Routing Based Workflow for Construction of Distributed Applications*

Wanjun Huang, Xinhua Zhang, Uwe Roth, Christoph Meinel
Department of Computer Science, University of Trier
{huang, zhang, roth, meinel}@ti.uni-trier.de

Abstract

Dynamic reconfiguration is absorbing more and more research focus for its increasing demand in inconstant distributed application. In this paper we propose a routing based workflow to model the dataflow, runtime state and control management of cooperating components. Routing based workflow successfully realizes dynamic reconfiguration by the way of modifying routing structure and simplifies the hard problem of maintaining consistence into rather easy issue of synchronization. A detailed analysis is also given to show the great flexibility for construction of software architecture and potential applications.

1. Introduction

Distributed applications and services increase greatly as the popularization and development of Internet. The complexity of distributed system grown with dimension of application scope and areas makes the design and implementation of distributed system rather difficult. At the same time because of high speed updating of computer hardware and invention of new network technologies, new application services are frequently required to be integrated into existing distributed system, which demand the constitution of distributed applications easy to be extended. When designing and implementing a distributed application, people always try to complete all functionalities and services in advance. But if all components are loaded into memory when starting, the fat server will cost much unnecessary resources for some seldom used components. Architecture description language provides possibility to customize components for specific services according different circumstances and application areas, but this offline configurability can not satisfy increasing requirements. Dynamic reconfiguration is gradually becoming an indispensable feature of large scale system. Especially for the system which provides crucial services, it will produce big loss if the system shut down or restart. But sometimes faults are unavoidable, so defective component has to be replaced. Component parts may also have a new version, or even a new component is requested to integrate into the system. In this case, if distributed system has capability of dynamic reconfiguration, the loss can be decreased to minimum.

Workflow is first proposed to model the automation of a business processes according to a set of procedural rules where the document, information or task are passed from one participant to another for action. The core task for construction of distributed system is to design different components and organize these components working together. The kernel of workflow is processes management that is similar in essence to components collaboration. So we can also apply workflow to direct, coordinate and monitor execution of relevant core components in a distributed applications. For the dynamic reconfiguration, there are also plenty of proposals trying to give a final answer. These approaches do address some problems, but at the same time they also leave other problems. From idea of oil pipeline, we know, once the pipeline has been established, the oils can automatically flow to everywhere that the pipeline can reach. If we want to transport oils to a new oil station, what need to do is just to lengthen the pipeline to the new place, and other oil stations will not be affected. This idea looks apparently very simple and obvious, but it has not been realized in area of distributed application system. Now most approaches for dynamic reconfiguration adopt the idea of safe state that contains three steps: the first step is to hold incoming requests to wait all updating involved components into a safe sate, and then perform reconfiguration, finally restore to process held requests using new updated components. The kind of approach

* Proceedings of The Ninth IEEE Symposium on Computers and Communications (ISCC 2004). Alexandria, Egypt. June 28 - July 1, 2004. pp 80 – 85.

can deal with normal case, but it may wait a long time to process the reconfiguration when concerned component involves into a long time interaction or processing. According to the mentioned idea of oil pipeline, we propose a routing based workflow where a routing is constructed as pipeline of dataflow. When reconfiguration operation comes, what need to do is to modify the structure of routing that simplifies the hard problem of maintaining consistence into rather easy issue of synchronization. In this paper we first will introduce the concept and construction of routing based workflow. Then we will analyze and discuss in detail about features of routing based workflow for distributed application. At last related works will be given and compared to our solution.

2. Routing based Workflow

As indicated in Figure1, routing based workflow includes three parts: workflow manager, routing manager and component repository. Workflow manager is responsible to transfer an arrived request to a specified routing manager and get back the response from routing manager. When the routing manager receives a request from workflow manager, it dispatches a routing processor to concretely execute the routing for this request.

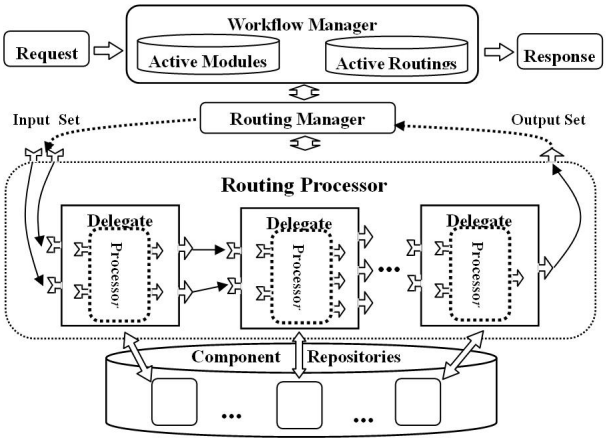

Figure1. Architecture of Routing Based Workflow

2.1 Description of Routing

Routing is a concept appeared with Internet technologies. In network protocols routing is an action of moving information across an internet from a source to a destination, whose functionalities are to determine optimal routing path and to transport information packet. In distributed environments, in spite of underlying system components or application services, there exist similar issues. To flexibly manage the inner structure of system or application services, we model the functional component as computational module, and model the running dataflow and control relations between different components as routing. Routing is data flow pipeline of workflow constructed from information contained in routing schema. Before the creation of active routing, all relevant modules have to be created and instantiated firstly. Then module delegates are constructed to encapsulate module processor and form the entity of routing. A routing is ready to execute only after virtual binding is performed, and real binding will occur at the momentary of request execution.

2.2 Routing Composition

Composition elements of routing are communication port, module delegate and input/output set. Input set indicates what are needed for execution of routing, and output set provide the way to export results produced by routing.

2.2.1 Communication Port

To achieve more convenience for modification of a routing, each computational module should keep in a state as independently as possible. The communication between different modules is realized by communication port. From the type of passed data, port can be distinguished as operation port or stream port, and from IO direction communication port can be identified as in port and out port. Operation port is used for conventional communication conformable to pattern of *request-reply*. Stream port is imported to process the continuous stream media, such as video, audio or other stream data. Inside of routing communication port is the unique data exchange media among different components. Communication port can be used in module processor and module delegate. For module processor, ports are not detachable. When the design and implementation of a module finish, the ports are fixed to processor. But for module delegate, these ports are detachable, through which delegate can represents different modules in different period. Two important part of communication port are destination object and source object. Destination and source object can be ports or named object. Just as in Figure1, all coupled ports should be bound to create a pipeline of dataflow. Destination indicates the next object where the passed data of port should be transferred, and source tells port

where the data can be got from. When ports are bound, there are some rules that have to been met to ensure the action of binding is correct. For example, if the owner of an in port is a module processor, the destination of in port must be a named object that is directly consumed by processor.

2.2.2 Module Delegate

Module Delegate encapsulates IO behaviors of module that it delegates, and represents the module for all management tasks in a routing. That means what can be seen in a routing is delegate instead of processor entity. All dependences operation and management concerned with a module can be done by its delegate. Inside of delegate, a mandatory module manager has to been given to indicate which module is being delegated. Delegate is like a container. What delegate contains is a module manager through which processor instance can be fetched or returned. Class of module delegate is unique, but the type of module that it delegates is countless. The detachable ports of delegate provide ability for this possibility. When module delegate instance is created, the ports of delegate should also be created according to ports of represented module. It's possible to hold different module delegates that represent the same module in different routings. In this case a module instance may serve for multiple routings synchronously.

Delegate's primary responsibilities are to help managing and controlling a routing in workflow. Two such major tasks are checking requirements and dependences management. To ensure a routing can be processed successfully, there are many requirements needing to be satisfied. Some are for routing, such as input set and output set should be a valid type and can be bind to its owner delegates. Some are for module processor that ensures it can be executed correctly. Dependences management is another task that delegate has to take charge. There are two dependences existing in routing based workflow. One is dataflow dependence has already been realized by binding of communication ports. The other is control dependence that is created according dataflow dependency and indicates the control relationship between current module and its neighboring modules.

2.3 Computational Module

Computational module consists of module manager, module processor and component repositories. Their essential responsibilities are to create connection between module delegate and its processor entity.

Module processor is the real processing component that implements the functionality what it provides. Module processor doesn't need to care about any functionality of collaboration and management with other modules. In addition to functionality implementation, the processor needs to complete two tasks. One is to create communication ports for its supplier and consumer to exchange data with itself. Another task is trigger mechanism for activating the execution of processor. In a routing processor will be loaded after real binding, and begin to execute when all in port data arrive.

Module manager is used to manage pool of module processors and keep contact with routing. Inside of manager, there is a pool to store processor instances that are available for multiple requests synchronously. Whenever there is no more available processor instance in pool and the size of pool has not reach its maximum, a new processor instance will be created to deal with new request. Also, if a processor has finished its execution for one routing, it will be reinitialized and retrieved back to pool. To optimize the usage of memory resource, module repository is designed to a two-layer storage structure. One is static repository that stores all available modules Schema. Another is active repository that stores instances of module processor that has joined in one or multiple routings.

2.4 Routing Execution

Routing execution is a procedure of serial real binding and unbinding, and only after virtual binding routing is ready to execute.

2.4.1 Virtual Binding

Virtual binding is a process to make the delegates of routing connected. After virtual binding the dataflow still can not run immediately in routing. Virtual binding has just established the pipeline of dataflow, and created the contact with specific module processor. Communication still needs a performing of real binding. The concrete tasks of virtual binding include port binding and validation check for rules. When a routing is created, the composition of routing are separated module delegates. Port binding is to connect delegates according provided binding pairs of coupled ports. In a pair of coupled ports supplier should be set to the source of its consumer and consumer should also be set to the destination of its supplier. If all ports of all delegates in a routing have been bound, a dataflow pipeline of routing from input set objects to output set objects has been created. To further enhance the

success rate of routing, some rule requirements should also be checked to validate the routing after port binding.

2.4.2 Real Binding

Virtual binding has created the dataflow pipeline of routing. But just like the example of oil pipeline, warehouse of oil station should also connect to pipeline. The responsibility of real binding is to bind the "oil station" with " oil pipeline" , and here the " oil station" is component repository and "oil pipeline" is a ready to execute routing. For the delegate and processor, the source of their in port and the destination of their out port are involved in the processing with outer delegates, so we call these as outer behaviors of its owner. Similarly the destination of in ports and the source of out ports are concerned with processor or the inner named object of processor, so we call these as inner behaviors of its owner. In the description of virtual binding, we have known that the outer behaviors of delegates have been bound to form dataflow pipeline. But after the stage of virtual binding, the inner behaviors of delegates are still separated. The concrete operations of real binding are to connect the inner behaviors of delegates to the outer behaviors of module processors. The inner behaviors of processor has already bound and fixed when it was created. So after the real binding, the routing can be said really ready for request processing. For a module processor, only when it is its turn to execute, it then processes the real binding and performs concrete functionality. After execution, processor will also process the operation of unbinding to release module processor.

3. Analysis and Discussion

In this section we analyze and discuss in detail the features and flexibilities of routing based workflow for construction of distributed applications.

3.1 Routing Structure

Routing is a graph oriented structure that enables routing bases workflow to flexibly establish different kinds of distributed application. Different routing structure represents varied potential software structure of application. The most common routing is depicted in Figure2 where each module delegate stands for respective module processor. In a valid routing out ports may point to in ports of different delegates and an out port may also be bound to multiple in ports, but each in port can only be bound to one out port.

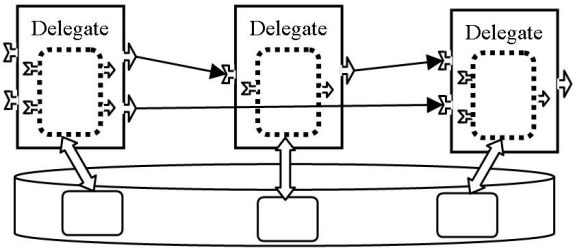

Figure2. Common Routing

One module processor may be shared by different delegates, and these delegates may also belong to different routings or the same routing. An example of this case is given in Figure3. Because of the enough independence of module processor and its contact way with delegate, the usage of shared module is same as normal one and need not any special measure and management except for pool synchronization.

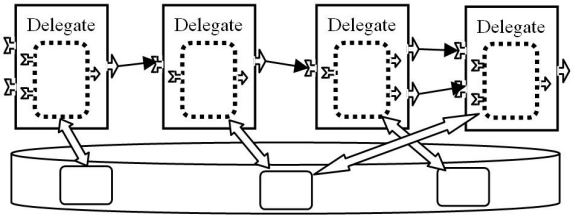

Figure3. Component Reused Routing

In some case, application may be involved in circulating operation between different components. In traditional solution these different components have to be merged into one module. In our routing based workflow, we can model this case using circulating routing as in Figure4. In this case one out port points to in port of its prior delegate or itself. Certainly the condition to break circulating has to be defined in module processor.

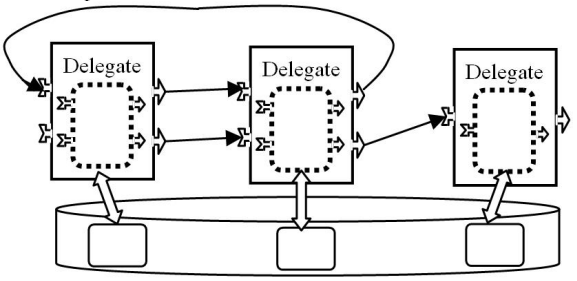

Figure4. Circulating Routing

3.2 Dynamic Reconfiguration

The motivation of routing based workflow is to acquire the capability of dynamic reconfiguration and it embodies in distributed system as changing the inner structure of software system, namely adding, deleting or modifying an inner component or methods of a

component. The difficult of dynamic reconfiguration is not just to realize these meta-operations, but also maintain consistence for involved components and recover the dependencies between updated component and its surrounding. The success of routing based workflow is that it has rather perfectly modeled the inner structure of system software and dependencies of involved components. In routing based workflow, the dynamic reconfiguration of distributed applications can be realized by manipulation on routing. And reconfiguration operations can be classified into three categories, routing updating, module updating and port updating. Routing updating involve the operations of adding a new routing and deleting a routing. Here we have not offered operation of modifying a routing because it can be achieved by the operation of module and port updating. Module updating includes adding a module, deleting a module and replacing the implementation entity of a module. When a module is added for a routing, the real operation is to add a delegate of this module into the routing. Also, deleting a module means to delete delegate of the module in a routing. Operations of port updating only contains enable a port and disable a port. We have introduced that the ports of module processor are fixed after the programming of module is completed. So here the port updating only refer to the port of module delegate. When disabling a port of a delegate, it means to hide the port of this module in current routing. Only after a port is already disabled, the operation of enabling for this port is available. One of big problems of traditional solutions is how to maintain consistency, because the updating component may be interacting with client and its neighbour components. Here our proposal transforms the consistency of involved components to the routing that simulates runtime environment for involved components. When updating a routing, it will first be duplicated. Only if the updating on duplicated routing success, the old routing will be replaced by updated one. So the hard problem of consistency is turned to easy issue of synchronization on routing.

3.3 Coupling degree

From observation of the law of nature, we detect that the configurability of everything is rather close with the coupling degree of its compositional parts. When the coupling degree is higher, the configurability is lower if appropriate method it has chosen. On the other side, if the coupling degree is lower, the configurability may be higher. Software developing also follows this rule. So we try to minimize the coupling degree of compositional components, not only inner components itself and also the management and control of components. Lowering the coupling degree of components does not mean decreasing its out behaviours. What we have done is just turning the black, uncontrollable communication way of components to a clear and manageable one and then naturally acquires higher flexibility and reconfigurable ability.

3.4 Application Analysis

Routing based workflow has reflected the dynamic dataflow, control dependencies and runtime sate for cooperating components. It can be applied for system level software development, such as middleware architecture construction that we are doing now. In this case routing based workflow may only model parts of system. Other components still have to be kept as non-reconfiguration part. Such as, communication protocol can not be modeled into routing based workflow for the nature of its function. In addition to system software, routing based workflow can also be applied for application level development. For example, web services can be assembled to a composite service or services application with the help of routing based workflow.

4. Related Works

Allen et al [1] separate the dynamic re-configuration behavior of architecture from its non-reconfiguration functionality, and provide a notation to precisely interpret each of these aspects. Through analysis to the notation for consistency and completeness, it is possible to guarantee that reconfiguration occur only at points in the computation permitted by the participating components and connectors. Allen's approach based on the Architecture Description Language (ADL) is only passive to know when it's better to make the reconfiguration and has not addressed the problem of consistency maintaining. In [2], [3], Almeida et al in detail analyze the situation when dynamic reconfiguration occurs and subdivide the problems to three concrete requirements: structural integrity, mutually consistent states and application state invariants and propose individual solutions for different requirements. This approach can be said rather successfully solved the problem of maintaining consistency, but it is also not available for extreme case. Goudarzi et al [5] also present a similar approach as Almeida to preserving mutual-consistency, and give a description of the intended changes, automatically

identify and forces components which will be affected by the change into a safe state. In [9], [10], Kon et al design a C*omponent Configurator* to model and manage the runtime dependencies of components. *Component Configurator* just records the dependences of component, but it has no consideration of the consistency, and the communication way between components is also not available. In our approach the design of control dependency is influenced by the concept of *Component Configurator*. Rather similar idea with us can be found in [11], Shrivastava et al present a workflow based model for distributed applications. In their model, workflow schema is used to express the structure of tasks in a distributed application and task controller is used to represent temporal dependences between constituent tasks. In our approach the design of module delegate are much effected by the idea of task controller. But Shrivastava et al have not modeled the structure of workflow execution and they also directly bind the component implementation to task controller, so the multi-solutions supporting are not available in his solution.

5. Conclusion

For the attractive application prospect, dynamic reconfiguration of distributed system absorbs quite a number of research activities, but there is still no a recognized stable solution that has addressed a serial of problems arisen by dynamic reconfiguration. In this paper we have proposed a routing based workflow for construction of distributed applications. Routing based workflow has successfully modeled the inner structure and dependencies of cooperating components. In a routing all involved components are represented by their own delegate. Only when it's time to execute, the module processor will be loaded, and it will again be released to component repository when execution finishes. Because of low coupling degree of components, all components have gained much independence and at the same time do not lose the flexibility of collaboration. Through modification of routing structure, the capability of dynamic reconfiguration is automatically realized. At the same time some hard problems, such as maintaining consistency etc., are skillfully simplified.

References

[1] R. Allen, R. Douence and D. Garlan. *Specifying and analyzing dynamic software architectures*. In Fundamental Approaches to Software Engineering, volume 1382 of LNCS, pages 21-37. Springer-Verlag, 1998.

[2] J.P.A. Almeida, M. Wegdam, L. Ferreira Pires, M. van Sinderen. *An approach to dynamic reconfiguration of distributed systems based on object middleware*. In Proceedings of 19th Brazilian Symposium on Computer Networks (SBRC'01), Florianó polis - Santa Catarina, Brazil, May 2001.

[3] Maarten Wegdam, Joao Paulo A. Almeida, Marten J. Van Sinderen, Lambert J.M. Nieuwenhuis. *Dynamic Reconfiguration for Middleware–based Applications*. Submitted to IEEE Transaction on Parallel and Distributed System, Special Issue on Middleware, 2003.

[4] T.Batista and N.Rodriguez. *Dynamic Reconfiguration of Component-Based Applications*. In Proceedings of International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE 2000), Limerick, Ireland, June 10- 11, 2000.

[5] K. Moazami-Goudarzi. *Consistency-Preserving Dynamic Reconfiguration of Distributed Systems*. PhD thesis, University of London, Department of Computing, Imperial College of Science, Technology and Medicine, 180 Queen's Gate, London SW7 2BZ, UK, 1997.

[6] C. Hofmeister and J. Purtilo. *Dynamic Reconfiguration in Distributed Systems -Adapting software modules for replacement*. In Proceedings of the 13th International Conference on Distributed Computing Systems, pages 101–110, Pittsburgh, May 1993. IEEE Computer Society Press.

[7] K. H. Kim, Chittur Subbaraman. *Dynamic Configuration Management in Reliable Distributed Real-Time Information Systems*. IEEE Transactions on Knowledge and Data Engineering 11(1): 239-254, 1999.

[8] J. Suzuki, T. Nakano, K. Fujii, N. Ikeda and T. Suda, *Dynamic Reconfiguration of Network Applications and Middleware Systems in the Bio-Networking Architecture*. Proc. of IEEE Workshop on Large Scale Real-Time and Embedded Systems, Austin, TX, December 2002.

[9] Fabio Kon, Manuel Román, Ping Liu, Jina Mao, Tomonori Yamane, Luiz Claudio Magalhã es, and Roy H. Campbell. *Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflevtive ORB*, IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000). New York. April 3-7, 2000.

[10] Fabio Kon and Roy H. Campbell, Dependence Management in Component-Based Distributed Systems. IEEE Concurrency, 2000. 8(1): p. 26-36.

[11] Shrivastava, S. and Wheater, S., Architectural Support for Dynamic Reconfiguration of Large Scale Distributed Applications. The 4th International Conference on Configurable Distributed Systems (CDS'98), Annapolis, Maryland, USA, May 4-6 1998.