Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 13 (12): 2217-2224, 2013
ISSN 1812-5654 / DOI: 10.3923/jas.2013.2217-2224
© 2013 Asian Network for Scientific Information

Research on the Improvement of Software Design Pattern Based on Substance-field Analysis and Standard Solutions of TRIZ

[1,2]LI Munan
[1]School of Business Administration, South China University of Technology, Guangzhou, 510641, P.R. China
[2]Guangdong Key Lab of Innovation Methods and Decision Management, South China University of Technology, Guangzhou, 510641, P.R. China

Abstract: The accuracy descriptions and definitions of requirements and contradictions in the engineering projects and the analysis and modeling of resource conflicting are the common challenges and important issues to us. Aimed to the problems and the contradictions in the process of the software design, this study combined the model of substance-field and the analysis methods related with the traditional methodologies of software design. We tried to find a novel approach of representation and description for the requirements of the software engineering and the strategies of improvement on the design methods and the patterns. In the last section of this study, we analyzed the improvement of design pattern of the CAI software based on the MVC pattern. Finally, we can argue that the methods of problems modeling and the improved strategies of the design pattern combined with the TRIZ are meaningful and valuable for other engineering science and the research domain of system science.

Key words: TRIZ, substance-field, analysis, design patter of software

INTRODUCTION

Staring from requirements engineering, software engineering involves multiple stages or parts, such as architecture design, detailed design, coding, testing and deploying and also it needs lots of team communication, coordination and task scheduling, progress monitoring, quality assurance, etc. Software design patterns are put forward upon the summary of mature and successful design experiences in software designing process, intending to integrate the experiences with the design concepts and methods of code reuse and system maintenance (Gamma *et al*., 1994; Yao *et al*., 2003). Successful design patterns have played certain promoting roles in developing software engineering theory and cutting cost in software development and maintenance. But because of the lack of intuition, it is difficult to be understood and mastered by most of the common designers (Gamma *et al*., 1994).

In a large software system of commercial customer, the system quality is not only to meet the demand of function, but the need of such non-functional points as: Maintainability, concurrency, safety etc., should also be satisfied. Give a very simple example: Put forward very high requirements for information security in e-government systems, the design of information encryption mechanism will improve the safety of the software, but will reduce the processing performance of the system. Kluender (2011) believes that the current existence of conflict problem between the functional design of typical and non functional quality index in the design of large software systems. With the current software system more and more complex, various quality indexes of the increasingly high demand, the lack of the effective theory framework and tools to solve these conflicts increasingly prominent (Wang *et al*., 2011). Perhaps, for the simple and intuitive, TRIZ theory and method gradually began to get more and more attention and was introduced into the more and more engineering field besides the design of industrial products and the patent analysis (Jiang *et al*., 2011; Prickett and Aparicio, 2012; Blackburn *et al*., 2012). Wang *et al*. (2011) put forward a solution of trusted software based on the combination between QFD and TRIZ from the view of technical conflict during software development.

Besides, there are not enough effective guiding ideology and methods for the evolution strategy and direction of design pattern itself. At present, there are more than twenty kinds of software design patterns, but the design pattern itself is less intuitive, difficult to be most common design staff fully understand and master. In addition, the evolution strategy and direction of design model also lack the guidelines and effective method (Gao and Niu, 2011; Ma *et al*., 2005). For example: At present, many software systems are proclaimed using

MVC pattern in the design stage. But in practical applications, the problems of poor code maintainability and low code reuse still exist, because developers usually apply fixed open source programming framework without understanding the nature of MVC. And, as a result, the actual code and modules of the software cannot fully comply with the MVC design concept (Cao and Hu, 2005; Li and Cui, 2005).

The substance-field analysis and standard solutions of TRIZ provide a more intuitive way to describe and solve problems, with certain reference significance in improving software design patterns and design process (Tan, 2004; Liu *et al*., 2009). Contradictions and conflicts between environmental constraints and user requirements should be taken into consideration when choosing software design patterns and design process.

SOFTWARE DESIGN AND SUBSTANCE-FIELD ANALYSIS

Actually, there are many analogies and similarities between software designing methods and industrial products. Environment constraints are challenges for both industrial products design and software design (Bonnardel, 2000; Suwa *et al*., 2000). The unpredictable design requirements, as well as the user's requirements of visualization are important means to clarify the user's design requirements further (Woo *et al*., 2005). Huang *et al.* (2009) put forward a novel knowledge network architecture model based on TRIZ. Zou *et al.* (2009) put forward a method to solve the problem of coupled design based on TRIZ.

In traditional software designing process, people mainly use words and UML (Unified Modeling Language) to describe problems and requirements. And because the same word probably means different to different people, the most common communicating way is UML. UML is an object-oriented language for application systems modeling. Though the twelve diagrams defined in UML is in the same level, the best practical experience is used to take the fist-class legend combination as priority, in which structure diagram is equal to <class diagram, object diagram, component diagram, deployment diagram>. In consequence, it is inevitably that business objects have to be defined prior to transactions in business process defined in UML, which a narrow approach is leading to not-fully satisfied business analysis and business process reengineering, as well as potential problems in coping with risk brought by changes in demand. The current UML is too complex and not intuitive enough and is lack of a concise profile, some semantic definition is not precise enough, even contains two meanings and the lack of mechanism of domain modeling requirements (Shao *et al*., 2003). Yao and Wang (2010) also suggested a lack of demand for conflict without ambiguity solution in the traditional UML.

As one of the important problem resolution tools, substance-field analysis is suitable for constructing problems of complex requirements to substance-field model related to the existing technology systems and then, by combining with standard solutions, the best solution can be found. TRIZ theory and method is also becoming the tool and object of integration and combination in many fields (Martin, 2005). Substance-field analysis transforms the technical system to the combination of substance and field and the modeling result is intuitive and easy to understand.

Description of substance-field model and software design problems: Substance-field Model of TRIZ holds the viewpoints that: (1) All of the functions can be broken down into three basic elements, (2) An existing function must be structured by three basic elements, (3) A function will come out from three organically interacted basic elements (Tan, 2004; Zhu *et al*., 2010). As shown in Fig. 1, the most basic Substance-field Model is composed by two substances and one field and more complex models can be structured by adding other substances, fields in series and parallel.

Substance-field Model provides a more intuitive way to model requirement problems in software engineering, for example: Any of the smallest functional design units can be a triple like:

$$\text{F\_unit}=\langle \text{Requestor, Service/Field, Responder} \rangle \quad (1)$$

In Eq. 1, Requestor and Responder are the two basic substances, while Service is the basic field. For the problem description of software design, the smallest function unit should contain at least one “requester agent”-Requestor and at least one “responder agent”-Responder and the realization of this function is a “service field”-Service. For example, the simple function

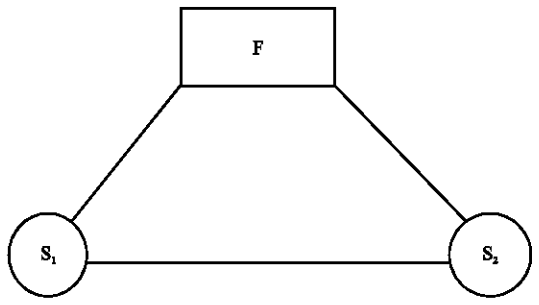

Fig. 1: Basic substance-field model (three elements)

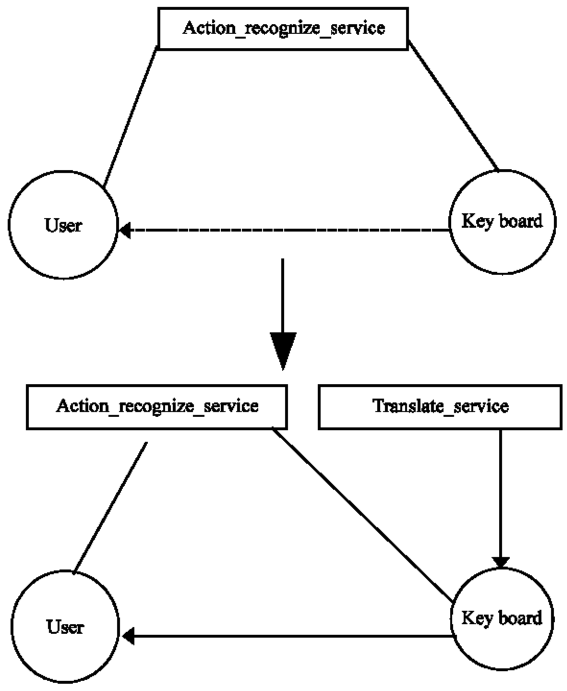

Fig. 2: Adding field in coping with requirement change

"keyboard input" can be described as <user, type and capture services, keyboard>, as shown in the following equation:

$$\text{Keyboard Input}=<\text{User, Field, Keyboard}> \quad (2)$$

If the requirements need future changes, like action recognition service only capture the typing exactly, but what if there is a need for electrical pulse signal conversion service, such as encoding conversion between electrical pulse and Chinese words. And here, another field needs to be introduced to meet the requirements, as shown in Fig. 2.

Substance-field model transformation and design patterns of software architecture: There are five principles in substance-field model transformation (Tan, 2004; Liu *et al.*, 2009), which are important references for problem solving, which also can be used to improve the software design process and design pattern.

Principle 1: Introducing the missing elements to form a complete substance-field model.

Figure 1 is such an example introducing missing elements, but it is just an abridged general view, in a more real circumstance, a "Monitor"- substance should be introduced to display the output of keyboard typing, as shown in Fig. 3.

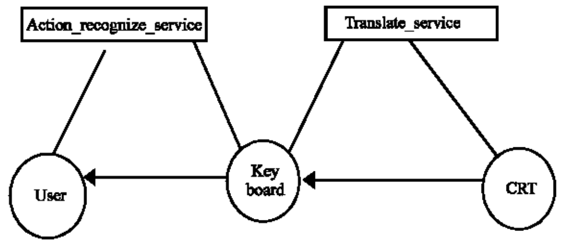

Fig. 3: Introducing "Monitor" to improve substance-field model

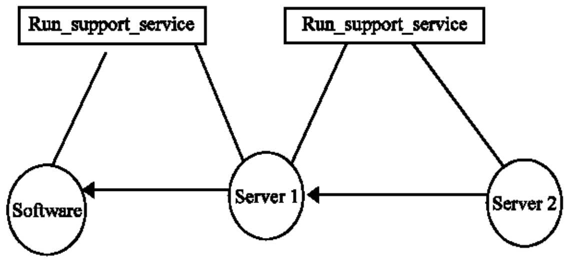

Fig. 4: Functon and performance extension of software systems by adding hardware platform

Principle 2: To extend the existing substance-field model by connecting to substances able to produce fields, to enhance the effect of the current system.

Such extension requirement is relatively common in the software design process, such as, extending software functions to meet the evolving needs of enterprise informatization; adding and updating servers and network equipment to improve system performance and stability.

The substance-field model for function and performance extension of software systems is shown in Fig. 4.

Principle 3: For measurement problems, two fields can be produced by extending, one is input and the other is output.

Testing should be taken into consideration in the software designing process, such as test goal, test plan and test cases. The principle provides a way to test those software problems which are relatively difficult. Additionally, as the complexity of software design is directly related to the workload and cost of the entire software project, how to exactly measure and control the cost of the entire software development process is a problem which should be thought over in the architecture design stage.

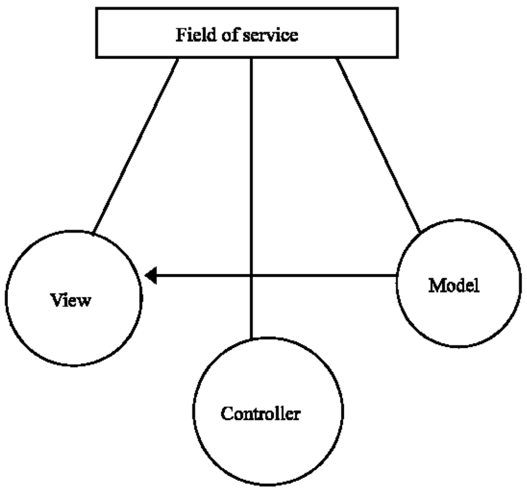

Fig. 5: Substance-field model designed in MVC pattern

Principle 4: Introducing the third element is the most effective way to eliminate a harmful, unnecessary and unwanted substance or field.

In the process of software design, this principle can be understood as the most effective way to eliminate code risk or system security vulnerability and other hidden dangers is introducing a third party, sophisticated encryption software and decryption software and code vulnerability monitoring software. For example, TCP protocol and the 1475 port are used in software system. In order to prevent the 1475 port from being used by hackers, encryption and decryption operations can be done to the TCP protocol packet communicating by this port and thus eliminate safety hazards which may be existing in the system.

Principle 5: If field F2 is a necessary output of the system, F1 is a necessary input and then add a substance as the intermediary of F1-F2.

The intermediary principle of input-output can be reflected to classical MVC software design pattern. The early traditional software design patterns generally present the data and results to users based on the user'ss access and demand. The most serious problem caused by this kind of patterns is that business logic changes may lead to huge changes in the code, eventually resulting in system collapse without maintainability. As the most popular design pattern, MVC patterns manage and monitor user access and view of calculating results effectively using an intermediary (a controller) and improve the ability to handle concurrent access to the system and effective management of limited resources of database connection pool, which is the biggest advantage of MVC. Substance-field model designed in MVC pattern is shown in Fig. 5.

CORRESPONDING ANALYSIS OF COMMON OBJECTS-FIELD MODEL AND SOFTWARE DESIGN PATTERNS

Incomplete substance-field model and singleton design pattern: Incomplete substance-field model means that in the model one or two elements are missing and the requirements cannot be realized. Singleton pattern provides unique access point for object-orient application software, with the greatest advantage of sharing concept for the designing and developing team. It is difficult for a Singleton object class to be sub classed, only if the father class has not been instantiated. In addition, in the developing language and environment of C# (.net), the connection object of database – Connection cannot be designed as a Singleton object, because if Connection is called and shared by application software, there will be exception of connection overflow. And there are lots of problems of Singleton pattern for the common databases. That's why other means or mechanisms should be introduced to make up this defect when using Singleton Pattern.

Disadvantage, incomplete substance-field model and MVC design pattern: With loosely coupled design philosophy, MVC design is to solve the system scalability and maintainability issues, using a relatively independent controller module to control user access and results view. Traditional software systems usually base on C/S or B/S architecture, causing harmful problems, such as occupation of large amounts of data connection resources, low efficiency of system operation, poor readability of code and low maintainability. MVC can partially response or solve the problems of traditional two-tier structure through loosely coupled design philosophy.

Insufficient substance-field model and façade design pattern: Insufficient Substance-field model is such kind of models with complete elements, but without enough functions to meet the user's needs. Façade pattern provides an easy public access to deal with complex subsystems, without reducing functions of subsystems. Façade eliminates the complex of subsystem, direct links between customers and subsystems and links between subsystems. Each of the subsystems has its own Façade pattern, by which they get access to other subsystems. The disadvantages of Façade are limiting the personalized needs of customers and reducing the system flexible, which will lead to the conflicts between users and software in practical operations. Such conflicts seldom can be resolved under Façade pattern, which is typical problem of lacking effectiveness. And the improving

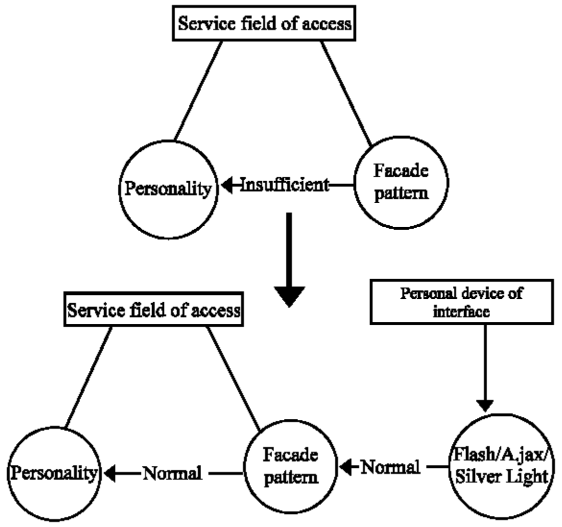

Fig. 6: Introduce personalized service field and specified page technology to meet users' needs

method for this problem which is based on insufficient Substance-field model is shown as below in Fig. 6.

In the traditional Façade pattern, add a service field "user friendly interface" to provide a friendlier user perception and interaction, for example, add Flash element or take Ajax technology into the original static pages (Jsp, Aspx, Html, etc.,) to improve the interactive performance and personalization of the system interfaces under develop environment of Let, the frame of SilverLight also can be considered to improve the interactivity and flexibility of interfaces without destroying the original design model.

STANDARD SOLUTIONS APPLICATION IN SOFTWARE DESIGN PATTERNS

In TRIZ, there are 76 standard solutions in correspondence with the results of substance-field analysis, which are standards solutions for invention in essence, not especially for problems in software design process. But there are no big differences in basic application processes and the process of software design is the process solving the conflicts between user's needs and resources (constraints of developing cost, tools, hardware and networks and interfaces, etc.,).

In Fig. 7, the numerical codes represent the codes of TRIZ material-field model standard solutions correspondingly (Tan, 2004). For example, 1.1.1 means the solution is "transferring to complete substance-field model from uncompleted one" and 1.1.2 means "establishing internal synthetic substance-field model by introducing appendages to the substance internally", etc.

INSTANCE: DESIGN OF CAI SOFTWARE BASED ON MVC MODEL

Computer Aided Innovation-CAI is a high-tech integrating innovating problem-solving ways, modern design method, semantic processing and computer software technology, which can be presented as the following triple (Yang *et al*., 2009; Lan *et al*., 2008):

$$\text{CAI} = \langle\text{innovation theory, Innovation technology,Information technology}\rangle \quad (3)$$

In Eq. 3, the innovation theory mainly is TRIZ; Innovation technology includes problem analyzing, semantic processing, knowledge base, patent query and program evaluation, etc., and IT means the technologies related to analyzing, designing, coding, testing and deploying in the CAI software implementation. MVC is one of the currently mainstream software design models, which has been widely promoted and used, like SSH Framework (Struct-Spring-Hibernate) based on Java and "yii" or "codeigniter" developed using PHP. Traditional MVC model takes loose coupling between modules into consideration and puts emphasis on the independence of user request to respond.

CAI software is often sorted into innovative method platform and innovative knowledge platform. Suppose that it refers to Pro/Inovator (Yang *et al*., 2009) in requirements analysis. Pro/Innovator is one kind of CAI software belonging to American IWINT Company, which is one of the comparatively mature software products in CAI technology application and research and the famous CAI software domestic is Invention Tool, developed by Hebei University of Technology (Yang *et al*., 2009). There are mainly ten modules or functions in Pro/innovator, including project/module navigation, technology systems analysis, problem decomposition, solutions, innovation principles, patent search, patents generation, program evaluation, report generation and knowledge base editor. Suppose that the system takes MVC model based on PHP, like Framework codeigniter (similar to Framework SSH based on Java) to design CAI software, then the MVC model is shown in Fig. 8.

As MVC model shown in Fig. 8, each function module is separated into three parts, controller, model and view. Take problem decomposition and patent query in the CAI software requirements for example, supposing these two modules are designed as:

Problem_Decompose=<Problem_Decompose_controller, Problem_Decompose_Model, Problem_Decompose_View>;
Patent_Query = <Patent_Query_Controller, Patent_Query_Model, Patent_Query_View>

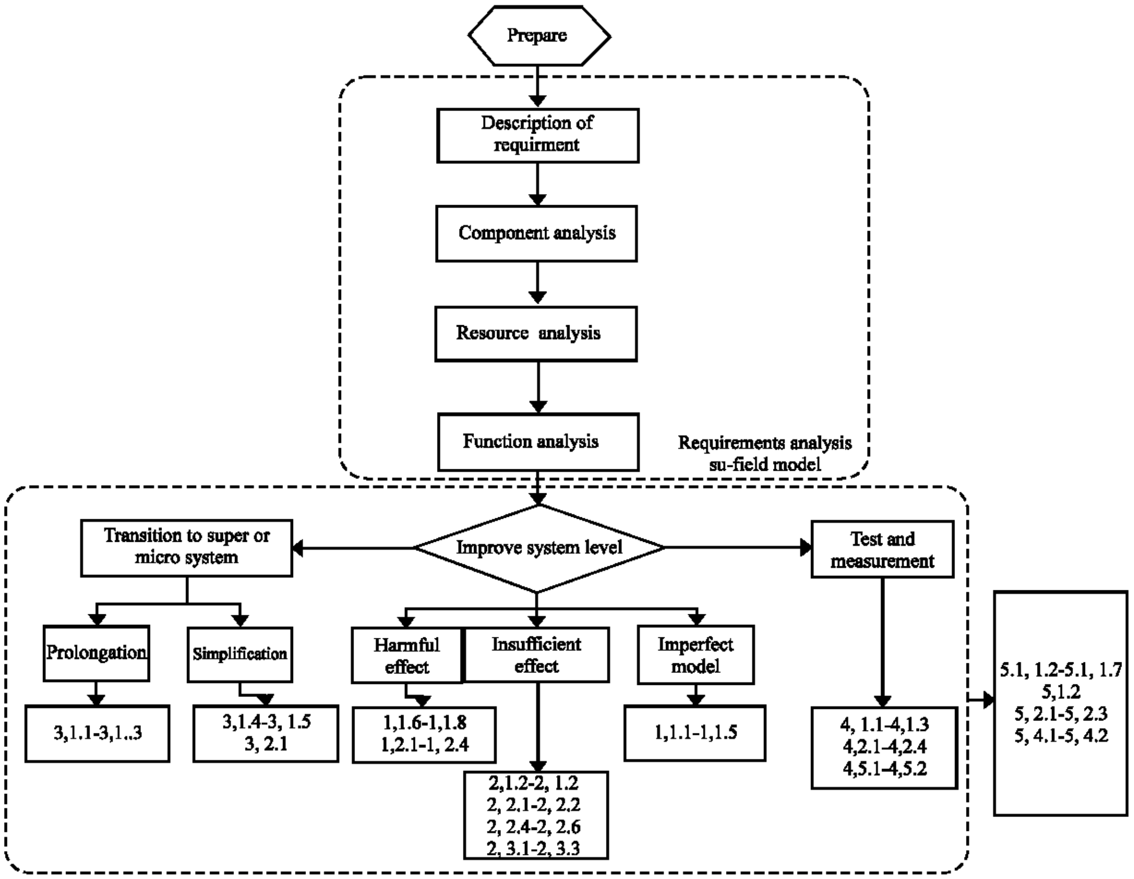

Fig. 7: Software requirements analysis and design based on su-field model and standard solutions of TRIZ

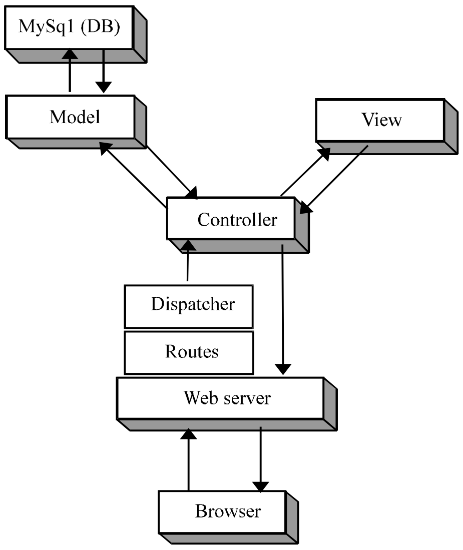

Fig. 8: Software designing based on MVC

When "Inventive problem decomposition" is undertaken, the Problem_Decompose_Controller is called first, then Business Model is visited by Controller, then Business Model interacts with data and then feedback is shown the user by view and controller through rendering. In this design model, the functional modules are separated with loose coupling. When it is required to embed "Patent_Query Module" in "Problem_Decompose Module", the details of "Patent_Query Module" (Model and view) can be copied into Problem_Decompose, which is not economic with low maintenance in code and the code maintenance later will be extremely complicated and costly if all modules in need of embedded Patent_Query in this way. And this indicts that horizontal communication capacity between modules is weak for traditional MVC Model, lacking interaction and communication effect between modules. Using the design process and model in Fig. 3, the standard solution can be found, as in Table 1.

According to the standard solutions 2.1.1, 2.2.4, 2.2.6 in Table 1 and then we can get the improved substance-field analysis is shown in Fig. 8, by introducing view-driven TextPattern Framework on the basis of codeigniter framework of traditional MVC model and enhancing interact and communicate effect between

Table 1: Improvements of MVC software design model based on non-efficient standard solutions

Name of standard solutions	Sub-solutions	Corresponding Improvements
2.1 Transfer to complex substance-field model	2.1.1 Introduce substance and transfer to tandem substance-field model	Introduce new design pattern or developing framework to make up horizontal interaction
	2.1.2 Introduce field and transfer to parallel substance-field model	Introduce new interface interactive technology and service-field, like Ajax, Silver Light to drag and layout the interfaces flexibly
2.2. Enhanced substance-field model	2.2.1 Use field easier to control as an alternative	Persuade client to give up the requirements of embedded modules
	2.2.2 Enhance the transfer of tool, type of substance, to microscopic control	Give up MVC model, design functions at a smaller particle size, turn to view-driven models to meet user requirements, but with the potential risk of tight coupling
	2.2.4 Improve the dynamic of substance	Introduce layering concept, breakdown function particle size, introduce view-driven
	2.2.5 Add new field	Introduce new technology and interface interactive service
	2.2.6 Add new substance	Add new design framework, modify the original MVC design
2.3 Use frequency to realize enhanced substance-field model coordinately	2.3.1 Match the frequency of the field and substance in the mode _ior don't match_j	N/A
	2.3.2. Match the frequency of the field and field in the mode _ior don't match_j	N/A
	2.3.3. use periodic principle	N/A

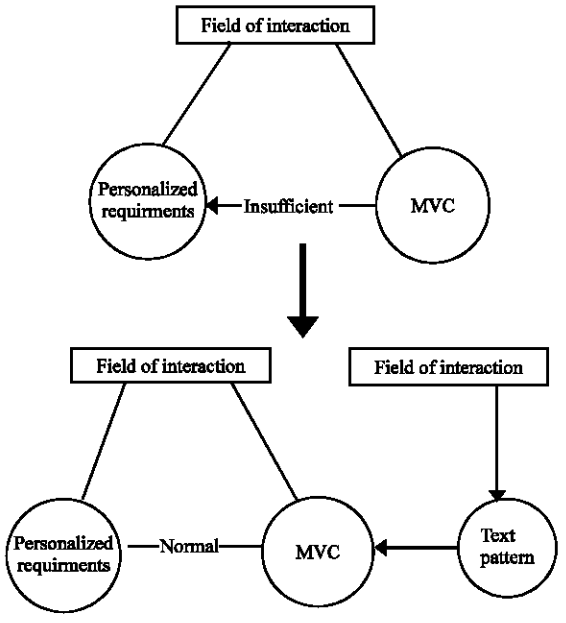

Fig. 9: Improvement of MVC design pattern of CAI software

different modules without lowing the loose coupling of the MVC model.

In Fig. 9, an improvement strategy is given, based on view-driven combing MVC, which is not the unique solution and in fact, MVC model can be altered to view-driven model combing with sampling-factory model, or hierarchical MVC, namely HMVC etc.

CONCLUSION

Currently, there are very few fruits of research combining TRIZ and software engineering and software design pattern is an evolving system itself. In the study, a relatively new solving and improving solutions are discussed about software design pattern from the perspective of substance-field model and standard solutions of TRIZ. Compared with the traditional software engineering modeling languages, the substance-field of TRIZ is in advantage, more intuitive and easier to understand. Besides, substance-field analysis emphases on describing and breaking down problems and conflicts, making up disadvantages of traditional process modeling software, like UML, in describing requirement problems and resource conflicts. The standard solutions, based on substance-field, expand software design ideas and requirement problem solving strategy greatly. But, the combination of TRIZ theory and methods with software engineering is quiet a new study area and the correspondence of solving strategies for TRIZ problems and solving methods for software design problems needs further research. Because of the limited professional ability and knowledge field, there may be some controversies about the concepts and models, but as a positive attempt, this research is hopefully to be the basis for further study.

ACKNOWLEDGMENT

This study is partly sponsored by the Major Project of China National Social Science Fund (No. 11 and ZD154), Open Research Funds of Guangdong Province Key Laboratory (No. 2011A06090100101B) and Fundamental Research Funds for the National Universities (No. 2013XMS03).

REFERENCES

Blackburn, T.D., T.A. Mazzuchi and S. Sarkani, 2012. Using a TRIZ framework for systems engineering trade studies. Syst. Eng., 15: 355-367.

Bonnardel, N., 2000. Towards understanding and supporting creativity in design: Analogies in a constrained cognitive environment. Knowledge-Based Syst., 13: 505-513.

Cao, C.P. and D.M. Hu, 2005. Design of Web test system on MVC design pattern. J. Univ. Shanghai Sci. Technol., 27: 459-462.

Gamma, E., R. Helm, R. Johnson and J. Vlissides, 1994. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Boston, USA.

Gao, S. and Z.Y. Niu, 2011. Practice of agile principles and design patterns. J. Comput. Appl., 31: 147-152.

Huang, S.Q., F.Y. Xu and F. Dai, 2009. Integrated architecture for TRIZ based on knowledge network and its key technologies. J. Zhejiang Univ. (Eng. Sci.,), 43: 2244-2249.

Jiang, J.C., P. Sun and A.J. Shie, 2011. Six cognitive gaps by using TRIZ and tools for service system design. Expert Syst. Appl., 38: 14751-14759.

Kluender, D., 2011. TRIZ for software architecture. Procedia Eng., 9: 708-713.

Lan, F., B. Qin and Y.J. Liang, 2008. The product innovation tool-research on CAI technology. Equipments Manufacturing Technol., 4: 108-111.

Li, Y. and D. Cui, 2005. Improvement and application of MVC design patterns. Comput. Eng., 31: 95-97.

Liu, X.Z., Y.J. Ji and G.N. Qi, 2009. Application of systematic design approach based on theory of inventive problem solving. J. Zhejiang Univ. (Eng. Sci.,), 43: 2244-2249.

Ma, Z., Y. Zhou and S.B. Xie, 2005. Research and applications of MVC design pattern in NMS. J. UEST China, 34: 638-641.

Martin, G.M., 2005. How combinations of TRIZ tools are used in companies-result of a cluster analysis. R D Manage., 35: 285-296.

Prickett, P. and I. Aparicio, 2012. The development of a modified TRIZ technical system ontology. Comput. Ind., 63: 252-264.

Shao, W.Z., Y.B. Jiang and Z.Y. Ma, 2003. The present problems and roadmap of UML. J. Comput. Res. Dev., 40: 509-516.

Suwa, M., J. Gero and T. Purcll, 2000. Unexpected discoveries and s-inventions of design requirements. Design Stud., 21: 539-567.

Tan, R., 2004. Theory of Inventive Problem Solving. Science Press, Beijing.

Wang, S.H., D. Samadhiya and D. Chen, 2011. Software development and quality problems and solutions by TRIZ. Procedia Comput. Sci., 5: 730-735.

Woo, J.Y., S.M. Bae and S.C. Park, 2005. Visualization method for customer targeting using customer map. Expert Syst. Appl., 28: 763-772.

Yang, B., Y. Tian, R. Tan and J. Ma, 2009. Research and development of CAI system based on standard solutions. China Mechanical Eng., 20: 704-708.

Yao, S., H. Guo and T. Wang, 2003. Description of object-oriented software architecture using design patterns. J. South China Univ. Technol. (Nat. Sci. Edn.,), 31: 15-22.

Yao, Q.Z. and J. Wang, 2010. Software formalization requirements analysis and verification based on UML. Comput. Eng., 36: 30-33.

Zhu, H.F., Y. Li and W.Q. Li, 2010. Based on the idealized design of substance-field model. Machinery Design Manufacture, 12: 6-8.

Zou, F., Z. He and A.B. Yu, 2009. Research and application of the method for uncoupling design by TRIZ. J. Harbin Inst. Technol., 41: 265-268.