Jrl Syst Sci & Complexity (2006) 19: 157–181

DESIGN METHODOLOGY OF NETWORKED SOFTWARE EVOLUTION GROWTH BASED ON SOFTWARE PATTERNS*

Keqing HE · Rong PENG · Jing LIU · Fei HE · Peng LIANG · Bing LI

Received: 13 December 2005

Abstract Recently, some new characteristics of complex networks attract the attentions of scientists in different fields, and lead to many kinds of emerging research directions. So far, most of the research work has been limited in discovery of complex network characteristics by structure analysis in large-scale software systems.

This paper presents the theoretical basis, design method, algorithms and experiment results of the research. It firstly emphasizes the significance of design method of evolution growth for network topology of Object Oriented (OO) software systems, and argues that the selection and modulation of network models with various topology characteristics will bring un-ignorable effect on the process of design and implementation of OO software systems. Then we analyze the similar discipline of "negation of negation and compromise" between the evolution of network models with different topology characteristics and the development of software modelling methods. According to the analysis of the growth features of software patterns, we propose an object-oriented software network evolution growth method and its algorithms in succession. In addition, we also propose the parameter systems for OO software system metrics based on complex network theory. Based on these parameter systems, it can analyze the features of various nodes, links and local-world, modulate the network topology and guide the software metrics. All these can be helpful to the detailed design, implementation and performance analysis. Finally, we focus on the application of the evolution algorithms and demonstrate it by a case study.

Comparing the results from our early experiments with methodologies in empirical software engineering, we believe that the proposed software engineering design method is a computational software engineering approach based on complex network theory. We argue that this method should be greatly beneficial for the design, implementation, modulation and metrics of functionality, structure and performance in large-scale OO software complex system.

Key words Complex networks, evolution growth design method, growth characteristics of software patterns, networked software, OO software network, types and modulation of preferential attachment.

Keqing HE · Rong PENG · Jing LIU
State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, *China*.
Email: hekeqing@public.wh.hb.cn; rongpeng@sklse.org.
Fei HE
Fukazawa Laboratory, Graduate School of Computer Science, Waseda University, Tokyo 169-8555, *Japan*.
Peng LIANG · Bing LI
State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, *China*.
*Supported by the National Natural Science Foundation of China under Grant No. 60373086; ISO/IEC SC32 Standardization Project No. 1.32.22.01.03.00; "Tenth Five-Year Plan" National Key Project of Science and Technology under Grant No. 2002BA906A21; Hubei Province Key Project under Grant No. 2004AA103A02; Wuhan City Key Project under Grant No. 20021002043; Open Foundation of SKLSE under Grant No. SKLSE05-19.

1 Introduction

During a long period, researchers have tended to adopt ER (Entity-Relation) random model to study complex networks, such as Internet, WWW, or the relationship networks in human society, but had very little progress. Consequently, we knew little about the structures and characteristics of these complex networks. The works [1] and [2] of Duncan Watts and Steven Strogatz, in 1998, and Albert Barabási and Reka Albertand, in 1999, discovered that: there exist amazing disciplines in those real world complex networks, showing that most of them are not random network, but SW (Small-World) networks, SF (Scale-Free) network, or networks of high clustering type. All these new features attract the attentions of scientists in various fields, and lead to many kinds of emerging research directions.

In Duncan Watts's work [1], he pointed out that the links between the nodes in most of existing complex networks are neither regular nor random, but follow the rule of "friend's friend" relationship, and consequently reflect the SW features: the relatively short average distance of the complex network and high clustering coefficients. After the analysis of huge amount of network data, Barabási and Albert (BA) published their paper [2], and pointed out that most of the existing complex networks are SF networks which demonstrate power-law distribution for the system degree, and then they furthered the research on those characteristic quantities for this type of networks[3,4]. The complex network model proposed by Barabási and Albert abstracted two basic characteristics of SF feature in complex networks (such as Internet, WWW, and scientist cooperation networks): exponential growth and preferential linking of nodes[5]. Although BA model can be used for the description of many kinds of complex networks, the usage is still limited. For example, BA model can only generate networks with exponential scale 3 for degree distribution, which are different from some existing complex networks with exponential scale of degree distribution ranging from 1 to 3, and the feature of non power-law (such as exponential termination, and small variable saturation). In order to solve these issues and discover the factors that affect the network topology and evolution in microcosmic level, a lot of trials and efforts have been made. So far, the knowledge about network evolution has been developed considerably, especially the types of preferential links, growth, local events, and competition, etc.[6−12] A local world evolution network model was proposed in [13]. But little research has been conducted on the evolution factors from the view of software engineering.

Software systems, especially those in large scale, are kinds of artificial designed complex systems, and the structure of software can be seen as a kind of complex network topology. The combination of software modelling methods and complex networks forms a quite new research direction. For example, the big concern for software engineers is how to design robust, flexible, and scalable software system structures. Here, with our research on the software network models, its evolution designs, the corresponding complex network characteristics, and parameter system of modulation, we come up with a systematic solution. Meanwhile, the traditional targets of the software test and metrics are the finished program and software systems. We think: why couldn't we change the situation from passive to active way? We try to guide the software metrics and test in advance by investigating the evolution design process of software network topology and the complex network characteristics with the parameter system of modulation in this process, and propose the solution in this paper.

Presently, the combination of software engineering and complex networks has attracted the attention of the scientists in related fields, and many valuable research papers and experimental results have been published. For example, with the research on large-scale software systems, we understand that the internal structures of these systems are not random, but follow the complex network characteristics: SW and SF. The large-scale software system can be decomposed into collections of information-components, classes, objects, modules, etc. And the computation

functionality is implemented by the interaction of these units. We can build the complex network models for the large-scale software systems by regarding these components, classes as nodes, and the relationships between them as links.

As we know, there are many important relations between structures and functions. For example, in the biological domain, it is already a plain fact that the structures and arrangements of DNA can determine the property and function of the body. And we argue that there must be corresponding important inherent relations between the software functions and structures similarly. What's more, the combination between software and network technology becomes inseparable with the development of network application, and Networked Software becomes the mainstream in this trend. Networked Software is a complex software system operating on network environment, whose topology structure and behavior can evolve dynamically. Its development and application platform is on complex network, and the representative case is Internet. The compositions of networked systems are much more independent and their providers can be random (same composition of the system can be provided by different providers). They are associated by SSOA (Semantics Service Oriented Architecture) mechanism to realize information resource serving and sharing. This kind of systems has the following characteristics: loose coupling, dynamic evolution (e.g., system topology structure, dynamics behavior and performance, etc.), due to user uncertainty, requirement continuous growth, and multiple targets. It also has the features of non-boundary of requirements, customization and interoperability of services to satisfy user personalization necessity. It is an important issue to investigate the modelling theory, method and modulation strategy of the networked software growth in order to understand the formation and development of networked software, and guide the future practices.

Up till now, several large-scale software systems have been investigated based on complex network models. For example, for the JDK-A (Java Development Kit) class diagram[14], there are 1,376 classes (nodes) in this complex network with degree distribution of 2.5, average distance of 6.39, and clustering coefficient of 0.06. Another example is Pro Rally[15], an entertainment video software. It includes over 1,993 classes, which belong to several modules of image processing, system simulation, sound effect, GUI and expense counting management. Both of the two software systems demonstrate the SW and SF features[14,16−19]. So we can say that applying the research achievements of complex network theory in the fields of software engineering, and developing corresponding methodologies for quarantining the quality of software have become an emergency theme[20,21].

The combination research of software engineering and complex networks is still in its early status, and much limitation still exists. Most research work is limited in the following aspects: network topology of software systems, the demonstration of the complex network characteristics in OO software system structures[22], the software system metrics and evaluation by simply applying complex network characteristics, or the case study of software architectures through the network motifs in computational graphs[23]. So far, in the field of software engineering, no research work on the combination of complex network characteristics and evolution design has been done.

In our paper, we regard software patterns as structure constraint on the evolution of software network that has un-negligible effects on the network formation. Given the initial network model according to the requirement model (e.g., use case model in UML), the OO software network evolves with software pattern constraints.

Base on this recognition, we analyze the growth feature of software pattern network, and propose both the network evolution growth method and its evolution algorithms based on the growth capability of software patterns. By the computation of network topology, we can perform the evolution design from OOA (OO Analysis) to OOD (OO Design), and provide the characteristics of OO system network topology and its parameter systems for the statistical

analysis and computation of topology characteristics in the evolution process. We also propose the strategy of selection and modulation, policy of metrics and test for the stability design of system architecture. We argue that this method is an explorative method in the field of computational software engineering, compared with experience methods.

In this paper, we present the similarity between development of software modelling methods and the evolution of network models with different topology characteristics in Section 2 and analyze the growth features of software pattern topology in Section 3. In Section 4, we offer the software-pattern based modulation on the degree of preferential attachment of OO software network topology. In Section 5, we firstly propose the network evolution growth method and its evolution algorithm based on both growth capability of software patterns and preferential attachment mechanism in Section 4; and then, the characteristics of OO system network topology and its parameter system are provided. Aiming at adjusting the capability (coupling and robust (robust, yet fragile)) of the designed system and guiding the detailed design and implementation, we propose corresponding strategies and methods based on the above parameter system. In Section 6, we focus on a specific application case of the algorithm, the experiment we conduct and the results obtained as well as their implications. Finally, we draw the conclusion from the ideas, the methods to the implementations of the whole work.

2 The Evolution of Network Models and the Development of Software Modelling Methodology

We argue that there is certain similarity between the development of software modelling method and the evolution of network models with different topology characteristics essentially.

First, we illustrate the view of our argument and analysis: the network models (random networks, SW networks, SF networks, and hierarchical clustering networks, and regular networks) and its corresponding degree distribution (Poisson distribution, exponential distribution, power-law distribution, and delta distribution). As an extreme case (featured as entropy increment), the links between nodes in a random network are random, and consequently the degree distribution is homogeneous and democratic, and the Poisson distribution of degree has the expected scale, so it is not an SF network. As the other extreme case (featured as entropy reduction), the links between nodes in a regular network are regular, centralized, and presented as delta distribution. When long distance links are added to a regular network in a "friend's friend" manner, the network will present the SW feature: being relatively short in average distance. Then the degree distribution of those SF networks between the above two extreme cases follows the 80/20 way, and has no expected scale. Actually, the exponential scale of complex networks in our real world is quite different (basically, ranging from 1 to 3), and the feature of non power-law is quite normal for them. When these network models are sorted by their topology characteristics and illustrated in Figure 1, we can find out the A-B-C roadmap presents the evolution discipline of "negation of negation and compromise" in the history of network knowledge. Let us explain the evolution discipline in details. In the initial stage, all the network topologies are classified in two types, the random network and regular network. But it is difficult to apply the random network model due to its uncertainty in structure and performance, and then the construction of regular network (negation of random network) is promoted in order to get better structure and performance. With the development of technology and practices, the regular network is still limited in depicting the characteristics of real-world networks (negation of regular network), and then the small-world and scale-free features in complex networks are found. These features can reflect the characteristics of real-world network topology, and these network topologies are neither random nor regular totally, so it is regarded as compromise. A

local world evolution network model was proposed in [13], this model can transform freely between exponential distribution and power-law distribution, and the complex networks based on this model show strong robustness against intentional attacks. Due to the needs in real world networks, it is necessary to pay more attentions to the network characteristics of this evolution type and build up the corresponding software modelling method.

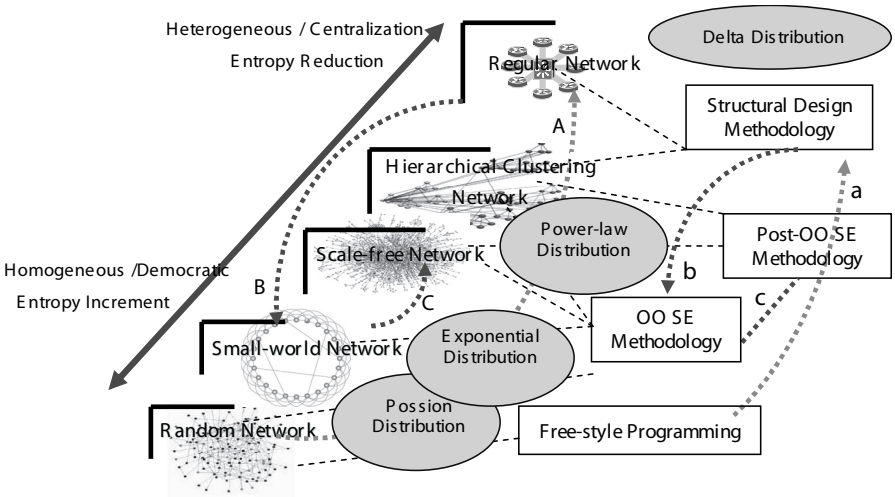


Figure 1 Network model evolution and software modelling method development

In the other aspect, the increase of software complexity also promotes the development of software engineering methodologies, from free style programming, structural programming, OO programming and methodologies, to component-technology based Post-OO methodologies. As an extreme case, free style programming does not limit the usage of "GOTO" in programs; consequently the network models of free style programs are randomly structured. As the other extreme case, the design idea of structural programming is based on hierarchy and stepwise refinement, and its modularization limits the usage of "GOTO" and enhances the cohesion of a module. We regard the statements and modules in a structural program as the nodes in network models, and regard the invocations, sequential links, and selective links as directed edges. While inside modules, most of the structural program topologies demonstrate the high clustering feature, its hierarchical rules affect the presentation of software network topology characteristics naturally. Between the two extremes, there are OOSE and Post-OOSE; for OO methods and UML, classes and objects hide their contents (attributes and operations) from outside, and form perfect nodes for network topology; the relationships (dependence, inheritance, composition, aggregation and message passing) between classes are regarded as directed links; then a directed network topology with OO contents is formed, whose natural features make it easy for wide understanding. Meanwhile, the SF network characteristic of OO software topology has been proved by lots of experimental results[14,16−19]. Corresponding to the A-B-C roadmap, a-b-c roadmap, as shown in Figure 1, also presents the evolution discipline of "negation of negation and compromise" in the history of software development methodologies. Therefore, our research on the types of preferential attachment links, on the methods of evolution growth which support OO software network topology from the view of software engineering, and on the discovery of the complex network characteristics such as SF, SW, high clustering, and its boundary conditions thereof in the evolution process of network models of OO software systems, are of great significance.

Actually, software development is a process full of creation, and endless iteration and evolution. The selection and modulation of network models with different topologies will bring un-ignorable influence on the OO software system designs, implementations and metrics.

3 Analysis of Software-Pattern Based Growth Characteristics

The meta-model of software pattern is the model describing the software patterns[24,25]. In essence, each software pattern is composed of roles of classes and its association relationship between those roles is as shown in Figure 2 (a). The illustration of Composite pattern is presented in Figure 2 (b), the ellipse dash-line represents the dependency association relationship (e.g., the relationship in Composite pattern), and arrow dash-line and its name refer to the element class in the pattern and the roles thereof (e.g., the Component, Leaf and Composite in Composite pattern). The essential characteristics are shown in Figure 2. And in Table 1, we present the icons and their descriptions used in the notation of network topology of software patterns.

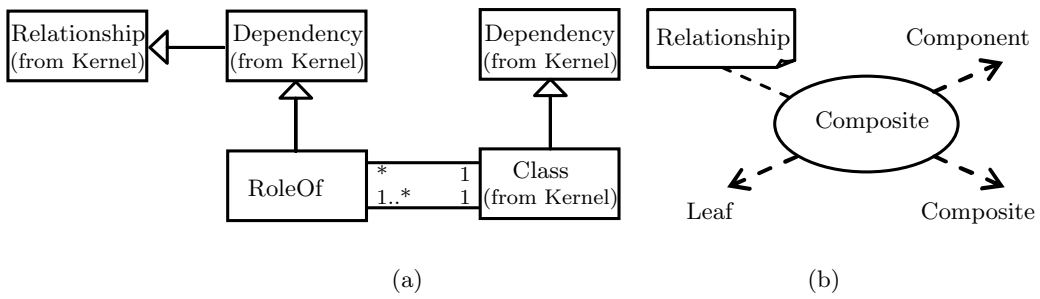


Figure 2 Meta-model and notation of software pattern

Table 1 OO topology icons for software patterns

Icon	Description
	Object note (if necessary, its role-name, which is associated with other nodes, shall be attached to it)
	Relationship: Dependence association
	Relationship: Directed dependence association
	Aggregation and Composition; While the direction of the relationship is from object A to object B; A covers the behaviors or functions of B; This whole-part relationship between A and B is aggregation; The co-existing relationship of all the aggregations between A and B is composition.
	Implementation Inheritance; While the direction of the relationship is from object A to object B; B presents a requirement and the existence of its extension parts; A presents the growth of the requirement from the extension parts.

Software patterns are the reusable knowledge in software development, and each of them is composed of two parts, the Frozen spots and Hot spots. Frozen spots provide the coordinative dependency relationship between roles of pattern classes, and can not be modified; Hot spots provide the rules for extension and modulation in pattern applying and can be regarded as the "Local World" for growth of pattern network topology, e.g., in Figure 3, the concrete objects A, B in the context group and contents group which are the parts possible to grow in pattern topology. Furthermore, there are normally several links with high betweenness[26] between groups A and B. To cut off any links between them, two sub-local-worlds will be formed.

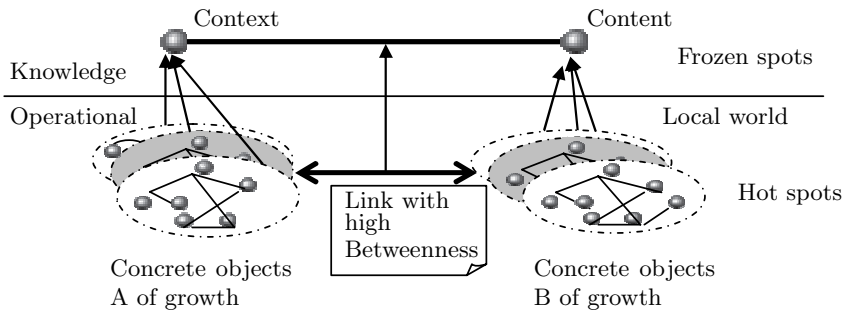


Figure 3 OO topology icons for software patterns

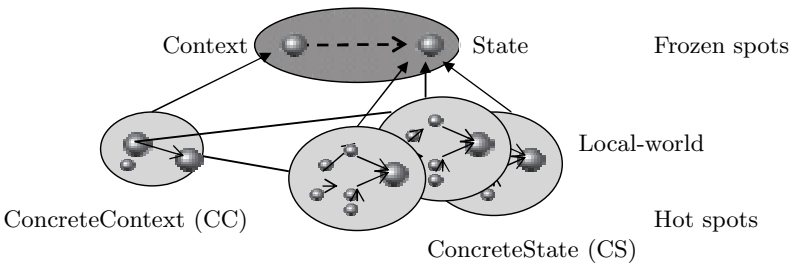


Figure 4 Growth type of state patterns

Based on the growth characteristics of software patterns, we can find the following growth characteristics of preferential attachment links in the network topology of software pattern:

1) Inheritance (copy) growth: Topology nodes (e.g., Context and State in Figure 4) in growth will inherit the attributes and operations of corresponding nodes (classes) in Frozen spots with preferential attachment links, and the growth of topology links shall also be constrained by the corresponding links in Frozen spots. In other words, the local worlds (e.g., the ConcreteContext (CC) and ConcreteState (CS) in Hot spots, Figure 4 grouped by these nodes and links in growth will grow-up under the constraints of Frozen spots. In the full length, inheritance growth is also called the growth of inheritance preference attachment links, see Figure 3;

2) Aggregation growth: The objects in the growing Hot spots are constrained by the aggregation relationships between whole- and part- objects in Frozen spots. In the full length, aggregation growth is also called the growth of aggregation preference attachment links;

3) Composition growth: The objects in the growing Hot spots are constrained by the composition relationships between whole- and part- objects in Frozen spots. In the full length, composition growth is also called the growth of composition preference attachment links.

Figure 3 illustrates that the Hot spots are constrained by Frozen spots (composed of Context, Content and the dependence association relationships between them) in software pattern, and the exact form of growth is released by the local worlds of Hot spots which are the inheritance of Frozen spots.

After the analysis of patterns and design adaptability[27], we define a software pattern as a type of preferential attachment growth, and provide the typical software pattern topology and its growth characteristics in the Appendix.

4 Preferential Attachment Modulation on the SF Network Evolution Growth of OO Software Systems

As for a general network evolution, what shall be put into consideration normally includes five major factors[26]: adding node, adding edge, reconnecting, removing node, and removing edge (considering that the probability of reconnecting event is much larger than the product of the probabilities of removing and adding events, we see reconnecting as important as other 4 factors).

1) Adding node is the most normal behavior that will be completed by the sequence of adding a new node to Graph $G(t-1)$ at time t, and also the responding edges derived from the new node;

2) Adding edge means adding some new edges to connect certain existing nodes in $G(t-1)$ at time t;

3) Reconnecting is the combination of first removing and then adding edges, but only when the removed and added edges are limited to the same node;

4) Removing node is the reverse of adding node;

5) Removing edge is the reverse of adding edge.

To research the design methodology of evolution growth of OO software network topology, we choose software pattern as one of the evolution growth type for the SF network of OO software systems. And according to the result of our experiments, we consider preferential attachment linking is a good mechanism to the formation of SF networks.

$$\mathrm{prod}(k_i) = \frac{k_i}{\sum\limits_j k_j}. \tag{1}$$

BA networks hold two characteristics: growth ability and preferential attachment based on degree with the formula (1). Its degree distribution is $P(k) \sim x^{-\gamma}$ ($\gamma = \gamma_{BA} = 3$), and its scale exponent γ is exactly 3. However, as to real world SF networks, normally, $\gamma \neq 3$, and $1 < \gamma < 3$.

In the research of modulation problems on preferential attachment, S. N. Dorogovtsev and J. F. F. Mendes pointed out that actually, only the linear preferential attachment linking would yield SF networks[28]. Linear attachment linking is, in time t, the probability of a new node attracting a linking to node v with degree $k(v,t)$ is in direct ratio with linear combination. And they also gave a modulation model of the linear combination of preferential attachment linking for the formation of SF networks:

$$f(k,v,t) = G(v,t)k + A(v,t), \tag{2}$$

in which, $A(v,t)$ is called the additional attractiveness, $G(v,t)$ is called the fitness. And in (2):

1) If, $A = 0$, $G(v,t) = 1$, then (2) has the same meaning as BA model (1);

2) If, for certain node v, setup $G >> 1$, then (2) can greatly strengthen the attractive fitness of node v;

3) If, $A > 0$, we can modify and make those nodes with low degree have certain preferential attractive power;

4) In the case of OO software, whose topology is directed network, we shall further the consideration on In and Out links of a node.

In the models of Dorogovtsev et. al [29], they discussed the growth model of the case: when a new node is added, the newly added m links are not derived from the new node. And in order to keep the preferential attachment unchanged, they offered the concept of In degree based attractive power, where the attractive power of node v is $A_v = A + k_v^{\mathrm{in}}$, which presents the degree distribution of power-law, and the power exponent is also adjustable: $\gamma = 2 + \frac{A}{m}$.

In the process of evolution growth of software-pattern based OO software SF networks, we can use the linear combination model of preferential attachment linking (2) to setup node's $G(v,t)$ and $A(v,t)$, and adjust the preference growth in the evolution process of an OO software's SF network.

In above Figure 4, we offer the growth type of state pattern for the preferential attachment in the evolution growth of OO software SF network.

1) Under the constraint of Frozen spots, the growth of a node class ConcreteContext (CC) will require growth of lots of node ConcreteState (CS), so the preferential attachment of node State shall be greater than that of node Context, namely, the $G(v,t)$ and $A(v,t)$ of node State should be greater than those of node Context;

2) In the state machine of local world ConcreteState (CS), with the existence of core state node for state transferring, we shall configure and make sure the core state node's $G(v,t)$ and $A(v,t)$ are greater than those of other nodes;

3) While setting A as a constant (> 0), we can ensure that the growth attachment of software pattern local worlds will hold certain degree of attractiveness;

4) Within the topologies of OO software systems, we can find lots of local worlds different in scale and quality, with the different requirements of users, and we can setup different constants A to adjust the attractiveness for different local worlds.

Based on this preferential attachment scheme, we develop the following evolutional method and algorithms of OO software network.

5 Design Method of OO Software Network Evolution

In most of OO software systems using software patterns, their network topologies illustrate the heterogeneous feature obviously due to growth capability of software patterns when they grow to certain scale. Consequently, the usage of multiple software patterns will determine the basic features of system network topology evolution.

However, the network topology of a large-scale OO software system with high efficiency in reality is provided with different kinds of network models for different parts, so normally it has the properties of multiple network models. However, we argue that it is quite important to pay more attention to the complex network model which determines the integral features of the software network topology, such as "scale-free", "small-world", and "high clustering". It is also critical to do the research of design method of evolution growth from OOA to OOD.

5.1 The Idea of Evolution from OOA to OOD

It is a difficult problem for the transformation from system requirement specification to system design specification in software engineering because of the technical obstacle between them. The objective of this method is to support the preliminary design of OO software system and guide the detailed design and implementation in UML through the evolution of network topology from OOA to OOD after the determination of OOA in requirement specification phase.

For this, we give the assumptions in evolution design method.

Initial network model: the OOA use case models for user requirement.

Due to the self-similarity feature among the software pattern network topology (see Appendix), network topology of OOA use case models, and OOD software network topology, we assume that OOA use case models can be regarded as the initial model of OOD network evolution.

Evolution destination model: a customizable network topology model of OO software preliminary design, which satisfies user requirement. Moreover, this model can provide system

network topology features with its parameter systems and the strategies for modulation of system performance.

Network topology growth types based on software patterns: the preconditions for the formation of OOD "scale-free" network are growth and preferential attachment in network topology. The preferential attachment rules of software network evolution from OOA to OOD are based on software patterns. The growth is achieved by the growth of nodes and links in the hot spots of software patterns. Then it is possible to obtain a "scale-free" network topology. The representative software pattern network topologies and their growth types definitions can be found in Appendix.

5.2 Algorithms of OO Software Network Evolution Growth Based on Software Patterns

The algorithms is composed of PaNEGA and PaGA.

PaNEGA Algorithm

The algorithm PaNEGA uses software patterns as seeds of growth, OOA requirement analysis models (use case models) as input, network topology of OOD software system as output. It works as follows:

1) Initial topology model generation: generate use case network topology graph (a directed graph) $U = \langle V, L\rangle$ of the system according to its OOA, where $V = \{v_i\}$ is the set of nodes representing the use cases in OOA, $L = \{(v_i, v_j)\}$ is the set of links representing use case v_i is associated with use case v_j in OOA. Assume $|V| = m_0$, $|L| = l_0$ and for every node v_i, v_i.scale represents its growth scale;

2) Network evolution.
   a) Local-world and software patterns identification: According to the intrinsical characteristics, such as context and contents, of software patterns (see Appendix), we identify the "Local World" in U that can use various patterns to expand the design. Assume the use cases in Local World are in set U' and the usable patterns are in list PList;
   b) Preferential attachment based on software patterns: When choosing the pattern in PList as the growth seed of U', we assume that the probability $\prod_p$ that U' will use pattern A as seed depends on the use frequency f_A of pattern A, such that $\prod_p(f_A) = \frac{f_A}{\sum_{k\in \mathrm{PList}} f_k}$, $A \in$ PList;
   c) Growth scale acquirement: According to the use case graph U, we acquire the growth scale v_i.scale for each use case node v_i in U';
   d) Network Growth: Call the pattern growth algorithm PaGA(A,U') to get the new topology U'' generated according to software pattern A;
   e) Local modification (Optional): User may modify the network topology U'' according to his requirement;
   f) Reconnection: If the node v_i in U and the node v_j in U' have link l_{ij} or l_{ji}, and the node v_j has evolved to the node set $S = v_{j1}, v_{j2}, \cdots, v_{js}$, $s \neq 1$, in U'', then re-link it:
      i) Delete the link l_{ij} or l_{ji};
      ii) Preferential attachment based on degree of nodes: When choosing the node to which v_i connects, we assume that the probability $\prod_d$ that v_i will be connected to node j ($j \in S$) depends on the degree k_j of node j, such that $\prod_d(k_j) = \frac{G_j k_j + A_j}{\sum_{t\in S}(G_t k_t + A_t)}$, $j \in S$, where $G_j(G_t)$ and $A_j(A_t)$ are the fitness and additional attractiveness of node $j(t)$ respectively, and the default values of G, A are 1 and 0 respectively;
   g) Frequency increase: $f_A + +$.

3) Parameters calculation: According to related formula in next section, we calculate the network topology graph features of complex network.

a) Parameter features in Local World U'': Calculate the degree, in-degree, out-degree of each node in Local World to get corresponding degree distributions; Calculate average connection coefficient; Calculate average shortest distance; Calculate betweenness for each node and link; Calculate the scale of U'';

b) Parameter features in U (network topology): Calculate the degree distribution, average connection coefficient, average shortest distance, scale distribution of local world and system scale;

4) Global modification (Optional): User may modify the network topology U according to statistical data and actual requirement (such as changing name of nodes and links, etc.);

5) Repeat 2), 3), 4), until there is no suitable pattern for use cases in U, or users terminate to use pattern for expanding the design;

6) Use cases extension: According to users' requirement, we generate new classes gradually for those use cases and related connections without suitable pattern;

7) Parameters calculation: the same as 3)–b);

8) Optimization (Optional): Propose the modulation strategy according to the data produced by 7);

a) If user agree with the modulation strategy, go to 2);

b) Otherwise, algorithm terminates.

PaGA(A,U') Algorithm

The growth algorithm based on software pattern PaGA(A,U') uses software pattern A as the seed of growth, and local world U' as growth base, and outputs a new local world U'' which consists of corresponding classes. PaGA(A,U') works under the following assumptions:

1) Use case network topology graph (a directed graph) $U' = \langle V, L\rangle$, where $|V| = m_1$, $|L| = l_1$ and for every node v_i, v_i.scale represents its growth scale;

2) The graph G_A of software pattern A: $G_A = \langle V_A, L_A\rangle$, where $|V_A| = m_2$, $|L_A| = l_2$;

3) Matching paradigm (v_i, v_j): The node v_i in U' will grow according to the growth paradigm represented by node v_j in A.

PaGA(A,U') algorithm works as follows:

1) Nodes replacement: According to the matching paradigm (v_i, v_j), permute node v_i to v_j.

a) If v_j is a stable frozen node, replace v_i with v_j directly;

b) If v_j is a increasable frozen node, replace v_i with v_j and generate s new nodes $v_{j1}, v_{j2}, \cdots, v_{j_s}$ according to v_i.scale (assuming v_i.scale $= s$);

c) Otherwise, namely v_j is a node in hot spots, replace v_i with v_j directly;

2) Nodes growth: According to the pattern growth specification in Appendix, generate nodes, which have not been matched in matching paradigm;

3) Link growth: Generate certainty links between nodes according to Appendix;

4) Link replacement: Assume link $l_i = v_{i1}, v_{i2}$ in L, and the corresponding nodes in pattern A are $v_{j1}(v_{j1_1}, v_{j1_s})$, $v_{j2}(v_{j2_1}, v_{j2_t})$, generate new links.

a) If there are preferential attachment connections between nodes $v_{j1}(v_{j1_1}, v_{j1_s})$, $v_{j2}(v_{j2_1}, v_{j2_t})$ in pattern A, then for every node v_{j1_p} in v_{j1}, choosing the node v_{j2_q} in v_{j2} to which v_{j1_p} connects, we assume that the probability $\prod_d$ that v_{j1_p} will be connected to node v_{j2_q} depends on the degree $k_{v_{j2_q}}$ of node v_{j2_q}, such that $\prod_d(k_{v_{j2_q}}) = \frac{G_{v_{j2_q}} k_{v_{j2_q}} + A_{v_{j2_q}}}{\sum_{m\in[1,2,\cdots,t]}(G_{v_{j2_m}} k_{v_{j2_m}} + A_{v_{j2_m}})}$, where G, A are respectively the fitness and additional attractiveness of the corresponding node and the default values of G, A are 1 and 0 respectively;

b) If there are stable connections between nodes $v_{j1}(v_{j1_1}, v_{j1_s})$, $v_{j2}(v_{j2_1}, v_{j2_t})$ in pattern A, then ignore it;

c) If there is no connection between nodes $v_{j1}(v_{j1_1}, v_{j1_s})$, $v_{j2}(v_{j2_1}, v_{j2_t})$ in pattern A, then generate full connections;

5) Link amendment: Generate probability connections which is not matched in 4)–a) according to Appendix;

6) Output graph U'': The graph U'' consists of all nods and links generated in the above four steps;

7) Algorithm terminates.

5.3 Characteristic of OOD Software Network Topology and Based Its Modulation Design

OOD software network topology is a kind of directed network, which is different from the undirected network, such as Internet. According to the features of OOD software topology, we provide its parameter systems referring to the definitions of complex network parameter systems, and the parameter systems provide the guideline for the modulation design of OOD software topology. The objective of modulation design is to improve the system robustness through the location and protection of core structures, which provide the primary functions, to improve capability for software reuse, refactoring, and reformation, and to form the scalable structure to facilitate the quick recovery to software system. The OOD software system network model by evolution growth based on software patterns is a kind of multiple local-world coupling model essentially, and it controls the topology feature and performance of software system.

The OOD software system network model by evolution growth based on software patterns is a kind of multiple local-world coupling model essentially, and it controls the topology feature and performance of software system.

The system structure stability becomes the main aspect in system modulation design besides the function for user requirement in system modelling. Compared with current empirical software engineering methodology from use case model to UML models, the method proposed in this paper provides a method, which can select topology structure, and modulate system performance, in the process of network topology evolution.

5.3.1 Characteristic of directed network in OOD software topology

Features of directed network in OOD software topology can be measured in several different guidelines. Since some features are measured in directed graph, and others in undirected graph, we propose the parameter guideline Definitions 1–13 to guide the features analysis based on the following assumptions:

1) According to the directed association types defined in Table 1, we can define directed links δ^{uv} according to the association type between u and v.

a) Dependency association $\delta_1 : \delta_1^{uv} = 1 \Rightarrow \delta^{uv} = \delta^{vu} = 1$ (the undirected association between u and v will be viewed as bidirectional links);

b) Directed dependency association $\delta_2 : \delta_2^{uv} = 1 \Rightarrow \delta^{uv} = 1,\ \delta^{vu} = 0$;

c) Clustering or compositional association $\delta_3 : \delta_3^{uv} = 1 \Rightarrow \delta^{uv} = 1,\ \delta^{vu} = 0$;

d) Implementation inheritance association $\delta_4 : \delta_4^{uv} = 1 \Rightarrow \delta^{uv} = 0,\ \delta^{vu} = 1$ (since the inheritance association means u will have new characteristics based on the copy of v, u has the features of v and can be accessed by v);

2) According to the directed association types defined in Table 1, we can define undirected edges θ^{uv} according to the association type between u and v: If there are one or more kinds of associations δ_1, δ_2, δ_3 and δ_4 between nodes u and v, $\theta^{uv} = \theta^{vu} = 1$; otherwise $\theta^{uv} = \theta^{vu} = 0$;

3) Features defined in Definitions 1–13 use both directed graph dG and undirected graph uG:

a) Directed graph $dG = (V, L)$, where $V = \{v_1, v_2, \cdots, v_n\}$ is set of nodes; $L = \{(u, v)|u, v \in V \wedge \delta^{uv} = 1\}$ is set of links.

b) Undirected graph $uG = (V, E)$, where $V = \{v_1, v_2, \cdots, v_n\}$ is set of nodes; $E = \{(u, v)|u, v \in V \wedge \theta^{uv} = 1\}$ is set of edges.

Under above assumptions, define and analyze the following thirteen feature parameters of directed and undirected network in OOD Software Topology.

Degree and degree distribution

As to directed network, it is important to analyze the In-degree and Out-degree of nodes and the corresponding degree distributions for evaluating the performance of OOD software topology. And as to undirected network, the parameters Degree and Degree distribution are also crucial.

Definition 1 The In-degree of node u is $d_u^{\text{in}} = \sum_{u \in V} \delta^{vu}$, and the In-degree distribution of directed network is $P^{\text{in}}(k) = \frac{|u|d_u^{\text{in}}=k|}{|V|}$.

Definition 2 The Out-degree of node u is $d_u^{\text{out}} = \sum_{u \in V} \delta^{uv}$, and the Out-degree distribution of directed network is $P^{\text{out}}(k) = \frac{|u|d_u^{\text{out}}=k|}{|V|}$.

Definition 3 The Degree of node u is $d_u = \sum_{u \in V} \theta^{uv}$, and the Degree distribution of undirected network is $P(k) = \frac{|u|d_u=k|}{|V|}$.

Clustering coefficient

The clustering coefficient measures the proportion of network clustering, namely the fraction between the mean number of edges (links) between the neighbors of a node and the mean number of possible edges (links) between those neighbors.

Definition 4 In-group clustering coefficient of node u is defined as follows: Let the In-group neighbors of node u be $V_u^{\text{in}} = \{v \in V|\delta^{vu} = 1 \wedge v \neq u\}$, let $n = |V_u^{\text{in}}|$, and let the number of links between the nodes in V_u^{in} be $M = \sum_{x,y \in V_u^{\text{in}}, x \neq y} \delta^{xy}$, then $C_u^{\text{in}} = \frac{M}{P_n^2} = \frac{M}{n(n-1)}$. If $n = 0$ or 1, $C_u^{\text{in}} = 0$. And the average In-group clustering coefficient of directed network is $C^{\text{in}} = \frac{1}{|V|} \sum_{u \in V} C_u^{\text{in}}$.

Definition 5 Out-group clustering coefficient of node u is defined as follows: Let the Out-group neighbors of node u be $V_u^{\text{out}} = \{v \in V|\delta^{uv} = 1 \wedge v \neq u\}$, let $n = |V_u^{\text{out}}|$, and let the number of links between the nodes in V_u^{out} be $M = \sum_{x,y \in V_u^{\text{out}}, x \neq y} \delta^{xy}$, then $C_u^{\text{out}} = \frac{M}{P_n^2} = \frac{M}{n(n-1)}$. If $n = 0$ or 1, $C_u^{\text{out}} = 0$. And the average Out-group clustering coefficient of directed network is $C^{\text{out}} = \frac{1}{|V|} \sum_{u \in V} C_u^{\text{out}}$.

Definition 6 Clustering coefficient of node u is defined as follows: Let the neighbors of node u be $V_u = \{v \in V|\theta^{uv} = 1 \wedge v \neq u\}$, let $n = |V_u|$, and let the number of edges between the nodes in V_u be $M = \frac{1}{2} \sum_{x,y \in V_u, x \neq y} \theta^{xy}$, then $C_u = \frac{M}{C_n^2} = \frac{2M}{n(n-1)}$. If $n = 0$ or 1, $C_u = 0$. And the average clustering coefficient of undirected network is $C = \frac{1}{|V|} \sum_{u \in V} C_u$.

Average path length

The average path length provides a global characterization of network organization and communication. Low average path length always indicates faster communication and higher performance. Therefore, for many systems, it is useful to keep the average path length very low. Typically, it is defined in undirected graph as Definition 7.

Definition 7 Average path length uD in undirected graph uG is defined as follows: Choose a pair of nodes (u, v) and then trace a path from v_i to v_j while traversing the minimum number of edges. Let d_{uv} represent the length of the shortest path between u and v, then $uD = \frac{1}{|V|(|V|-1)} \sum_{u,v \in V, u \neq v} d_{uv}$.

Definition 8 Average path length dD in directed graph dG is defined as follows: Choose a pair of nodes (u, v) and then trace a path from v_i to v_j while traversing the minimum number of links. Let d_{uv} represent the length of the shortest path between u and v. Suppose there are N reachable pairs of nodes (u, v) then $dD = \frac{1}{N}\sum_{u,v\in V, u\neq v} d_{uv}$.

Betweenness

The betweenness also provides a global characterization of software topology. It consists of node betweenness and link betweenness. The node betweenness of node u is defined as the number of shortest pathes, which pass through u. It embodies the power of node u. The link betweenness of link l is defined as the number of shortest paths, which pass through l. It can be used to analyze the clustering relations between nodes.

Definition 9 Let S_{ij} represent the shortest path set of node i and node j. Node betweenness of node u is defined as $B_u = \sum_{i,j} \frac{\sum_{l\in S_{ij}} \delta_l^u}{|S_{ij}|}$.

Definition 10 Let S_{ij} represent the shortest path set of node i and node j. Link betweenness of link e is defined as $B_e = \sum_{i,j} \frac{\sum_{l\in S_{ij}} \delta_l^e}{|S_{ij}|}$.

Scales of local worlds and scale distribution

Definition 11 The scale of local world S_w is defined as the number of nodes in local world.

Definition 12 The scale distribution of local world in software topology P_w is defined as follows: Suppose the directed network in OOD software topology is composed of N local worlds, and V_i $(i = 1, 2, \cdots, N)$ represents each local world, then the scale distribution $P(k) = \frac{|i||V_i|=k\wedge i\in[1,2,\cdots,N]|}{N}$.

5.3.2 Analysis of robustness and coupling in OO software network and its modulation strategies

This section proposes the modulation strategies based on the analysis of robustness and coupling of network topology.

Robustness analysis and its modulation

The term "Robustness of OO software network" here mainly refers to the robustness of network topology. And its modulation is also achieved by modulating the network topology. Since the dynamic characteristic in "Scale-Free" network is controlled by the few nodes with high degree, the few links with high link betweenness and the key local-worlds affected the non-even characteristics of network, identify those key nodes, key links and key local-world is the key point.

The algorithm PaNEGA can not only automate the design process from OOA to OOD, but also provide the local world and global parameter systems, such as degree, clustering coefficient, betweenness for each node and each link in OO software network topology. These parameters can not only be the proof for the discovery of key nodes, key links and key local-world, but also be the evidences for the system Robustness determination.

1) Identification and modulation of key nodes and key links

In Table 2, we present not only the features of various nodes (links) in parameter systems, but also the degree of invalidation effects to other nodes. In order to promote the robustness of the system, we propose several the modulation strategies to change its topology. Other measures related with security, such as access control, encryption, are out of our research scope.

For example, a key node u in local world (Local key node) is usually with high degree (d_u), high node betweenness (B_u) in pattern local-world; and if it invalidates, other nodes in this pattern local-world are affected considerably, such as degree (d_u), average clustering coefficient (C), and average path length (uD) in local-world parameter system. Similarly, a global key node u in network topology is usually with high node betweenness (B_u) in global parameter system;

Table 2 The corresponding features of various nodes and links

	Feature				Invalidation Effects						Modulation Strategy
	Local		Global		Local			Global			
	d_u	$B_{u/e}$	d_u	$B_{u/e}$	d_u	C	uD	d_u	C	uD	
Local key node	H	H	-	-	H	H	H	-	-	-	BP
Global key node	-	-	H	H	-	-	-	H	H	H	BP;BU;Clone
General node	L	L	L	L	L	L	L	L	L	L	None
Local key link	-	H	-	-	L	H	H	L	-	-	BP
Global key link	-	-	-	H	L	-	-	L	H	H	BP;Clone
General link	L	L	L	L	L	L	L	L	L	L	None

Note: H (High), L (Low), - (Indefinite), BP (Behavior Pattern), BU (BackUp).

and its invalidation will cause the huge change in corresponding global parameter system. So, if needed, we should take some additional protection method or modulation strategy to change the situation, such as using behavior pattern to decrease the effect of local key nodes, cloning the key node and its associated links, and creating its backup periodically. For general nodes, since the invalidation effects are little, we should not take any additional measures.

Similarly, a local (global) key link l in network topology is usually with high edge betweenness (B_e) in local-world (global) parameter system, and its invalidation will cause the huge change in average clustering coefficient (C) and average path length (uD), but the change to degree (d_u) is little since it only effects its source and destination. The corresponding modification strategies include using behavior patterns and cloning its source and destination node with it. For general links, we take no protection measures for the same reason as general nodes;

2) Identification and modulation of key local-world

It turns to identify the key local-world after finding out the global key nodes and links. The feature of key local-world is that the proportion that global key nodes and global key links belong to it is relatively high. If the whole local-world is invalid, other nodes in network topology are affected considerably. The system will be divided into several independent connected groups in all probability, and consequently the system can not complete the basic function. So, the modulation strategy is also needed.

Additionally, besides the features that key nodes, key links and key local-world can be modulated, the scale of the local-world can also be modulated. By modulation, we can make the non-even characteristic that network topology present some special features, such as scale-free and high clustering, etc. By applying modification strategies to control the stability of the whole network by protecting the few big nodes and key links directly, we can promote the system capability against various attacks, such as intentional attacks or random faults, with scalable topology. We argue that it is quite important for the design of a robust OO software system.

Coupling Analysis and its Modulation

Coupling represents the dependency degree between design units (use cases, classes, and objects). The high coupling between design units will hinder their reuse. Consequently, the high cohesion and low coupling is very important rule for software design. When we regard the OOD system software network topology as directional network, the coupling refers to the coupling degree between nodes in the topology. The software system can have better performance when the coupling of its topology is reasonable. It is quite important to research the parameter system and the relativity between high cohesion and low coupling in directed network of OOD software topology, and perform the modulation accordingly.

Table 3 The relationship between the CK model and parameter systems of PaNEGA

Measurement Items	CK	Parameter Systems of PaNEGA
Localization	N	Y (P_w in global parameter system)
Encapsulation	N	Y (S_w; d_u, C_u, uD, B_u, B_e in local-world parameter system)
Responsibility	N	Y (uD, dD in global parameter system)
Weighted Methods per Class	Y	Y (B_u in global parameter system)
Depth of Inheritance Tree	Y	Y (the inheritance association length)
Number Of Children	Y	Y (d_u^{out} produced by inheritance association)
Coupling Between Objects	Y	Y (B_e in global parameter system)
Response For a Class	Y	Y (d_u^{in} in global parameter system)
Lack of Cohesion of Methods	Y	N (but the property of patterns can ensure its cohesion)

According to the parameter systems, we find out that the link betweenness B_e can primarily reflect the coupling degree between its source node and destination node. The higher B_e is, the higher the coupling degree of the two nodes connected by this link will be. We provide the strategies for coupling modulation based on this fact as follows:

1) Use behavioral patterns to decrease the coupling reasonably

In the process of OO network evolution design, we can use the behavioral pattern intentionally to decrease and modulate the coupling degree between nodes besides recognize and use patterns according to the design requirements. For instance, since the Observer pattern designs the loose binding relationship between one "Subject" and several "Observers", we can still add "Middle Observer" nodes besides Subject, Observer to decrease the coupling degree between them: For Command pattern, it creates a subclass for each link marked the role "receiver", which can reduce the coupling degree; For Mediator pattern, it reduce the coupling between nodes in a indirect way;

2) Modulate the network topology to decrease the coupling inside software patterns properly

The relationship between nodes in the Frozen Spots of software pattern always presents the strong coupling. The strong coupling may hinder the reuse and package of software component. Consequently, we need to concentrate on these key links. We should modulate the network topology through various balance strategies, such as increasing branch node and refining function in order to obtain a stable system.

5.3.3 The relationship between OO software network evolution growth method and OO software metrics

Presently, some research results have been achieved for the theory and methods in OO software system metrics. The CK model based on class metrics[15] proposed by R. Chidamber and Chris F. Kenerer is a representative one.

Since the topology feature parameter systems have formed in the evolution process, it can be the proofs for the OO software metrics. The relationship between them is shown in Table 3. Our method provides the local-world scale distribution (P_w) for the measure of system localization, the scale of each local-world (S_w) to measure its encapsulation, and the average path length in topology for responsibility metrics.

It's worth explaining that the metrics LCOM (Lack of Cohesion of Methods) is the number of the methods which access one or several same attributes. Although there is no direct reflection of LCOM in our metrics, the constraint growth based on pattern Frozen Spots can guarantee scientificity and rationality of the cohesion of methods.

6 Case Study

This section will focus on a specific application case of the method and algorithms, the experiment we conduct and the results obtained as well as their implications. It is organized as follows: Firstly, we give a brief description of the requirement of the studied subsystem; And then we elaborate on the procedures of the evolution design and analyze the obtained results in succession.

6.1 Requirement Description of "Account Sheet Processes" Subsystem of e-Commerce

In a financial application, the system requirement specifies that a subsystem, "Account Sheet Processes", should be capable of extending its functionality with the increase in business variety. At the same time, every function should be able to be constructed and configured dynamically according to the requirement of business. We should be able to add dynamically to the types and the methods of business processing when the target subsystem is in operation, so as to dynamically add to the function of the "Account Sheet Processes" component.

According to the user specification, the "Account Sheet Processes" subsystem has 30 functional types in need of extension, including use cases "Sales Account Sheet Processes", "Audit Account Sheet Processes" and etc. Figure 5 is the subsystem use case model[1].

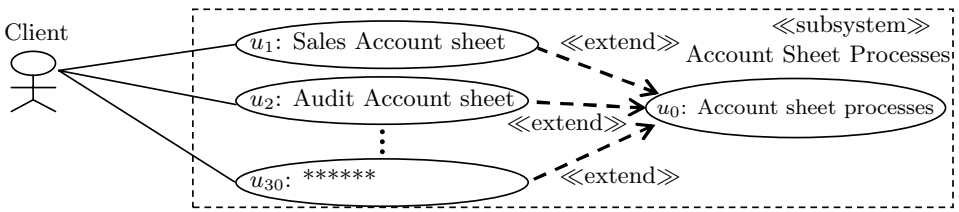


Figure 5 Use case graph of "Account Sheet Processes"

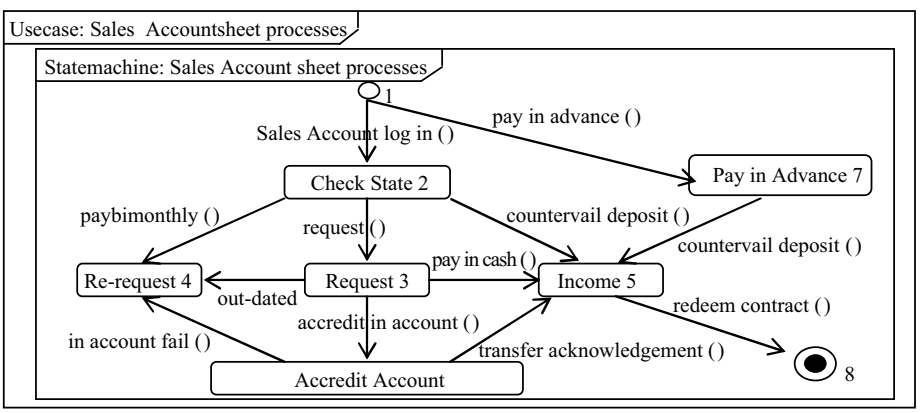


Figure 6 Use case graph of "Sales Account Sheet Processes"

As an example, we give the state machine of u_1: "Sales Account Sheet Processes" in Figure 6. It is obvious that the scale of the state machine is 8, denoted by $\mathrm{ST}_1.\mathrm{scale} = 8$.

By the same way, we can obtain the state machines and the scales of other use cases in "Account Sheet Processes" following the user requirement. For detailed specification of the subsystem see [30].

6.2 Evolution Design Based on State Pattern

6.2.1 Design goal

Design the "Account Sheet Processes" subsystem according to the above specification, it should be able to dynamically construct various function processing types, define and configure their specific functionality.

According to the specification, there are two hot spots in the subsystem: Firstly, the type of "Account Sheet Processes" (u_1Context) can be added as needed to handle various kinds of accounts; Secondly, state nodes can also be added to the corresponding state machine of different "Account Sheet Processes".

6.2.2 Evolution design process

On the basis of the above analysis and setting, we consult the table in the Appendix and apply algorithms PaNEGA and PaGA.

The general process of the evolution growth design is as follows:

1) Initial topology model generation

The use case graph of "Account Sheet Processes" in Figure 6 can be transformed into the following initial topology model in Figure 7, with a total node number 61: $m_0 = 61$; and a total link number of 60: $e_0 = 60$;

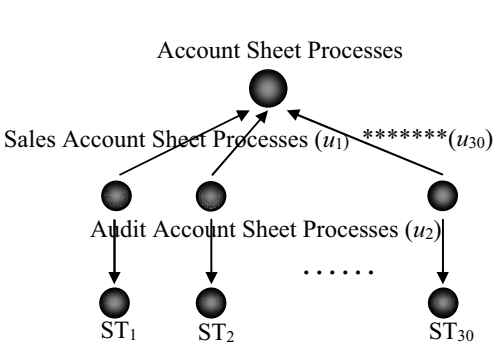


Figure 7 Initial topology of subsystem use case

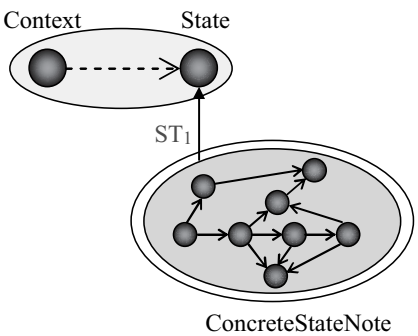


Figure 8 Matching State Paradigm

2) Network evolution

Through the matching procedure, we choose State Pattern to match the use case. Figure 8 illustrates the matching paradigm. And the matching paradigm of the state pattern is $(u_1, \text{Context}), (\text{ST}_1, \text{CSN})$;

3) Parameters calculation

Calculate the parameter features in Local World and system topology according to algorithm PaNEGA;

4) Pattern application

Following the same procedure, State pattern is applied to $u_2, \text{ST}_2, u_3, \text{ST}_3, \cdots, u_{30}, \text{ST}_{30}$, and the following logic network topology is as Figure 9 and its generated network topology is as Figure 10.

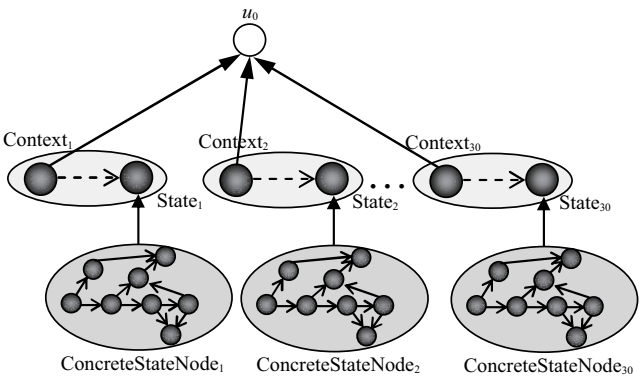


Figure 9 The evolution process of local world

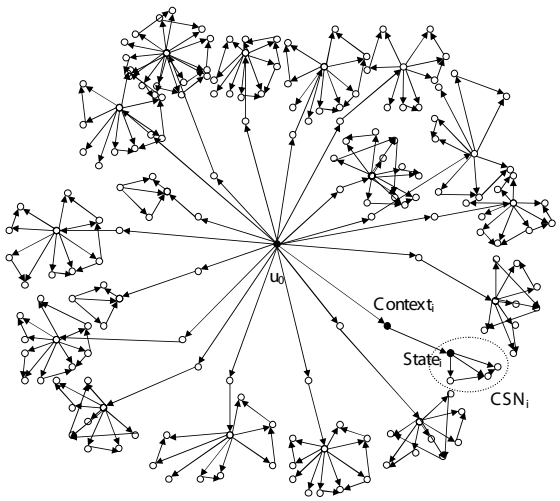


Figure 10 The generated network topology

6.3 Characteristic Analysis

According to the parameters defined in Section 5, we analyze the resultant directed network of "Account Sheet Processes" topology, and find the following results.

6.3.1 The link number

In the process of network evolution, the link number grows linearly with the node number. It means that every time a new node adds to the network, it connects to approximately a constant number of other existing nodes. This is probably because that the evolution process is constrained by the regulation of software pattern, and that the number of links and the number of nodes follow linear relationship as each pattern applies, and that the whole network also reveal linear relationship in the evolution process (see Figure 11).

6.3.2 Average path length

As the evolution process goes on, the average path length of the "Account Sheet Processes" directed network tend to be constant ($uD \approx 5.7$, $dD \approx 3.4$) (see Figure 12). And the aver-

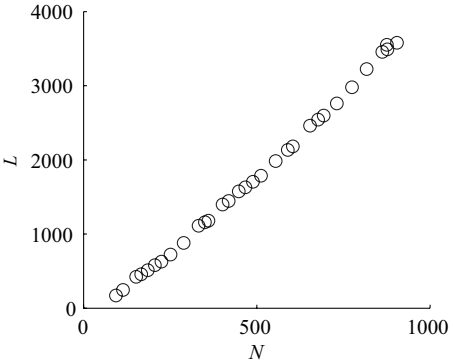


Figure 11 Scatter plot of the number of links (L) vs. the number of nodes (N)

age path length is approximately proportional to the logarithm of the graph size. These are indications of the small average path length feature of SW network[18]. Keeping the average path length low is useful for many systems, since it means in spite of the large nodes quantity, communication between nodes will be much efficient.

The parameter uD has the same order of magnitude as the parameter d from table I in [18], which suggests the resultant network form evolution design tallies with the artificially designed one. As we can see from Figure 12, the resultant undirected version of the network has a larger average path length uD than that of its directed counterpart (dD). And dD is a more exact indication of how far objects have to travel to visit each other (most system calls pass through → Context → State → ConcreteState).

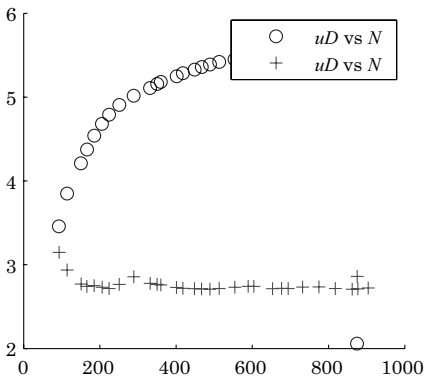


Figure 12 Average path length

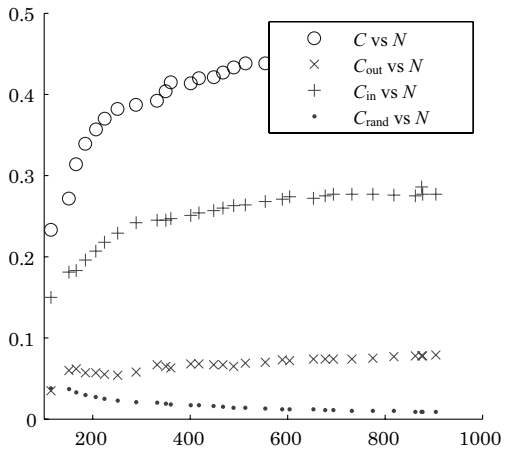


Figure 13 The average clustering coefficient vs. the number of nodes

6.3.3 Average clustering coefficient

As the evolution process goes on, the average clustering coefficient of the "Account Sheet Processes" directed network tend to be constant($C \approx 0.568,\ C_{\text{in}} \approx 0.288,\ C_{\text{out}} \approx 0.012$), and they are much larger than the average clustering coefficient of the random graph $C_{\text{rand}} \approx 0.004$ (see Figure 13). This result indicates the large average clustering coefficient feature of the SW

network. Combining the results of average path length and average clustering coefficient, we can infer that the resultant network displays SW features.

6.3.4 Degree distribution

As the evolution process goes on, the cumulative degree distribution of "Account Sheet Processes" network demonstrates SF feature. As is shown in Figure 14, on the logarithm scale, the cumulative degree distribution of the 30th iteration is close to a decreasing line, which is an indication of power law feature of SF network. And the exponent is approximately $\gamma \approx 2.64$.

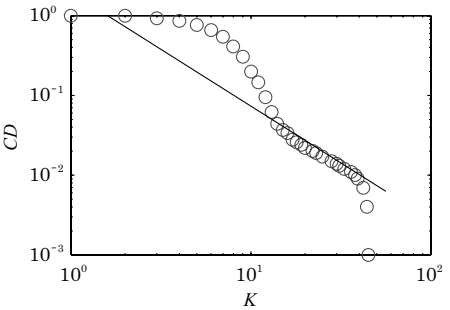


Figure 14 Cumulative degree distribution (CD) vs. degree (K) (30th iteration)

6.3.5 Betweenness

We discover that the betweennesses of frozen nodes and links between them are on average greater than those of hot spot nodes and the links between them (see Figure 15). This result accords with the rules of software patterns.

The betweenness could be used as a measure to identify key nodes or links in the network, in this case, the State node, the Context node and the link between them. As we illustrate in Section 5.3.2, high node betweenness is often an indication of key node, on which additional protection measures should be taken; high link betweenness, on the other side, suggests the high degree of coupling between connected nodes. These results enable us to carry out strategies of the modulation design to enhance overall structural robustness.

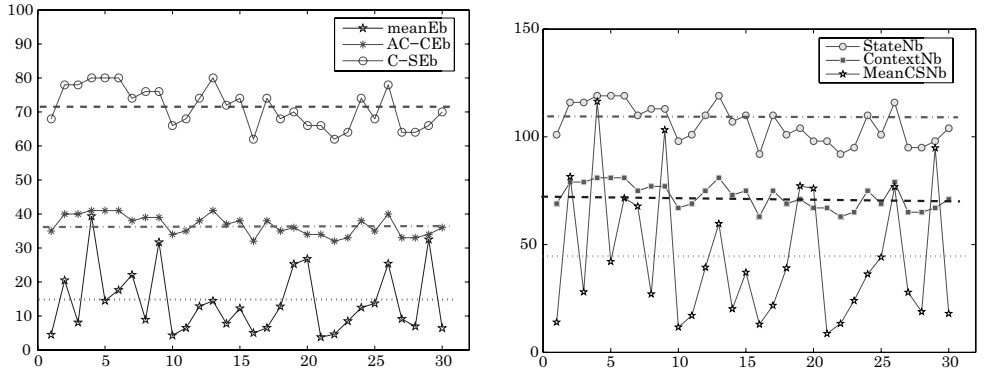
6.4 Result Analysis

Through the statistical results of the resulted directed network of the "Account Sheet Processes" topology in the evolution process, it presents the following facts:

1) Link number grows linearly with the node number;
2) Average path length tends to be constant;
3) Average clustering coefficient tends to be constant;
4) Degree distribution tends to demonstrate power law feature;
5) Betweennesses of frozen nodes and links between them are on average greater than those of hot spot nodes and the links between them.

From these facts, we can infer that the resulted network is of both SW and SF features. From the statistical results we obtain in our design process, we can get overview of the overall network characteristic. We also provide parameters for further modulation to enhance the robustness of the software structure.

Since this example is only a part of the overall OO software system requirement specification and only applying the State pattern, there may be discrepancies between the resultant



(a) (b)

Figure 15 Betweenness vs. the iteration times

parameters and the parameters of the overall system. However, the design process of the local evolution growth and the resultant OO software network topology proves the feasibility of the OO software network evolution growth design and the algorithm as well as the select-ability and maintainability of the topology. It can be used to improve the delicacy of system topology structure and enhance the quality and efficiency of system design.

7 Conclusion and Future Works

General speaking, the method proposed in this paper is an OO software design method based on complex network theory and computation compared with the traditional methods in empirical software engineering. In this paper, we present the theoretical basis, design method, algorithms and an case study of the research in the paper.

In the theoretical basis, it has emphasized the significance of network topology evolution design of OO software systems, developed a conclusion that the selection and modulation of network models with different topology characteristics have un-ignorable influence on the design and implementation of OO software systems, and also analyzed the similar discipline of "negation of negation and compromise" between the evolution of network models with various topology characteristics and the development of software modelling methods. To research on the software engineering approach of preferential attachment in SF network growth of OO software systems, it analyzes the growth characteristics of software patterns and its topology thereof, proposes network topology and local-world growth types of software patterns, and develops a software-pattern based modulation model.

In the description of method and algorithms, it uses requirements (UML use cases) as the guidelines, the types of software pattern as the preferential attachment growth basis, and "calculates" the desired detailed design semi-automated. This kind of method has the following benefits:

1) The trace of software pattern usage recorded clearly in evolution process is beneficial to resolve the traceability problem of pattern usage in system implementation;

2) The local-world in network topology by evolution demonstrates complex network features of multiple hierarchies and multiple granularities, and these features are beneficial for the transformation from previous design to component-based design;

3) The process of network topology evolution growth can proceed automatically or customizably, and then designer can select proper network topology according to user requirements.

Since it can provide parameter systems of OO software network topology in the design process of evolution growth, it can analyze the system performance in the midway, which is beneficial to design a secure and reliable system accordingly. In addition, the modulation strategies and metrics system the method proposed can also very helpful to acquire desired designs.

Finally, we demonstrate the application of this method by a case study in e-Commerce system. Our experiments results prove that this computational method is a useful one for the structure design of OO software systems.

As we can see, the research in this filed is in the starting status, and still many problem are going to be investigated. From our research perspective, the future works mainly cover the following aspects:

1) Interoperability organization, aggregation and fusion of information resource in Networked Software modelling;

2) Validation and verification of Networked Software system;

3) Research on the dependability, reliability and survivability of Networked Software system;

4) Research on the evolutionary design methods of Networked Software systems in real world and conduct corresponding experiments;

5) Research on dynamics behaviors in Networked Software system;

6) Besides software patterns, find out other types in software engineering which provide the SF features in the preferential attachment growth, such as component reuse, framework reuse, public foundation class reuse for system implementation;

7) Develop the tools system to support the topology evolution and growth of OO software network.

References

[1] D. J. Watts and S. H. Strogatz, Collective dynamics of 'small world' networks, *Nature*, 1998, **393**: 400–442.

[2] A. L. Barabási and R. Albert, Emergence of scaling in random networks, *Science*, 1999, **286**: 509–512.

[3] A. L. Barabási, R. Albert, and H. Jeong, Mean-field theory for scale-free random networks, *Physica A*, 1999, **272**: 173–187.

[4] A. L. Barabási, R. Albert, H. Jeong, and G. Bianconi, Power-law distribution of the World Wide Web: response, *Science*, 2000, **287**: 2125a.

[5] R. Albert and A. L. Barabási, Statistical mechanics of complex networks, *Reviews of Modern Physics*, 2002, **74**: 47–97.

[6] G. Bianconi and A. L. Barabási, Topology of evolving networks: local events and universality, *Phys. Rev. Lett.*, 2000, **85**: 5234–5237.

[7] G. Bianconi and A. L. Barabási, Competition and multiscaling in evolving networks, *Europhysics Letters*, 2001, **54**(4): 436–442.

[8] J. Jost and M. P. Joy, Evolving networks with distance preferences, *Phys. Rev. E*, 2002, **66**, 036126: 1–7.

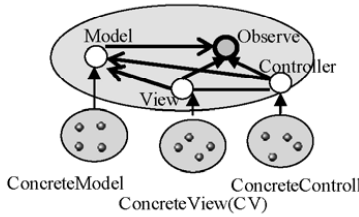
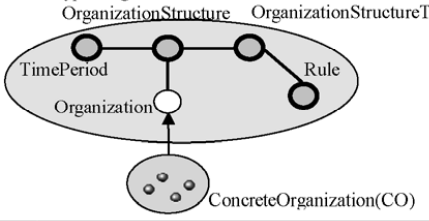
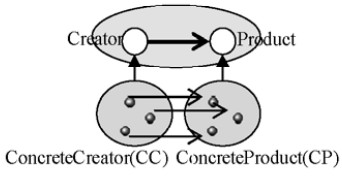
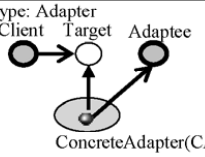
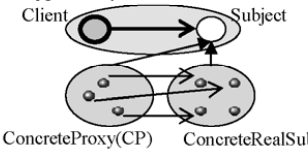
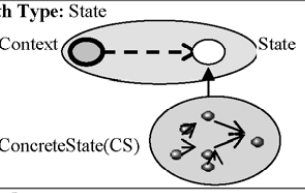
[9] P. L. Krapivsky, S. Redner, and F. Leyvraz, Connectivity of growing random networks, *Phys. Rev. Lett.*, 2000, **85**: 4629–4632.

[10] S. N. Dorogovstsev and J. F. F. Mendes, Evolution of reference networks with aging, *Phys. Rev. E*, 2000, **62**: 1842–1845.

[11] S. N. Dorogovstsev and J. F. F. Mendes, Scaling behaviour of developing and decaying networks, *Europhys. Lett.*, 2000, **52**: 33–39.

[12] S. N. Dorogovstsev and J. F. F. Mendes, Scaling properties of scale-free evolving networks: continuous approach, *Phys. Rev. E*, 2001, **63**, 056125: 1–19.
[13] X. Li and G. R. Chen, A local-world evolving network model, *Physica A*, 2003, **328**(1–2): 274–286.
[14] S. Valverde and Ricard Solé, Hierarchical small-worlds in software architecture, Santa Fe Institute working paper SFI/03-07-044, 2003.
[15] R. S. Chidamber and C. F. Kenerer, A metrics suite for object oriented design, *IEEE Transactions on Software Engineering*, 1994, **20**(6): 476–493.
[16] C. R. Myers, Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs, *Physical Review E*, 2003, **68**, 046116: 1–15.
[17] D. Braha and Y. Bar-Yam, The topology of large-scale engineering problem-solving networks, *Physical Review E*, 2004, **69**, 016113: 1–7.
[18] J. Watson, H. A. Abbass, C. Lokan, and P. Lindsay, Software engineering for artificial life, complex systems, and agent-based distillation, in *Proceedings of the 7th Asia-Pacific Conference on Complex Systems Cairns Convention Centre*, Cairns, Australia, 6–10th December 2004.
[19] L. Li, D. Alderson, R. Tanaka, J. Doyle, and W. Willinger, Towards a theory of scale-free graphs: definition, properties, and implications, Technical Report CIT-CDS-04-006, Engineering & Applied Sciences Division, California Institute of Technology, Pasadena, CA, USA, 2005.
[20] N. F. Schneidewind, Modularity considerations in real time operating structures, *COMPSAC* 77: 397–403.
[21] R. W. Wolverton, The cost of developing large-scale software, *IEEE Transactions on Computer*, 1974, **C-23**(6): 615–636. Also in: Tutorial on Programming Productivity: Issues for the Eighties, IEEE Computer Society, Second Edition, 1986.
[22] B. Liu, D. Y. Li, J. Liu, and F. He, Classifying Class and Finding Community in UML Metamodel Network, Springer LNCS 3584, 1st International Conference, Proceedings pp. 690–695, Advanced Data Mining and Applications, 2005.
[23] S. Valverde1 and R. V. Sole, Network Motifs in Computational Graphs: A Case Study in Software Architecture, Dynamically Evolving, Large-scale Information Systems Technical Report 0206: 1–9 (DELIS-TR-0206), Santa Fe Institute.
[24] E. Gamma, R. Helm, R. Johson, and J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*, Soft-Bank Publishers (in Japanese), 2000, 325–333.
[25] M. Fowler, *Analysis Patterns: Reusable Object Models*, Addison-Wesley Publishers Japan, Ltd, 1998.
[26] J. S. Wu and Z. R. DI, Complex networks in statistical physics, *Progress Physics* (in Chinese), 2004, **24**(1): 18–44.
[27] C. W. Irving and D. Eichmann, Patterns and design adaptability, *Pattern Languages of Programs*, 1996, **2**: 1–10.
[28] S. N. Dorogovtsev and J. F. F. Mendes, *Evolution of Networks From Biological Nets to the Internet and WWW*, Oxford University Press, 2003.
[29] S. N. Dorogovtsev, J. F. F. Mendes, and A. N. Samukhin, Structure of growing networks with preferential linking, *Phys. Rev. Lett.*, 2000, **85**: 4633–4636.
[30] K. Q. He, S. Ying, and F. He, A constructive state reflective pattern, *Journal of Software* (in Chinese), 2001, **12**(8): 1242–1249.

Appendix

Architecture Pattern	
Growth Type: MVC 	
FS	δ_2(Model, Observer) $\wedge\delta_2$(Controller, Model) $\wedge\delta_2$(View, Modle) $\wedge\delta_1$(View, Controller) $\wedge\delta_4$(View, Observer) $\wedge\delta_4$(Controller, Observer)
CM	$\forall$cm$\in$CM, δ_4(cm, Model)
CV	$\forall$cv$\in$CV, δ_4(cv, View)
CC	$\forall$cc$\in$CC, δ_4(cc, Controller)
Analysis Pattern	
Growth Type: Organization 	
FS	δ_1(TimePeriod, OrganizationStructure) $\wedge\delta_1$(OrganizationStructure, OrganizationStructureType) $\wedge\delta_1$(OrganizationStructureType, Rule) $\wedge\delta_1$(OrganizationStructure, Organization)
CO	$\forall$co$\in$CO, δ_4(co, Organization)
Design Pattern (Creational Patterns)	
Growth Type: Factory Method 	
FS	δ_2(Creator, Product)
CC	$\forall$cc$\in$CC, δ_4(cc, Creator)
CP	$\forall$cp$\in$CP, δ_4(cp, Product)
Prob.	$\forall$cc$\in$CC $\forall$cp$\in$CP δ_2(cc, cp)
	Constraint: $\forall$cc$\in$CC ($\exists$cp$\in$CP δ_2(cf, cp))$\rightarrow$ ($\forall$cp'$\in$CP$\wedge$cp$\neq$cp') $\neg\delta_2$(cf, cp')
Design Patterns (Structural Patterns)	
Growth Type: Adapter 	
FS	δ_2(Client, Target)
CA	$\forall$ca$\in$CA, δ_4(ca, Target) $\wedge$ δ_2(ca, Adaptee) $\wedge$ |CA|=1
Growth Type: Proxy 	
FS	δ_2(Client, Subject)
CRS	$\forall$crs$\in$CRS, δ_4(crs, Subject)
CP	$\forall$cp$\in$CP, δ_4(cp, Subject) $\wedge$ (|CP| $\leq$|CRS|)
Prob.	$\forall$cp$\in$CP $\forall$crs$\in$CRS δ_2(cp, crs)
	Constraint: $\forall$crs$\in$CRS ($\exists$cp$\in$CP δ_2(cp, crs))$\rightarrow$ ($\forall$cp'$\in$CP$\wedge$cp$\neq$cp') $\neg\delta_2$(cp', crs)
Design Pattern (Behavioral Patterns)	
Growth Type: State 	
FS	δ_3(Context, State)
CS	$\forall$cs$\in$CS, δ_4(cs, State)
Prob.	$\forall$cs1, cs2$\in$CS δ_2(cs1, cs2)
	Constraint: None

Note: Stable frozen node; Increasable frozen node; Hot spot in which node is increasable

FS: Frozen Spots Prob.: Probability

The definitions of δ_1, δ_2, δ_3, δ_4 refer to Section 5.3.1.