ID of a teacher. This field is the foreign key of table teacher.

Table A.8: Table studentandteacher

Field	Type	Comments
dId	int(11)	The primary key of this table, auto increment.
pId	varchar(32)	The ID of the teacher. This field is the foreign key of table teacher.
fromDate	varchar(10)	The opening date of a topic.
toDate	varchar(10)	The closing date of a topic.
approval	int(11)	The teacher can set up or cancel time limits
type	varchar(10)	The name of types, e.g gate, mux or kmap

Table A.9: Table dataaccess

Field	Type	Comments
eId	varchar(32)	The primary key of this table.
pId	varchar(32)	The ID of the teacher. This field is the foreign key of table teacher.
uId	varchar(32)	The ID of the teacher. This field is the foreign key of table student.
eType	varchar(20)	The type of topic of self-assessment, e.g kmap, mcclusky, and so on.
eTime	varchar(25)	The date of self-assessment
drawNodeAndLineId	varchar(32)	Store the FSM diagram for testing.
inputCount	int(10)	The total number of questions that the student answered.
correctCount	int(10)	The number of questions that the student answered correctly.

Table A.10: Table evaluation

Field	Type	Comments
fId	int(11)	The primary key of this table, auto increment.
questionNumber	varchar(32)	The number of questions.
inputDate	varchar(10)	Rating time .
grade	int(11)	The scores, between 1 to 5.

Table A.11: Table feedback

B

APPENDIX II GRAPHICAL FUNCTION LIBRARY OF DLD-VISU

B.1 THE LIBRARY DVNL

DVNL consists of two components – State and Connection – which are respectively used to draw the circle component and the line component.

B.1.1 *State*

Because the State class is designed for drawing a FSM, it contains different constructors for setting various properties of the state. If the user only needs to draw a simple circle, he only needs to implement the default constructor. Table B.1 shows all the constructors.

Constructor parameters	Description
State(x)	@param number x: The coordinate of the midpoint of state. The radius of State is the system default value.
State(x,y)	@param {number} x: The abscissa of the midpoint of state. @param {number} y: The ordinate of the midpoint of state. The radius of State is the system default value.
State(x,y,r)	@param number r: The radius of State.
State(x,y,r,objName)	@param string objName: display name on State.
State(x,y,r, objName, objActiveoutput)	@param string objActiveoutput: when FSM is a moore FSM, Output is displayed on State.
State(x,y,r, objName, objActiveoutput, isStart)	@param boolean isStart: When isStart=true, State is shown with two concentric circles. The default value of isStart is false.
State(x,y,r, objName, objActiveoutput, isStart,name)	@param string name is the id of this graphic. Users can apply the statement getElementById of JavaScript to obtain this graphic object.

Table B.1: The constructor list of State class.

In addition to the constructor, the State class also contains methods for displaying animations. Table B.2 shows all the methods.

B.1.2 *Connection*

Table B.3 shows all the methods of the class State.

Method	Description
changeState(isMoore)	@param boolean isMoore: Change the type of the state. The Moore state contains a straight line in the circle.
setColor(color)	@param string color: Set the color of the circle, it can be the name of the color, for example: white, black.. or the hexadecimal form, for example: #oooooo, #ffffff
setObjectName(name)	@param boolean name: Set the name of the circle. The name will be displayed in the middle of the circle. The color of the text is the same as the color of the circle.

Table B.2: The method list of State class.

Constructor parameters	Description
Connection(startCircle endCircleName)	@param string startCircleName,@param string endCircleName: The name of source / target circle.
Connection(startcircle, endcircle)	@param State startcircle, @param State endcircle: The state object of source / target circle.
Connection(startcircle, endcircle, objName, objActiveOutput,name)	@param {string} objName: the name of the object. @param {string} objActiveOutput: The active output of the transition (only for mearly FSM). @param {string} name: The name of the line, the text is displayed above the line.
Connection(startcircle, endcircle, objName, objActiveOutput,name,boolean withhelper)	@param boolean withhelper: If withhelper is true, the center of the line display a small circle, in order to facilitate users to select this line.

Table B.3: The constructor list of Connection class.

In addition to the constructor, the Connection class also contains *setColor* method for changing the color of a line.

B.2 THE LIBRARY DVEC

The library DVEC is used to draw the circuit diagram. It can draw a total of 14 components.

1) Logic gate: And; Or; Nand; Nor; Invert; Xor; Xnor
2) Mux
3) Decoder
4) Lookup-Table
5) Flip-Flop: D Flip-Flop; T Flip-Flop; JS Flip-Flop
6) Path

B.2.1 *Logic gate*

All logic gates contain the same form of constructor, table B.4 only lists the AND gate constructor.

Constructor parameters	Description
AndGate ([x,y]|,Board)	@param {number} x,y are the coordinate of the midpoint of AND-gate. The length/width adopt the system default values. @paramJXG.Board Board is a drawing board object on which AND-gate is drawn. If Board is not designated, AND-gate is going to be drawn on a default board.
AndGate ([[x,y],width,heigh	@param {number} width, height are the width and height of AND-gate.
AndGate ([x1,y1,x2,y2]|, Board)	@param {number} x1,y1 are the coordinate of the top left corner of AND-gate. @param {number} x2,y2 are the coordinate of the bottom right corner of AND-gate.

Table B.4: The constructor list of And class.

All logic gates only include the body and output-line, does not contain input-line. Because the user can customize the number of input ports, the program draws the input-line at the corresponding position based on the number of user inputs. So all of the classes of logic gates contain an *setInputLine(int n)* method that specifies the number of inputs. DVEC currently supports up to 10 input-lines.

AndGate, OrGate and XorGate classes also have a *addNotCircle* method. The user can first draw an AND-gate on the web page, then call its *addNotCircle* method, draw a small circle at the front of this AND-gate, convert this AND-Gate to NAND-Gate. Of course, users can also directly call the NandGate class to create a NAND-gate. The reason to design this method is that users may need to use animations to display AND-gate and NAND-gate.

B.2.2 *Flip-Flop*

The constructor of the Flip-Flops and logic-gate are the same. The Flip-Flop classes include a *setText(string text)* class, this method is used to set the input signal of the Flip-Flops.

B.2.3 *Mux*

The constructor of the Mux is shows in the table B.5. Table B.6 shows the methods of Mux classes.

Constructor parameters	Description
Mux (board, position, width, height, attr)	@param {JXG.Board}: board The board the new multiplexer is drawn on. @param {Array} position: The coordinate array [x,y] of the middle point of the new multiplexer. @param {number} width: The width of the new multiplexer. @param {number} height: The height of the new multiplexer. @param {json} attr: The attributes of the new multiplexer.

Table B.5: The constructor of Mux class.

B.2.4 *Decoder*

Table B.11 and B.7 show the constructor and methods of the Decoder class.

B.2.5 *Lookup-Table*

Table B.12 and B.8 show the constructor and methods of the Lookup-Table class.

B.2.6 *Path*

The constructor of Path is *LogicPath([x1,y1]|point1, [x2,y2]|point2,x_offset|,Board)*. The path start at point1, it will take a vertical angle at the place with x_offset horizontal offset to point1 then it goes further to point2.

B.3 THE LIBRARY OF DOM OBJECTS

We have developed 2 sets of methods that users can use them to quickly create style-consistent web page components. A set of methods are used to create flowcharts, and other set of methods are used to create tables.

Table B.9 shows the methods for creating and controlling flowcharts.

Table B.10 shows the methods for creating and controlling tables.

Methods	Description
moveTo (position)	Move the component to a new position. @param position {array} x, y position
setMainAttribute (attributes)	Set the attributes of the component. @param attributes {json} for example: color, highlight color...
setMuxText (text, xOffset, attributes)	Set 0,1 or 1,0 ports of multiplexer. @param text {string[]} Set 0,1 text to the multiplexer. @param xOffset {int} Set the xoffset to the text. @param attributes {json} Set the color attributes to the text.
setInput (number, length, isAllHide, attributes)	Set input-line. @param number {int} The number how many inputs there are. @param length {int} The length of input line. @param isAllHide {boolean} Whether hide the input line when they are created. @param attributes {json} Set the color attributes to input-line.
setOutput (length, isHide, attributes)	Set output-line. @param length {int} The length of output line. @param isHide {boolean} Whether hide the output line when they are created. @param attributes {json} Set the color attributes to output-line.
setSelector (number, length, isAllHide, isAtBelow,attributes)	Set selector @param number {int} The number how many selectors there should be. @param length {int} The length of selector lines. @param isAllHide {boolean} Whether hide the selector line when they are created. @param isAtBelow {boolean} Set the selector lines upper of below. @param attributes {json} Set the color attributes to selector.
setInputText (no,text, xOffset, yOffset, attributes)	Set or create a text for a input-line. @param no {int} the id number. @param text {string} The text to be created or set. @param xOffset {int} Set the xoffset to the text. @param yOffset {int} Set the yoffset to the text. @param attributes {json} Set the color attributes to the text.
setOutputText (text, xOffset, yOffset, attributes)	Set or create a text for a output-line. @param text {string} The text to be created or set. @param xOffset {int} Set the xoffset to the text. @param yOffset {int} Set the yoffset to the text. @param attributes {json} Set the color attributes to the text.
setSelectorText (no,text, xOffset, yOffset, attributes)	Set or create a text for a Selector. @param no {int} the id number. @param text {string} The text to be created or set. @param xOffset {int} Set the xoffset to the text. @param yOffset {int} Set the yoffset to the text. @param attributes {json} Set the color attributes to the text.
getInputPoint (no)	Get the outermost point of a input-line. @param no {int} the id number.
getOutputPoint (no)	Get the outermost point of a output-line. @param no {int} the id number.
getSelector (no)	Get the outermost point of a selector. @param no {int} the id number.

Methods	Description
moveTo (position)	Move the component to a new position. @param position {array} x, y position
setMainAttribute (attributes)	Set the attributes of the component @param attributes {json} for example: color, highlight color...
setInput (number, length, isAllHide, attributes)	Set input-line @param number {int} The number how many inputs there are. @param length {int} The length of input line. @param isAllHide {boolean} Whether hide the input line when they are created. @param attributes {json} Set the color attributes to input-line.
setOutput (length, isHide, attributes)	Set output-line @param length {int} The length of output line. @param isHide {boolean} Whether hide the output line when they are created. @param attributes {json} Set the color attributes to output-line.
setControll (length, isAllHide, direct, attributes)	Set output-line @param length {int} The length of control line. @param isAllHide {boolean} Whether hide the output line. @param direct {string} The control line is in the top or bottom of the decoder. @param attributes {json} Set the color attributes to control-line.
setInputText (no,text, xOffset, yOffset, attributes)	Set or create a text for a input-line @param no {int} the id number. @param text {string} The text to be created or set @param xOffset {int} Set the xoffset to the text. @param yOffset {int} Set the yoffset to the text. @param attributes {json} Set the color attributes to the text.
setOutputText (text, xOffset, yOffset, attributes)	Set or create a text for a output-line @param text {string} The text to be created or set @param xOffset {int} Set the xoffset to the text. @param yOffset {int} Set the yoffset to the text. @param attributes {json} Set the color attributes to the text.
setSelectorText (no,text, xOffset, yOffset, attributes)	Set or create a text for a Selector @param no {int} the id number. @param text {string} The text to be created or set @param xOffset {int} Set the xoffset to the text. @param yOffset {int} Set the yoffset to the text. @param attributes {json} Set the color attributes to the text.
getInputPoint (no)	Get the outermost point of a input-line @param no {int} the id number.
getOutputPoint2 (no)	Get the outermost point of a output-line @param no {int} the id number.

Table B.7: The methods list of decoder class.

Methods	Description
moveTo (position)	Move the component to a new position. @param position {array} x, y position
setMainAttribute (attributes)	Set the attributes of the component. @param attributes {json} for example: color, highlight color...
setInput (number, length, isAllHide, attributes)	Set input-line. @param number {int} The number how many inputs there are. @param length {int} The length of input line. @param isAllHide {boolean} Whether hide the input line when they are created. @param attributes {json} Set the color attributes to input-line.
setOutput (length, isHide, attributes)	Set output-line. @param length {int} The length of output line. @param isHide {boolean} Whether hide the output line. when they are created. @param attributes {json} Set the color attributes to output-line.
setsetTableConter (no,text, xOffset, yOffset, attributes)	Set or create a text for a input-line. @param no {int} the id number. @param text {string} The text to be created or seted. @param xOffset {int} Set the xoffset to the text. @param yOffset {int} Set the yoffset to the text. @param attributes {json} Set the color attributes to the text.
setInputText (no,text, xOffset, yOffset, attributes)	Set or create a text for a input-line. @param no {int} the id number. @param text {string} The text to be created or set @param xOffset {int} Set the xoffset to the text. @param yOffset {int} Set the yoffset to the text. @param attributes {json} Set the color attributes to the text.
setOutputText (text, xOffset, yOffset, attributes)	Set or create a text for a output-line. @param text {string} The text to be created or set. @param xOffset {int} Set the xoffset to the text. @param yOffset {int} Set the yoffset to the text. @param attributes {json} Set the color attributes to the text.
getInputPoint (no)	Get the outermost point of a input-line. @param no {int} the id number.
getOutputPoint (no)	Get the outermost point of a output-line. @param no {int} the id number.
getSelector (no)	Get the outermost point of a selector. @param no {int} the id number.

Table B.8: The methods list of lookup table class.

Method	Description
setChart(info)	@param array info: The String array contains the name of each step. Program creates steps of flowchart according to the size of the array
showStep(currentStep)	@param int currentStep: The program highlights the text in step "currentStep" and turns all steps before "currentStep" grayed out.

Table B.9: The method list of flowchart.

Method	Description
setTable(info)	@param {array} info: Create a table based on two-dimensional array contents. @returns {array}: Returns an array of html label.
mergeRowAndColumr	@param int x1,x2,y1,y2: Merge all the cells which start from x1, y1 and end with x2, y2.
getTable(tablearray)	@param array tablearray: Convert two-dimensional array to html code.

Table B.10: The method list of flowchart.

Constructor parameters	Description
Decoder (board, position, width, height, attr)	@param {JXG.Board}: board The board the new decoder is drawn on. @param {Array} position: The coordinate array [x,y] of the middle point of the new decoder. @param {number} width: The width of the new decoder. @param {number} inputNum: The number of input signal of the new decoder. @param {json} attr: The attributes of the new decoder.

Table B.11: The constructor of Decoder class.

Constructor parameters	Description
Lut (board, position, width, attr)	@param {JXG.Board}: board The board the new Lookup-Table is drawn on. @param {Array} position: The coordinate array [x,y] of the middle point of the new Lookup-Table. @param {number} width: The width of the new Lookup-Table. @param {json} attr: The attributes of the new Lookup-Table.

Table B.12: The constructor of Lookup-Table class.

BIBLIOGRAPHY

[1] H. M. El-bakry A. M. Riad H. K. Elminir and H. A. El-Ghareeb. "Supporting Online Lectures with Adaptive and intelligent Features." In: *International Journal of Research and Innovation, Advances in Information Sciences and Service Sciences* 3.1 (2011), pp. 47–53.

[2] Hazem M. El-Bakry A. M. Riad and Samir M. Abd El-razek. "Simulation of Medical Consultation on Virtual Patient." In: *International Journal of Computational Linguistics and Natural Language Processing* 1.3 (2012), pp. 60–64.

[3] *AVL-Tree*. Website. https://www.cs.usfca.edu/~galles/visualization/AVLtree.html.

[4] H. Z. Abidin, M. Kassim, K. A. Othman, and M. Samad. "Incorporating VHDL in teaching combinational logic circuit." In: *2010 2nd International Congress on Engineering Education*. 2010, pp. 225–228. DOI: 10.1109/ICEED.2010.5940796.

[5] O. B. Adamo, P. Guturu, and M. R. Varanasi. "An innovative method of teaching digital system design in an undergraduate electrical and computer engineering curriculum." In: *2009 IEEE International Conference on Microelectronic Systems Education*. 2009, pp. 25–28. DOI: 10.1109/MSE.2009.5270837.

[6] Hazem M. El-Bakry Ahmed A. Saleh and Taghreed T. Asfour. "Design of Adaptive E-Learning for Logic Operations." In: *International Journal of Education and Information Technologies* 4 (2010), pp. 49–56.

[7] Hazem M. El-Bakry Ahmed Abou Elfetouh S. "A Novel Adaptive Mobile E-Learning Model." In: *International Journal of Computer Applications* 63.14 (2013). ISSN: 0975 – 8887.

[8] Mustafa Ahmed. "Implementation of Course Scoring System Based on Spring MVC and Hibernate." In: (2016). http://repository.stcloudstate.edu/csit_etds/13/.

[9] Abdullah Y Al-Zoubi, Sabina Jeschke, Nicole Martina Natho, Jarir Nsour, and Olivier Frederic Pfeiffer. "Integration of an online digital logic design lab for it education." In: *Proceedings of the 9th ACM SIGITE conference on Information technology education*. ACM. 2008, pp. 237–242.

[10] J. N. Amaral, P. Berube, and P. Mehta. "Teaching digital design to computing science students in a single academic term." In: *IEEE Transactions on Education* 48.1 (2005), pp. 127–132. ISSN: 0018-9359. DOI: 10.1109/TE.2004.837048.

[11] Ron Baecker. "Two Systems Which Produce Animated Representations of the Execution of Computer Programs." In: *SIGCSE Bull.* 7.1 (Jan. 1975), pp. 158–167. ISSN: 0097-8418. DOI: 10.1145/953064.811152. URL: http://doi.acm.org/10.1145/953064.811152.

[12] *Binary Search Tree*. Website. http://visualgo.net/bst.html.

[13] Marc H. Brown. *Algorithm Animation*. Cambridge, MA, USA: MIT Press, 1988. ISBN: 0-262-02278-8.

[14] Carl Burch. "Logisim: a graphical system for logic circuit design and simulation." In: *Journal on Educational Resources in Computing (JERIC)* 2.1 (2002), pp. 5–16.

[15] Michael D Byrne, Richard Catrambone, and John T Stasko. "Evaluating animations as student aids in learning computer algorithms." In: *Computers & education* 33.4 (1999), pp. 253–278.

[16] Joel L. Hartman Charles D. Dziuban and Patsy D. Moskal. "Blended Learning." In: *Educase Connect* (2004).

[17] Chaomei Chen and Yue Yu. "Empirical studies of information visualization: a meta-analysis." In: *International Journal of Human-Computer Studies* 53.5 (2000), pp. 851–866.

[18] *Chipmunk Documentation*. Website. http://www.cs.berkeley.edu/lazzaro/chipmunk/..

[19] Shih-Wei Chou and Chien-Hung Liu. "Learning effectiveness in a Web-based virtual learning environment: a learner control perspective." In: *Journal of computer assisted learning* 21.1 (2005), pp. 65–76.

[20] Sarah A.Douglas Christopher D.Hundhausen and John T. Stasko. "A Meta-Study of Algorithm Visualization Effectiveness." In: *Journal of Visual languages and Computing* 13.1 (2002), pp. 259–290.

[21] James M Clark and Allan Paivio. "Dual coding theory and education." In: *Educational psychology review* 3.3 (1991), pp. 149–210.

[22] Douglas Crockford. "The JSON saga." In: *YUI Theater video* (2009).

[23] D DiNucci. "Fragmented Future." In: *Print 53* (1999).

[24] Eckehard Doerry. "An empirical comparison of copresent and technologically-mediated interaction based on communicative breakdown." PhD thesis. University of Oregon, 1995.

[25] Giuliano Donzellini and Domenico Ponta. "A simulation environment for e-learning in digital design." In: *IEEE Transactions on Industrial Electronics* 54.6 (2007), pp. 3078–3085.

[26] Hazem El-Bakry and Mohamed Hamada. "Adaptive E-Learning for Data Encoding and Computer Networks based on Learner's Styles." In: *International Journal of Computer Networks and Security* 22.12 (2012), pp. 333–342.

[27] Martin Fowler. "Inversion of control containers and the dependency injection pattern." In: (2004).

[28] Jesse James Garrett. "Ajax: A new approach to web applications, February 2005." In: *URL http://adaptivepath. com/ideas/essays/archives/000385. php* 7.3 (2007).

[29] Robert Glaser and Lauren B Resnick. *Knowing, learning, and instruction: Essays in honor of Robert Glaser*. Psychology Press, 1989.

[30] Vlado Glavinic, Mihael Kukec, and Sandi Ljubic. "Digital design mobile virtual laboratory implementation: A pragmatic approach." In: *Universal Access in Human-Computer Interaction. Addressing Diversity*. Springer, 2009, pp. 489–498.

[31] Vlado Glavinic, Mihael Kukec, and Sandi Ljubic. "Digital design mobile virtual laboratory implementation: A pragmatic approach." In: *Universal Access in Human-Computer Interaction. Addressing Diversity* (2009), pp. 489–498.

[32] Howard E Gruber and J Jacques Vonèche. "The essential Piaget: An interpretive reference and guide." In: *The essential Piaget: An interpretive reference and guide* (1977).

[33] J. S. Gurka and W. Citrin. "Testing effectiveness of algorithm animation." In: *Proceedings 1996 IEEE Symposium on Visual Languages*. 1996, pp. 182–189. DOI: 10.1109/VL.1996.545285.

[34] Judith S Gurka and Wayne Citrin. "Testing effectiveness of algorithm animation." In: *Visual Languages, 1996. Proceedings., IEEE Symposium on*. IEEE. 1996, pp. 182–189.

[35] CHRISTOPHER D. HUNDHAUSEN, SARAH A. DOUGLAS, and JOHN T. STASKO. "A Meta-Study of Algorithm Visualization Effectiveness." In: *Journal of Visual Languages and Computingv* 13.3 (2002), pp. 259 –290. ISSN: 1045-926X. DOI: http://dx.doi.org/10.1006/jvlc.2002.0237. URL: http://www.sciencedirect.com/science/article/pii/S1045926X02902375.

[36] Charles Hacker and Renate Sitte. "Interactive teaching of elementary digital logic design with WinLogiLab." In: *Education, IEEE Transactions on* 47.2 (2004), pp. 196–203.

[37] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. "The WEKA Data Mining Software: An Update." In: *SIGKDD Explor. Newsl.* 11.1 (Nov. 2009), pp. 10–18. ISSN: 1931-0145. DOI: 10.1145/1656274.1656278. URL: http://doi.acm.org/10.1145/1656274.1656278.

[38] M. Hamada. "Web-based Tools for Active Learning in Information Theory." In: *Proceedings of the 38th SIGCSE technical symposium on Computer science education*. Vol. 38. 2007, pp. 60–64.

[39] M. Hamada. *Supporting Materials for Active e-Learning in Computational Models*. Springer-Verlag Berlin, Heidelbergm, 2008, pp. 678–686.

[40] Rosilah Hassan, Nazlia Omar, Haslina Arshad, and Shahnorbanun Sahran. "A design of virtual lab for digital logic." In: *The 7th WSEAS Int. Conf. On EACTIVITIES (E-Learning, E-Communities, E-Commerce, E-Management, E-Marketing, E-Governance, Tele-Working)(E-ACTIVITIES'08)*. 2008, pp. 29–31.

[41] Norman Hendrich. "A Java-Based Framework for Simulation and Teaching: HADES—the Hamburg Design System." In: *Microelectronics Education*. Springer, 2000, pp. 285–288.

[42] Christopher D. Hundhausen. "Toward Effective Algorithm Visualization Artifacts: Designing for Participation and Negotiation in an Undergraduate Algorithms Course." In: *CHI 98 Conference Summary on Human Factors in Computing Systems*. CHI '98. Los Angeles, California, USA: ACM, 1998, pp. 54–55. ISBN: 1-58113-028-7. DOI: 10.1145/286498.286526. URL: http://doi.acm.org/10.1145/286498.286526.

[43] *Introducing JSON*. Website. www.json.org/.

[44] Tobias Isenberg, André Miede, and Sheelagh Carpendale. "A Buffer Framework for Supporting Responsive Interaction in Information Visualization Interfaces." In: *Proceedings of the Fourth International Conference on Creating, Connecting, and Collaborating through Computing (C^5 2006)*. IEEE, 2006, pp. 262–269. ISBN: 978-0-7695-2563-1.

[45] *JSXGraph-Dynamic Mathematics with JavaScript*. Website. http://jsxgraph.uni-bayreuth.de/wp/index.html/.

[46] Sabina Jeschke, Thomas Richter, Harald Scheel, Ruedi Seiler, and Christian Thomsen. "The experiment in eLearning: Magnetism in virtual and remote experiments." In: *Conference Proceedings ICL*. 2005.

[47] Lilian Jiang. "The web application using SpringMVC." In: *Computer and Modernization* 11 (2013), pp. 167–168.

[48] Rod Johnson. *Expert one-on-one J2EE design and development*. John Wiley & Sons, 2004.

[49] Rod Johnson et al. "Introduction to the spring framework." In: *TheServerSide. com* 21 (2005), p. 22.

[50] Charles Kann, Robert W. Lindeman, and Rachelle Heller. "Integrating Algorithm Animation into a Learning Environment." In: *Comput. Educ.* 28.4 (May 1997), pp. 223–228. ISSN: 0360-1315. DOI: 10.1016/S0360-1315(97)00015-8. URL: http://dx.doi.org/10.1016/S0360-1315(97)00015-8.

[51] *Karnaugh Map Explorer 2.0*. Website. http://www.ee.calpoly.edu/media/uploads/resources/KarnaughExplorer_1.html.

[52] *Karnaugh-Veitch Map*. Website. http://www.mathematik.uni-marburg.de/~thormae/lectures/ti1/code/karnaughmap/.

[53] Tony Karrer. "Understanding eLearning 2.0." In: *Learning Circuits* 7.2007 (2007).

[54] Glenn E. Krasner and Stephen T. Pope. "A Cookbook for Using the Model-view Controller User Interface Paradigm in Smalltalk-80." In: *J. Object Oriented Program.* 1.3 (Aug. 1988), pp. 26–49. ISSN: 0896-8438. URL: http://dl.acm.org/citation.cfm?id=50757.50759.

[55] S. Kubisch, R. Rennert, H. Pfueller, and D. Timmermann. "LoGen – Generation and Simulation of Digital Logic on the Gate-Level via Internet." In: *2006 1ST IEEE International Conference on E-Learning in Industrial Electronics*. 2006, pp. 46–51. DOI: 10.1109/ICELIE.2006.347210.

[56] James A Kulik, Chen-Lin C Kulik, and Peter A Cohen. "A meta-analysis of outcome studies of Keller's personalized system of instruction." In: *American Psychologist* 34.4 (1979), p. 307.

[57] Budi Kurniawan. *Struts 2 design and programming: a tutorial*. Brainy Software Inc, 2007.

[58] Ulrich Lampe, Markus Kieselmann, André Miede, Sebastian Zöller, and Ralf Steinmetz. "A Tale of Millis and Nanos: On the Accuracy of Time Measurements in Virtual Machines." In: *Proceedings of the Second European Conference on Service-Oriented and Cloud Computing (ESOCC 2013)*. Springer, 2013, pp. 172–179. ISBN: 978-3-642-40650-8.

[59] Ulrich Lampe, Qiong Wu, Ronny Hans, André Miede, and Ralf Steinmetz. "To Frag Or To Be Fragged – An Empirical Assessment of Latency in Cloud Gaming." In: *Proceedings of the Third International Conference on Cloud Computing and Services Science (CLOSER 2013)*. 2013, pp. 5–12. ISBN: 978-898-8565-52-5.

[60] Andrea Williams Lawrence. "Empirical Studies of the Value of Algorithm Animation in Algorithm Understanding." UMI Order No. GAX94-08921. PhD thesis. Atlanta, GA, USA: Georgia Institute of Technology, 1993.

[61] Andrea Williams Lawrence. "Empirical Studies of the Value of Algorithm Animation in Algorithm Understanding." UMI Order No. GAX94-08921. PhD thesis. Atlanta, GA, USA, 1993.

[62] Kalloniatis M and Luu C. "The Perception of Color." In: *Webvision: The Organization of the Retina and Visual System* (2007). ISSN: 0018-9359.

[63] *Market Share Statistics for Internet Technologies*. Website. https://www.netmarketshare.com/.

[64] Vladimir Mateev, Svilena Todorova, and Angel Smrikarov. "Test system in digital logic design virtual laboratory: tasks delivery." In: *Proceedings of the 2007 international conference on Computer systems and technologies*. ACM. 2007, p. 80.

[65] Michael Mattsson, Michael Mattsson, and Michael Mattsson. *Object-Oriented Frameworks - A survey of methodological issues*. 1996.

[66] Peter M Maurer. "Electrical design automation: An essential part of a computer engineer's education." In: *Frontiers in Education Conference, 1998. FIE'98. 28th Annual*. Vol. 2. IEEE. 1998, pp. 620–627.

[67] Richard E Mayer and Richard B Anderson. "Animations need narrations: An experimental test of a dual-coding hypothesis." In: *Journal of educational psychology* 83.4 (1991), p. 484.

[68] Richard E Mayer and Richard B Anderson. "Animations need narrations: An experimental test of a dual-coding hypothesis." In: *Journal of educational psychology* 83.4 (1991), p. 484.

[69] Dean A McManus. "The two paradigms of education and the peer review of teaching." In: *Journal of Geoscience Education* 49.5 (2001), pp. 423–434.

[70] André Miede. "Theses and other Beautiful Documents with `classicthesis`." In: *TUGboat – The Communications of the TEX Users Group* 31.1 (2010), pp. 18–20. ISSN: 0896-3207.

[71] André Miede, Gökhan Şimşek, Stefan Schulte, Daniel F. Abawi, Julian Eckert, and Ralf Steinmetz. "Revealing Business Relationships – Eavesdropping Cross-organizational Collaboration in the Internet of Services." In: *Proceedings of the Tenth International Conference Wirtschaftsinformatik (WI 2011)*. Vol. 2. 2011, pp. 1083–1092. ISBN: 978-1-4467-9236-0.

[72] Felix Mödritscher, Silke Spiel, and Victor Manuel García-Barrios. "Assessment in e-Learning environments: A comparison of three methods." In: *Society for Information Technology & Teacher Education International Conference*. Association for the Advancement of Computing in Education (AACE). 2006, pp. 108–113.

[73] M Mohandes, M Dawoud, S Al Amoudi, and A Abul Hussain. "Online Development of Digital Logic Design Course." In: *Information and Communication Technologies, 2006. ICTTA'06. 2nd*. Vol. 1. IEEE. 2006, pp. 42–47.

[74] Paul Mulholland. "A principled approach to the evaluation of SV: a case study in Prolog." In: *Software visualization: Programming as a multimedia experience* (1998), pp. 439–452.

[75] W. A. Awad N. S. Abu El-Ala and H. M. El-Bakry. "Cloud Computing for Solving E-Learning Problems." In: *International Journal of Advanced Computer Science and Applications* 3.12 (2012), pp. 135–137.

[76] Thomas L. Naps. "Algorithm Visualization in Computer Science Laboratories." In: *Proceedings of the Twenty-first SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '90. Washington, D.C., USA: ACM, 1990, pp. 105–110. ISBN: 0-89791-346-9. DOI: 10.1145/323410.323422. URL: http://doi.acm.org/10.1145/323410.323422.

[77] Thomas L. Naps. "Algorithm Visualization in Computer Science Laboratories." In: *SIGCSE Bull.* 22.1 (Feb. 1990), pp. 105–110. ISSN: 0097-8418. DOI: 10.1145/319059.323422. URL: http://doi.acm.org/10.1145/319059.323422.

[78] P Negretti, G Bianconi, and A Finzi. "Visual image analysis to estimate morphological and weight measurements in rabbits." In: *World Rabbit Science* 15.1 (2010), pp. 37–41.

[79] Allen Newell. "Physical symbol systems." In: *Cognitive science* 4.2 (1980), pp. 135–183.

[80] Allen Newell, Herbert Alexander Simon, et al. *Human problem solving*. Vol. 104. Prentice-hall Englewood Cliffs, NJ, 1972.

[81] Allan Paivio and Mark Sadoski. "Lexicons, Contexts, Events, and Images: Commentary on From the Perspective of Dual Coding Theory." In: *Cognitive science* 35.1 (2011), pp. 198–209.

[82] B.A. Price. *A Framework for the Automatic Animation of Concurrent Programs*. Canadian theses. Thesis (M.Sc.)–University of Toronto, 1990. ISBN: 9780315655126. URL: https://books.google.de/books?id=x8jUSgAACAAJ.

[83] G. Puvvada and M. A. Breuer. "Teaching computer hardware design using commercial CAD tools." In: *IEEE Transactions on Education* 36.1 (1993), pp. 158–163. ISSN: 0018-9359. DOI: 10.1109/13.204837.

[84] J. R. Quinlan. "Induction of decision trees." In: *Machine Learning* 1.1 (1986), pp. 81–106. ISSN: 1573-0565. DOI: 10.1007/BF00116251. URL: https://doi.org/10.1007/BF00116251.

[85] Mehrak Rahimi. *Handbook of Research on Individual Differences in Computer-Assisted Language Learning*. Idea Group,U.S., 2015, 2015, p. 571. ISBN: 9781466685192.

[86] David Rashty. "Traditional Learning vs. eLearning." In: (2003). URL: http://www.addwise.com/articles/traditional_learningv.vs.e-learning/.

[87] T Reenskaug. "A note on DynaBook requirements." In: *Xerox PARC, Available on http://heim. ifi. uio. no/~ trygver/themes/mvc/mvc-index. html (accesed: 21 March 2013)* (1979).

[88] G. C. Roman and K. C. Cox. "A taxonomy of program visualization systems." In: *Computer* 26.12 (1993), pp. 11–24. ISSN: 0018-9162. DOI: 10.1109/2.247643.

[89] Guido Rössling and Bernd Freisleben. "AnimalScript: An Extensible Scripting Language for Algorithm Animation." In: *SIGCSE Bull.* 33.1 (Feb. 2001), pp. 70–74. ISSN: 0097-8418. DOI: 10.1145/366413.364541. URL: http://doi.acm.org/10.1145/366413.364541.

[90] Ahmed A Saleh, Hazem El-Bakry, and Taghreed T Asfour. "Design of adaptive e-learning for logic operations." In: *International Journal of Education and Information Technologies* 4.2 (2010), pp. 49–56.

[91] Robert G Scanlon. "Individually prescribed instruction." In: *Educational Technology* 10.12 (1970).

[92] Chr Schaffer, RJ Raschhofer, and A Simma. "EaSy-Sim: a tool environment for the design of complex, real-time systems." In: *International Conference on Computer Aided Systems Theory*. Springer. 1995, pp. 358–374.

[93] Randal L. Schwartz, brian d foy, and Tom Phoenix. *Learning Perl*. O'Reilly, 2011. ISBN: 978-1-4493-0358-7.

[94] Nan C Shu. *Visual programming*. Van Nostrand Reinhold New York, 1988.

[95] *Smartsim*. Website. http://smartsim.org.uk/downloads/manual/smartsimusermanual.pdf..

[96] *Some components of Spring*. Website. https://spring.io/.

[97] Kaye Stacey, Sue Helme, Shona Archer, and Caroline Condon. "The effect of epistemic fidelity and accessibility on teaching with physical materials: A comparison of two models for teaching decimal numeration." In: *Educational Studies in Mathematics* 47.2 (2001), pp. 199–221.

[98] Z. Stanisavljevic, V. Pavlovic, B. Nikolic, and J. Djordjevic. "SDLDS - System for Digital Logic Design and Simulation." In: *IEEE Transactions on Education* 56.2 (2013), pp. 235–245. ISSN: 0018-9359. DOI: 10.1109/TE.2012.2211598.

[99] Zarko Stanisavljevic, Bosko Nikolic, and Jovan Djordjevic. "A module for automatic assessment and verification of students' work in digital logic design." In: *Engineering of Computer Based Systems (ECBS), 2012 IEEE 19th International Conference and Workshops on*. IEEE. 2012, pp. 275–282.

[100] John Stasko. *Software visualization: Programming as a multimedia experience*. MIT press, 1998.

[101] John Stasko, Albert Badre, and Clayton Lewis. "Do Algorithm Animations Assist Learning?: An Empirical Study and Analysis." In: *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*. CHI '93. Amsterdam, The Netherlands: ACM, 1993, pp. 61–66. ISBN: 0-89791-575-5. DOI: 10.1145/169059.169078. URL: http://doi.acm.org/10.1145/169059.169078.

[102] John Stasko, Albert Badre, and Clayton Lewis. "Do Algorithm Animations Assist Learning?: An Empirical Study and Analysis." In: *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*. CHI '93. Amsterdam, The Netherlands: ACM, 1993, pp. 61–66. ISBN: 0-89791-575-5. DOI: 10.1145/169059.169078. URL: http://doi.acm.org/10.1145/169059.169078.

[103] John Stasko, Albert Badre, and Clayton Lewis. "Do Algorithm Animations Assist Learning?: An Empirical Study and Analysis." In: *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*. CHI '93. Amsterdam, The Netherlands: ACM, 1993, pp. 61–66. ISBN: 0-89791-575-5. DOI: 10.1145/169059.169078. URL: http://doi.acm.org/10.1145/169059.169078.

[104] S. A. Steele and G. G. Balazs. "A Course Sequence for the Teaching of Digital Systems." In: *IEEE Transactions on Education* 9.4 (1966), pp. 198–201. ISSN: 0018-9359. DOI: 10.1109/TE.1966.4321988.

[105] Hsin-Yi Tsai, Melanie Siebenhaar, André Miede, Yu-Lun Huang, and Ralf Steinmetz. "Threat as a Service? Virtualization's Impact on Cloud Security." In: *IEEE IT Professional* 14.1 (2012), pp. 32–37. ISSN: 1520-9202.

[106] *Usage of JavaScript libraries for websites*. Website. https://w3techs.com/technologies/overview/javascript_library/all.

[107] Michael Wache. "E-Learning - Bildung im digitalen Zeitalter." In: *Bonn: Bundeszentrale für politische Bildung* (2003), p. 32.

[108] Gregory S Wolffe, William Yurcik, Hugh Osborne, and Mark A Holliday. "Teaching computer organization/architecture with limited resources using simulators." In: *ACM SIGCSE Bulletin*. Vol. 34. 1. ACM. 2002, pp. 176–180.

[109] Norul Huda Yusof and Rosilah Hassan. "Flash notes and easy electronic software (EES): New technique to improve Digital Logic Design learning." In: *Electrical Engineering and Informatics (ICEEI), 2011 International Conference on*. IEEE. 2011, pp. 1–6.

[110] A. Zemva, A. Trost, and B. Zajc. "A rapid prototyping environment for teaching digital logic design." In: *IEEE Transactions on Education* 41.4 (1998), 8 pp.–. ISSN: 0018-9359. DOI: 10.1109/13.728275.

[111] Y. Zhu, T. Weng, and C. K. Cheng. "Enhancing Learning Effectiveness in Digital Design Courses Through the Use of Programmable Logic Boards." In: *IEEE Transactions on Education* 52.1 (2009), pp. 151–156. ISSN: 0018-9359. DOI: 10.1109/TE.2008.921796.

[112] Elena Zudilova-Seinstra, Boris Van Schooten, Avan Suinesiaputra, Rob van der Geest, Betsy van Dijk, Johan Reiber, and Peter Sloot. "Exploring individual user differences in the 2D/3D interaction with medical image data." In: *Virtual Reality* 14.2 (2010), pp. 105–118.

[113] *jQuery: The write less, do more, JavaScript library*. Website. https://jquery.com/.