Developing Software Using A Novel Event-Condition-State Pattern Framework: Taking Mobile Input Method System As A Case Study

Fei Zhu[1,2]
1 School of Computer Science and Technology, Soochow University
2 Provincial Key Laboratory for Computer Information Processing Technology, Soochow University
Suzhou, Jiangsu, China, 215006
Email: zhufei@suda.edu.cn

Hongjun Diao[1,2*] and Wei Huang[1,2*]
1 School of Computer Science and Technology, Soochow University
2 Provincial Key Laboratory for Computer Information Processing Technology, Soochow University
Suzhou, Jiangsu, China, 215006
Email: {hjdiao, huangwei}@suda.edu.cn

***Abstract*—Design patterns add more reliability, flexibility and reusability to a software system. Taking advantage of design patterns is usually beneficial to software design and makes software development relatively easier. The state pattern proposed by GoF is not a determined design and can hardly be transformed into implementation code directly. It is disable to cope with sophisticated response event either. Hereby we propose a novel event-condition-state pattern framework, an event-based finite state machine. The framework improves GoF state pattern by re-segmenting Context into two classes, one for management and the other for operation interface. Essential components for implementation, such as management, state transferring, and event triggering mechanism, are also taken into consideration in the framework. We use pattern framework to develop a thirty party input method system which has a Windows Mobile system version and a Symbian system version separately. We rewrote Context part related with system interfaces in the two different systems, and encapsulated the part related with internal logic of input method by concrete subclasses, thus avoiding redundant developing work in both systems and providing convenience for later synchronous updating and maintenance of two input methods. With the pattern framework, much effort during software development is saved. The development logic seems simpler and clearer. As a result, the whole development process is simplified.**

***Index Terms*—software architecture, design pattern, event condition state, mobile developing, input method**

I. INTRODUCTION

Design pattern is widely used in software development, such as Internet game, utility software, and system software. As a behavioral software design pattern, state pattern is deeply favored by software developers. It is used to represent the states of an object and is often applied to develop software that has multiple states. State pattern is a clean way for an object to partially change its type at runtime. For example, documents in office automation system with workflow typically have multiple states, including editing, waiting for auditing, being audited, replying and forwarding[1][2]. Microsoft Pinyin input method system[3] can be in one of the states, including initial state before input, input state, and selecting state after input. Obviously, state pattern can work in these two kinds of software.

The state design pattern proposed by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides[4] is one of milestones in software engineering domain. However it is not a determined design that can be transformed directly into code and when coping with sophisticated response event, the state design pattern shows its nature disability. One of the common methods to implement the state design pattern is to use IF-ELSE-IF statement provided by most programming languages. However too many branches of IF-ELSE-IF will make the program structure and logic ambiguous, making it harder for future potential modification and updating. Meanwhile, taking advantage of event-response which is provided by modern programming technology, it is of benefit to refine the similar functional structures and change IF-ELSE-IF statement to event-response ones so as to promote readability and reusability as well as relieving design efforts.

In this paper, we propose a novel event-condition-state pattern framework, which is an event-based finite sate machine, for software development. With the pattern framework, we developed a third party input method system which has a Microsoft Mobile version for Microsoft Mobile system and a Symbian version for Symbian system. We found it easy to fulfill development with the help of the pattern framework. We did not carry out two developing activities for input method system. Instead, we utilized common part of the systems with sub pattern model and then rewrote related interfaces to

doi:10.4304/jsw.6.8.1506-1512

different systems. By this, we spared lots of redundant efforts in developing, and gained convenience in later synchronous updating and maintenance.

II. Related Work

In software engineering, a design pattern is a general reusable solution to a commonly occurring problem in software design. It is a description or framework for how to solve a problem that can be used in many different situations but not a determined design that can be transformed directly into code. It explains why a particular situation causes problems, why the proposed solution is considered a good one, and when it is applicable [5].

Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, referred as GoF, in their software engineering book [4], described recurring solutions to common problems in software design. Design pattern is especially important in software engineering. Many researchers have devoted themselves to study on design pattern.

In [8], Shahir, Kouroshfar and Ramsin proposed a method for using design patterns in the context of model transformation, refactoring real-world models through application of these patterns. The results showed its applicability to real-world models.

Zamani and Butler figured out in [9] that in designing with patterns, three aspects of the pattern language that should be taken into consideration: Structural, Syntactic, and Semantic. They proposed formalisms for representing the Structural, Syntactic, and Semantic aspects of a pattern language. And finally, they selected a pattern language in the domain of enterprise application architecture, and showed how to describe the pattern language using the proposed formalism, thus achieving automatic design model checking.

Bayley and Zhu focused on the composition of design patterns in [10], defining a notion of composition of patterns with respect to overlaps based on two operations on design patterns, which was illustrated by the composition of Composite, Strategy and Observer patterns. In paper [11], they advanced an approach that used first order predicate logic to specify design patterns by capturing the dynamic behaviour represented in sequence diagrams.

Medve and Ober addressed the pair-wise direct mapping of service factories from the problem domain to POSA2 features in [12], where the software components were implemented in the context of the solution domain by weaving GoF features. They took advantage of SDL's allowance to describe rigorously systems at the model level, which has been successfully used on large scale developments using intensive simulation and code generation.

Design-pattern is a reusable solution to a commonly occurring problem in software design. If design-patterns could be captured and reused in reverse engineering, the reverse engineering would be very helpful to developers and maintainers of the software. Thus there have been many attempts to detect design-patterns during reverse engineering. Lee and his group, in paper [13], proposed taxonomy of GoF design patterns that could guide the reverse-engineering process, which did not only combine static analysis with dynamic analysis but also added the implementation-specific analysis. They applied the approach to a number of applications, and demonstrated the reverse engineering process more accurate.

Shi and Olsson presented an automated pattern detection approach in [14], which was based on reclassification of the GoF patterns by their pattern intent. They argued that the GoF pattern catalog classifies design patterns in the forward-engineering sense and their reclassification was better suited for reverse engineering. The experiments showed their implementation could detect patterns in programs and package including Java AWT, JHotDraw, Swing and Apache Ant.

In [15], Forbrig and Lämmel discussed an experimental programming language PaL, which was based on a compilation to Eiffel and developed a PaL library with the 23 GoF patterns. They also introduced a new form of abstraction so as to capture reuse schemes for patterns.

Dlamini, Olivier and Sibiya, in [16], extended work on a forensic model for Logical Traffic Isolation (LTI) based on Differentiated Services (DiffServ) and designed the LTI model, a three-tier architecture, using different design patterns. They focused on three design patterns in modeling the LTI architecture to achieve reusability and flexibility of the LTI model, which was composed of tDiffServ network and the sink server.

So far the most common technique used to implement the pattern is by object oriented state design pattern, in which states are represented as descendants of a common interface class that declares event handler functions. A context class delegates all events for processing to the current state object. State transitions are accomplished by changing the current state object. [17] and [18] extended and refined this basic pattern and presented several patterns to solve problems including implementing state transition mechanisms and composite states, etc.

III. Design Pattern Improvement

GoF design pattern is not a kind of particular technique. It is an idea or a kind of thought. It does not only display flexible application of interface as well as abstract class and its intelligence, but also iteratively emphasize on making the program reusable, which has prove to be an important trend. However it is a challenge to current state as software requirement keeps changing so quickly and frequently that corresponding designs cannot keep up with the changes. Therefore it is a wise way to find out invariable part and then separate it from variable part.

As we can see in figure 1[19], although GoF described a reasonable state pattern, their framework only gave relations among context objects and state objects in short, which is not enough for implementation. There normally are many response events, causing the structure of handling method very much complex, which will lead to inefficiency in software development.

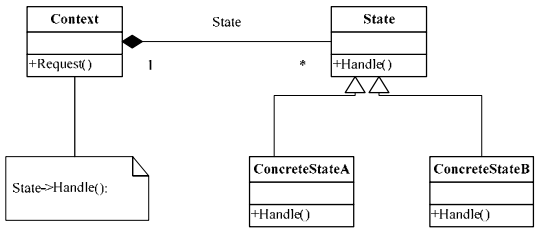

Figure 1. State pattern class diagram which merely shows relations among context object and state object without interpreting details of implementation

As GoF state pattern did not give sufficient details for implementation, developers can complete in various ways. Some prefer setting properties of Context in method Handle. Some use state transferring table in method Request of object Context to retrieve state table to fulfill state transferring, which separates concrete rules out of state pattern.

Context in GoF state pattern is used as user operation interface and also used for state class management. User operation part is usually related with dedicated application, while state management part is normally unchanged in each state pattern program. Therefore, we can separate it to two objects. One is for client object of user operation interface, changing with applications. The other is for state management, achieving state transferring by retrieving and interpreting state transferring table. Classes and their relations of the whole state pattern are shown as figure 2 [19].

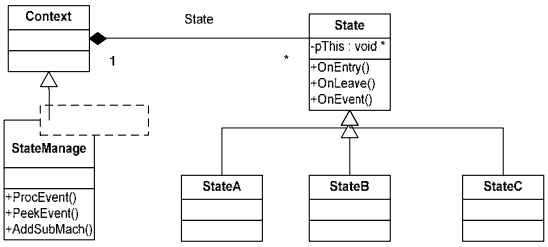

Figure 2. Novel state pattern class relation diagram which demonstrates relations among Context, State and StateManage as well as details of implementation

In figure 2, state management class StateManage inherits from client class Context, while StateManage has an abstract class State to represent state. Taking advantage of StateManage, we can separate class Context of dedicated application from the framework to maximal extent, including functionalities in class State that are related with state management and indexing state transferring table, and embrace user event mapping mechanism in class State. As these functionalities normally keep unchanged, we can place them in event-condition-state general framework as constant part.

A. State Management

States should be maintained properly. StateManage is mainly used for that. It has three methods: ProcEvent, PeekEvent and AddSubMach. ProcEvent is entrance to the user defined processing of event. PeekEvent is used to see if the user event has related processing method and if it would result in state transferring. AddSubMach is responsible for adding state transferring table to current state management object.

In ProcEvent, some mechanism, initially, seeks for executing OnEntry. Otherwise, if no OnEntry is found, a default processing mechanism will be automatically woken up. Then state transferring table will be queried according to existence of state transferring. The query result will be used to determine whether state transferring is essential.

PeekEvent queries user event processing method mapping table of current state and then finds out whether default processing method for event will deal with current event. Finally PeekEvent decides whether user event and current query state transferring table will cause state transferring.

State transferring table has five elements: CID, iFirst, iLast, iCond and NID. CID and NID are identifiers of some state object, using object address, where CID is used as current identifier and NID represents next identifier. iFirst and iLast are used to represent the range of one particular kind of event that triggers the same transferring. iFirst is the starting value and iLast means the ending. iCond is the transfer event condition that user's processing will effect transferring event.

B. State Representation

Abstract class State, designed to represent state, has two properties: pthis and EventMap, as well as three methods: OnEntry, OnLeave and OnEvent. The property pthis tracks address of object Context for convenient access to state values. Property EventMap, an array of TEvent structure, is used to store mechanism of mapping user event to event processing function in concrete class. OnEntry, OnLeave and OnEvent are designed to deal with corresponding event.

Structure TEvent has three elements: iFirst, iLast and lpDelegate. iFirst is used for representing the starting value of class transfer event. iLast keeps the ending value. lpDelegate is a pointer of M_Delegate type, which is an abstract class with function delegation calling mechanism.

Event processing function mapping in concrete state class is implemented by class M_Delegate that inherits a template class CKeyDelegate[20][21]. The definitions[19] of M_Delegate and CKeyDelegate are as follows.

Definition of M_Delegate:

```
class M_Delegate{
public:
   virtual int Invoke(short sPara ,long lPara)=0;
   virtual void DestorySelf()=0;
};
```

Definition of CKeyDelegate:

```
template<class T>
class CKeyDelegate:public M_Delegate{
typedef int (T::* LPF)(short,long);
public:
   CKeyDelegate(T* lpObj,LPF lpFuc){
      lpOO=lpObj;
      lpMethod=lpFuc;
   }
```

```
    virtual int Invoke(short wParam,long lParam){
          return (*lpOO.*lpMethod)(wParam,lParam);
        }
        virtual void DestorySelf(){
          delete this;
        }
  private:
    T* lpOO;
    LPF lpMethod;
  }
```

Here CKeyDelegate is a template class. It overrides method Invoke in M_Delegate. Any method in concrete state class with signature of int (T::* LPF)(short,long) type can be encapsulated as function delegation type and be triggered to execute by method Invoke in M_Delegate. Such kind of function delegation composes mapping relation of user event and concrete state class together with iFirst and iLast in TEvent structure.

Method OnEvent maps user event to related processing function. It queries user event processing method to map current state table. If the processing method is retrieved, related processing method will be executed and result will be returned. Otherwise, a null value indicating no related function found will be returned.

Moreover, method AddEvent can be used to add related processing function for some user defined event. By the mechanism, entering current state will trigger method OnEntry and leaving current state will trigger method OnLeave accordingly.

IV. Design Mobile Input Method With Event-Condition-State Pattern Framework

Demands for intelligence in mobile phone urge development of intelligent mobile phone. Popularity of operation system in mobile phone makes it possible to develop a third party input method system for it. Meanwhile there are various mobile models and systems presently, making it difficult to develop an input method software for them. Moreover, the future update and maintenance of input method will get into trouble as well.

As we can see, although a mobile phone may have different input methods, mobile phone will eventually be in one of the three states which are numeric number input state, English input state and non-English input state. These three states have similar state transferring rules to the ones in event-condition-state pattern. At the same time, each input state can be described by a sub state model. For example, English ABC, an English input method, will be in initial state before user input and in input state after user input. If there is no input within a predefined interval, the system will return to initial state. The whole view of state transferring is shown in figure 3.

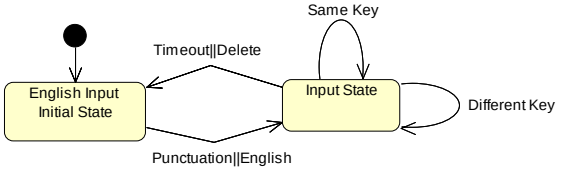

Figure 3. State transferring diagram of English input method

There are usually far less keyboards in mobile than in PC, usually including numeric key 0 to numeric key 9, key shift and key star. To save as many keys as possible, one single key is normally given several alternatives in different cases, e.g. numeric key 0 to numeric key 9 represent numeric 0 to 9 separately in numeric number input state. However, they are used as candidate keys in other situations, e.g. they are used as English candidate keys in English input mode. There will be even more complicated combinations in non-English inputting methods. For instance, in Chinese character input method, they are treated as input candidate keys and they will be turned to selecting keys when a user is selecting from candidate Chinese character. So it is the same as other keys. Key shift is used as shifting and key star is used to input character star. Meanwhile they have other meanings in different input methods.

Although such one key with multiple interpretations is sophisticated, it conforms to the event-condition-state pattern. Thus, it is possible to develop input method systems in mobile with the framework proposed.

Here, we use client interface class Context to encapsulate interfaces to mobile operation system, internal data and message processing component, of which two main interfaces are PressKey which receives and processes information when the key is pressed, and CommitResult which delivers character that is selected by the user. Figure 4 shows general view of state transferring process.

However there are still exceptions in practice. For example, some special inputs merely use part of the transferring rules, and some rules in use do not pertain to rules shown in figure 4. To gain maximum portability and dynamic capability, we use XML to describe state transferring table. Once rules are changed, what we should do to adapt to the changes is just to alter values, add, or remove some items in XML-based transferring state table description document, which will not affect the whole input model. Part of the XML-based transferring state table description example document is as follows.

Example of XML-based transferring state table description:

```
<author: dataAttribute xmlns: author ="FEI ZHU">
    <ID name="Begin">
        <iEvent iFirst=”-1” iLast=”-1” iCond=”-1”
        nID=”Chinese”/>
    </ID>
    <ID name="Search">
        <iEvent iFirst=”0” iLast=”9” iCond=”1”
        nID=”Input”/>
        <iEvent iFirst=”0” iLast=”9” iCond=”0”
        nID=”Warr”/>
    </ID>
</ author: dataAttribute>
```

Here ID and nID are state identifiers. iFirst and iLast are used to represent range of events that will be accordingly triggered and can be classified into one category. iCond is condition of transferring event.

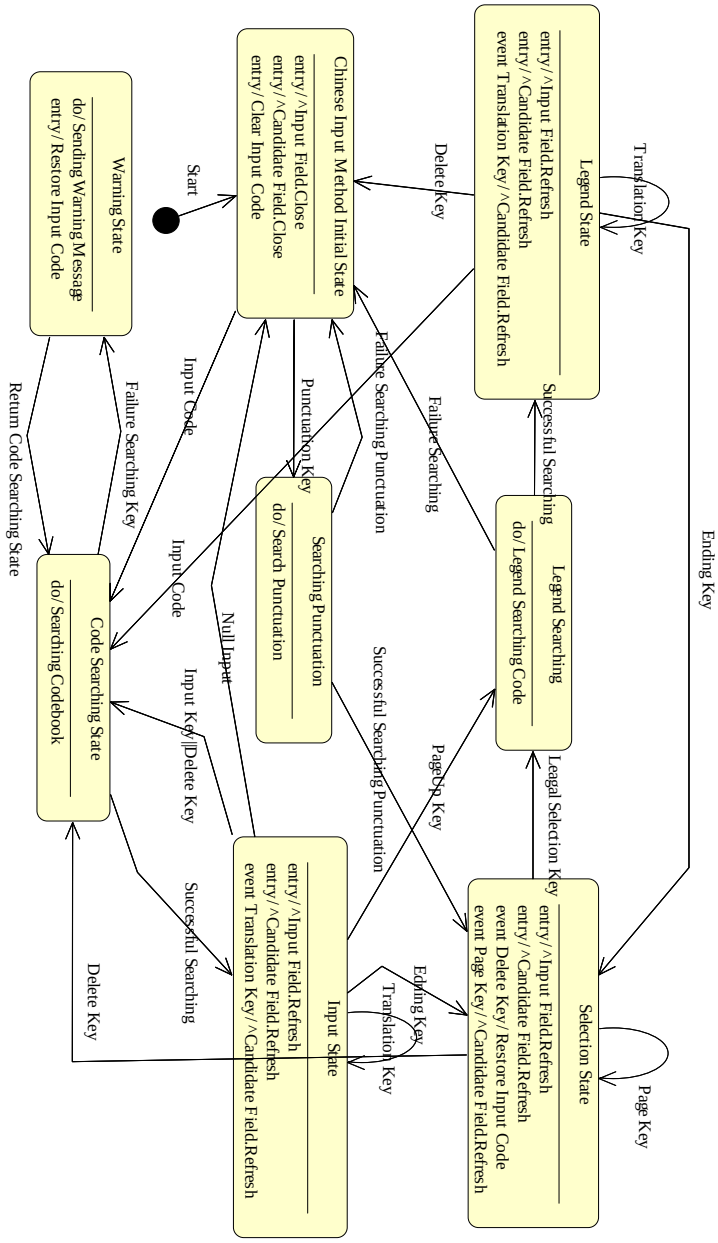

Figure 4. State transferring diagram inside input method which shows main states in non-English input method, and state transferring processes

V. DEVELOP MOBILE INPUT METHOD USING THE PATTERN

A. For Windows Mobile System

Windows Mobile is an open and intelligent mobile system. It provides developer a set of interface functions that are similar to those of Windows system[22][23]. These interface functions are dynamic libraries between system and application. They get user's pressing key message, and send one or more characters to application after processing. Their relations are shown in figure 5.

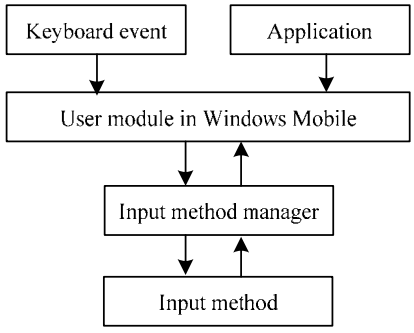

Figure 5. Relation between input method and system in Windows Mobile system

We can see from figure 5 that it is input manager that interacts with input model directly. User's pressing key message is passed to input method model through input method manager. Characters from input method are also delivered to application through input manager.

In fact, input method manager can interact by input method user interface or by function interface of dynamic library. The difference is that input method user interface is the interface for input manager, while function interface is interface for input manager to maintain the whole input method modules. When applying the model in Windows Mobile system, a Context object is necessary to implement these two kinds of interact interfaces requiring by input method manager. The following is part of key code of Context in Windows Mobile system.

```
class Context:public CWinIME{
    PreRequest(int iEvent,short wParam,long lParam);
    Request(int iEvent,short wParam,long lParam);
    :
}
```

Here CWinIME is an UI class for input method, which encapsulates all UI message processing codes for input method. Class Context provides UI by inheriting from class CWinIME. Method Prerequest and method Request are two main entries for users to manipulate state machine. Prerequest is used to query how state machine will deal with current message and Request is used to deliver current message to state machine for processing.

B.For Symbian System

Symbian OS is an operating system designed for mobile devices, with associated libraries, user interface frameworks and reference implementations of common tools. It provides an API for input method, called FEP (Front-End Processor), to developers[24]. Developers can implement the interface by inheriting from class CcoeFep which exists in form of polymorphous and dynamic link library[25]. It receives user's pressing key message, and sends characters to application for processing after input method system completes coping with the message.

The whole input method system in Symbian is composed of FEP and FEP server[24][25]. FEP deals with internal logic of input method. FEP server acts as input method server and is in charge of data resource related processing. Symbian is a multitasking operation system, and there may be several input method instances running simultaneously. Different input method instances can share one thesaurus or corpus without adding more overhead, with which Symbian system achieves data share among input method instances by client server architecture. The relations among user, FEP and application are shown in figure 6.

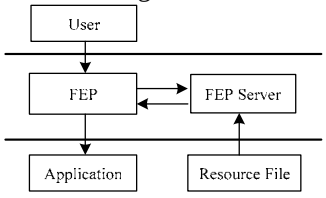

Figure 6. Relations among user, FEP, FEP server and application in Symbian system

We implement FEP API in Context to apply the input method model in Symbian system. The following is part of key code of Context in Symbian system.

```
class Context:public CSybFep{
    TKeyResponse OfferKeyEventL (const
    TKeyEvent& aKeyEvent, TEventCode
    aEventCode);
    int Request(int iEvent,short wParam,long lParam);
    :
}
```

Here CSybFep is derived from base class CcoeFep, of which method ConstructL will generate a CSybFepControl object that has the highest priority to handle input event so that it can deal with input event ahead of application[25]. Context implements API in Symbian system by inheriting from class CSybFep. Method OfferKeyEventL and method Request are designed to receive key pressing message and manipulate state machine. OfferKeyEventL overrides method OfferKeyEventL of CSybFep, gets key pressing message with CSybFepControl, and calls method OfferKeyEvent in Context to transform data. Then OfferKeyEvent transmits transformed data to method Request, which delivers its current event to concrete sub state model for further processing.

VI. CONCLUSION

Before we develop a thirty party input method system for Windows Mobile system and Symbian system, we analyzed different versions of an input method in both systems, finding all the internal logics are basically the same except interfaces to the two systems are different, which has the obvious feature to fit the event-condition-state pattern.

Therefore we use idea of state pattern in design and other phases in development. We rewrote Context part related with system interfaces of different systems. To avoid redundant developing work in both systems, we encapsulated the part related with internal logic of input method by concrete subclasses. The approach proved to be of benefit for providing convenience for later synchronous updating and maintenance of two different versions of input method.

The event-condition-state design pattern proposed in the paper is not only fit for input method development, but also can be used in other software design and development.

ACKNOWLEDGMENT

Funding: This work was supported by School of Computer Science and Technology, Soochow University and Provincial Key Laboratory for Computer Information Processing Technology, Soochow University.

Corresponding authors: Hongjun Diao and Wei Huang.

REFERENCES

[1] Fei Zhu, Xiaoxu Zhu, Qiang Lv, Yiyong Shi, Yang Yang and Qiaoming Zhu, "ZFlow: Workflow for Cooperative

Editing System", Journal of Software, Vol.4, NO. 4, p. 339-347, June 2009.
[2] Fei Zhu, Qiang Lv, "ACEAC: A Novel Access Control Model for Cooperative Editing with Workflow", 2008 International Symposium on Electronic Commerce and Security, pp.1010-1014. IEEE Press, New York, 2008.
[3] Microsoft Developer Network Simplified Chinese MSPY, http://msdn.microsoft.com/en-us/library.
[4] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, "Design Patterns:Elements of Reusable Object-Oriented software," Addison Wesley Pearson, June, 2000.
[5] http://en.wikipedia.org/wiki/.
[6] Elisabeth Freeman, Eric Freeman, Bert Bates, Kathy Sierra, "Head First Design Patterns," O'Reilly, November, 2005.
[7] Frank Buschmann, Kevlin Henney, Douglas C. Schmidt, Pattern Oriented Software Architecture: On Patterns and Pattern Languages, Wiley, June 11, 2007.
[8] Hamed Yaghoubi Shahir, Ehsan Kouroshfar, Raman Ramsin, "Using Design Patterns for Refactoring Real-World Models," seaa, pp.436-441, 2009 35th Euromicro Conference on Software Engineering and Advanced Applications, 2009.
[9] Bahman Zamani, Greg Butler, "Describing Pattern Languages for Checking Design Models," apsec, pp.197-204, 2009 16th Asia-Pacific Software Engineering Conference, 2009.
[10] Ian Bayley, Hong Zhu, "On the Composition of Design Patterns," qsic, pp.27-36, 2008 The Eighth International Conference on Quality Software, 2008.
[11] Ian Bayley, Hong Zhu, "Specifying Behavioural Features of Design Patterns in First Order Logic," compsac, pp.203-210, 2008 32nd Annual IEEE International Computer Software and Applications Conference, 2008.
[12] Anna Medve, Ileana Ober, "From Models to Components: Filling the Gap with SDL Macro-patterns," cimca, pp.1252-1257, 2008 International Conferences on Computational Intelligence for Modelling, Control and Automation; Intelligent Agents, Web Technologies and Internet Commerce; and Innovation in Software Engineering, 2008.
[13] Hakjin Lee, Hyunsang Youn, Eunseok Lee, "Automatic Detection of Design Pattern for Reverse Engineering," sera, pp.577-583, 5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA 2007), 2007.
[14] Nija Shi, Ronald A. Olsson, "Reverse Engineering of Design Patterns from Java Source Code," ase, pp.123-134, 21st IEEE International Conference on Automated Software Engineering (ASE'06), 2006.
[15] Peter Forbrig, Ralf Lämmel, "Programming with Patterns," tools, pp.159, Technology of Object-Oriented Languages and Systems (TOOLS 34'00), 2000.
[16] Innocentia Dlamini, Martin Olivier, Sihle Sibiya, "Pattern-Based Approach for Logical Traffic Isolation Forensic Modelling," dexa, pp.145-149, 2009 20th International Workshop on Database and Expert Systems Application, 2009.
[17] Yacoub S M, Ammar H H, "A pattern language of statecharts," PLOP'98, The Fifth Conf on the Pattern Languages of Program, 1998.
[18] Kent Beck, "Implementation Patterns," Addison Wesley Professional, November, 2008.
[19] Hongjun Diao, Fei Zhu, "A novel general event-condition-state pattern framework," ICIEA, pp. 1024-1027, Industrial Electronics and Applications, 2009.
[20] Jeffrey Richter, Christophe Nasarre, "Windows via C/C++," Microsoft Press, September, 2008.
[21] Microsoft Developer Network, http://msdn.microsoft.com/en-us/
[22] Andy Wigley, Daniel Moth and Peter Foot, "Microsoft Mobile Development Handbook," Microsoft Press, March, 2008.
[23] Microsoft Mobile Developer Center, http://msdn.microsoft.com/en-us/windowsmobile/.
[24] Symbian Developer Network, http://developer.symbian.com/.
[25] Ben Morris, "The Symbian OS Architecture Sourcebook: Design and Evolution of a Mobile Phone OS ," Wiley , July, 2008.

Fei Zhu was born in Suzhou, China, in 1978. He got his master's degree in 2006, majoring in computer science and technology. He is now a PhD student of Soochow University, studying on bioinformatics and systems biology. His interests include machine learning, biological text mining, biological and biomedical network analysis and construction.