International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:8, 2010

A Functional Framework for Large Scale Application Software Systems

Han-hua Lu, Shun-yi Zhang, Yong Zheng, Ya-shi Wang and Li-juan Min

***Abstract*—**From the perspective of system of systems (SoS) and emergent behaviors, this paper describes large scale application software systems, and proposes framework methods to further depict systems' functional and non-functional characteristics. Besides, this paper also specifically discusses some functional frameworks. In the end, the framework's applications in system disintegrations, system architecture and stable intermediate forms are additionally dealt with in this in building, deployment and maintenance of large scale software applications.

***Keywords*—**application software system, framework methods, system of systems, emergent behaviors

I. Introduction

APPLICATION software systems are playing an important role in many society activities. Nowadays, more and more functional requirements for application software systems are being presented, while construction and maintenance jobs have become more and more difficult. For example:

- It requires a relatively short time span to build or rebuild in order to make quick responses to real-time requirements;
- Functional requirements are ambitiously expanding;
- More and more uncertain requirements have come up, which requires adaptations in actual executions;
- Systems need to become more dynamic and more open.

Consequently, the functions and technologies of application software are more and more intricate and the scale accordingly turns out to be larger. Additionally, associations among application software systems are more and more complex, while more systems are involved in the ultimate goal of the whole application. With the rapid and emerging development of SOA, cloud computing, ubiquitous networks and Ambient Intelligence (AMI), the constructions, deployment, execution and maintenance have been significantly changed and updated. This garners advantages, but at the same time, there are also emerging problems which never existed in mere monolithic systems. New technologies and methods are required to solve these problems.

Applications' functions are integrated under the context of SOA and cloud computing, while the constituents of these functions are required to be provided by autonomous systems. It is a growing popular tendency in current applications. Take actual applications for example, more and more people would like to use meta-search engines in order to obtain better results and better performances; more and more application integrators are inclined to use GIS services from the Internet as position functions for their customers' IT systems; and more and more enterprises are utilizing CRM based on SaaS in order to offer support for their businesses.

Coupled with applied technologies in order to get more convenient services and advanced efficiencies, as well as to implement flexible services with low costs, the dependency on these autonomous systems also gave its share of headaches. For example:

- It is difficult to predict the applications' performance due to the invisibility of external autonomous systems. However, performance problems may directly influence the application goals when it comes to crucial situations.
- Both associations among complex systems and their openness enable security situations to be more intricate. Assorted attacks and misuses could possibly destroy the whole application.
- It is difficult to control external autonomous systems when using them. Emergent behaviors which were not observed before and are beyond our expectations may result in business processing failures.

To sum up, current large scale application systems are in the process of switching from isolated monolithic systems to the more open process of system of systems. This characteristic has aroused software industries' attention. The concepts of system of systems and emergent behaviors [1] are proposed and introduced to figure out solutions for these problems.

According to the concepts of system engineering, systems with the characteristics below can be categorized as SoS (System of Systems) [2-3] :

- Obtain operational independence ;
- Obtain managerial independence ;
- Obtain geographic distribution ;
- Perform emergent behavior;
- Perform evolutionary development

Han-hua Lu is with Nanjing University of Posts & Telecommunications, Nanjing, Jiangsu Province 210003 China (corresponding author, phone: +86-25-83491312; fax: +86-25-83491372; e-mail: luhh@njupt.edu.cn).
Shun-yi Zhang is with Nanjing University of Posts & Telecommunications, Nanjing, Jiangsu Province 210003 China (e-mail: dirzsy@njupt.edu.cn).
Yong Zheng is with Nanjing University of Posts & Telecommunications, Nanjing, Jiangsu Province 210003 China (e-mail: zy_njupt@hotmail.com).
Ya-shi Wang is with Nanjing University of Posts & Telecommunications, Nanjing, Jiangsu Province 210003 China (e-mail: wangyashi@njupt.edu.cn).
Li-juan Min is with Nanjing University of Posts & Telecommunications, Nanjing, Jiangsu Province 210003 China (e-mail: minlj@njupt.edu.cn).

Emergent behaviors usually manifest the following characteristics [4、5]: "The common characteristics are: (1) a radical novelty (features not previously observed in systems); (2) a coherence or correlation (meaning integrated wholes that maintain themselves over some period of time); (3) on a global or macro "level" (i.e. there is some property of "wholeness"); (4) the product of a dynamical process (it evolves); and (5) are also "ostensive" (it can be perceived). For good measure, Goldstein throws in supervenience -- downward causation."

Obviously, large scale application software systems also have these characteristics, as they integrate autonomous constituents. This tendency requires the following applications, including cloud computing and SaaS to face these problems below：

- The system can only use the function of constituents, but cannot maintain full control；
- It is difficult to totally define constituents' functions as well as the predictions of functions evolution；
- The system is invisible to its constituents' structures and technologies, and therefore also to their monitoring and maintenance.

Due to the fact that large scale application software owns characteristics of SoS, and its application goals are implemented by integrated autonomous systems, therefore, descriptions and definitions should be conducted during both developing and executing stages, so its constituents and the whole system can satisfy business goals. Additionally, SoS performs evolutionary development, so characteristics requirements may be continuously changing in the process of development. It requires certain abstract-leveled descriptions to ensure the systems' stability and consistency.

Functional abstract descriptions are further discussed in this paper. These abstract descriptions can be continuously specified and formulized in the processes of system building and reconfiguration in order to guarantee functional integrity and consistency, as well as the stability of the long-term evolutionary development and quick responsibility towards requirement changes for application goals.

II. Basic Requirements for Software Systems

In order to guarantee basic application functions, a software system is required to obtain these characteristics:

- It can meet functional requirements of application goals；
- It can be executed stably and continuously；
- It can run with satisfactory performance；
- It can ensure security both for applications and for data.

The characteristics above can be divided into two scopes – functional and non-functional. For functional scopes, as depicted by Figure 1, there are two orientations for system to perform evolutionary development – The first one is from abstract to detailed development in order to ultimately satisfy systems' functional requirements; while the other depicts deformalization to formalization in order to ultimately satisfy systems' execution requirements. No matter from perspective of the whole system and life cycle, or from a view of constituents' maintenance or development, it will always perform in the same way.

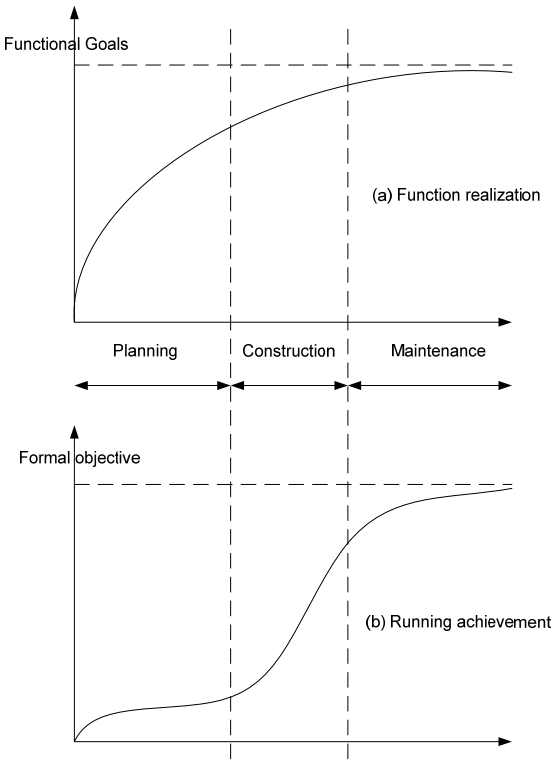

Fig. 1 System's Evolutionary

Because of the uncertainty of requirements and the changes of systems execution architecture, the ultimate goal of these two orientations could not be achieved at the same time. This turns out to be significantly difficult, especially when it comes to large scale application software systems. The specific reasons can be listed as follows:

- Systems have a long time life cycle, and they also need to be evolutionarily developed；
- On the condition of no controls, constituents' emergent behaviors cannot easily coincide with the two goals indicated above；
- Understandings of business goals are always changing in the lifecycles of systems；
- Business goals are continuously changing through different periods；
- Constituents are supposed to run and be maintained independently；
- As a SoS form, its constituents' goals are not consistent with the scopes of the system；

- Constituents are required to be alternated or reconfigured according to dynamic requirements ;
- System environment is intricate and is frequently subject to change.

Based on the situation and problems indicated above, we can point out that it is difficult to totally guarantee the consistency of the application of SoS and the entire business goals, efficiently control and utilize the systems' emergent behaviors, reasonably organize systems' functions and information, and ensure agile business support.

III. System Abstraction and Functional Framework

We can consider an application software system as a Discrete Event Dynamic System (DEDS) [6], whose inner status and outputs can be described by functions related to past status and inputs. From this view, we can describe these software systems in such a way: these systems will accept inputs or stimuli, then create their own outputs and change their status. This behavior can be depicted in the formulas below:

$$O(n) = F_o(I(n), S(n)) \quad (1)$$

$$S(n) = F_s(I(n), S(n-1)) \quad (2)$$

In the formulas above, O, I and S are all vectors:

$$O = (o_1, o_2, ..., o_i) \quad (3)$$

$$I = (i_1, i_2, ..., i_j) \quad (4)$$

$$S = (s_1, s_2, ..., s_k) \quad (5)$$

Specifically, O stands for inputs, while I for outputs, and S equals to the internal status of systems. These vectors and their components are all time sequences; F_O represents the functions which can be observed outside the system, F_S stands for processing functions for the system's permanent data storage; subscript 'n' signifies the current value of discrete series, while 'n-1' stands for the last value. The system abstractions are depicted in Figure 2.

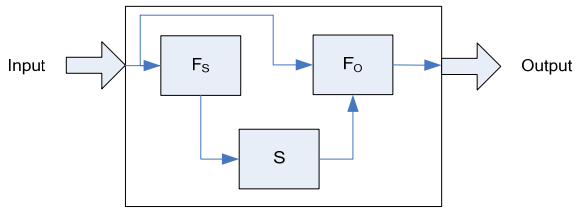

Fig. 2 System abstractions for application software

Furthermore, if F_S and F_O's processes towards S can be combined to formulate an information object S_F, and represent the separated F_O as F, then the whole system can be depicted by Figure 3 which is an object-oriented representation. In this view, a software system can be represented by the input vector (I), the output vector (O), the inner status vector (SF) and outer behaviors (F) [7].

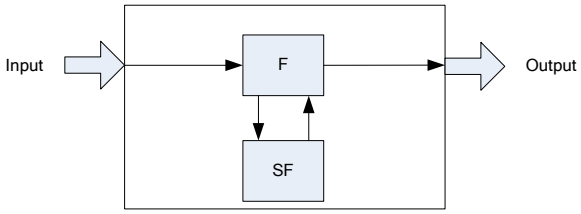

Fig. 3 System's object-oriented representation

In the case of SoS, though there may be complex constructs within the boundaries of the whole system, we can still define the framework with these abstractions from the perspective of business functional requirement satisfactions. Ideally, we can consider the results of SF concretion and formulizations as the systems' basic components or constituents, while viewing F as the orchestration or composition of these basic components or constituents, and assume F to be stateless.

Ideally, both F and SF should be fully established according to business requirements, to meet these requirements. However, when it comes to large scale application software systems, this ideal condition does not exist, due to the following reasons:

- System requirements cannot be totally scheduled in the process of constructing. Growing evolutionary development is needed to satisfy dynamic business requirements ;
- The structures of S and SF change frequently along with the development of requirements or technologies, during the whole life cycle of the system ;
- Understandings of business requirements are always changing as well. We cannot schedule a long-term stable system according to current temporary requirements ;
- As a SoS, a large scale information system always relies on the functions or services of different autonomous systems, while the goals of these autonomous systems may not coincide with those of SoS.

Therefore, the continuous reconfiguration according to dynamic business requirements and computing environments is required, and this reconfiguration should not interfere with system operation.

In order to attain these goals, we can make an effort to make use of system association techniques [8], to reconfigure the system rapidly, in accordance with system specification guidelines, and use the specification in system development and maintain activities, to get system stabilities and greater processing consistency.

Regarding the system function, popular techniques used for reconfiguration are workflow technologies, generalized programming, rules engine and strategies technologies, machine learning, and resource discovery, etc. All the techniques largely improve systems' flexibilities. However, when it comes to large scale application software systems, the

precondition of the application of these techniques is the full understanding of systems' short-term and long-term goals, and knowledge of the flexible controls of systems' evolutionary developments. The description from abstractions to specifications is required in order to achieve this criterion. Functional framework here is the corresponding abstract description for system functions.

In reference [7], we proposed a framework architecture composed with functional requirement framework based on enterprise or organization processes, application framework abstractly depicted system's behaviors, and the information framework describing systems' information, which can be depicted in Figure 4.

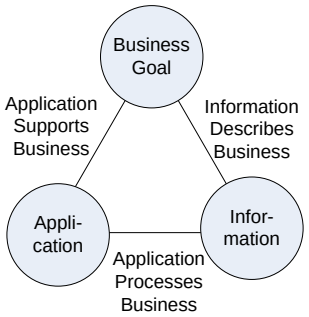

Fig. 4 Systems' Functional Framework

Comparing this framework architecture with previous indicated abstractions, application framework is for system behaviors (F) abstraction, while information the frameworks for Inputs (I), Outputs (O) and System Status (SF), and the business framework are requirements for applications.

The ultimate function of these frameworks is to specify system functions. These specifications can be executed in the system planning phase, analysis and design phases, as well as in the maintenance phase. There are two functions for these specifications:

- Targeted objects classifications. For example, through business frameworks, collected requirements can be classified into specified process catalogues, to standardize these requirements. These classifications can guarantee system functional consistency in the whole life cycle.
- Targeted objects associations. Association relationships among the three parts of functional framework will be taken advantage of in order to establish associations among targeted objects. For example, associations between application framework and business framework can help make sure which application units can support classified business units to which the requirements is referred. These associations can guarantee the orderliness and coherence in the process of development and maintenance, in order to further guarantee resource reusages and the growing evolutionary development, and also prevent components and codes from unreasonable expansion.

The specifications of this framework enable us to obtain descriptions of system requirements, applications and information. The pith of the framework is to establish different levels of abstractions and apply them to diverse scopes, such as industrial, entrepreneur or organizational, either with project or system planning, all of which need more specific maintenance.

IV. Disintegration and Specification based on Frameworks

Layered-structure is an efficient solution to fix complexity problems. However, in case of SoS, it is difficult to realize layering towards system controls, due to the autonomous characteristic of its constituents. Therefore, layered-abstractions are useful especially when facing system complexity problems. And the process from abstractions to specifications does well to system implements.

Specifying functional implements is the main function of framework. Specifically, in the process of system builds and maintenance activities, framework is used to specify requirements, classifications and associations between information and applications. Specifications and formalizations are also conducted under the guidance of frameworks. In order to realize this goal, disintegration and specification based on frameworks should be further researched on.

The disintegration here is on the basis of system abstractions in the Figure 3, that is:

$$O = F(SF, I) \tag{6}$$

O, SF and I are all discrete time series vectors.

As indicated before, F is stateless. It integrates SF and I, and then creates outputs. Therefore, disintegrations based on frameworks firstly emerge on the disintegration of SF.

SF can be disintegrated into several sub-vectors:

$$SF = (sf_1, sf_2, ..., sf_n) \tag{7}$$

And then there are two methods to proceed.

1) Consider sf1, sf2, …, sfn as atom vectors which cannot be further disintegrated. Then integrate them according to certain rules to formulate information system (as Figure 5 described):

$$SF = \bigcup_{i=1}^{m} SF_i \tag{8}$$

$$SF_i = (sf_{i1}, sf_{i2}, ..., sf_{ij}, ..., sf_{ik}), sf_{ij} \in SF \tag{9}$$

$$SF_i \subseteq SF \tag{10}$$

The combined sets formulated by SF_i could be orthogonal, which means:

$$\forall SF_i \rightarrow SF_j \cap SF_k = \varnothing, j \neq k \tag{11}$$

Of course, these combined sets can be not fully orthogonal also:

$$\exists SF_j, SF_k \rightarrow SF_j \cap SF_k \neq \varnothing, j \neq k \tag{12}$$

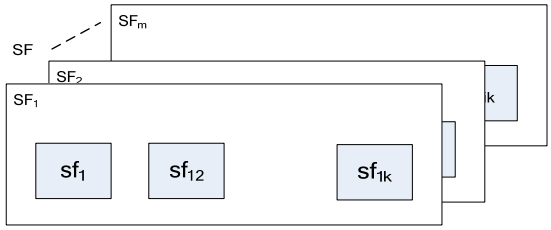

Fig. 5 SF's disintegration and integration

2) Due to that SF contains data and its processing, we can consider sf1, sf2, …, sfn as a lower-layer system, and further conduct disintegrations. In case of every sf, we can also use its inputs (Isf), outputs (Osf), behaviors (Fsf) and status (SFsf) to describe, which can be depicted by Figure 6. What should be noticed is that, Fsf is stateless, while SFsf stands for its status :

$$Osf = Fsf(SFsf, Isf) \tag{13}$$

This kind of disintegration can be continued continuously until a required specific extent is achieved.

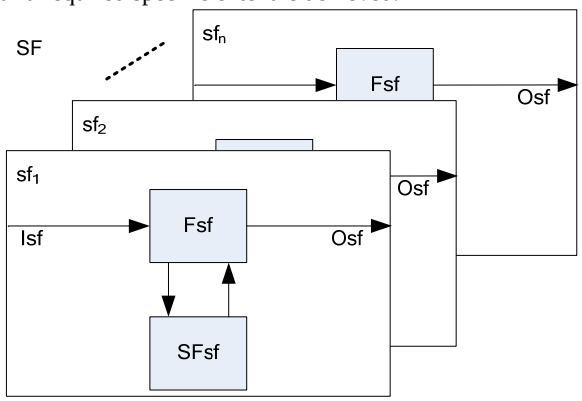

Fig. 6 SF's disintegration

Method 1) is mainly used in system integrations. According to certain rules, system information SF will be divided into different constituents (SF_i), then functional units defined by framework will be classified to include in these constituents. The only parts for implement are combinations and invoking sequence processed by SF. Ideally, F can be realized by generalized programming, such as workflow and rules, etc. Web Services programming using BPEL can be viewed as one of these implements.

Method 2) is applied in system functions analysis and implements. We can further disintegrate information of sf to develop its information processing functions and algorithms. These disintegrations can be designed as recursive ones. The disintegration results can be used for deeper-layered framework descriptions, and also can be implement model and design for further integrations. These disintegrations are the specifying process indicated in chapter one. Traditional object-oriented software development can be considered as the implement of these disintegration processes.

If object-oriented approach is used to describe business requirements and business process, then business goals framework can be conducted through similar disintegrations and specifications.

V. System Architecture and Framework Methods

As indicated before, one of the most important characteristics of SoS is the autonomy of its constituents. Reference [9] pointed out that higher-leveled SoS can only pose influences on these constituents, but cannot control them. This is the most significant difference between system and SoS.

However, from actual perspective, these influencing and controlling relationships between SoS and its constituents are relative, which means that there are different leveled-influences, and there is gradient processes between influences and controls, instead of absolute separated ones. To integrate as a SoS, there exists parts totally be controlled, as well as parts totally be autonomous. Of course, parts interposed in these two situations are also allowed.

In case of large scale application software system, it is impossible to fully not include controlled parts. For example, An order management system is required to communicate with products supplier systems, in order to perform a full functions including product ordering and product applications. We can consider this system as a SoS, while product supplier system and geographical information system can be view as the autonomous constituents. However, order management system requires essential status management and controls, such as customer information and product information, which may be provided by outer systems, but whose inner status is controlled by the enterprise. In other words, these constituents are owned by the enterprise.

Based on this situation, as well as the processing method 1) indicated in chapter four, we can separate sf into two types – the first one is being controlled constituents SF_C, while the second one is constituents without controls SF_U:

$$SF = SF_U + SF_C \tag{14}$$

$$SF_U = (sf_{U1}, sf_{U2}, ..., sf_{Ui,...,} sf_{Um}) \tag{15}$$

$$SF_C = (sf_{C1}, sf_{C2}, ..., sf_{Cj,...,} sf_{Cm}) \tag{16}$$

As a SoS for large scale software system, these two parts need difference processes to proceed.

Regarding SF_U, it can be processed by method 1) indicated in chapter four, due to its unknown inner structures. It is a method which only integrates outer functions to formulate all the required functions for the whole system. It is the common approach applied in SOA and cloud computing. In this way, functions integrated by sf and its integrations are views as stateless ones whose outer functions are only our focused attentions on. The very problems about processing of sf_U and its integrations are not the basic functions implements, but the special problems come up from SOA and cloud computing. These problems can be listed as follows:

- How to define and discover functions or services ;
- How to notice changes of current functions ;
- How to ensure system's stability when changes emerge in outer autonomous constituents ;
- How to guarantee business supports during system recombination ;

- How to process outer service errors and exceptions ;
- How to guarantee the security and performance of outer services, as well as the quality of services ;
- …

Additional functions and structures, such as Web Service, SCA and structures mentioned in reference [1] are required to solve these problems.

Problems are not that obvious when it comes to SF_C. Instead, the attention should be paid on function implements, inner status maintenance and guarantees which can be processed by tradition approaches. Method 2) pointed out in chapter four is suitable for these problems. Specifications of sf_C should be conducted until we get the required functions. Due to that it belongs to the normal software engineering scope, this paper will not indicate more about that.

VI. Stable Intermediate Forms and Framework Methods

Because of the dynamic changes of context and requirements, in macro-view, changes are always continuous during the life cycle of large scale software systems. That how to keep stability and decrease the risk brought by changes turns out to be a crucial issue for large scale application software systems.

Due to the complexity of large scale SoS and the uncertainty of context, if Stable Intermediate Forms (SIF) [10-11] can be established in the process of builds and maintenance, and further be developed in a revolutionary way, it will facilitate the overall goals' achievements. Iterative and incremental developments are all outcomes of these methods. In case of SoS composed by several autonomous systems, SIF has been involved in not only development jobs, but also executions and maintenance stages.

When establishing SIF, constructed SIF requires evaluations in order to grasp the development and maintenance progresses and the overall situations. Based on this purpose, framework can be viewed as the most efficient tool for these evaluations. Therefore, functions evaluations are recommended to conduct from perspective of framework disintegrations.

Regarding the evaluation contents, functions coverage and functions specifications can be the main two orientations.

Currently, the normal coverage detection method is on the basis of Petri Nets which will be established according to mapping of requirements [7] and peer-to-peer process flow of requirements. The established net system N is as follows:

$$N = (P, T; F) \tag{17}$$

P stands for the system status SF, while T for the system function F.

Regarding status, we have disintegrated as follows:

$$P = SF = \{sf_i\}, i = 1, 2, ..., n; \tag{18}$$

In order to establish Petri Nets, F is also required to be integrated:

$$T = F = \{f_j\}, j = 1, 2, ..., m; \tag{19}$$

In the formula above, f_j equals to the No. j step in the whole process. As indicated before, on ideal conditions, system does not exist the entity of F which is actually an integration of steps. Every step integrates several sf, and then conducts processing to implement corresponding functions. In real SOA practice, BPEL defined sets using for node processing can be viewed as F, while all Web Service sets revoked by it can be considered as SF. Through these static analyses, it can be used for functional coverage evaluations for the system.

What is worthy of noticing is that, Petri Nets indicated above represent relationships between processing steps and information processing instead of the whole process, though Petri Nets are established for peer-to-peer processes. These relationships are reflected by Pre-sets and Post-sets. If sf_i is the essential information depended by f_j processing, then sfi belongs to fj's Pre-set. If sf_i's status is changed after f_j's processing, then sfi belongs to fj's Post-set. It is similar to the concepts of data flow graph. But its targeted object is not data. Figure 7 gives an example of Petri Nets representation.

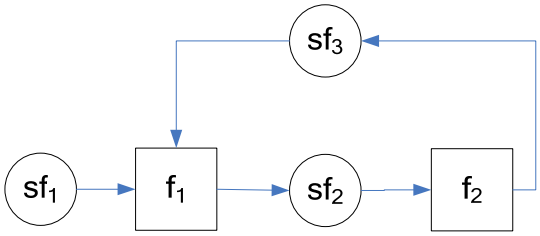

Fig. 7 Petri Net Representation

In this example:

$$P = \{sf_1, sf_2, sf_3\} \tag{20}$$

$$T = \{f_1, f_2\} \tag{21}$$

$$F = P \times T \cup T \times P = \{(sf_1, f_1), (sf_2, f_2), (sf_3, f_1), (f_1, sf_2), (f_2, sf_3)\} \tag{22}$$

There are two processing steps in this figure – f1 and f2. The Pre-set of f1 (C_{f1}) is:

$$C_{f1} = \{sf_1, sf_3\} \tag{23}$$

The Pre-set of f2 (C_{f2}) is:

$$C_{f2} = \{sf_2\} \tag{24}$$

In Figure 7, f1's processing depends on sf1, sf2 and sf3. It requires revoking sf1 and sf2 to obtain relevant information. After that, the status of sf3 will be changed accordingly.

In this representation, a place without pre-sets means the processing of the whole process poses no influences on its status. If no sf owns pre-sets in the whole processing graphs, then this sf can be member of SF_U indicated before, because there are totally no influences on its status, not to mention controls; on the contrary, if a sf owns pre-sets and also tends to be a member of SF_U, other conditions, especially mainly on semantics will be required.

Additionally, characteristics below can be also concluded from this representation:

- A f without pre-sets only relies on inputs to complete processing functions, which has nothing to do with system status ;
- The implement of a f without post-sets mainly

equals to output processing ;

- A sf without post-sets is independent information for system. And it only provides interfaces for other systems.

The conclusions above are the observation results of the whole system processing.

Based on these representations, assorted analysis methods in Petri Nets can be applied to evaluate the implement levels of peer-to-peer processes functions in systems. When it comes to that some sf are unavailable or incomplete, it can be used to further evaluate the specific influences of peer-to-peer processes in systems. These evaluations can be viewed as the basis for establishing stable intermediate forms, and further be used to judge the status of functional implements.

The building of N does not rely on the design and implement of sf. Instead, it only depends on the descriptions of information processing functions which is the main jobs of frameworks.

On the condition of incomplete sf, method 2) mentioned in chapter 4 which needs sf subdivisions will be required to provide more precise evaluations. The scope of this job is the part in Fsf related to complete set of processes. The specific method is similar to previous indicated ones. Besides, there are two points worth noticing:

- Through subdivisions, we can get the quantitative values of system-defined abstraction levels. After weight processing, we can make sure the extent of applicable implement of current system functions, which will further provide references for building stable intermediate status ;
- The net system indicated above can be concluded from Petri Nets for system processes. Subdivision can also be implemented by Petri Nets methods. If cross-layers are in need, the concepts of sub nets could be applicable. Reference [13] provided an approach for master-slave workflow system. Layered-dyeing in this approach will be also beneficial for subdivision jobs.

VII. Conclusion

This paper discusses normal concepts of framework methods for large scale application software systems, and specifically introduces the principles and constituents of functional frameworks. The functions of this functional framework on system disintegrations and integrations, system architecture under SOA and cloud environment, and stable intermediate form buildings are also discussed in this paper.

Of course, functions of framework are not limited to these ones. Except from what we have pointed out, it also plays an important role in software engineering processes and system specifications, which we have introduced in references [7, 12].

Current research contents towards framework methods also include:

- Research on methods for system characteristics sets descriptions. How to formally represent the overall and separated functions will be the main theme.
- The applications of functional framework in the service discovery areas. Based on characteristics sets, clustering algorithms will be applied to find suitable services.
- Quantitative measurement towards the mapping relationship among business application goals, system applications and system information. Through this quantitative measurement, the support extents for business goals can be precisely evaluated.
- Mapping and measurement relationships among business goals, application and information units in functional framework.
- Structures and descriptions of security and performance frameworks.

The ultimate goal of these research jobs is to enable a large scale application software system owing SoS characteristics to better satisfy business requirements; be able to conduct quick recombination according to dynamic changes of requirements and contexts; and also guarantee stable executions during the long-term evolutionary developments.

Acknowledgment

This paper is supported by National Key Technology R&D Program in the 11th Five year Plan of China (No.2007BAH17B04), National High-Tech Research and Development Plan (863) of China (No. 2009AA01Z212, 2009AA01Z202), Natural Science Foundation of Jiangsu Province (No.2007BK603), High-Tech Research Plan of Jiangsu Province (No.BG2007045) and Talent Introduction Foundation of Nanjing University of Posts & Telecommunications (No.NY2007044). At the same time, thanks to the support of Fujian Fujitsu Communication Software Co., Ltd. (FFCS). With their support, this study works well and is processed in a smooth way.

References

[1] Radu Calinescu and Marta Kwiatkowska, Software Engineering Techniques for the Development of Systems , In *Proc. 15th Monterey Workshop on Foundations of Computer Software*, pages 86--93. September 2008

[2] Sage, A.P. and C.D. Cuppan, On the Systems Engineering and Management of Systems of Systems and Federations of Systems, *Information, Knowledge, Systems Management*, Vol. 2, No. 4, 2001, pp. 325-345.

[3] Maier, M.W., “Architecting Principles for System of Systems”, Systems Engineering, Vol. 1, No. 4, 1998, pp. 267-284

[4] Corning, Peter A. (2002), "The Re-Emergence of "Emergence": A Venerable Concept in Search of a Theory" , Complexity, Vol. 7, No.4, 2002, pp.18-30

[5] Goldstein, Jeffrey (1999), "Emergence as a Construct: History and Issues", *Emergence: Complexity and Organization* 1 (1): 49–72

[6] Xi-Ren Cao (Editor-in-Chief), Discrete Event Dynamic Systems, Theory and Applications, ISSN: 0924-6703, Springer, 2008

[7] Lu Hanhua, Wang Yashi and Min Lijuan, Framework Method Used for Large Scale Application Information Systems, INC2010 , May 2010

[8] Lu Hanhua, Zheng min, Component and Service Integrating Technologies in OSS, Chinese Journal of Telecommunications Science, No.9, 2006, pp.42-46

[9] David A. Fisher, An Emergent Perspective on Interoperation in Systems of Systems, CMU/SEI-2006-TR-003 ESC-TR-2006-003, March 2006
[10] Kevlin Henney. Stable intermediate forms: A foundation pattern for derisking the process of change. In Proceedings of the Ninth European Conference on Pattern Languages of Programs, 2004.
[11] H.A. Simon. The Sciences of the Artificial. MIT Press, 1996.
[12] Lu Hanhua, Wang Yashi, Min Lijuan, Huang Zhenqi, OSS/BSS Framework Based on NGOSS, Chinese Journal of Telecommunications Science, No.10, 2009, pp.57-62
[13] Lu Hanhua, Min Lijuan, Wang Yashi, Approach to master-slave workflow system and its Petri-net modeling, Chinese Journal on Communications, Vol. 31, No.1, 2010, pp.92-99