Influence Control for Dynamic Reconfiguration of Data Flow Systems

Wei Li
School of Computing Sciences, Central Queensland University, Rockhampton, Australia
Email: w.li@cqu.edu.au

Zhikun Zhao
School of Computing Sciences, Central Queensland University, Rockhampton, Australia
Email: z.zhao@cqu.edu.au

Abstract—**Influence control is a very challenging issue in dynamic reconfiguration and still not well addressed in the literature. This paper argues that dynamic reconfiguration influences system execution in four ways: functional update, functional side-effect, logical influence on performance and physical influence on performance. Methods including version control, transaction tracing, switching reconfiguration plan, and reconfiguration scheduling have been proposed for controlling the influence. These methods are integrated into the Reconfigurable Data Flow (RDF) model, which is designed to support the dynamic reconfiguration of stateless data flow systems. The RDF platform is an implementation of the RDF model on Java platform. The RDF platform is implemented as an open frame for different reconfiguration planning algorithms and scheduling policies to be simulated and their influence to be quantitatively compared on a single platform. Using the Data Encryption and Digital Signature System as a case study, tests have been done on the RDF platform to examine the influence of different reconfiguration planning algorithms and scheduling policies. Experimental results show that the methods proposed in this paper is effective in controlling the influence of dynamic reconfigurations.**

Index Terms—**dynamic reconfiguration, influence control, data flow**

I. Introduction

Currently, Dynamic Reconfiguration (DR) i.e. changing a system from one configuration to another at run-time [11] is becoming a necessary feature of software. This trend originates from two facts: 1) increasing need for systems to be online 24-hour daily as Internet usage rapid increases and 2) continuous update is one of the inherent properties of software. DR technique makes it possible to evolve a system without disruption.

Although DR does not need a system to shut down, it does have some influence on the running of the system. Typically, the system will inevitably suffer a performance decline during the reconfiguration period due to the necessity of the DR performing blocks to some parts of the system [17]. This also involves processor time to execute the reconfiguration operations [6]. A DR will lose its benefits if it causes severe impact on the system performance because it has no essential difference with a Static Reconfiguration (SR, reconfiguration after shutdown) from an end user's point of view. An end user will encounter the *unavailability* of a system if the system is going through a SR while encountering the *no-response* of a system if the system is going through a DR with severe performance decline. In comparison, the no-response state has no significant advantage over the unavailability state. Therefore, influence control, i.e. the ability to preserve system correctness and maintain system performance (throughput or response time), is one of the most important aspects that have to be considered in designing a DR.

This paper will discuss the ideal way of DR, which satisfies four constraints: 1) *Correctness-preservation*. A DR should preserve the correctness of the system functionality while achieving the appropriate change. 2) *Performance-maintenance*. A DR should restrict its impact on the system performance to zero if possible or to an extent that is acceptable to end users. 3) *Minimal-application-contribution*. The DR mechanisms should be ideally transparent to component developers. 4) *Automation*. A DR should operate automatically where the administrator only needs to input the target configuration and the system will complete the remaining requirements including reconfiguration planning and execution.

However, such an ideal way of DR does not exist in the literature. Although many DR models [2][5][8][9][15] have provided methods to correctness preservation, only a few [12][14] are concerned about the problem of performance-maintenance. And even fewer [14] have methods that can maintain the system performance to some extent. Finally, there is not a model that has integrated the methods to correctness preservation and performance maintenance into a single cohesive model and at the same time provided automation and the framework that requires minimal-application-contribution.

The most difficult problem in pursuing an ideal DR is the aspect of performance-maintenance. Typically DR models are based on software components. Software

component is suitable to be the basic entity in reconfiguration due to its modularity, well-defined interface and inter-connection dependency [16]. In a general DR scenario, to guarantee the correctness of the system functionality, the system has to be blocked and driven into a *safe* state before reconfiguration operations can be executed. In such a state, reconfiguration operations will not interrupt interactions between components or interfere with the constraint relationships between components. Since the blocking operation will have an influence on the system performance, researchers are striving to restrict the influenced area and period. But the influenced area and period are application relevant; therefore, there is theoretically a minimum influence for each general DR and it cannot be minimized further.

Since the ideal DR is not always available in general DR scenarios, this paper concentrates on whether it is available in restricted scenarios and how it can be achieved if available. The stateless dataflow model is researched in this paper for the goal, and the four characteristics are unified into the Reconfigurable Data Flow (RDF) model to demonstrate the feasibility of the proposed mechanisms for influence control. This paper is based on the previous work in [18].

This paper is organized as follows. Section II overviews the related DR models and compares their works on influence control. Section III discusses how DR influences the running of a system, how the influence can be eliminated or restricted, and what constraints the DR scenario should satisfy for the influence control. Section IV presents the RDF model to demonstrate the workability of the methods for influence control proposed in section III. The implementation of the RDF model and the experimental results, based on the case study of the *Data Encryption and Digital Signature System (DEDSS)* are also given in this section. Finally section V concludes the paper and points out the future works.

II. Related Works

Since Kramer and Magee's early work [8], many models have been proposed for DR. Several representative models are selected for review in this section. This includes the following models: Kramer and Magee [8], SOFA 2 [3], Mitchell [12], OpenRec [7], and Naveed [14].

These models are compared from several aspects, including correctness preservation, performance maintenance, application contribution, automation, along with application scenario and openness of implementation. 1) Application scenario reflects the applicability of the models. 2) Correctness preservation and performance maintenance are the two aspects for influence control. 3) Application contribution and automation indicate the amount of human intervention needed by the models in DR. 4) Openness of implementation means whether the implementation is *open* i.e. other algorithms/methods are easy to be integrated into and compared within the same platform, or conversely, *closed*.

The characteristics of the models are listed in TABLE I. Kramer and Magee's model is a general scenario model. The problems relevant to correctness preservation are well addressed in the model, such as structural integrity [16], global consistency [15] and state transfer [17]. These problems are solved by using a blocking mechanism to drive the system into a safe state before applying reconfiguration operations. The application contribution it requires is minimal. The drawbacks are: the system performance is not maintained because of the blocking; the reconfiguration progress is not automated; and its implementation is closed. SOFA 2 uses several patterns to assure the correctness of reconfiguration operations, but it is hard to determine that it can preserve the correctness as the problem of global consistency is not considered. Mitchell proposed a special DR model for codec chain replacement in multimedia systems. In this special application scenario, global inconsistency never occurs because a DR always replaces an entire codec chain with a new one. Performance is maintained by establishing the new code chain before removing the old one. However, the application scenario of Mitchell's model is too restricted and the DR progress is not automated. Hillman presented the OpenRec framework for the purpose of comparing different reconfiguration algorithms quantitatively on a single platform. It can plan a DR automatically using the reconfiguration algorithms provided by the users. It is an open platform but itself lacks the supports for the algorithms for correctness

TABLE I Comparison of The DR Models

Characteristics / DR Models	Application Scenario	Correctness	Performance	Application Contribution	Automation	Implementation
Kramer's model	General	Preserved	-	Minimum	-	Conic, Darwin -Closed
SOFA 2	General	Global consistency is not considered	-	Minimum	-	SOFA 2 -Closed
Mitchell's model	Codec chain replacement	Global inconsistency is supposed to not occur	Maintained	Minimum	-	DJINN -Closed
OpenRec	General	-	-	-	Planning	OpenRec -Open
Naveed's model	General	-	-Considered but not maintained	Minimum	Planning	Planit -Closed
RDF	Stateless data flow	Preserved	Maintained	Minimum	Planning / Execution	RDF Platform -Open

"-" means the corresponding aspect is not mentioned in the literature.

preservation and performance maintenance. Naveed's model applies AI planning technique to DR. It achieves an automated DR but does not provide solution to the correctness preservation and performance maintenance.

Among these models, only Mitchell's model can maintain system performance in DR. However, it is restricted to a very special application scenario, in which global consistency never occurs. Other models are general DR scenario models but none can maintain the system performance in DR. Therefore, the aim of this paper is to design a DR model that can be applied to a certain range of application scenarios and at the same time cares for correctness preservation, performance maintenance, minimum application contribution, automation and open implementation. Based on this consideration, the Data Flow model is chosen as the underlying component model to design the RDF. The characteristics of the RDF are also listed in TABLE I for comparison. The methods/mechanisms that support these characteristics of the RDF will be explained in the following sections.

To compare the positions of these DR models in the study on influence control for DR, a coordinate system can be drawn as Fig.1. The X axis represents the application scenario and the Y axis represents the degree of performance maintenance. Karmer's model, SOFA 2 and OpenRec locate at the lower right because they are general models but do not consider the performance maintenance. Naveed's model is located at the middle right as it is a general model and the performance maintenance is considered but not solved. Mitchell's model and the RDF model locate at the top as both are able to maintain the performance during DR. The RDF is located to the right of Mitchell's model as it has a wider application scenario.

III. Influence Control for Dynamic Reconfiguration

A. The Four Ways That Dynamic Reconfiguration May Influence System Execution

The influence to be discussed is the effect caused by DR on the overall system and just not the effect on a single component or a subsystem. As overall system performance is the direct indicator that an end user encounters, this performance should be the key indicator to assess the influence of a DR on the system execution.

A DR may influence a system in four ways: *functional update*, *functional side-effect*, *logical influence on performance*, and *physical influence on performance*.

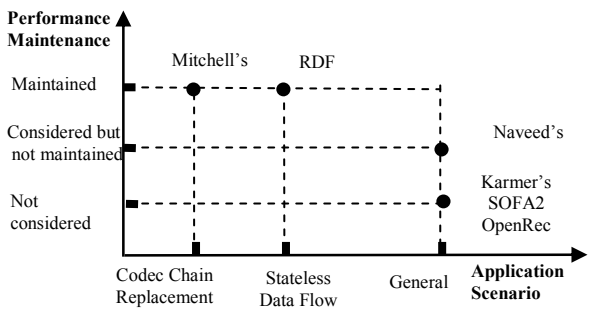


Figure 1. The positions of the DR models.

Functional update and functional side-effect are the influence of DR on system functionality and they are relevant to the system correctness. If the purpose of a DR is to upgrade the functionality, the end users will see these new functions after the DR. This is the influence termed as functional update, which is what the DR designers expect. However, changing the configuration of a running system is usually 'unsafe' as the reconfiguration may interfere with the ongoing interactions and transactions in the system. Even if the system works correctly under the original configuration and the target configuration, it may generate incorrect results during the reconfiguration. This is termed as the functional side-effect of DR.

Along with the configuration evolving from phase to phase, a change will typically affect the system performance. The performance indicators considered in this paper are *response time* (the period between receiving the input and producing the output) and *throughput* (the amount of data items handled per time interval). A DR may influence the system performance in two ways. The first is logical influence, which is the theoretical influence that may be caused by a DR. Usually a DR needs to perform a reconfiguration plan, which consists of a sequence of atomic operations. Each operation may change the system from one configuration to another, the system undergoing a sequence of interim configurations before subsequently reaching its goal. During this time, it is very likely for the system to have different performance characteristics under these different configurations. The second is the physical influence, which can be caused by the execution overhead of a DR. The execution of a DR itself also needs some processor time, and therefore, the DR procedure may compete with the system's ongoing functional procedures for the processor time. If the computational capability of the hardware is limited, the reconfiguration procedure consuming some processor time means the functional procedures loses some processor time, i.e. a decline in performance. As this competition is likely to take place, the execution of a DR may have in actual influence taking place even if its reconfiguration plan has no logical influence in theory.

According to the above analysis, influence control for DR includes three aspects: 1) avoiding functional side-effect, 2) minimizing logical influence on performance, and 3) restricting physical influence on performance.

For further study on to what extent a DR may influence a system, formal definitions are given below. Using these definitions, the influence of a DR can be estimated or measured quantitatively.

Definition 1. The functionality of a system s is represented as a function F, which means s outputs $F(e)$ for an input e. F can be stateful or stateless.

Definition 2. The functional update by a DR r is defined as $F_{origin} \rightarrow F_{target}$, where F_{origin} is the system functionality before r and F_{target} is the one after r.

Definition 3. A DR r with functional update $F_{origin} \rightarrow F_{target}$ has functional side-effect if and only if r causes the system output neither $F_{origin}(e)$ nor $F_{target}(e)$ for an input e.

Definition 4. The performance indicators of a system include the response time and the throughput.

Definition 5. The influence of a DR on system performance is a set

$$\{ iop \mid iop = | p - p_{origin} |, p \in P_{reconfig} \}$$

where p_{origin} is the performance indicator of the original system and $P_{reconfig}$ is the set of performance indicators the system undergoes in the DR.

Although functional correctness is the precondition for DR to be meaningful, the requirement for minimizing logical influence on performance has the determinative effect on the design of DR models. For this reason, minimizing logical influence on performance will be discussed first, and then avoiding functional side-effect and finally restricting physical influence on performance will be discussed thereafter.

B. Minimizing Logical Influence on Performance

To control the logical influence on performance requires the theoretical performance indicator of the system to be maintained in an acceptable range during DR. Blocking should not be allowed because it causes the theoretical throughput of the system down to zero in that period. The lowest and highest points of the range depend on the QoS (Quality of Service) requirement of the application scenario. In the ideal DR, the designers do not want the end users to feel any unnecessary delay. Therefore, the lowest and highest points of the range should be the performance indicators of the original system and that of the target system.

Formally, suppose the system undergoes a sequence of interim configurations $c_1, c_2, \ldots, c_n$ in a DR and the theoretical performance indicators under these configurations are $p_1, p_2, \ldots, p_n$, the ideal minimum influence requires $p_i \in [p_{origin}, p_{target}], (1 \le i \le n)$, where p_{origin} is the performance indicator of the original system and p_{target} is that of the target system.

To satisfy this requirement, such reconfiguration operations that change the actual system configuration and last for a period should not appear in DR. All reconfiguration operations can be categorized into three types according to their impact on the system. Type I operations are the operations that do not change the actually effective part of the system. Thereby they will not influence the theoretical performance indicator of the system. For example, starting a component but not connecting it to other part of the system is such an operation. Type II operations are the operations that influences the actual system configuration but their execution is instantaneous, such as adding a connector, which is a very simple memory operation like passing the reference of a component to another. The transition from one configuration to another caused by a type II operation is instantaneous, and therefore, the theoretical performance indicator of the system changes from one value to another without any intermediate progress. Type III operations are the operations that influence the actual system configuration and their execution will last for a period, such as transferring state variables between old and new components. This operation can not be considered instantaneous because how much time it takes is relevant to the application scenario. Transactions relevant to the part changed by type III operations have to be blocked during the execution period; otherwise functional side-effect is very likely to appear because the result of these transactions will be unpredictable. Although the blocking period may be very short (usually it is decided by the application scenario and reconfiguration algorithm), theoretically the response time of the system in that period could be extremely longer than normal and the throughput of the system could be zero. It means a violation to the acceptable range.

On the component/connector model, there are five elementary reconfiguration operations, which are *start component*, *transfer state variables*, *stop component*, *add connector* and *remove connector*. Among them, *start component* and *stop component* are type I operations because the execution of these two operations do not change the theoretical performance indicator of the system. *Transfer state variables* is a type III operation. It changes the configuration (new component takes the place of the original component) and will last for a period. *Add connector* and *remove connector* can be considered type II operations. State variable transfer is not allowed in a non-blocking DR unless it is such a simple operation that can be implemented/considered instantaneously. That is a requirement to the application scenario for the implementation of the ideal DR.

If the application scenario does not need state variable transfer, the key problem to minimize its logical influence on performance is on the arrangement of the operations in a reconfiguration plan. It is obvious that the stopping of original components first and then starting new components is not a practical plan as the system will be unable to respond to any requests in the period after the original configuration is shut down and before the new one can be initiated. The appropriate reconfiguration plan should be a *switching plan*, which is comprised of three stages: 1) establish the new configuration, 2) switch from the original configuration to the new configuration, 3) shutdown and remove the original configuration. The progress of the switching plan is shown in Fig.2.

In the switching plan, the establishment of the new configuration can be comprised of type I operations so that it does not influence the theoretical performance

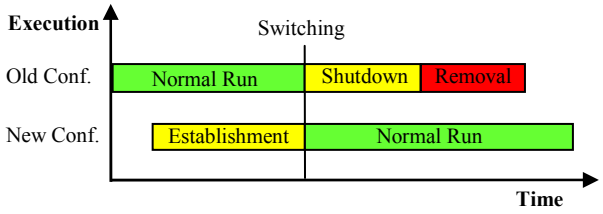


Figure 2. The switching plan for DR.

indicator of the system. The theoretical performance indicator of the system in this stage is p_{origin}. The switching can be achieved through type II operations so that it can be considered as instantaneous. After the switching, the theoretical performance indicator of the system becomes p_{target}. And the shutdown (waiting for old transactions to complete) and removal of the old configuration can be implemented by type I operations. Since the theoretical performance indicator of the system is kept in the range of $[p_{original}, p_{target}]$, the switching plan has the minimum logical influence on performance.

C. Avoiding Functional Side-effect

The switching reconfiguration plan has the minimum logical influence on performance, but other challenges may follow: avoidance of functional side-effect. This is because the original configuration works in parallel with the new configuration during its shutdown period and it may share some components with the new configuration.

This problem needs to be discussed based on the concept of transaction. A transaction refers to the processing to a request from the time it is received to the corresponding result being generated. And it is usually a composition of the internal processing of several components and the interactions between them. Here interaction refers to the communication between two components through connectors. In DR, to avoid functional side-effect is to assure the correctness of every single transaction. This transactional correctness consists of two aspects: *transactional non-interleaving* and *transactional completeness*.

Transactional interleaving refers to the phenomena that transactions belonging to different configurations may interfere with each other. Architectural interleaving (a component participates in two different configurations) is allowed, but transactional interleaving should be avoided.

For instance, in the data encryption/decryption system, a DR is to replace the encryption component and decryption component with new algorithms (Fig.3). Transactional interleaving would appear if a data package was encrypted by the original encryption component but decrypted by the new decryption component or vice versa. The reason that the transactional interleaving is easy to happen is due to the functional dependencies between components and the sharing of components between configurations may break these dependencies. In this data encryption/decryption system, the decryption component and the encryption component have dependency relationship. The sharing of the packing component and unpacking component may break the dependency relationship between the encryption and decryption. In literatures, *global consistency* [15] is also used to describe this problem.

Transactional completeness means that a transaction should complete once it is instantiated. It requires any reconfiguration operations not to stop or destroy any ongoing interactions or transactions. Particularly it should be assured that all transactions belonging to the original configuration complete during the shutdown period

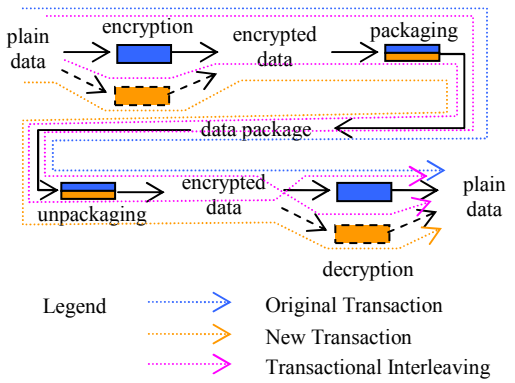


Figure 3. Transactional interleaving.

before the removal of components/connectors could take place during the removal period (reference to Fig.2).

To assure transactional correctness needs the support of the underlying component model. Transactional non-interleaving requires a version control mechanism for components that can distinguish the transactions belonging to different configurations and make them run independently. Transaction completeness requires 1) a synchronization mechanism that can guarantee any reconfiguration operation that may interrupt an ongoing interaction being executed after the interaction completes; and 2) a tracing mechanism that can make sure that a component or connector is not being used currently and no longer used in future by a transaction so that it can be removed safely.

D. Restricting Physical Influence on Performance

DR may have physical influence on performance because the reconfiguration procedure may compete with the ongoing functional procedures for processor time. How to control this influence can be analyzed from two conditions.

If the system has free processor time, the reconfiguration procedure can be executed using the free processor time and other functional procedures can run as if there was no reconfiguration procedure. Such a scheduling mechanism relies on *preemptive* scheduling with the functional procedures assigned a high priority and the reconfiguration procedure assigned a low priority. Under this scheduling, the reconfiguration procedure should not obtain processor time until there is no functional procedure acting in the system and it should yield the processor to any functional procedures once they are ready for execution. Using the preemptive scheduling can totally avoid the physical influence of DR on performance under the condition that free processor time is available.

If the system has no free processor time, the preemptive scheduling does not work because the reconfiguration procedure has no chance to get processor to execute. Instead, time-slice scheduling is a practical way for influence control under this condition. Under the time-slice scheduling, the processor time is separated into many small slices. In some slices the reconfiguration procedure are allowed to run and compete with other functional procedures for processor time. In other slices it

is not allowed to do so. Viewing from a longer period, the physical influence of DR can be restricted through controlling the amount of slices in which the reconfiguration procedure is allowed to run.

Two examples are helpful in understanding the executions of DR and their influence under these two scheduling mechanisms. Fig.4(A) shows an example of the preemptive scheduling. Suppose that four data items will be available at time t, t+3(seconds), t+5 and t+8 separately. The processing to each of these data items takes 2 seconds. A reconfiguration process starts at time t+2 and it will take 2 seconds if it possesses the processor exclusively. Comparing with the execution without DR, the data processing of the preemptive scheduling remains the same but that of the non-preemptive scheduling is different: the processing to the second and the third data item is delayed. Fig.4(B) demonstrates the time-slice scheduling. Data items will be available every 2 seconds; therefore, there is no free processor time. With the absence of proper control, the reconfiguration procedure will compete with functional procedures for processor time during its life-span. It prolongs the processing to the second and third data items and also delays the processing to the following data items. With time-slice scheduling, the execution of the reconfiguration procedure is separated into several parts, which are distributed in a longer period so that the influence to each single processing is restricted. Even though this also prolongs the processing of the second to fourth data items and delays the processing to the following data items, but the influence is smaller than the execution without control.

The precondition of applying these two DR scheduling mechanisms is that the reconfiguration procedure is able to be rescheduled at any time. As pointed out in part B of this section, only type I and type II operations are allowed in DR. Because theoretically type II operations are instantaneous, its execution will not delay the rescheduling. Type I operation will last for a period and its execution may be interrupted at any time by functional procedures and needs to be resumed in future. This interruption will not cause functional side-effect because type I operations do not change the actual configuration.

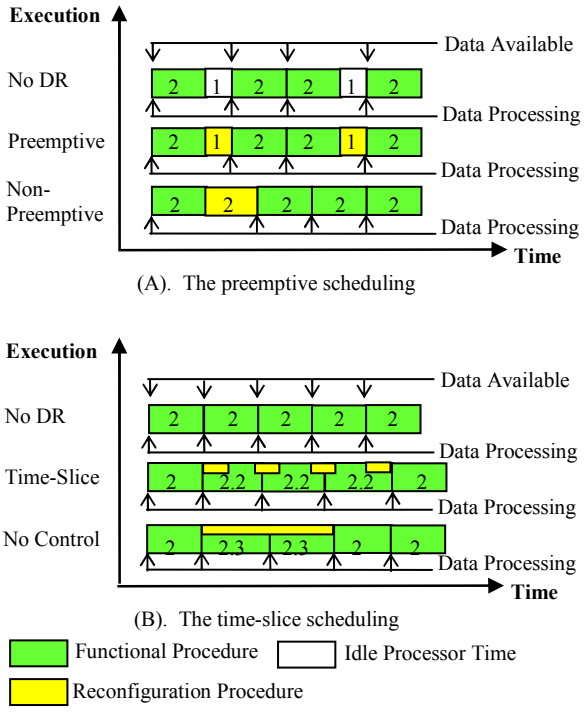


Figure 4. DR scheduling.

E. Summary for Influence Control

The discussion on the influence of DR in this section is summarized in TABLE II.

IV. The Reconfigurable Data Flow

A. Motivation

The Reconfigurable Data Flow (RDF) is designed to show the feasibility of the methods presented above.

The RDF is an extension to the conceptual Dataflow Process Networks (DPNs) [4] so that it firstly supports the fundamental dataflow semantics defined by the DPNs. The motivation to use the DPNs as the base model for reconfigurable extension comes from three aspects. First, the mechanisms for influence control can be implemented on the DPNs, including version control, synchronization, transaction tracing, switching reconfiguration plan, schedulable type I reconfiguration operations, and preemptive and time-slice scheduling. Second, over the past decades, the DPNs have been proved to be a widely used optimized model to describe highly concurrent applications in the areas of signal-processing, linear or nonlinear control systems, image processing for geographical information systems, or other stream-oriented application systems [13]. Third being compatible with the conceptual DPNs, the RDF is able to make full advantage of research outcomes realized by the DPNs communities in the future.

TABLE II. INFLUENCE CONTROL

Influence	Functional side-effect	Logical influence on performance	Physical influence on performance
What should be down	Avoidance	Minimization	Restricting
How to achieve	Assuring transactional correctness	Using the switching reconfiguration plan	Restricting the competition on processor time
Control applied in what stage	Reconfiguration design	Reconfiguration planning	Reconfiguration. execution
Supports needed from the underlying model	Version control Synchronization Transaction tracing	The switching reconfiguration plan	Type I operation schedulable, Preemptive scheduling Time-slice scheduling
Requirements to the application scenario		No need for type III operations (e.g. state variables transfer)	

B. Basic RDF Elements and an Application Scenario

The basic elements of the RDF are *process*, *data-store*, and *data-path*. A process is such a software component that has a single threaded working progress, consuming data through its entrances, then processing them, and finally producing results 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 data-store through which the process can consume data as its input from the data-store or save the produced data to the data-store. In the RDF, processes' internal processing to data items do not interfere with each other, and the interactions between processes are the transmissions of data items through data-stores. A data-store has to be used to buffer the data items in a transmission because the receiver process may be not ready to consume data items when the sender process produces the data items.

The reconfiguration operations of the RDF include addition and removal of processes/data-stores (type I operations), and addition and removal of data-paths (type II operations).

Graphically, a RDF model could be represented as a bi-partite directed graph, in which rectangles represent processes, circles represent data-stores, and arrows represent data-paths.

Fig.5 shows an example, the PKI Data Encryption and Digital Signature System (DEDSS) [1]. In the system, the data items in data-store *d1* will be delivered to the corresponding receivers after being digitally signed and encrypted. The encryption progress and the signature progress can be carried on in parallel on the sender side as well as the decryption progress and the signature verification progress on the receiver side.

A DR to the system might be to add data compression/decompression function and replace the digesting algorithm as shown with dashed line in Fig.5.

Such a reconfiguration scenario can expose the main issues in avoiding functional side-effect. There are three pairs of functional dependencies that may be broken in transactional interleaving in DR. 1) dataEncryption encrypts data items with the receiver's public key and dataDecryption decrypts data items with the receiver's private key; 2) mDigest2 and mDigest1 use the same message-digesting algorithm; 3) digestEncryption encrypts the digest with the sender's private key and digestDecryption decrypts the digest with the sender's public key.

In addition, to apply the switching reconfiguration plan, the DEDSS is designed as it does not need state variables transfer in the DR. The data compression processes, the data decompression processes and the digesting processes are all stateless.

C. Mechanisms to Support Dynamic Reconfiguration

The RDF provides several mechanisms for DR including synchronization, version control, transaction tracing, reconfiguration planning and reconfiguration scheduling.

The synchronization mechanism of the RDF is able to protect the consuming/producing operations of processes from being interrupted by the addition/removal of data-paths. All data transmissions in the RDF are through the data consuming or data producing operations of processes. All reconfiguration operations that may interrupt data transmissions are the addition or removal of data-paths. The removal of a process/data-store will not interrupt data transmissions because it requires the data-paths connected to the process/data-store to be removed first. Therefore, such a synchronization mechanism can protect the data transmissions from being interrupted by reconfiguration operations. In another aspect, the synchronization will not delay the execution of processes. As discussed in the previous sections, theoretically the addition and removal of data-paths are instantaneous so that their execution will not delay the data consuming or data producing operations of processes. Other reconfiguration operations such as addition and removal of processes/data-stores have no conflict with any

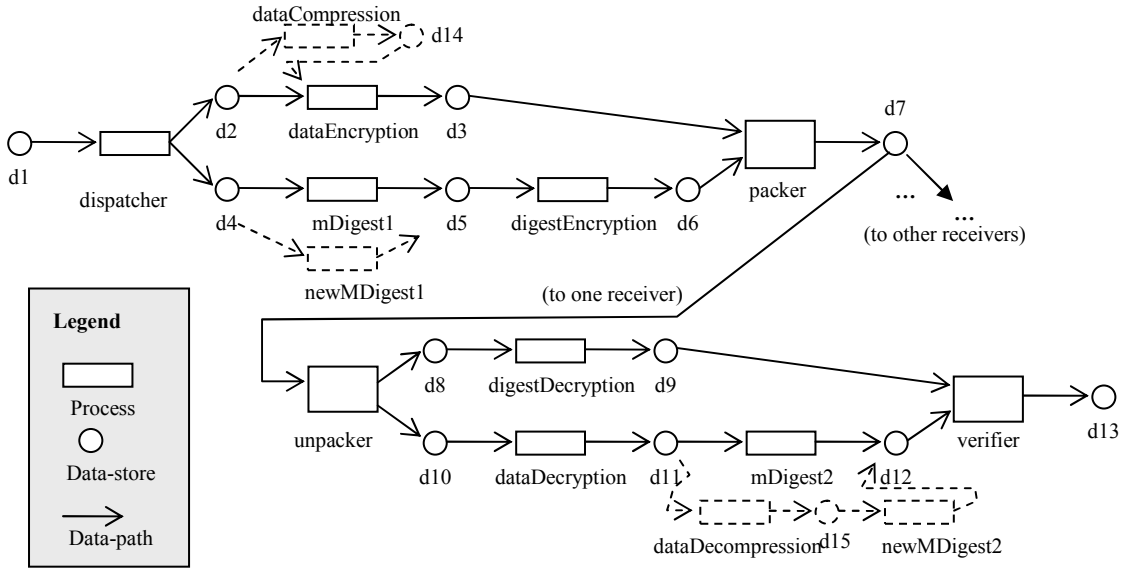


Figure 5. The DEDSS.

ongoing data transmissions so that they do not need to be synchronized with the consuming or producing operations of processes. Hence, these are schedulable in any time period.

The version control mechanism of the RDF prevents transactional interleaving by isolating the data flows belonging to different configurations. The RDF is a data flow based model, therefore, the basic idea of the version control is to assign every data item (or package) a version tag and let processes be able to recognize the version tag. The data items of the transactions belonging to the original configuration are assigned one version tag and those belonging to the new configuration are assigned another version tag, therefore, they can be distinguished and processed by processes independently. To let the version control be transparent to the process developers, the version tag is designed as a virtual tag so that the process developers do not need to concern about it. They can develop processes following the consume-process-produce pattern without any concern about the versions of the data items.

The version control mechanism is supported by the following rules:

1) A process has three version modes: *strict*, *transparent* and *filter*. Under strict mode, denoted as *(x,y)*, a process can consume data items of version *x* and produce data items of version *y*. Under transparent mode, denoted as *(*,*)*, a process can consume data items of any versions and produce data items of the same version as consumed. Under filter mode, denoted as *(*,z)*, a process can consume data items of any version but produce data items of version *z*. Changing a process from one version mode to another is instantaneous and thereby it is a type II reconfiguration operation.

2) A data-store can store data items of any versions. Different versions of data items are stored separately in a data-store so that they can be distinguished although there is no real version tag on each of the items.

Transactional interleaving can be avoided by utilizing version control. In the data encryption/decryption example in Fig.3, transactional interleaving will not happen if the processes are set to the proper version modes as shown in Fig.6. The data items encrypted by the original algorithm will have version *v1*. This virtual tag will be kept till these packages reach data-store *d4*. And then they can only be consumed by the original decryption algorithm. By the same principle, the data items encrypted by the new algorithm will have version *v2* and they can only be consumed by the new decryption algorithm. For each DR that may have transactional interleaving, such a *restrict segment* can be found and version control can be applied to avoid the transactional interleaving.

The transaction tracing mechanism of the RDF is based on the following idea. If the incoming data-paths of a process (data-paths that connect data-stores to the entrances of the process) are removed, the process will become idle after its ongoing processing finishes and will never be triggered in future. And if the incoming data-paths of a data-store (data-paths that connect the exits of processes to the data-store) are removed, the data-store will become empty as its data items will be consumed up by the subsequent processes and there is no new data item to flow into the data-store. The removal of such idle processes or empty data-stores will not influence the completeness of any transactions.

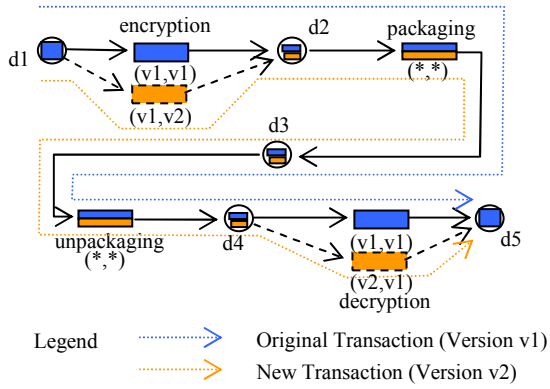


Figure 6. Version control.

Two operations are designed to support the transaction tracing.

1) *Wait-finishing(p,v)*, which waits for process *p* to finish the current processing of version *v* data items.

2) *Wait-empty(d,v)*, which waits for data-store *d* to be empty with version *v* data items.

During the original configuration shutdown period in DR, all incoming data-paths to the original configuration are closed so that there will be no data item to flow into this part of the system. Then for each original transaction, the *wait-finishing* operation can be used to wait for the processes to finish and then the *wait-empty* operation can be applied to the subsequent data-stores. This progress continues until all original version data items are assured to flow out of the original configuration, which means that the processes and data-stores belonging to the original configuration are never to be used so that they can be safely removed. In addition, these two operations will not block the system because all new data items can be processed by the new configuration during this stage.

For the example in Fig.6, the completeness of the original transactions can be assured by the following progress.

(remove data-path d1→encryption;)
wait-finishing(encryption, v1);
wait-empty(d2,v1);
wait-finishing(packaging, v1);
wait-empty(d3,v1);
wait-finishing(unpackaging,v1);
wait-empty(d4,v1);
wait-finishing(decryption,v1);

The reconfiguration planer of the RDF is able to generate switching plans for DRs. First of all, a reconfiguration must have a declarative specification, denoted as $C_{origin} \rightarrow C_{target}$, where C_{origin} is the original configuration of the system and C_{target} is the target one. Then a declarative specification can be changed to an operational plan automatically by the planner provided on the RDF.

The configuration of a system is represented as a *route map*, which consists of all the *routes* in the system. A route is a sequence of processes and data-stores that data items must go through in a transaction. A route has the format of $[d_1, (en_1,p_1,ex_1),d_2,(en_2,p_2,ex_2),...]$, which means a data item in d_1 can flow into process p_1 through its entrance en_1, and the result will flow out of p_1 through its exit ex_1 and flow into d_2. Then it can flow into process p_2 through its entrance en_2, and so on. From the route map representation, it is easy to know what processes and data-stores comprise of the system, how they are connected by data-paths, and particularly how transactions are carried on. The information is necessary for the version control and transaction tracing to be applied in the DR.

The plan generated by the RDF planner is a switching plan with the influence control methods integrated, i.e. version control is used to assure transactional correctness and transaction tracing is used to assure transactional completeness. Such a plan is comprised of six steps and its progress is shown in Fig.7.

T1. Apply version control to all the processes once they are in the system.

T2. Add new processes, new data-stores and new data-paths into the system.

T3. Start data supply to the new configuration and stop data supply to the old (to-be-removed) configuration.

T4. Wait for the remaining data flow to go through the old configuration.

T5. Remove the old processes, old data-stores and the old data-paths from the system.

T6. Cancel version control.

As an example, the original and target route maps of the DEDSS, and the generated reconfiguration plan are given in appendix A.

The reconfiguration scheduling mechanism of the RDF is able to control the execution of DR. Since the scheduling mechanism is based on the thread scheduling mechanism of the underlying platform, it first requires the underlying platform (the operation system or middleware) supporting the priority-based preemptive scheduling. Given a reconfiguration plan, the preemptive scheduler will run the plan in a separate thread that is assigned a lower priority than other functional threads. Under the priority-based preemptive scheduling of the underlying platform, the reconfiguration will be executed using the free processor time between functional threads. Therefore, it will not cause physical influence on performance. The time-slice scheduler will run the plan in a thread with the same priority to the functional threads so that the reconfiguration thread will have chance to get processor time. And the time-slice scheduler will periodically suspend the reconfiguration thread so that the physical influence of the reconfiguration thread is restricted. Choosing which scheduler depends on the application scenario, i.e. whether there is free processor time and the QoS requirement.

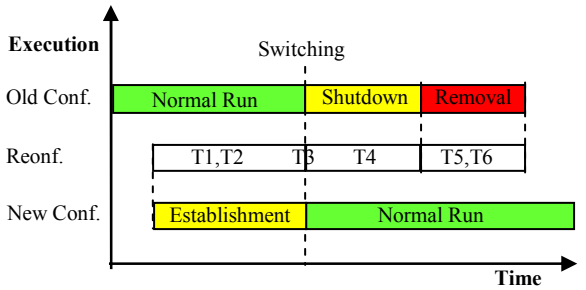


Figure 7. Reconfiguration state graph.

D. Implementation and Experimental Results

The RDF platform is an implementation of the RDF model on Java platform. It integrates all the mechanisms supported by the RDF model. Fig.8 depicts its functions and Fig.9 shows its Graphical User Interface (GUI).

In Fig.8, *constraints check* checks whether functional dependencies are satisfied in architectural configurations. This function can help designers in finding errors in architecture design. *Runtime setting* configures the running environment for a system, including the data-sources (feeding the system with data items), the verifiers (verifying the functional correctness of the system), and the monitors (monitoring the performance indicator of the system). *Run* starts a system according to its configuration and thereby the functional correctness and performance indicator can be detected. *Reconfiguration setting* establishes a reconfiguration scenario, including the original and target configurations, the planner and scheduler to be used, and when the reconfiguration starts. Then *planning* generates the reconfiguration plan using the designated planner and *scheduling* controls the execution of the reconfiguration using the designated scheduler. *Verification* verifies the functional correctness

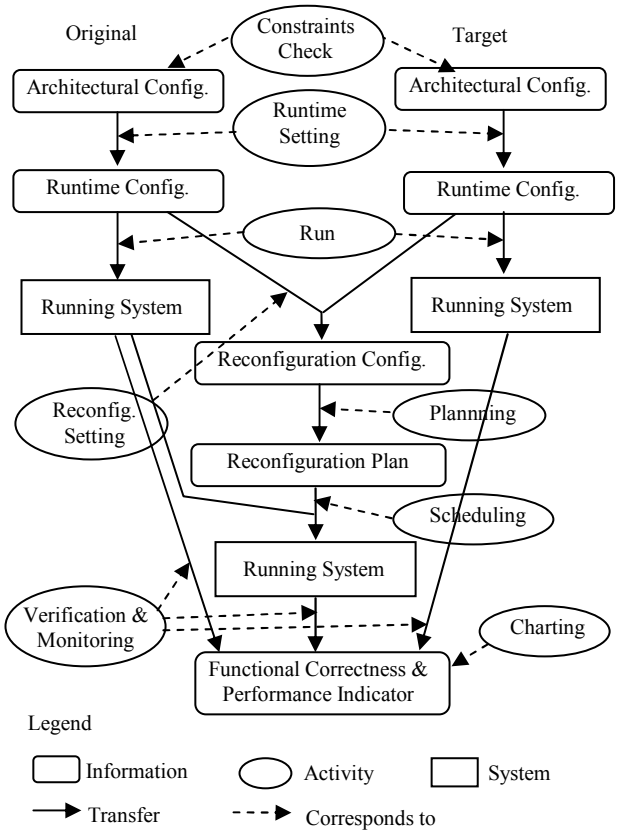


Figure 8. Functions of the RDF Platform.

of the system using the verifiers predefined in *runtime setting*, and *monitoring* records the performance indicator using the monitors. Finally, *charting* collects all the results and draws the performance indicators in a line chart, from which the influence of DR on system performance is easy to uncover.

The RDF platform is very flexible in that: 1) it has well-defined framework for processes and an open process library. Therefore, developers can easily implement processes needed by applications and put them into the library; 2) application specific data-sources and verifiers can be designed following the predefined interfaces; 3) the architecture, running environment and reconfiguration scenario of the system to be tested can be set up through the GUI interface and all settings can be saved in XML document format; 4) the reconfiguration planner and scheduler are pluggable so that other planners and schedulers can be imported as plug-ins and be tested and compared based on the same reconfiguration scenario on the platform.

Based on the RDF platform and using the DEDSS as a case study, two groups of tests have been done to examine the effectiveness of the mechanisms for influence control. Group A has been done under the condition that the system has no free processor time and group B has been done under the condition that the system has free processor time. Group A consists of five tests. Test 1 examines the influence of the DR when these mechanisms for influence control are all applied, including the version control, the transaction tracing, the switching plan, and the time-slice scheduling. Test 2 examines the influence of the DR when the version control and the transaction tracing are not used. Its settings are same to test 1 except that it changes the system from the original configuration to the target one directly without using the version control and the transaction tracing. By comparing with test 1, it can be found whether the version control and the transaction tracing are able to avoid functional side-effect. Based on the same idea, test 3 uses the blocking reconfiguration plan and test 4 does not use reconfiguration scheduling while other settings are the same as test 1. Then comparing with test 1, it can be found to what extent the switching plan and the time-slice scheduling can restrict the influence of DR. Test 5 examines the different effects of the time-slice scheduling under different parameters. For a time-slice scheduling, 40% means the reconfiguration thread is allowed to compete for processor time for 40% in a time slice; and 20% means the competition for processor time is allowed for 20% of a time slice. The settings for group B are almost the same as group A. The difference is that group B uses the preemptive scheduling because the system has free processor time. The settings are listed in TABLE III. For each test, both the throughput and the response time of the system are recorded.

In the tests, functional side-effect appears in test 2 of both group A and group B. The DRs in other tests have no functional side-effect. For the DEDSS, the functional side-effect can be detected by checking whether the data items outputted to data-store *d13* are exactly the same to the data items inputted to data-store *d1*. Considering only test 2 does not apply the version control and transaction tracing, a conclusion can be drawn that the version

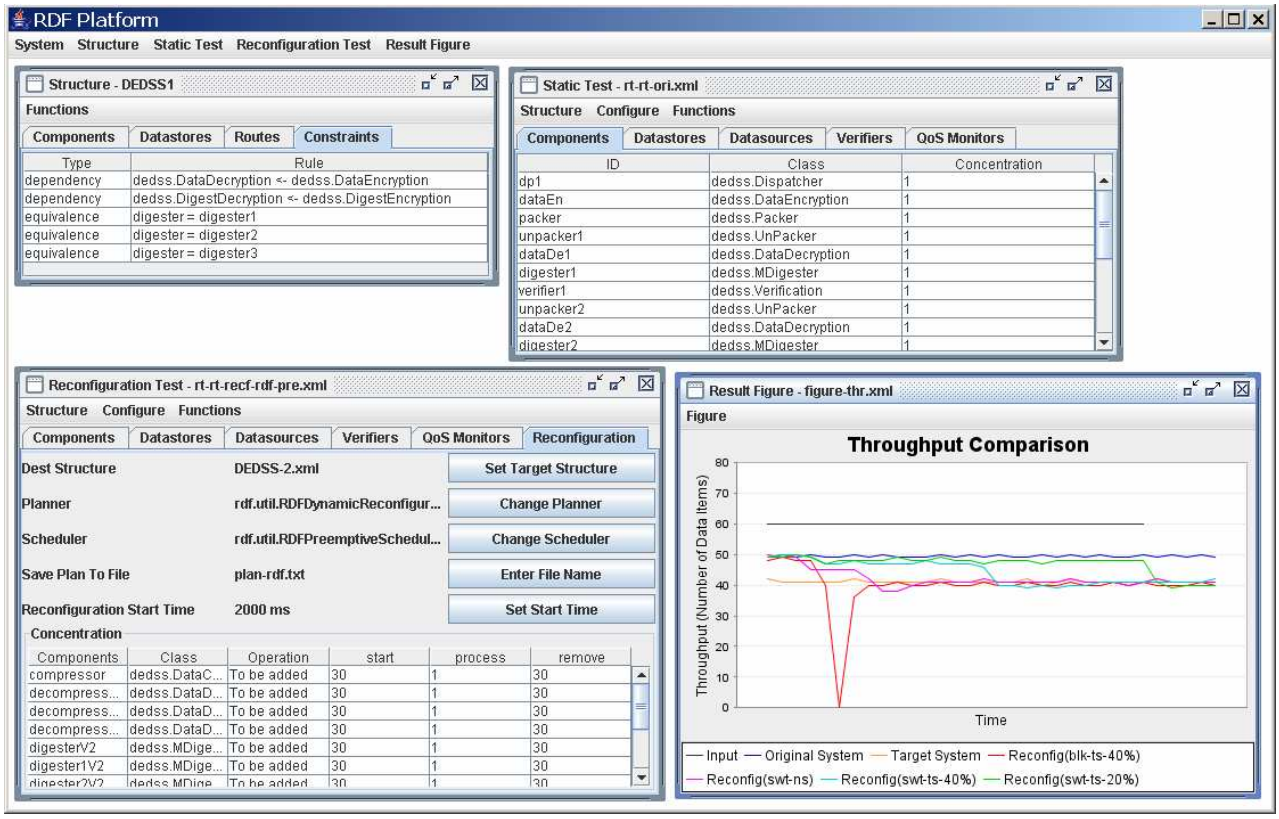


Figure 9. The GUI of the RDF platform.

TABLE III. SETTINGS FOR THE EXPERIMENT

(A) THE SYSTEM HAS NO FREE PROCESSOR TIME

Test	Version Control and Transaction Tracing	Plan	Scheduling	Influence to Test
1	Use	Switching plan	Time-slice(40%)	Restricted influence
2	Not use	Switching plan	Time-slice(40%)	Functional side-effect
3	Use	Blocking plan	Time-slice(40%)	Logical influence on performance
4	Use	Switching plan	No scheduling	Physical influence on performance
5	Use	Switching plan	Time-slice(20%)	Restricted influence

(B) THE SYSTEM HAS FREE PROCESSOR TIME

Test	Version Control and Transaction Tracing	Plan	Scheduling	Influence to Test
1	Use	Switching plan	Preemptive	No influence
2	Not use	Switching plan	Preemptive	Functional side-effect
3	Use	Blocking plan	Preemptive	Logical influence on performance
4	Use	Switching plan	No scheduling	Physical influence on performance

control and transaction tracing are able to assure the functional correctness of the system in DR.

The performance indicators of the system in the tests excluding test 2 are drawn in line charts in Fig.10. (It is no sense to discuss the performance indicator of test 2 because the precondition i.e. functional correctness is not satisfied in test 2). For the DEDSS, the response time is the time interval from the instance when a data item enters data-store *d1* to the instance when the data item is put into data-store *d13*. The throughput is the amount of data items consumed from data-store *d1* or produced to data-store *d13* in a time interval. The performance indicators of the original system and the target system are also drawn in the charts as references (the blue line and the orange line).

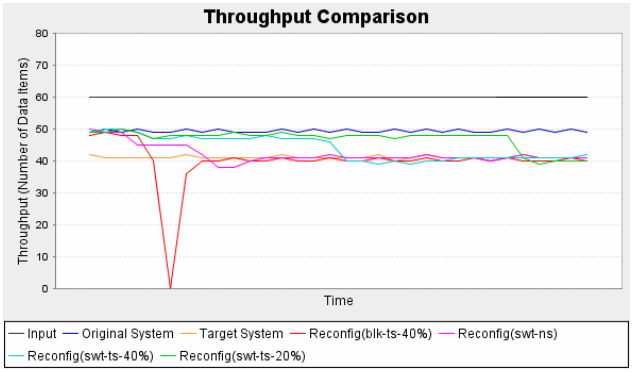
The max throughput of the original system is about *50 dips* (data items per second) and that of the target system is about *40 dips*. In group A, the system is fed with *60 dips*, which is larger than the max throughputs of both the original system and the target system so that the system has no free processor time. Fig.10(A) gives the throughput comparison of the tests. The blocking reconfiguration plan has a severe influence on the throughput (the red line). The DR without scheduling (the pink line) has an influence larger than the DR with the time-slice scheduling (the indigo line and the green line). And the DR under the 20% time-slice scheduling (the green line) has an influence smaller than the DR under the 40% time-slice scheduling (the indigo line). Fig.10(B) gives the response time comparison. As the feeding rate is larger than the max throughput, more and more data items are buffered in data-store *d1*. Therefore, the response time of the system to these data items becomes larger and larger and it results in slash lines in the chart. To make the performance indicators comparable under this condition, the starting time of the DRs are adjusted so that the new configuration starts to work at the same time. It is indicated by the chart that the DR using the blocking plan causes the fastest increasing rate to the response time (the red line), the DR without scheduling causes a relative slower increasing rate (the pink line), and the DR with time-slice scheduling causes the slowest increasing rate (the indigo line and the green line). The result of group A indicates that the mechanisms used in the RDF for influence control are effective under the condition that the system has no free processor time.

In group B, the system is fed with *38 dips*, which is smaller than the max throughputs of both the original system and the target system so that the system has free processor time. Fig.10(C) gives the throughput comparison of the tests. The lines for the input, the original system, the target system, and the DR using the switching plan and the preemptive scheduling are overlapped (only the green line can be seen because it is on the top, which corresponds to the DR using the switching plan and the preemptive scheduling). That means the DR has no any influence on the throughput. And the DR using the blocking plan imposes a large influence and the DR without scheduling causes manifest influence on the throughput. Fig.10(D) gives the response time comparison. As the feeding rate is smaller than the max throughput, there is no data item waiting in data-store *d1*. Therefore, the response time of the original system is horizontal (the blue line); and so is the target system (the orange line). It also can be seen that the DR using the switching plan and the preemptive scheduling changes the response time from the original level to the target level without other influence (the green line). And the DR using the blocking plan causes a very large response delay that is out of the chart (the red line) and the DR without scheduling causes a large influence too (the pink line). The result of group B proves the effectiveness of the mechanisms used in the RDF for influence control under the condition that the system has free processor time.

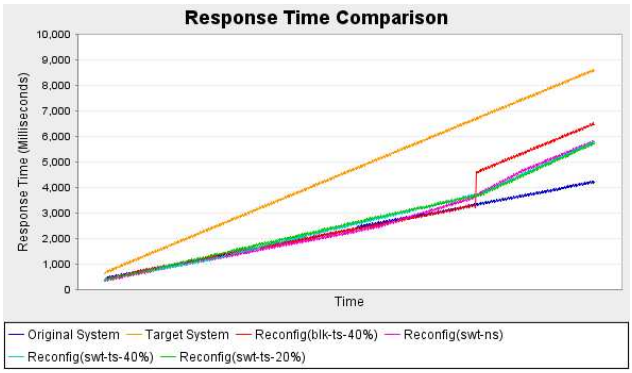
IV. CONCLUSION AND FUTURE WORK

This paper has three main contributions to the influence control for DR.

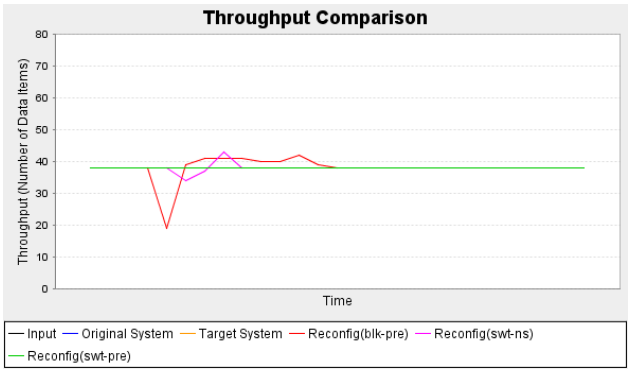
First, this paper theoretically analyzes how DR influences system execution and how the influence can be controlled. DR may influence system execution in four ways, which result in the functional update, functional side-effect, logical influence on performance, and physical influence on performance. To control the influence of DR, the functional side-effect should be avoided, the logical influence on performance should be minimized and the physical influence on performance should be restricted.



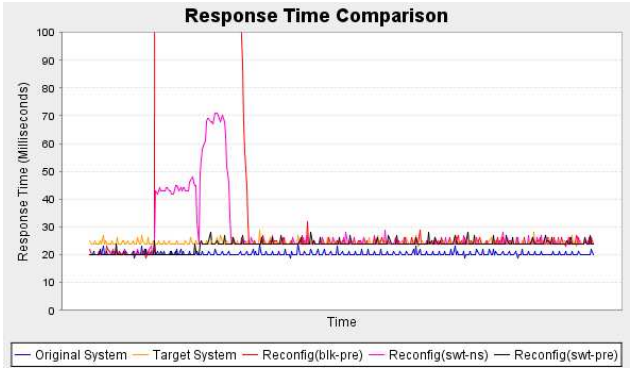
(A) Throughput comparison of group A



(B) Response time comparison of group A



(C) Throughput comparison of group B



(D) Response time comparison of group B

blk – the blocking plan
ns – no scheduling
pre – the preemptive scheduling
swt – the switching plan
ts – the time-slice scheduling

Figure 10. Experimental results.

Second, this paper proposes the RDF model to demonstrate the feasibility of the methods for influence control. The RDF has several mechanisms for influence control. The version control prevents transactions belonging to different configurations from interleaving. The transaction tracing assures the completeness of transactions. For a DR, the reconfiguration planner generates the switching reconfiguration plan, which has the minimal logical influence on performance. And the reconfiguration scheduler restricts the physical influence on performance using preemptive or time-slice scheduling policies.

Third, this paper also presents the RDF platform, which is the implementation of the RDF model on Java platform. With the support of the RDF platform and using the DEDSS as a reconfiguration scenario, a testing criterion is designed to simulate different reconfiguration plans and scheduling policies on the single platform and quantitatively compare their influence. Experimental results indicate that the methods for influence control are effective in the RDF model.

Future works will focus on two directions.

As the current RDF is limited in application to stateless system, the first direction of future research is to expand the application scenario of the RDF by adopting the methods to stateful data flows. Two problems have to be solved in the DR to stated systems with influence control: how the time cost of state variable transfers can be restricted and when these transfers can take place. The current exploration in this direction originates from two ideas. First, state variable transfer is possible to be implemented with object reference passing if stated components store their state in special objects. Second, the transaction tracing can be utilized to find the appropriate opportunity for state variable transfer.

The second direction is to expand the mechanisms of the RDF model for influence control to other component models. Besides flow based models, there are a lot of procedure-call based models [10], which are also being widely used in both academia and industry. The procedure-call based models differ from the flow based models in that a connector is a procedure-call, i.e. the caller asks the recipient (i.e. callee) to start a processing and wait for the recipient to return the result. There is correspondence between the two kinds of models. A procedure-call can be considered as two data-paths, one for the parameters passing when the invocation starts and one for the result returning when the invocation finishes. Based on this correspondence, the mechanisms of the RDF model will be further studied to apply to the procedure-call based models.

APPENDIX A THE RECONFIGURATION SPECIFICATION FOR THE *DEDSS* AND THE PLAN GENERATED

Although the DEDSS may have multiple branches, the specification and reconfiguration for one branch is enough to demonstrate the route map representation and the switching reconfiguration plan.

Specification (reference to Fig.5)

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]

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}, 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]

Reconfiguration Plan Generated

// T1. Apply version control

set verifier to filter(1) mode;

set dataEncryption, packer, unpacker, digestDecryption, mDigest1, digestEncryption, dataDecryption, mDigest2 to transparent mode;

// T2. Add new processes, data-stores and data-paths

start dataCompression, dataDecompression, newMDigest1, newMDigest2 and set them to strict(2,2) mode;

add 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);

// T3. Switch data supply

set dispatcher to strict(1,2) mode;

// T4. Transaction tracing

wait for version 1 data elements to flow out of the involved segments

// T5. Remove the old processes, data-stores, and data-paths

remove data paths (d2→0,dataEecryption), (d11→0,mDigest2), (mDigest2,0→d12), (d4→0,mDigest1), (mDigest1,0→d5);

remove processes mDigest1, mDigest2;

// T6. Cancel version control

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;

ACKNOWLEDGMENT

This work was supported by Central Queensland University, Australia under Research Advancement Awards Scheme (RAAS) grants, 2006~2007. Authors would like to thank Dr. Andrew Chiou for his proof reading of the paper.

REFERENCES

[1] C. Adams, S. Lloyd, "Understanding Public-key Infrastructure", Macmillan Technical Publishing, 1999.

[2] S. Ajmani, B. Liskov, and L. Shrira, "Scheduling and Simulation: How to Upgrade Distributed Systems" in 9*th Workshop on Hot Topics in Operating Systems (HotOS-IX)*, Hawaii, USA, 2003, pp.43-48.

[3] T. Bures, P. Hnetynka, F. Plasil, "SOFA 2.0: Balancing Advanced Features in a Hierarchical Component Model" in 4*th International Conference on Software Engineering Research, Management and Applications (SERA'06)*, Seattle, USA, 2006, pp. 40-48.

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

[5] P. Feiler, J. Li, "Consistency in Dynamic Reconfiguration" in *4th International Conference on Configurable Distributed Systems*, Annapolis, USA, 1998, pp.189-196.

[6] J. Gorinsek, S. Van Baelen, Y. Berbers and K. De Vlaminck, "Managing Quality of Service during Evolution using Component Contracts" in *ETAPS 2003 Workshop on Unanticipated Software Evolution (USE2003)*, Warsaw, Poland, 2003, pp.57-62.

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

[8] J. Kramer, J. Magee, "The evolving philosophers problem: Dynamic change management" in *IEEE Transactions on Software Engineering*, vol. 16, 1990, pp.1293-1306.

[9] S. Kulkarni and K. Biyani, "Correctness of Component-based Adaptation" in *International Symposium on Component-based Software Engineering*, Edinburgh, Scotland, 2004, pp.48-58.

[10] K.K. Lau and Z. Wang, "A taxonomy of software component models" in *31st Euromicro Conference on Software Engineering and Advanced Applications*, Porto, Portugal, 2005, pp.88–95.

[11] N. Medvidovic and R.N. Taylor, "A classification and comparison framework for software architecture description languages" in *IEEE Transactions on Software Engineering*, 26(1), 2000, pp.70–93.

[12] S.R. Mitchell, "Dynamic Configuration of Distributed Multimedia Components", *PhD thesis*, University of London, UK, 2000.

[13] W.A. Najjar et al, "Advances in the Dataflow Computational Model" in *Parallel Computing*, 25(13-14) 1999, pp.1907-1929.

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

[15] N.D. Palma, P. Laumay, and L. Bellissard, "Ensuring Dynamic Reconfiguration Consistency" in *6th International Workshop on Component-Oriented Programming (WCOP 2001)*, Budapest, Hungary, 2001.

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

[17] M. Wegdam, "Dynamic Reconfiguration and Load Distribution in Component Middleware", *PhD thesis*, University of Twente, Netherlands, 2003.

[18] Z. Zhao and W. Li, "Influence Control for Dynamic Reconfiguration" in *Australian Software Engineering Conference 2007*, Melbourne, Australia, 2007, pp.59-68.

Wei Li is a Senior Lecturer in Computer Science in the School of Computing Sciences at the Central Queensland University, Australia. He received his PhD degree from the Institute of Computing Technology, Chinese Academy of Sciences in July 1998. His research interests include dynamic software architecture and multi-agent systems.

Zhikun Zhao received his Ph.D. degree in Computer Science at the Graduate University of Chinese Academy of Sciences (GUCAS), Beijing, China in 2003. He has been a Postdoctoral Research Fellow of Central Queensland University (CQU), Rockhampton, Australia since 2006. His research interests include software dynamic reconfiguration and agent technology.