Dynamic Reconfiguration of Component Based Software Systems

D.John Aravindhar, V.Dheepa, D. Nancy Kirupanithi
School of Computing Science, Hindustan University, Chennai

Abstract- Dynamic reconfiguration technique is introduced to maintain Quality of Service, which is meant to reduce application disruption during the system transformation. This dynamic reconfiguration technique involves the ability to change the system’s functionality or topology while the system is running. This technique involves safe dynamic reconfiguration such as insertion, removal and replacement of components. Our major challenge for this dynamic reconfiguration technique is to maintain the Quality of Service during system transformation, which has been achieved. The true benefit of this technique is application consistency and service continuity. The motivation for this dynamic reconfiguration technique is adaptability and high availability, which are both Quality of Service driven characteristics. An adaptive system is capable of runtime reconfiguration and works in unanticipated environments. In order to achieve high reliability and availability [3], the distributed component software has to support dynamic reconfiguration in order to avoid downtimes caused by reboots of the system. Dynamic reconfiguration refers to the set of tools and services allowing these changes to be performed in a dynamic way, which is without stopping the entire application. The objective is to freeze only the part of the application concerned by the modification so that the overall penalty on the running application is minimized. Dynamic reconfiguration technique looks very much like traditional control system model of ‘sense-plan-act’. This project investigates the maintenance of Quality of Service for component based software system from three points of view. Starting with the whole spectrum of Quality of Service is defined. Then the logical and physical requirements for Quality of Service characteristics are analyzed and solutions to achieve them are proposed. Finally, the prior work is classified and realized by the abstract configuration strategies.

Index Terms: Adaptability, Component, Dynamic Reconfiguration, Quality of Service Characteristics, Transformation.

I. INTRODUCTION

The importance of reconfiguration of software systems was to fix bugs, improve performance, and extend functionality previously. Static reconfiguration required shutting down, recompiling, and rebooting the system in the middle of the process. Some significant disadvantages were identified in concern with static reconfiguration of the system with respect to Quality of Service. First, shutting down could cause the system to lose its state of the process. Second, unavailability is not accepted for mission critical systems. Third, unavailability also leads to poor adaptation. Dynamic reconfiguration was proposed as an alternative to static reconfiguration. Dynamic reconfiguration is a synonym of runtime evolution [10]. This means the ability to change a system’s functionality or topology while the system is running. Dynamic reconfiguration is the process of changing at runtime via addition, deletion, and replacement of components, or alteration of the topology in a component system. Application consistency and service continuity while the system is being updated are the benefits of dynamic reconfiguration. Adaptively and high availability are the motivation for dynamic reconfiguration. The two aspects are both Quality of service driven. An adaptive system frequently adapts to the behavior of the environment and works in unanticipated environments. Mobile and ubiquitous systems also require dynamic reconfiguration in the form of system updates while the system is active [6]. The output of this dynamic reconfiguration was to enable safely changing of the system at runtime. The unaffected components are operational and the affected part is suspended for reconfiguration [2]. A core concept relating to node which is the software components is that of quiescence. Where a node is not involved in a transaction and will neither receive nor initiate any new transactions. The transaction model can be independent or dependent. The definition of quiescence and the change rules remain the same as for independent transactions when they are supporting dependent transactions. The passivity set must be expanded to account for dependency. The result is that the size of an affected part depends on transaction models. They are more interdependent and more globally quiescent.

A. A Component Model and Its Reconfiguration Techniques

The abstract model and the reconfiguration have been explained in the following session. The components, connectors, services, and these terms are used for context of dynamic reconfiguration. The dependent concepts are introduced in terms of Quality of Service characteristics. A component is a processing entity that encapsulates a set of functionality and data within it. A component has a provided-interface to specify a set of services that it provides for use by other components. A component has a required-interface to specify a set of services. A service is a set of functionality that has policies that provides its usage. A connector is a directed connection from a required service of a component to a provided service of another component. It indicates the client/server relationship between the components. The client of its services can be an internal component of the same system.
or an external software agent for an individual component, as long as they follow the service specification to make a request. For an example, in Fig 1, the Signing component offers a digital signing service through its provided-interface and needs two services to function as indicated in its required interface such as message digesting, which is here satisfied by connection to a corresponding service in the provided interface of the Message Digest component, and digests encrypting, satisfied by the Digest Encryption component.

Fig 1. A Component System with a Set of Components

In providing services to each other, individual components may be assembled into a system. As a whole they may provide services to other component systems or external software agents. A component system consists of a set of components with directed connections between them by a set of connectors. The proposed model does not restrict the way to make a request, which could be a request-reply style or a method call. A configuration of a component system is described as the structural relationship between components [9]. It indicates by the layout of components and connectors. A reconfiguration is to modify the structure of a component system in terms of addition, deletion, and replacement of components and/or connectors. Therefore, while the reconfiguration transforms the structural view of a component system [1]. It changes the system’s functionality and service specification. A reconfiguration may consist of several individual updates or changes to components and connectors.

B. Motivation for the Work

This project explains the benefit of dynamic reconfiguration which is to minimize application disruption. It is the Quality of Service assurance feature of dynamic reconfiguration. It essentially differentiates it from any other static reconfiguration techniques. The poor Quality of Service in a dynamically reconfiguring system has no advantage over the unavailability of a stastically reconfiguring system. Dynamic reconfiguration has been studied for its consistency, availability, and coexistence, but also somewhat studied for continuity or Quality of Service assurance. The gap between the state-of-the-art and Quality of Service assurance is significant due to the lack of the following

1. The whole spectrum of Quality of Service characteristics is studied and defined.
2. The Quality of Service assurance mechanisms is explored other than consistency and availability.
3. Quantitative benchmarking of related work is done for a clear understanding of a system’s Quality of Service in the aspects of different reconfiguration mechanisms.

C. Problem Statement

This project aims at bridging the gap by addressing the whole spectrum of Quality of Service assurance problems in terms of:

1. The whole spectrum of Quality of Service characteristics are described through detailed study.
2. Logical and physical requirements and conceptual modeling of each Quality of Service characteristic are analyzed.
3. Application-independent reconfiguration strategies, and realization of related work are designed by the strategies [4].
4. Generic reconfiguration benchmark to present the complete Quality of Service problems which are designed.
5. The reconfiguration strategies are benchmarked for their Quality of Service assurance abilities.
6. Given a criterion for quantitative evaluation and comparison of testing results.

Previously it has identified the limitations of existing work in Quality of Service impact analysis. It provided a framework to evaluate dynamic approaches for their impacts on the Quality of Service of running software systems [8]. This project will focus on the theoretical and technical aspects of Quality of Service assurance for dynamic reconfiguration of component based software systems. The contribution of this project is the identification of the properties of reconfiguration frameworks namely, dynamic version management (DVM), reconfiguration timing control, stateless equivalence, and controllability of overheads that are potentially useful for maintaining Quality of Service.

D. Objective of the Work

The main objective of the project work is to dynamically reconfigure a software system without any disruption of the application. The dynamic reconfiguration should be in a way that the Quality of Service is been maintained between the systems. There is service continuity in spite of the dynamic reconfiguration process over the components. The system is available during dynamic reconfiguration and there is no need for rebooting or shutting down of the system.

II. EXISTING SYSTEM

In static approach, a typical software update occurs by stopping a system to be updated by performing update of code and restarting the system. The system is not active during reconfiguration. The system gets to reboot when there is any updating made to the system. Three significant disadvantages with static approaches with respect to Quality of Service (QoS):

1. Shutting down could cause the system to lose accumulated state during the ongoing execution of the system;
2. Unavailability is unacceptable for long-lived or mission critical systems; and
4. In IT organization task assigned Process are confused and duplicate will occur in that process.
5. Static approaches lose its service continuity when there are any updating made to the software.

III. PROPOSED SYSTEM

Dynamic approaches are hence proposed as an alternative to static reconfiguration. Dynamic reconfiguration is a synonym of runtime evolution the ability to change a system’s functionality and/or topology while it is running. Dynamic reconfiguration typically involves changes at runtime via addition, deletion, and replacement of components, and/or alteration of the topology of a component system [5]. The advantage is application consistency and service continuity for the system while it is being updated. The motivation for dynamic reconfiguration comes from two aspects: adaptively and high availability, which are both Quality of Service driven. An adaptive system works in unanticipated environments and regularly adapts its behavior in reply to a changing environment. In such systems, scattered caching services, message disintegration services, and reliability necessities are subject to dynamic adaptation. In order to maintain system sensitivity (a Quality of Service requirement), services should only be deployed or removed under certain circumstances, and thus dynamic adaptation is needed. The advantages of the system is given as follows
1. To reduce the confusion of task assigned process.
2. Service continuity maintained
3. Adaptively to the changing environment
4. Continuous availability of software system.
5. To decrease the gap between economical and security viewpoints within an organization.

IV. ARCHITECTURAL PROPERTIES

In this section, the architectural properties are proposed for reconfiguration frameworks that are potentially useful for maintaining Quality of Service characteristics.

A. Modeling Service Continuity

a) Dynamic Version Management

Two versions (vold and vnew) are sufficient for version management of protocol-dependent transactions. DVM uses the following elements as the version carriers. The system as a whole is versioned; both connectors and application threads (representing transactions) are versioned. On this basis, DVM is designed to apply versioning to three aspects: shared structure, independent workflow, and timing. The feature of independent workflow on partially shared structure is fundamental to DVM. Structurally, in versioned connectors permit that a required-service can be connected to two provided-services (called a bi-link), of which one is the old service to be removed and the other is the new service just added. Consequently, the versioned connectors are to model a partially shared structure between the old and the new subsystem. While a component can be shared by the two subsystems, the versioned connectors are able to distinguish the two networks of components during coexistence. In workflow, a versioned application thread just works for a single protocol: either vold or vnew. If a required service is linked to a unique provided service by a single connector, the version of the connector is ignored by the thread. In this case, the application thread just follows the connector to invoke the service. However, if there are bi-links, the thread makes the decision, choosing the connector whose version matches its own version. Under such versioning, the partially shared structure supports two independent invocation chains, of which one works for the old protocol and the other works for the new protocol. These two invocation chains do not interfere with each other. Furthermore, because a new protocol can accommodate any number of protocol-dependent components, the feature supports dependent upgrades naturally.

b) Timing Control

The independent workflow is able to ensure that a data item is being processed by only one protocol, timing is important to ensure that the workflow is not physically blocked. We discuss the details of reconfiguration timing control in this section. The new and old application protocols are distinguished by two invocation chains. At the shared component, an application thread makes dynamic selection of an invocation chain by matching its own version with a connector’s version. The system determines timing (when starting the new version or stopping the old version). The system’s version is used as the global clock for timing control. First of all, suppose that the system keeps running in version vold. In this situation, all the connectors have version vold and all newly created application threads have vold. (A thread’s version remains unchanged throughout its life cycle.) For a system running in this situation, reconfiguration can start at any time. The first task of reconfiguration is to make (i.e., to load and initialize the new components, add the new connectors, and set the version of the newly added connectors to vnew) the new subsystem standby system’s version.

B. Modeling Quality of Service Assurance

Quality of System assurance requires reconfiguration to further possess both stateless equivalence and overhead controllability.

a) Stateless Equivalence and Constraints

Modeling stateless equivalence has two aspects. First, state should be transferred by redirecting references to state variables from the replaced component into the replacing component. If many objects make up the component’s state, these objects can be encapsulated into a single wrapper object. Consequently, state can be transferred by redirecting only a single reference, i.e., the reference to the wrapper object. By such encapsulation and redirection, state size is independent of application and a state transfer can be treated as virtually
instantaneous. Second, state sharing between the replaced component and the replacing component is also applicable to the problem of unintentional blocking. State sharing requires the following constraints:

1. The replacing component should provide backward compatibility of the state format for the replaced component.
2. Access to the shared state must be mutually exclusive.
3. The application logic permits state sharing.
4. The replacing and replaced components reside on the same physical machine.

Mutual exclusion is needed for access to the shared state and ensures atomicity of the state update and no corruption of state. However, whether the state is being used by the old or the new component, processing is contributing to the overall Quality of Service of the system, i.e., maintaining Quality of Service during the reconfiguration. The third constraint implies that state sharing is application dependent. State sharing is not applicable to every application, but we assume it can be used for a variety of applications. State sharing may be impossible and unintentional blocking could occur. We assume that state sharing is possible in most cases.

b) Controllability of Overheads

To address the controllability of reconfiguration overheads, a reconfiguration should be divided into three sequential phases which group different reconfiguration operations by CPU intensiveness.

1. The installation phase performs operations to load and initialize new components (including state sharing) and sets up new connectors.
2. The transformation phase switches workflow from version vold to version vnew, and traces ongoing transactions of version vold to completion.
3. The removal phase finalizes and garbage collects old components and deletes old connectors. Considering the operations in transformation phase, switching workflow versions amounts to resetting some flags in RAM and therefore can be treated as virtually instantaneous. Tracing the ongoing transactions to completion involves a wait operation. However, using thread synchronization, the reconfiguration thread yields its CPU via the wait operation and is awakened when a predefined condition, such as completion of a sub transaction at a component, is fulfilled. The wait operation is asynchronous and consumes little CPU time. Under such a classification, overheads of this phase can be ignored and it is safe to regard this phase as virtually instantaneous. However, both the installation phase and the removal phase involve CPU-intensive operations. On the basis of such a grouping, the controllability of overheads is achieved through thread prioritization and scheduling in consideration of current CPU usage. The components are divided with each individual work to be performed. Each component has a specific area of work flow. The work flow is continued only when the components are connected to each other through their interface. Each component is divided into their works that are illustrated below.

---

**Fig 2. System Architecture**

1. **Change Management**

Change management is transition of an existing state to new state. During the execution of program the desired state will be achieved by dynamic reconfiguration. Change management is closely related to configuration management, which is the process responsible for identifying, controlling, and tracking all of the elements of the system. Good configuration management ensures that only authorized components are used in the system and that all changes are recorded in a configuration management database. Effective change management allows you to introduce change into your system quickly and with minimal service disruption. Change management is responsible for changes in technology, systems, applications, hardware, tools, documentation, and processes, as well as changes in roles and responsibilities. A key goal of the change management process is to ensure that all parties affected by a particular change understand the impact of the impending change. Because most systems are heavily interrelated, any change made in one part of a system can have major impacts on others. Change management attempts to identify all affected systems and processes before the change is implemented so that adverse effects can be minimized.

2. **Component**

A component is a processing entity that encapsulates a set of functionality and data. Furthermore, a component has a provided-interface to specify a set of services that it provides for use by other components. Similarly, a component has a required-interface to specify a set of services that it needs. A service is a set of functionality, along with the policies that dictate its usage. A connector is a directed connection from a required-service of a component to a provided-service of another component, indicating the client/server relationship between the components. For an individual component, the client of its services can be an internal component of the same system or an external software agent, as long as they follow the service specification to make a request.
3. Dynamic Reconfiguration

Dynamic reconfiguration is a synonym of runtime evolution, the ability to change a system’s functionality and/or topology while it is running. Dynamic reconfiguration typically involves changes at runtime via addition, deletion, and replacement of components, and/or alteration of the topology of a component system. The benefit is application consistency and service continuity for the system while it is being updated. The motivation for dynamic reconfiguration comes from two aspects: adaptability and high availability, which are both Quality of Service driven. An adaptive system works in unanticipated environments and frequently adapts its behavior in response to a changing environment.

4. Component Transmitter

It is used to transfer file to every individual component. A stateful system can be regarded as stateless equivalent for reconfiguration if state transfer does not incur unintentional blocking of workflow.

5. Component Receiver

It is used to receive file from every individual component. The feature of independent workflow on partially shared structure is fundamental to DVM. Structurally, versioned connectors permit that a required-service can be connected to two provided-services (called a bi-link), of which one is the old service to be removed and the other is the new service just added. Consequently, the versioned connectors are to model a partially shared structure between the old and the new subsystem. The receiver component receives the data sent from the sender component. The whole process needs all the components to be connected with the interfaces of each component. In case the receiver component is not active even then the process is not stopped. Instead it intimates that the receiver is missing and once activated it starts the process from where the process was left. Then the completion of the process is done successfully.

V. RESULT

In this work, it has been analyzed that using dynamic reconfiguration the drawbacks of static reconfiguration has been solved. This dynamic reconfiguration technique solves the problem of missing the service continuity. Hence it keeps in track that all the components are connected to the interface for the service.

VI. CONCLUSION

Dynamic reconfiguration is a key aspect of application evolution, in terms of adding new functionalities, changing components relationships, and modifying the placement of components in a networked system. Reconfiguration refers to the modification of an application during execution, while preserving the service availability. Reconfiguration involves the creation, removal or replacement of components, the modification of interconnections and the migration of components. The challenge is to provide a dynamic reconfiguration mechanism that maintains application consistency while minimizing the impact on the running application. We have proposed two algorithms namely Symmetric Key Algorithm and Dynamic Reconfiguration Algorithm. Symmetric Key Algorithm is for encrypting and decrypting the file for transferring from various systems and it is checked whether the Quality of Service is maintained. The Dynamic Reconfiguration Algorithm is to dynamically reconfigure the system during the application disruption. Reconfiguration is thus reliable despite network or system failure. Furthermore, the reconfiguration algorithm does not have to care about global computation since it is achieved by a transaction manager with special isolation property from other transactions.

ACKNOWLEDGEMENT

Quality of Service assurance model the RCM which constitute future work. First-in-first-out order is important to some multimedia applications. RCM has no control over the order of request processing during coexistence and delegates this issue to the application layer. The state sharing issue is an issue for future research. It is to enable initial control of Quality of Service for state migration.

REFERENCES


