Quantitative Analysis of Design Decisions for Dynamic Reconfiguration

Zhikun Zhao and Pengfei Zhao
School of Computer Science and Technology, Shandong University of Finance and Economics
Jinan, Shandong, China, 250014
zhaozk@sdfi.edu.cn, zhaopf@sdfi.edu.cn

Wei Li
School of Computer Science, Central Queensland University
Rockhampton, Australia, 4702
w.li@cqu.edu.au

***Abstract*—Dynamic reconfiguration is becoming an important requirement of current software systems as they have a trend towards running continuously and updating frequently. The main purpose is to reduce the update cost caused by system shutdown or restart. Controlling the influence of dynamic reconfiguration on system performance is an essential yet difficult issue. Many factors have to be considered and various methods could be chosen to use, including underlying component model, state transfer, connector type, reconfiguration algorithm and reconfiguration execution. These issues are analyzed in this paper and special attention is focused on how these design decisions affect the influence of dynamic reconfiguration on system performance. A DSE system is used as a case to analyze the design choices for dynamic reconfiguration through out this paper. A comparison of the influences resulted from different design choices is made through recording the influence parameters of the system in reconfiguration under different design choices.**

***Index Terms*—dynamic reconfiguration, design decisions, influence control**

I. Introduction

The widespread use of continuously-running applications has raised the need for dynamic reconfiguration, i.e. changing system architecture at runtime [10]. These applications, such as banking systems and web services [17], need to be updated frequently because of the evolving user requirements, hardware technology, or outside environment [11]. Both users and service providers of these systems do not want to suffer the cost of system shutdown during the period of update [12]. Therefore, dynamic reconfiguration is the best solution.

One of the most important problems of dynamic reconfiguration is its severe threat to system performance, or in other words QoS (Quality of Service) [4]. If a dynamic reconfiguration causes the QoS of a system decline to zero and lasts for a relatively long period, dynamic reconfiguration will lose its significance because it has no much difference with system shutdown or restart [14]. Therefore, in many performance-critical systems, it is necessary to control the influence of dynamic reconfiguration on system QoS [8].

However, it is not easy to design a dynamic reconfiguration with QoS management. From underlying component model to reconfiguration plan, many factors have to be considered carefully. And for each factor, various methods could be chosen to use. Therefore a series of decisions need to be made in designing a system being dynamic reconfigurable. The problem is that it lacks of a systematic analysis of these factors that could help designers in making the appropriate decisions. Although many researchers have made their contributions to the area of dynamic reconfiguration since Kramer and Magee's early work [7], only a few works have been focused on the influence control for dynamic reconfiguration. So far as we know, Hillman and Warren have compared several reconfiguration algorithms [6], but the analysis has been restricted to the reconfiguration algorithms and other factors have not been considered.

Based on the previous works, different design choices and their influence on system QoS are analyzed in this paper. These design choices include: what component model can be used to construct a system, what algorithm can be used to achieve a reconfiguration, what method can be used to preserve transaction integrity, what policy can be used to schedule resources, and what interaction protocol can be used to synchronize reconfiguration operations among geographically distributed components. A real world application, the DSE (Digital Signature and Encryption) system, is used as a case throughout this paper to test the QoS influence of different design choices. Reconfiguration designers can benefit from this work in modeling dynamic reconfiguration and controlling the influence.

II. The Digital Signature and Encryption System

To analyze the design choices for a dynamic reconfiguration, the DSE system is used as an example

Manuscript received July 8, 2012.

through out this paper. A comparison of the influences resulted from different design choices can be achieved through recording the influence parameters of the system in reconfiguration under different design choices. This comparison can help find out which design choice can result to a small influence.

The DSE system is used to secure electronic communications. Data encryption helps the sender protect the information from being known by unauthorized users. And digital signature helps the receiver verify the authenticity of the information. The working progress of the DSE system is shown in Fig.1. To send a data package, the sender encrypts the data and attaches a digital signature. The package transferred is composed of two parts, the encrypted data and the digital signature. When receiving a package, the receiver can decrypt the data and verify the digital signature.

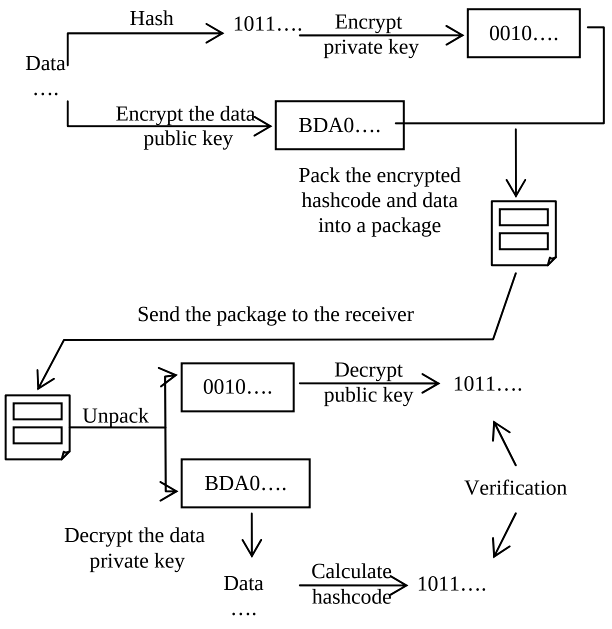

Hash
1011….
Encrypt private key
0010….
Data ….
Encrypt the data public key
BDA0….
Pack the encrypted hashcode and data into a package
Send the package to the receiver
Unpack
0010….
Decrypt public key
1011….
BDA0….
Verification
Decrypt the data private key
Data ….
Calculate hashcode
1011….

Figure 1. The DSE System.

The response time and reconfiguration time are chosen as the influence parameters. Increase in response time reflects how severe the influence is and reconfiguration time reflects how long the influence lasts. For the DSE system, the response time is the time interval between the instance at which a data package becomes ready to be sent at the sender and the instance at which the verification for the data package is finished at the receiver. Reconfiguration time is the time interval between the instance at which a reconfiguration starts and the instance at which the reconfiguration ends.

In some applications, throughput may be chosen as the QoS parameter. Throughput is the amount of data that pass through the system per time unit. A reconfiguration that has influence on response time also has influence on throughput. Longer response time means lower throughput. The length of the time unit also is an important parameter for throughput statistic. If the time unit is far longer than the reconfiguration period, the influence of the reconfiguration is hard to detect. In this paper, response time is chosen as the QoS parameter.

III. Design Decisions

Almost all the existing systems that support dynamic reconfiguration are component based systems. Component is suitable to be the elementary operational unit for structural reconfiguration because of its modularity, well-defined interfaces, and interconnection independence [15]. A component-based system is composed of components and connectors. A structural reconfiguration to such a system can be achieved through a series of operations that add and/or remove components and/or connectors.

Design decisions for dynamic reconfiguration to a component based system include decisions for component model and decisions for dynamic change. There are dependent relationships among these decisions, e.g. some decisions for dynamic change depend on the component model.

A. Design Decisions for Component Model

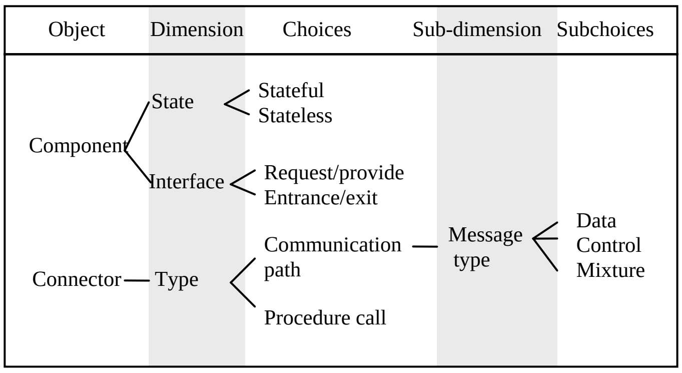
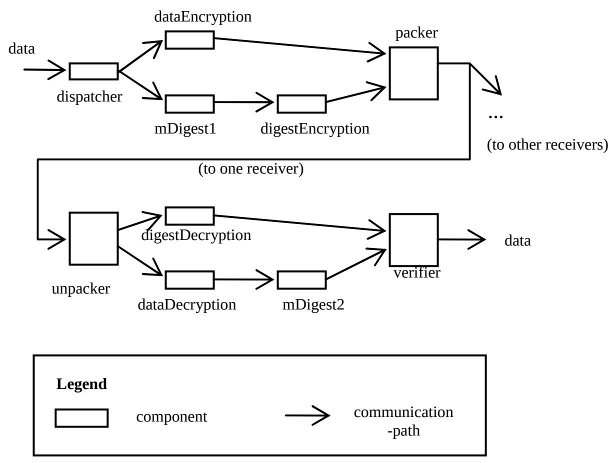


Figure 2. Design Decisions for Component Model

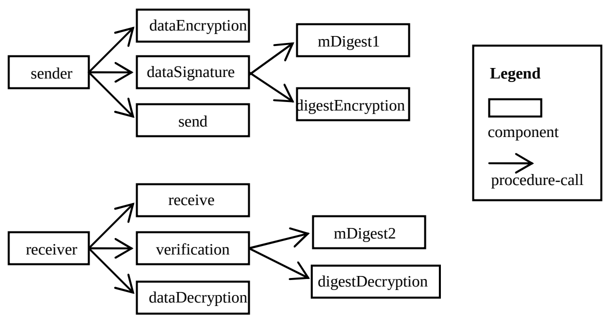
Briefly, component models can be divided into two categories: procedure-call model such as Fractal[1], Rapide[9], SOFA[13] and flow model such as Data Flow Network[2]. In a procedure-call model, components interact with each other through procedure-calls. Therefore a component has request/provide services as external interface and a connector represents procedure-calls. In a flow model, components are as filters and connectors are as pipes. Correspondingly the external interface of a component is the entrances and exits and a connector represents communication paths. What transferred through communication paths can be data, control, or mixture of these two. To show their difference, the DSE system based on different component models is shown in Fig.3.

Communication path is the basic connector between components. A procedure-call can be simulated by two communication paths that transfer the mixture of control and data. The caller transfers the control and the parameters to the callee through one path and the callee returns the control and the result to the caller through another path. More complicated connectors also can be simulated by communication paths.

Another important design decision for component model is whether components should be stateful or stateless [5]. A stateful component maintains an internal state, which makes the component possible to accumulate information over operations. And this information might be used in future operations. If a stateful component needs to be substituted in a reconfiguration, its internal state needs to be transferred to the new one to keep the system running correctly. On the contrary, a stateless component has no internal state and thereby there is no need for state transfer in reconfiguration.



a) The Flow Model



b) The Procedure-call Model

Figure 3. Different Component Models of the DSE System

B. Design Decisions for Dynamic Change

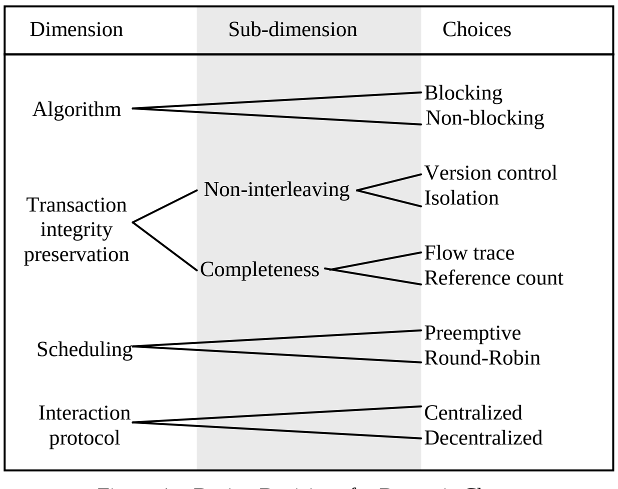


Figure 4. Design Decisions for Dynamic Change

Design decisions for dynamic change include what reconfiguration algorithm is used to achieve the change, what method is used to preserve transaction integrity, what policy is used to schedule resources, and what interaction protocol is used to synchronize reconfiguration operations among geographically distributed components.

Reconfiguration algorithm determines how reconfiguration is achieved by reconfiguration operations step by step. The reconfiguration algorithms currently in use can be classified into two classes, blocking algorithm and non-blocking algorithm. The blocking algorithm firstly waits or drives the system into a consistent state by blocking the components or connectors involved in the reconfiguration. Then it switches the system from the original configuration to the new configuration. Finally it resumes the system by non-blocking the involved components and connectors. The non-blocking algorithm firstly activates the new configuration by starting the new components and establishing the new connectors. And then it closes the entrance of the original configuration so that it can wait the transactions belonging to the original configuration to be completed. Finally it removes the part that belongs to the original configuration. The blocking algorithm provides a consistent system state for stateful components to transfer their states, but it may cause a severe influence on system QoS because new requests are suspended during the period of waiting for the consistent state and transferring components' internal state. On the contrary, the non-blocking algorithm does not support state transfer between original and new components because they need to run in parallel, but it is possible to achieve a zero-influence reconfiguration because the system keeps running during the whole reconfiguration period.

Preserving transaction integrity is a necessary prerequisite to ensure the functional correctness of a system [15]. A transaction usually means a sequence of work that the system must treat as a unit for the purpose of satisfying a request and for ensuring data integrity. Two constraints - non-interleaving and completeness - should be satisfied for transaction integrity preservation in reconfiguration. Transaction non-interleaving means that transactions belonging to the original configuration and transactions belonging to the new configuration should not interfere with each other. Transaction completeness means that a transaction should be guaranteed to complete once it starts. The optional methods to protecting transaction non-interleaving include isolation and version control. Isolation prevents transaction interleaving through avoiding the coexistence of original transactions and new transactions spatially or temporally. Version control solves this problem through assigning a version tag on every data. To ensure transaction completeness, reference counting [3] or flow tracing [16] can be used. Reference counting is known as a garbage collection algorithm mainly used in procedure-call systems where each component contains a count of the number of references to it held by other components. A component becomes removable when its reference count reaches zero and it is currently idle. Flow tracing is a method used in flow model where a component becomes removable when it is not being used or is never to be used by any flow.

Reconfiguration scheduling refers to the way that procedures are assigned priorities to run, including functional procedures and reconfiguration procedures. In functional procedures execute the functional codes of the system and in reconfiguration procedures execute the reconfiguration codes. The management is usually carried out by a scheduler. There are two typical classes of schedulers, preemptive and round-robin. A preemptive scheduler always arranges functional procedures run first, i.e. reconfiguration processes are suspended at any time a functional process arrives and are resumed after all running functional procedures are completed. A round-robin scheduler assigns time slice to each procedures in equal portion and in order, i.e. all procedures run in turn.

In distributed environment, interaction protocol determines how reconfiguration operations are synchronized among geographically distributed components. The protocol can be centralized or decentralized. In a centralized protocol, a central controller manages the reconfiguration progress. It interprets the reconfiguration plan, sends commands to components, and monitors the results of operations. In a decentralized protocol, every component knows how to behave and interact with others based on its role in reconfiguration. There is a coordinator who takes charge of the role assignment and component administration.

C. *Dependency between Design Decisions*

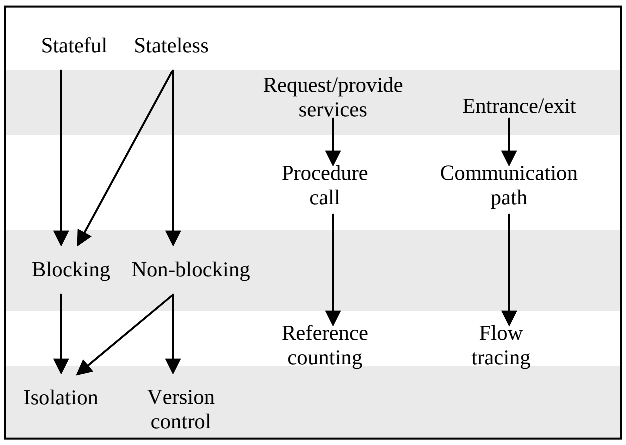


Figure 5. Dependencies between Design Decisions

There are dependencies between design decisions, which means some design decisions may become the reasons of other design decisions. these dependencies are shown in Fig.5.

If there are stateful components to be replaced in a reconfiguration, blocking algorithm should be chosen because a consistent system state is necessary for the state transfer between components. And thereby transaction non-interleaving is naturally guaranteed by isolation, i.e. transactions of new version and transactions of original version run in different period of time. If all the components to be replaced in reconfiguration are stateless, both blocking algorithm and non-blocking algorithm could be used. And if non-blocking algorithm is used, transaction interleaving could be avoided by isolation or version control.

What type of connector could be used depends on the interface that components provide. If components provide service-oriented interfaces, connectors should be procedure-calls. And correspondingly transaction completeness should be guaranteed using reference counting method. If components provide flow oriented interfaces, connectors should be communication paths and flow tracing should be used to ensure transaction completeness.

IV. Design Decisions and Influence of Dynamic Reconfiguration

Three design decisions have severe influence on the influence of dynamic reconfiguration. Reconfiguration algorithm has influence on system response time because the system may be blocked in reconfiguration. Reconfiguration scheduling also has influence on system response time because reconfiguration procedures may compete with functional procedures on CPU time. Interaction protocol for reconfiguration has influence on reconfiguration time because operations may be executed sequentially or in parallel.

To compare the influence of different design choices, all methods need to be executed under the same running environment. However, such comparison is very hard because existing implementations for these methods are embedded in different component models or middleware technologies. One solution is simulating these methods on the RDF (Reconfigurable Data Flow) model [16]. The RDF model is an extension to the widely used Data Flow Network model [2]. Several improvements make it a component model that supports dynamic reconfiguration.

Constructed on the RDF model, the structure of the DSE system is shown in Fig.6. Two structural changes are set for the reconfiguration, including replacing the message digesting algorithm and pulling in data compression/depression function. See the objects drawn with dashed line in Fig.6.

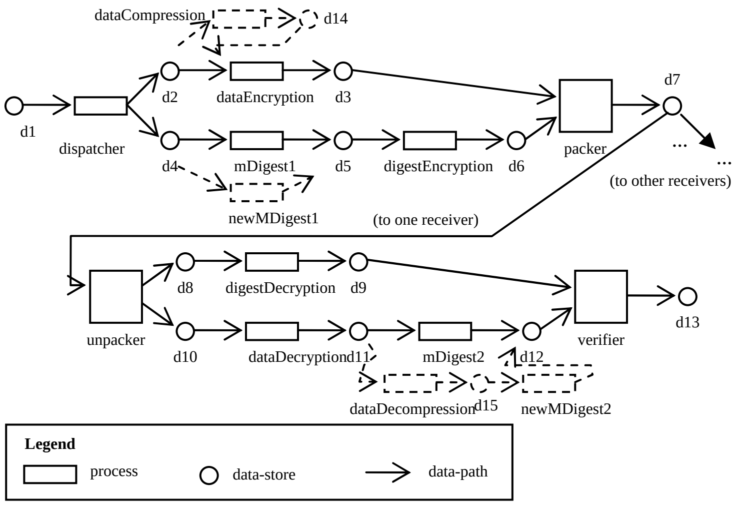


Figure 6. The DSE System Constructed on The RDF Model

A. Reconfiguration Algorithm and System Response Time

As mentioned previously, two categories of reconfiguration algorithms are currently in use. Both of them are tested on the RDF model and their influence on system QoS are compared. To exclude other factors that may influence the influence on QoS, the test is carried out with the following settings:

1) Two reconfigurations have been done to achieve the same structural change, one for the blocking algorithm and another for the non-blocking algorithm.

2) Every reconfiguration has taken place in a single node so that there is no influence from the interaction protocol.

3) Time-consuming reconfiguration operations are simulated with thread sleeping. Therefore the reconfiguration procedure does not compete with functional procedures on CPU time and there is no influence from the scheduling policy.

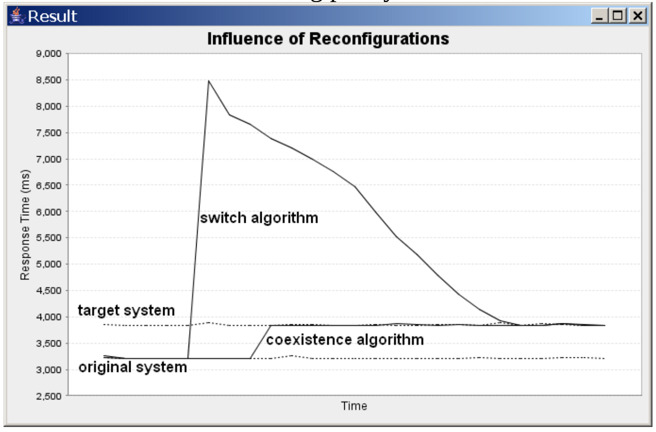


Figure 7. Influence of Different Reconfiguration Algorithms

The response time has been recorded during the system running period. See Fig.7. From the results, we can see that the system response time has an obvious increase in the test that uses the blocking algorithm. Therefore, to minimize the influence on system QoS, the non-blocking algorithm is prefered. And because the precondition for non-blocking algorithm is there is no state transfer between components, a principle for component design is every component that can be stateless should be stateless.

B. Reconfiguration Scheduling and System Response Time

To compare the QoS influence of preemptive policy and round-robin policy for reconfiguration scheduling, the following tests have been done:

1) Two reconfigurations have been done to achieve the same structural change, one for preemptive scheduling and another for round-robin scheduling. Reconfiguration operations really consume CPU time so that the influence of the scheduling policy will be reflected in the result.

2) These two reconfigurations use the same coexistence algorithm to exclude the influence of reconfiguration algorithm.

3) The reconfiguration has taken place in a single node so that there is no influence from the interaction protocol.

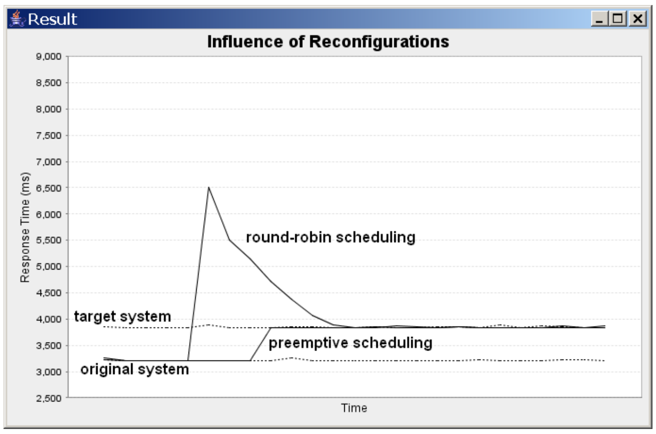


Figure 8. Influence of different scheduling policies

The result is shown in Fig.8. Using preemptive scheduling, the reconfiguration has no influence on the system response time. On the contrary, round-robin scheduling results to a severe influence on the system response time. The reason is preemptive scheduling can prevent functional procedures from being competed by the reconfiguration procedure. Therefore, preemptive scheduling is a good policy for reconfiguration scheduling, but the side effect is a longer reconfiguration time because the CPU time spent on the reconfiguration in a time unit is less than the round-robin scheduling.

C. Interaction Protocol and Reconfiguration Time

To examine the influence of centralized protocol and decentralized protocol on reconfiguration time, the following tests have been done:

1) Two series of tests have been done in a distributed environment, one for centralized protocol and another for decentralized protocol. In each series of tests, we recorded the reconfiguration time under different system sizes. Here we use different numbers of receivers to simulate different system sizes.

2) These two tests achieve the same reconfiguration with the same coexistence algorithm. This setting can exclude the influence of the reconfiguration algorithm.

3) Time-consuming reconfiguration operations are simulated with thread sleeping on each node. Therefore the influence from the scheduling policy is excluded.

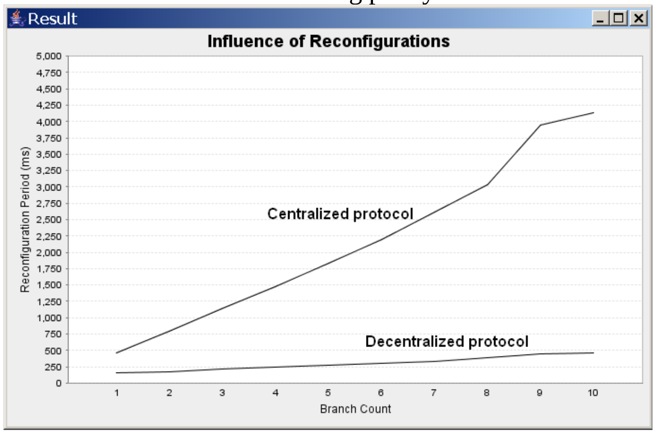


Figure 9. Influence of different interaction protocols

The result in Fig.9 shows that decentralized protocol has a much shorter reconfiguration time. This is because the reconfiguration can be carried out in a fully concurrent way in decentralized protocol. In centralized protocol, the reconfiguration operations are executed one by one sequentially. In decentralized protocol, several flows of reconfiguration operations can be executed in parallel. The disadvantage of decentralized protocol is its complication therefore it is harder to design and develop than the centralized one.

V. Conclusion and Future Works

Dynamic reconfiguration is a good solution for software systems that need to run 24 hours a day and 7 days a week and need to be updated frequently. Controlling the influence of dynamic reconfiguration is an important yet difficult work. Many factors need to be considered and various methods can be used. A systematic analysis of these factors and a comparison of theses methods can help designers in modeling dynamic reconfigurable systems.

The factors that need to be considered in designing dynamic reconfiguration have been analyzed in this paper. From the component interface, connector meaning to the reconfiguration algorithm, consistency preservation, scheduling policy, and interaction protocol, the key problems and the possible design choices for each of these factors are explained. Using the DSE system as an example and based on our RDF component model, the relationship between some design choices and the influence of dynamic reconfiguration are illustrated.

Future work will be focused on advanced tool support for analysis of dynamic reconfiguration. Besides influence, other aspects of dynamic reconfiguration such as memory usage and coding cost will also be considered. A richer analysis of the relationship between system properties and different design choices for dynamic reconfiguration will provide much aid to the reconfiguration designers.

Acknowledgment

This work was supported in part by the International Cooperation Program for Excellent Lecturers of 2011 by Shandong Provincial Education Department, P.R. China.

References

[1] E. Bruneton, e.t.al, "An open component model and its support in Java", Proc. 7th International Symposium on Component-Based Software Engineering, Edinburgh, UK, May 2004, pp.7-22.

[2] T. Demarco, Structured Analysis and System Specification, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[3] Divid F. Bacon, Perry Cheng, and V.T. Rajan. A real-time garbage collector with low overhead and consistent utilization. Proc. 30th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, ACM Press 2003, pp.285-298.

[4] J. Gorinsek, S. Van Baelen, Y. Berbers, and K. De Vlaminck, Managing quality of service during evolution using component contracts, Proc. 2nd international workshop on unanticipated software evolution, Warsaw, Poland, 2003, pp.57-62.

[5] Handte, Marcus; Schiele, Gregor; Urbanski, Stephan; Becker, Christian: Adaptation Support for Stateful Components in PCOM. Workshop on Software Architectures for Self-Organization: Beyond Ad-Hoc Networking at Pervasive 2005, München, Germany, 2005.

[6] Hillman, J., Warren, I.. Quantitative Analysis of Dynamic Reconfiguration Algorithms. Proc. of the International Conference on Design, Analysis and Simulation of Distributed (DASD) Systems, Virginia, USA (2004)

[7] Kramer J., Magee J.. The evolving philosophers problem: Dynamic change management. IEEE Transactions on Software Engineering, 16,11 (1990),1293-1306

[8] Linhai Cui, A Novel Approach to Hardware/Software Partitioning for Reconfigurable Embedded Systems, Journal of Computers, Vol 7, No 10 (2012), pp.2518-2525.

[9] D.C. Luckham et al, "Specification and analysis of software architecture using Rapide", IEEE Transactions on Software Engineering, 21(4), April 1995, pp.336-355.

[10] N. Medvidovic and R. N. Taylor, A classification and comparison framework for software architecture description languages. IEEE Trans. on Software Engineering, 26(1), 2000, 70–93.

[11] Paul Laird and Stephen Barrett, Towards Dynamic Evolution of Domain Specific Languages, Lecture Notes in Computer Science, 2010, Volume 5969/2010, 144-153.

[12] Peng Xiao, Zhigang Hu, Workload-aware Reliability Evaluation Model in Grid Computing, Journal of Computers, Vol 7, No 1 (2012), pp.141-146.

[13] F. Plasil, D. Balek, and R. Janecek, "SOFA/DCUP: Architecture for component trading and dynamic updating", Proc. International Conference on Configurable Distributed Systems, Annapolis, Maryland, USA, 1998, pp.43–52.

[14] Santambrogio M.D., Hoffmann H., Eastep J., Agarwal A., Enabling technologies for self-aware adaptive systems, 2010 NASA/ESA Conference on Adaptive Hardware and Systems, June 2010, Anaheim, CA, pp.149-156.

[15] Ian Warren B.Sc. A Model for Dynamic Configuration which Preserves Application Integrity, PhD thesis, Lancaster University, UK. 2000.

[16] Wei Li, Zhikun Zhao, Influence Control for Dynamic Reconfiguration of Dataflow Systems, Journal of Software, 2007(12).

[17] Ye Du, Jiqiang Liu, Ruhui Zhang, Jieyuan Li, A Dynamic Security Mechanism for Web Services Based on NDIS Intermediate Drivers, Journal of Computers, Vol 6, No 10 (2011), pp.2021-2028.

Zhikun Zhao was born in Qingzhou, Shandong province, China in 1975. He received his Ph.D. degree on computer software theory from the Graduate University of Chinese Academy of Sciences, Beijing, China in 2003.

He was an Associate Professor of the Graduate University of Chinese Academy of Sciences from 2003 to 2005. He worked as a Postdoctoral Research Fellow of Central Queensland University from 2006 to 2008. Currently he is an Associate Professor of Shandong University of Finance and Economics in Jinan, Shandong province, China. His research interests include dynamic software reconfiguration and multi-agent systems.