Dynamic Reconfiguration Planning with Influence Control

Zhikun Zhao and Wei Li
School of Computing Sciences, Central Queensland University
{z.zhao, w.li}@cqu.edu.au

Abstract

The automated way of dynamic reconfiguration has many advantages comparing with the manual way. But the existing approaches to automated dynamic reconfiguration are not integrated with the methods for influence control. In this paper, we propose an approach to dynamic reconfiguration planning with build-in influence control. A Reconfigurable Data Flow model is designed as the underlying model. Given a specification of the source configuration and the target configuration, a reconfiguration plan can be automatically generated by a planning algorithm. In the plan, new components are started before the removal of the original components in order to theoretically avoid any interruption to the system services. A version control mechanism is used to support the coexistence of original versioned transactions and new versioned transactions without interference. A flow tracing method is used to ensure the transaction completeness.

1. Introduction

Dynamic reconfiguration aims at runtime change of the architecture of software systems without interruption to the services they provide. Many efforts have been made on the underlying component model in support of dynamic reconfiguration [2][11][12][15]. But most of these models require administrators to write reconfiguration *plans* manually. This manual way of planning has several disadvantages. First, a human-written plan is usually error-prone and a single error in the plan is enough to put a system into an inconsistent state. Second, as system size increases, reconfiguration plan may become too complicated for the administrator to handle. Third, intelligent self-adaptation is difficult to achieve without an automated way of reconfiguration planning [14]. Therefore, it is necessary to find an approach to planning dynamic reconfiguration in a fully automated manner.

Some researchers have explored the automated methods for dynamic reconfiguration [1][4][8]. However, these methods are weak in controlling the *influence* of dynamic reconfiguration. Some other researchers have proposed methods for controlling the influence of dynamic reconfiguration on a system's functionality [5][6][17][18] and QoS (Quality of Service)[7][9][13]. But these method have not been automated, therefore many factors need to be concerned by the administrator in a reconfiguration.

In this paper, a novel approach is proposed to planning dynamic reconfiguration automatically with influence control incorporated. In our model to change the architecture of a system at runtime, an administrator only needs to specify the target configuration in an abstract specification. The system is able to automatically generate the reconfiguration plan by comparing its current configuration with the target configuration. Most importantly, the generated plan has three features on influence control. First, new components are started before the removal of original components to theoretically avoid any interruption to the system services. Second, a *version control mechanism* is introduced to support original versioned transactions and new versioned transactions running in parallel without interference. Third, a *flow trancing method* is used to ensure the *transaction completeness*.

2. Key issues and related works

2.1. Influence control

A dynamic reconfiguration may have two types of influence on a system, *functional influence* and *performance influence* [20]. The functional influence of a reconfiguration r can be represented as $F \rightarrow F'$, where F and F' are the functionalities of the system before r and after r. If r causes the system output neither $F(e)$ nor $F'(e)$ for an input e, r has *functional side effect*. Most reconfigurations have functional influence because changing functionality is one of the

0-7695-2841-4/07 $25.00 © 2007 IEEE

purposes for reconfiguration. But a reconfiguration should not have functional side effects.

Performance influence is the change on system QoS caused by reconfiguration. It can be reflected by the change on the response time or the throughput of the system. During a dynamic reconfiguration, a system undergoes a sequence of interim states. And the system very likely has different QoS features under these different states..

In the approaches that can eliminate the functional side effect of dynamic reconfiguration, various terms have been used, such as a reconfiguration that preserves the integrity of applications [17], a safe adaptation [18], a correct reconfiguration [5], or a consistent configuration [6]. To control the performance influence of dynamic reconfiguration, many approaches have been proposed, including component contracts [7], Mitchell's approach for multimedia system updating [13], and Hillman's OpenRec framework [9]. However, all these approaches for influence control have not been automated; therefore their efficiency depends on the administrators' capability on reconfiguration plan designing.

To control the influence of dynamic reconfiguration, we believe the following two principles should be applied to reconfiguration planning.

Principle 1. A reconfiguration should avoid *freezing* any part of the system because freezing leads to an interruption of system services. To do so, new components must be started before the removal of original components.

Principle 2. *Transaction completeness* should be preserved during reconfiguration. Once a transaction starts, the components it uses should keep working and the connectors between them should not be changed until it completes.

2.2. Underlying component model

Due to component's modularity, well-defined interface, and interconnection independence [17], component-based development has been a widely used technique for building reconfigurable software systems for over ten years [16]. For a component-based system, dynamic reconfiguration is achieved through manipulating components/connectors.

Many component models have been proposed to support dynamic reconfiguration, such as SOFA [15], Fractal [2], Darwin [12], and Rapide [11]. But the problem of how to write reconfiguration plans and how to manage the influence has been left to the system administrator.

The underlying component model should have the following properties in support of dynamic reconfiguration with influence control.

Property 1. A synchronization mechanism should be provided to prevent inter-component communications from being interrupted by reconfiguration operations.

Property 2. To ensure transaction completeness, a transaction tracing method should be provided to detect whether a component is being used or possible to be used in future by some transactions.

Property 3. A version control mechanism should be provided to allow the coexistence of original and new versions of transactions without interference.

Property 4. The programming framework for components should separate the reconfiguration codes from the functional codes for the purpose of Separation of Concerns (SoC) [10].

2.3. Reconfiguration specification and plan generation

For a system under current configuration C, an administrator can design a new configuration C' to meet the new requirements. Then $<C,C'>$ is a declarative specification of the reconfiguration as it only describes what change need to be done. Here configuration is the snapshot of system structure, which records the components running in the system and the connectors between them.

A declarative specification needs to be changed into an operational plan, which describes how to achieve the change step by step. An operational plan can be manually written by an administrator or automatically generated by a planner. The automated way has obvious advantages of error-free, no need of human intervention, and no limitation on the problem size. But the most challenging problem is how to integrate the influence control into the planning.

The existing approaches to automated dynamic reconfiguration include Chien's AI planning approach [4], Arshad's temporal planning techniques [1], and Hicks's dynamic patches [8]. However, the influence control mechanisms for reconfiguration have not been integrated in these methods.

To minimize the influence possibly caused by reconfiguration, the following criteria should be integrated into the plan generation.

Criteria 1. A plan should consist of two main sequential steps: 1) start the new components; 2) remove the original components.

Criteria 2. A plan should set components to proper version modes to make original and new versions of transactions coexist without interference.

Criteria 3. Before removing a component, a plan should ensure that it is not being used and will never be used by any transactions.

3. Dynamic reconfiguration planning

3.1. The reconfigurable data flow model

We have developed a Reconfigurable Data Flow (RDF) model, which can be used as the underlying component model for dynamic reconfiguration. It is an extension to the conceptual Data Flow Network model [3]. The elements of the RDF model are *process*, *data-store*, and *data-path*. A process is a software component that consumes data through its *entrances* and produces data through its *exits*. A data-store is a random-accessible data container with infinite capacity. A data-path is a connector between a process and a datastore through which data can flow. The reconfiguration operations of the RDF model include *addition* and *removal* of processes, data-stores and data-paths.

Graphically, a RDF model could be represented as a bi-partite directed graph. Fig.1 shows an example (the part marked with solid lines), a Digital Signature and Encryption (DSE) system constructed on the RDF model. In the system, the data elements in data-store *d1* will be delivered to the corresponding receivers after being digitally signed and encrypted. There are three pairs of processes that have dependency relationships: 1) *dataEncryption* encrypts data elements with the receiver's public key and *dataDecryption* decrypts data elements with the receiver's private key; 2) *mDigest1* and *mDigest2* perform the same functions, message-digesting; 3) *digestEncryption* encrypts the digest with the sender's private key and *digestDecryption* decrypts the digest with the sender's public key.

The RDF model supports all the four properties for dynamic reconfiguration described in section 2.2.

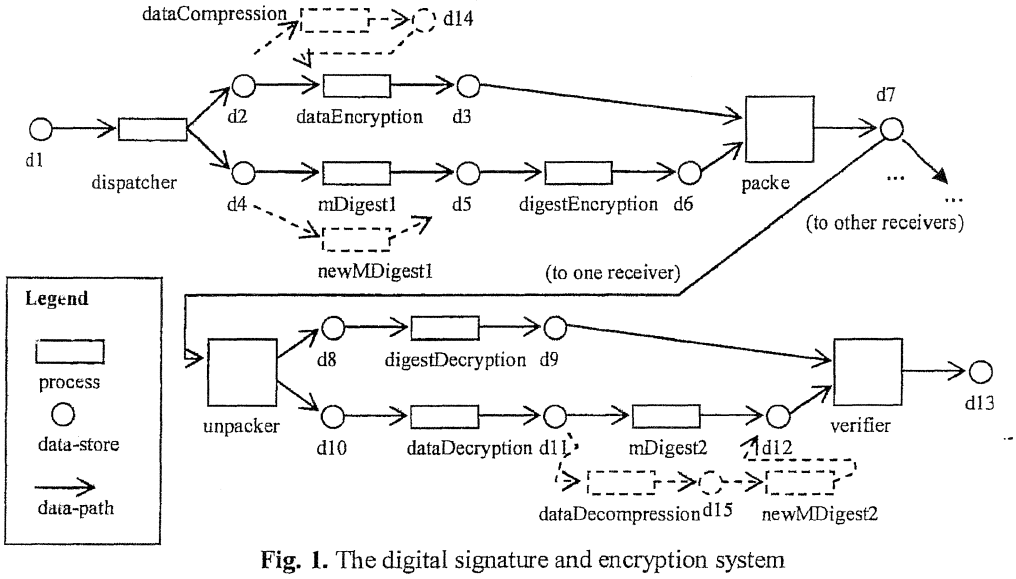

Fig. 1. The digital signature and encryption system

3.2. The reconfiguration algorithm

We design a planning algorithm for reconfiguration, which can generate reconfiguration plan with build-in influence control automatically. The input of the algorithm is $<R,R'>$, where R is the route map of the current system and R' is the route map of the target system. Following the discussion in section 2.3, the algorithm consists of two main sequential stages: 1) establish the routes in R'-R; 2) remove the routes in R-R'.

Between the two stages, there is a period for these routes coexisting in the system. Suppose $r \in R$-R' is a route to be removed and $r' \in R'$-R is a route to be added. If there is an intersection between them and there is a separation before the intersection and another separation after (Fig.2-a), the original versioned transaction supported by r and the new versioned transaction supported by r' may interfere with each other. The involved segment of r or r' is the part between the nearest splitting point before the intersection and the nearest joining point after the intersection. If there are several such intersections on a route, the involved segment should cover all these intersections. The version control mechanism should be used to prevent the interference. Suppose all the

data elements flowing in the original system are of version *m*. We can set the data elements flowing on the involved segments of *r* to version *m-1* and those on *r'* to version *m+1*.

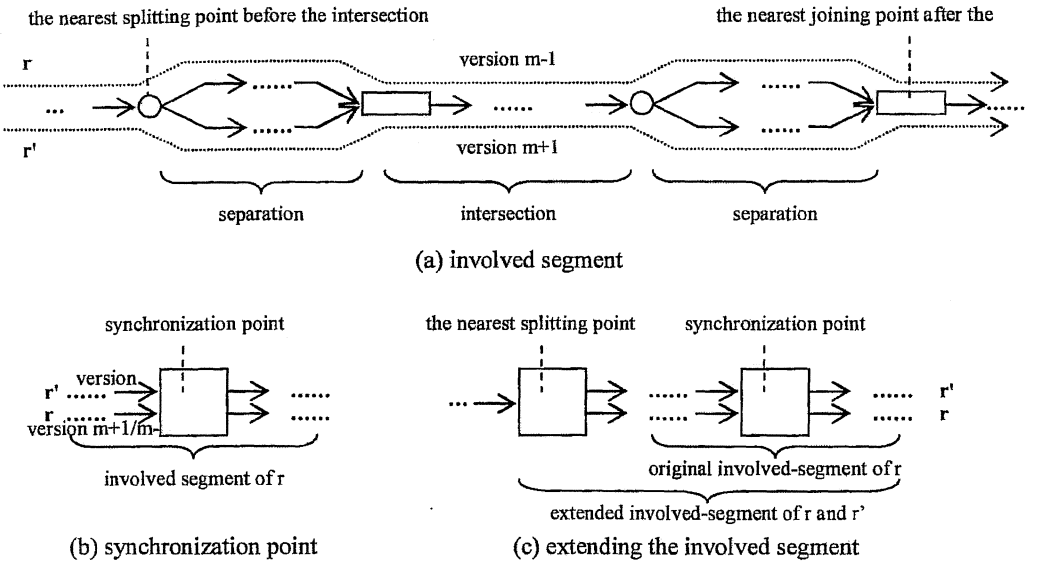

Fig. 2. Interference between routes

Suppose *p* is the first process of a segment and *q* is the last, to change the data elements flowing on the segment from version *x* to version *y*, the following progress can be followed. Suppose all the processes on the route run in *strict(x,x)* mode at the beginning. The first step changes *q* to *filter(x)* mode and the processes between *p* and *q* to *transparent* mode to allow version *x* and version *y* data elements coexisting on the segment. The second step changes *p* to *strict(x,y)* mode to change the data elements flowing into the segment to version *y*. The third step uses the tracing method to wait for all version *x* data elements to flow out of the segment to ensure all version *x* transactions being completed. The fourth step changes *q* to *strict(y,x)* mode and the processes between *p* and *q* to *strict(y,y)* mode.

Changing the version of the data elements flowing on a segment may affect the routes that have *synchronization points* with the segment. A synchronization point is a process that has multiple entrances or exits, where two routes intersect and need to be synchronized (Fig.2-b). The data elements flowing on the two routes should be changed to the same version together; otherwise, different versions of data elements can not pass through the synchronization point because different versions of data elements can not be consumed in a single firing. Therefore, the involved segment of the route should be extended. It starts from the nearest splitting point of these routes (Fig.2-c). If these routes are not originated from one point, the route map has a synchronization problem, which is a designing defect of the route map.

With the supports of the version control mechanism, the transaction tracing mechanism, and the above calculation, the reconfiguration algorithm is as follows:

Reconfiguration algorithm

Parameters:

R: route map of the current system

R': route map of the target system

The reconfiguration progress:

1. Calculate the involved segment for every route.
2. Change the data elements flowing on the involved segments of the routes in *R-R'* to version *m-1*.
3. Establish the routes in *R'-R* and set the data elements flowing on the involved segments to version *m+1*.
4. Wait for all version *m-1* data elements to flow out of the involved segments of the routes in *R-R'* and then remove the routes.
5. Change the data elements flowing on the involved segments of the routes in *R'-R* to version *m*.
6. Change all the processes in the system to *strict(m,m)* mode.

4. A case study

We use the DSE system as a case to examine our reconfiguration algorithm. Two objectives are set for the reconfiguration, replacing the message digesting algorithm with a new one and adding a data compression/depression function (the part marked with dashed line in Fig.1). Although the DSE system may have multiple branches, the reconfiguration planning for one branch is enough to demonstrate our algorithm. The input parameters of the algorithm are as follows:

Route map of the original system is $R=\{r1, r2\}$, where

r1=[d1,<1,dispatcher,1>,d2,<1,dataEncryption,1>,d3,<1,packer,1>,d7,<1,unpacker,2>,d10,<1,dataDecryption,1>,d11,<1,mDigest2,1>,d12,<2,verifier,1>,d13],

(It means a data element in d1 can flow into dispatcher through entrance 1, flow out dispatcher through exit 1, and then flow into d2, and so on....)

r2=[d1,<1,dispatcher,1>,d4,<1,mDigest1,1>,d5,<1,digestEncryption,1>,d6,<2,packer,1>,d7,<1,unpacker,1>,d8,<1,digestDecryption,1>,d9,<1,verifier,1>,d13],

Route map of the target system is $R'=\{r3, r4\}$ (new processes and data-stores are bolded),where

*r3=[d1,<1,dispatcher,1>,d2,<1,**dataCompression,1**>,**d14**,<1,dataEncryption,1>,d3,<1,packer,1>,d7,<1,unpacker,2>,d10,<1,dataDecryption,1>,d11,<**1,dataDecompression,1**>,**d15**,<**1,newMDigest2,1**>,d12,<2,verifier,1>,d13]*

*r4=[d1,<1,dispatcher,1>,d4,<**1,newMDigest1,1**>,d5,<1,digestEncryption,1>,d6,<2,packer,1>,d7,<1,unpacker,1>,d8,<1,digestDecryption,1>,d9,<1,verifier,1>,d13]*

The involved segment of each route is the part between *dispatcher* and *verifier*. And the generated reconfiguration plan is as follows:

// Step 1. change the data elements flowing on the involved segments of r1 and r2 to version 0
set verifier to *filter(1)* mode;
set dataEncryption, packer, unpacker, digestDecryption, mDigest1, digestEncryption, dataDecryption, mDigest2 to *transparent* mode;
set dispatcher to *strict(1,0)* mode;
wait for all version 1 data elements to flow out of the involved segments of r1 and r2;
set mDigest1, mDigest2 to *strict(0,0)* mode;
// Step 2. establish routes r3 and r4
start dataCompression, dataDecompression, newMDigest1, newMDigest2 and set them to *strict(2,2)* mode;
start data-stores d14, d15;
set up data paths (newMDigest2,0→d12), (d15→0,newMDigest2), (dataDecompression,0→d15), (d11→0,dataDecompression), (d14→0,dataEncryption), (dataCompression,0→d14), (d2→0,dataCompression), (newMDigest1,0→d5), (d4→0,newMDigest1);
set dispatcher to *strict(1,2)* mode;
// Step 3. remove routes r1 and r2
wait for all version 0 data elements to flow out of the involved segments of r1 and r2;
remove data paths (d2→0,dataEecryption), (d11→0,mDigest2), (mDigest2,0→d12), (d4→0,mDigest1), (mDigest1,0→d5);
remove processes mDigest1, mDigest2;
// Step 4. change the data elements flowing on the involved segments of r3 and r4 to version 1
set dataCompression, dataDecompression, newMDigest1, newMDigest2 to *transparent* mode;
set dispatcher to *strict(1,1)* mode;
wait for all version 2 data elements to flow out of the involved segments of r3 and r4;
set all processes to *strict(1,1)* mode;

To examine the influence of the plan, we build a simulating system based on our RDF model. In the experiment, we feed data elements to the system in a fixed frequency. And the following method is used to detect the influence of the reconfiguration:

1) The route that every data element has passed is compared with the predefined route to find any possible functional side effect.

2) The response time, i.e. the interval between the instance at which a data element enters into *d1* and the instance at which the data element reaches *d13* is recorded as the QoS parameter of the system.

We do the following three tests:

Test 1 simulates a reconfiguration without built-in influence control. In the test, we execute the reconfiguration after 'freezing' the system.

Test 2 simulates a reconfiguration with build-in influence control but without scheduling support [19]. In the test, we execute our reconfiguration plan under the *Round-Robin* scheduling. Under the scheduling, the reconfiguration procedure can compete for CPU time with the functional procedures.

Test 3 simulates a reconfiguration with both build-in influence control and scheduling support. In the test, we execute our plan under *Preemptive* scheduling. Under the scheduling, the reconfiguration procedure must give up CPU once a functional procedure is ready to run. Therefore, the reconfiguration procedure has no influence on functional procedures because it is scheduled to use the spare CPU time only.

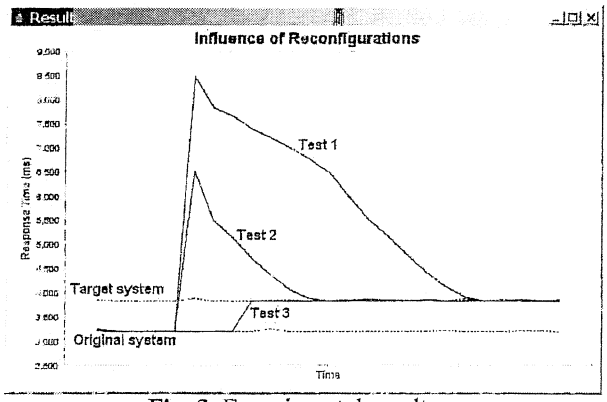

Fig. 3. Experimental results

From the experimental results, we come to the following conclusions. First, every data element has passed one of the predefined routes, *r1*, *r2*, *r3*, or *r4*. It proves that the reconfiguration has no functional side effect. Second, the plan generated by our algorithm has built-in influence control. The line-chart for system response time in the tests is given in Fig.3. Our reconfiguration method theoretically has no influence on system QoS. On the contrary, the method without build-in influence control or without scheduling support causes a significant impact on the system QoS.

5. Conclusion and future work

An approach to automated planning of dynamic reconfiguration is proposed in this paper. Using the approach, an administrator only needs to give the configuration of the target system. A planning algorithm can automatically generate the reconfiguration plan with build-in influence control. In the generated plan, the interruption to a system's services is theoretically eliminated by starting the new components before the removals of the original ones. With the route map representation of the system's configuration and transactions, a version control mechanism is used to prevent the new versioned transactions and the original ones from interfering with each other. And a transaction tracing method is used to ensure the transaction completeness.

The approach can be further improved from two aspects. First, the representation of routes, the tracing mechanism, and the algorithm could be extended to be able to handle cycles in the route map. Second, the centralized way of plan execution currently adopted by the approach can be changed to a decentralized way. The decentralized way of reconfiguration will be more efficient than the centralized way in a fully distributed environment.

Acknowledgements

This work was supported by Central Queensland University, Australia under Research Advancement Awards Scheme (RAAS) grants, 2006~2007.

References

[1] N. Arshad, et al., "Deployment and Dynamic Reconfiguration Planning for Distributed Software Systems", Proc. 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03), Sacramento, USA, 2003, pp.39-46.

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

[3] G. Cheng, *A Dataflow-Based Software Integration Model in Parallel and Distributed Computing and Applications*, Ph.D. thesis, Syracuse University, Italy, 1997.

[4] S.A. Chien,et al., "Using Artificial Intelligence Planning Techniques to Automatically Reconfigure Software Modules", Proc. 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Tasks and Methods in Applied Artificial Intelligence, Benicássim, Spain, 1998, pp.415-426.

[5] L. Desmet, et al., "Towards Preserving Correctness in Self-Managed Software Systems", Proc. the Workshop on Self-Managing Systems (WOSS'04), Newport Beach, USA, 2004, pp.34-38.

[6] P. Feile and J. Li, "Consistency in Dynamic Reconfiguration", Proc. 4th International Conference on Configurable Distributed Systems, Annapolis, USA, 1998, pp.189-196.

[7] J. Gorinsek, et al., "Managing quality of service during evolution using component contracts", Proc. 2nd international workshop on unanticipated software evolution, Warsaw, Poland, 2003, pp.57-62.

[8] Michael Hicks and Scott Nettles, "Dynamic Software Updating", ACM Transactions on Programming Languages and Systems, 27-6, 2005, pp.1049-1096.

[9] J. Hillman, and I. Warren, "An Open Framework for Dynamic Reconfiguration", Proc. 2004 International Conference on software Engineering (ICSE), Edinburgh, UK, 2004, pp.23-28.

[10] G. Kiczales, et al., "Aspect-Oriented Programming", ACM Computing Surveys(CSUR), 28(4), 1996.

[11] 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.

[12] J. Magee, J. Kramer, "Dynamic structure in software architectures", Proc. 4th ACM SIGSOFT Symposium on Foundations of Software Engineering, San Francisco, USA, Oct 1996, pp.3-14.

[13] S.R.Mitchell, *Dynamic Configuration of Distributed Multimedia Components*, Ph.D. thesis, University of London, 2000.

[14] P. Oreizy, et al., "An Architecture-Based Approach to Self-Adaptive Software", IEEE Intelligent Systems, vol.14, no.3, 1999, pp. 54-62.

[15] 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.

[16] C. Szyperski. Component software: beyond object oriented programming, 2nd edition, Addison-Wesley, 2002.

[17] I. Warren, *A Model for Dynamic Configuration which Preserves Application Integrity*, PhD thesis, Lancaster University, UK. 2000.

[18] J. Zhang, et al., "Adding safeness to dynamic adaptation techniques", Proc. ICSE Workshop on Architecting Dependable Systems, Edinburgh, Scotland, 2004, pp.17-21.

[19] Z.K. Zhao and W. Li, "Dynamic Reconfiguration with QoS Management", Proc. International Conference on Software Engineering and Applications. Dallas, USA. 2006, pp.387-392.

[20] Z.K. Zhao and W. Li, "Influence Control for Dynamic Reconfiguration", to be appeared in Proc. 18th Australian Conference on Software Engineering, Melbourne, Australia, 2007.