2002-07

Adaptive Middleware Framework for Distributed Embedded Applications

Luqi; Qiao, Y.; Luqi; Qiao, Y.

Naval Postgraduate School

Luqi and Y. Qiao,"Adaptive Middleware Framework for Distributed Embedded Applications", Technical Report, NPS-SW-02-010, NPS Monterey, CA, July 2002.
https://hdl.handle.net/10945/65204

NPS-SW-02-010

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Adaptive Middleware Framework for Distributed Embedded Applications

Technical Report

By

Luqi, Ying Qiao

July 2002

Prepared for: NSF

PROJECT SUMMARY

The object of proposed work is to provide an adaptive middleware framework for component-based applications in distributed embedded systems. Through this middleware, programmer can make service invocation with Quality of Service (QoS) requirements in embedded applications to give a run-time support for dealing with various QoS requirements such as real-time performance and dependability of the applications. To accomplish this task, a set of models, specification languages and methods will be developed in this middleware.

As the start point, the component model is developed. This model reflects the concurrency and communication abstractions and admits time as a first–class concept. Furthermore, QoS properties and dependency relations among components are introduced in this model.

To describe QoS requirements of service invocation, we plan to develop a service invocation model. This model focuses on describing the real-time performance and the desired dependability level of the service invocation. By this model, the timing constraints on the service invocation, the desired the dependability level of this service invocation, the dependability level actually provided by the system, system conditions needed to be monitored, possible measured or observed QoS and actions to take in case of the desired QoS requirement can not be satisfied will be revealed.

Basing on the component model and service invocation model, a QoS management mechanism is given out to improve the system robustness and dependability. QoS management is proposed to have two responsibilities. One is to adjust the system behavior in case of the QoS requirement of service invocation can not be satisfied, another is to adjust the end-system when system constraints are changed due to the change of interaction environment. In the first case, a framework is developed to support the QoS run-time monitor and system behavior adjustment. And following this framework, we focus on developing the fault-tolerance methods and model-based self-configuration methods. In the second cases, an adjustment mechanism should be presented to respond to the changed timing constraints on embedded applications.

Another feature of the proposed middleware is its support for the service invocation. This effort will give more flexibility to the applications since it allows components in the systems to be reconfigured transparently without having effecting on applications. For this purpose, a heuristic method should be studied to do the component connection so that "the most suitable component" can be selected to fulfill the invoked service.

To avoid the side-effect on the dependability of applications caused by dynamic dependency among components, we will provide a dependency management mechanism basing on dependency run-time monitoring.

In summary, the proposed work will give out followings:

(1) A component model written by "the modeling language".
(2) A service invocation model specified by a set of quality description language.
(3) A QoS management framework to adjust the system behavior when the desired QoS requirement of service invocation can not be satisfied
(4) An adjustment mechanism in response to the changed non-functional constraints on embedded applications.
(5) Fault-tolerance methods for each component and a model-based self-configuration method.
(6) Component connection method basing on heuristic searching.
(7) Dependency run-time monitoring mechanism to manage the dynamic dependency among the components.

Through using the proposed middleware in distributed embedded systems, the QoS requirements of remote invocation can be dealt with at run-time, and the embedded applications can maintain the performance in the change of interaction environments. These will give an efficient way to support the robustness and dependability of embedded applications at run-time. Moreover, this middleware also results in a more flexibility to the applications and makes it easy to upgrade the system. Another benefit of using this proposed middleware is that the burden on application programmers can be reduced since the middleware hide the tedious and error-prone programming related to ensure the QoS requirements of applications at run-time.

PROJECT DESCRIPTION

C.1 Objectives and Significance

The main objective of this proposed research is to provide an adaptive middleware framework to support the building of dependable component-based applications in distributed embedded systems.

Embedded systems require increasingly distributed and extremely dependable. At the same time, component-based software is an promising one in the development of distributed embedded system since it aims at decreasing development time and costs by creating applications from reusable, easily connectable and exchangeable components. Although a lot of methods have been developed to ensure the non-functional properties of component-based embedded applications before their deployment, run-time support is still needed to guarantee various Quality of Service (QoS) requirements such as real-time performance and dependability of these kinds of applications. This is because the interaction environment is always changed and a lot of behaviors such as the process of remote invocation can not be described before execution. Thus, we propose to present an adaptive middleware framework to provide run-time support for building dependable component-based applications in distributed embedded systems.

This middleware allows heterogeneous components in distributed embedded systems to interoperate by means of service invocation with QoS requirements. It provides an adaptive framework for embedded applications to adjust their behaviors according to the state of system QoS and the QoS which can be provided to the service invocation at run-time. This apparently improves the dependability and robustness of whole applications since it prevent the application from failure in the case of the service invocation can not be satisfied.

A main feature of embedded systems is their tightly couple with interaction environment. Their constraints may change at run-time because of the change of environment especially the timing constraint. Fail to response to the change of constraints will result in the undependability of embedded application. Thus, in this middleware, we will provide a mechanism to flexibly configure the resource allocation policies whereas to cope with the side-effect resulting from changed constraints.

C.1.1 Significance and Benefits

This proposed research will enable following to occur:

(1) Enable service invocation with QoS requirements among heterogeneous components in order to provide run-time support for the dependability of component-based embedded applications
(2) Enable adaptive adjustment of application behavior according to the state of system QoS and the provided QoS for service invocation whereas to prevent application from failure in case of failing to satisfy service invocation QoS requirements
(3) Eliminate side-effect resulting from changing constraints at run-time
(4) Allow transparently dynamical components reconfiguration in the distributed embedded systems without modification of applications
(5) Reduce the burden on application programmers by relieving them of tedious and error-prone programming related to ensure the QoS requirements of applications at run-time

C.1.2 Technical Barriers

Dependable component-based applications in distributed embedded systems are hard to build. This is because:

(1) QoS requirements maybe change during the execution of an application
(2) Some non-functional constraints of embedded applications such as timing maybe change because of the change of physical environment.
(3) Besides function constraints, there are several non-function constraints such as timing constraints, computational resource limitation during the interaction of components in distributed embedded systems.

(4) Dynamic dependency among components may cause side-effect on the dependability of embedded applications

However, conventional distributed object computing(DOC) middleware are limited for the building of dependable component-based applications in distributed embedded systems. The limitation is as followings:

(1) Components in embedded systems are active while object is passive. Thus, the object abstraction in conventional DOC middleware can not be mapped directly to the component abstraction.
(2) Conventional DOC middleware doesn't allow applications to specify their end-to-end QoS requirements.
(3) Conventional DOC middleware is lack of real-time features.
(4) Conventional DOC middleware is lack of performance optimizations.
(5) Conventional DOC middleware is lack of memory footprint optimizations.

Thus, what is needed is to provide an adaptive middleware framework to support the building of dependable component-based applications in distributed embedded systems at run-time. This middleware will allow heterogeneous components interoperate by means of service invocation with QoS requirement. And it can also provide capability to response to the changing constraints of embedded applications.

C.2 Technical Approach

C2.1 Component Model and Specification

In order to give out the adaptive middleware framework, component model should be studied at first. Unlike the component in general systems, component in the embedded systems is active and it is a process which heavily interacts with physical environment. Some actor-oriented component models [18] have been presented to emphasize the concurrency and communication abstractions and admit time as a first–class concept. In this case, the components are parameterized actors with ports. Ports and parameters define the interface of an actor. A port represents an interaction with other actors while parameters represent methods that the actor provides.

Our proposed research will extend this component model. Firstly, some QoS properties should be introduced to the parameters. These properties include the worst-case execution time, failure rate in last certain period of time etc. Secondly, since dynamic dependency among components has a direct impact on the dependability of embedded applications, we should describe the dynamic dependency in the component model. Thus, we can use the port to indicate this kind of relation. In this context, there are several "client ports" and "hook ports". Client ports describe all the components that possibly depend on this component and hook ports describe all the components on which this component possibly depends. Thus, all possible dynamic dependencies between this component and other heterogeneous components in the system can be revealed through these ports. Thirdly, in addition to using ports to describe the relation between the components, we will also include the non-functional properties such as timing constraints for each relation especially for the remote invocation.

Furthermore, we will develop a "component modeling language" to describe the component model mentioned above. Thus, the component model should be specified using this "component modeling language" in the design phase. Together with service invocation model, it will form the basis for component connection, dependency management and QoS management.

C2.2 Service Invocation Model and Specification

In our proposed work, we will use service invocation instead of method invocation to fulfill the remote invocation to the heterogeneous components. This effort will result in more flexibility of applications. To reduce development cost, component reconfiguration is popular in embedded systems. Components are often modified, added and deleted. Since service invocation does not explicitly indicate which components and which methods

should be involved in, it allows applications to execute without modifying the application code in case of component reconfiguration, i.e., using service invocation allows components in the systems to be reconfigured transparently without having effecting on applications.

In order to allow user to specify the service invocation, a service invocation model should be provided. This model not only describes the function of service invocation but also describes its QoS requirements. It includes the functional part and QoS part.

In functional part, users can only specify the service name. And the system will maintain a service list index by service name. Every entry in this list is a description for each service provided by the components in the systems. This description includes the required data to this service, the data they desire to get from this service and the process description of this service.

In QoS part, the model focuses on describing the real-time performance and the desired dependability level of the service invocation. To describe the real-time performance, the model should specify timing constraints on this service invocation. The timing constraints can be induced from the specification of component model. To describe the desired dependability of service invocation, different dependability level should be included in the model. However, how to define the dependability level is application-dependent. In the simplest case, we can define three levels of dependability which is active, correctness and timely. Furthermore, in QoS part, the service invocation model will also specify the dependability level that this service invocation actually obtains from the system, system conditions needed to be monitored, possible measured or observed QoS such as the failure rate of components according to desired dependability level, and actions to take in case of the desired QoS requirements can not be satisfied.

An important issue in the service invocation model is that we should study what system conditions should be monitored at the run-time. The selection principle is that the system conditions which have direct impact on the desired dependability level. For example, for the "active" dependability level, the occurrence of component's cash failure should be monitored while for the "timely" dependability level, the time the service invocation result is generated should be recorded.

Furthermore, in order to describe the service model, a set of quality description language will be provided. These quality description languages will specify all the factors mentioned above. Thus, the programming developer can use this language to specify the service model in applications whereas to allow applications to make service invocation with QoS requirements in the high level summary.

C2.3 QoS Management

QoS management is proposed to have two responsibilities. One is to adjust the system behavior in case of the QoS requirement of service invocation can not be satisfied, another is to adjust the system behavior when the system constraints are changed.

In the first case, QoS management is proposed to monitor the specified system QoS state during the service model running and give suitable adjustment policy according to the system QoS which can be provided for the service invocation. QoS management includes a QoS monitor and QoS controller. The QoS monitor is responsible to measure some specified QoS by checking specified system conditions and submit monitored event to the QoS controller. Thus, QoS controller derives the dependability level which can be provided for this service invocation from the monitored event. And also, QoS controller will select the suitable adjustment policy to adjust system behavior in response to the monitored event according to the desired dependability level specified in the service invocation model and the monitored system QoS state. In this proposed work, fault tolerance and self-configuration after the detection of fault is a typical process according to the framework of QoS management mentioned above.

Firstly, to improve the robustness and dependability of the system, we should hide fault-tolerance mechanism within this middleware. The fault tolerance methods are based on redundancy. We suppose to tolerate three types of

fault, i.e., crash fault, value fault and timing fault. As a basis, we can explore the replica of the component [1]. Thus, there are several replication groups, each group is composed of one or more identical components.

To tolerate crash fault, the method of broadcasting message among each member of replication group [2] can be used. To tolerate the value fault, we can provide a voter to compare the several computation results sent by replicas in the replication group [1]. And, to tolerate timing fault, we can use a monitor to record information regarding various times and omissions [1].

One feature in our fault tolerance is its adaptability. We expect to adjust the system configures according to desired dependability requirements of service invocation which is specified in the service invocation model, i.e., we will translate the high-level dependability requirements into a system configuration with an adequate level of fault tolerance. For this purpose, we should present a method to manage the replicas of components. And basing on this management mechanism, we should present a method to decide how to provide fault tolerance based on the desired dependability requirements of service invocation at run-time. This includes how to choose a style of replication, the type of voting to use, the degree of replication, the type of faults to tolerate, the location of the replicas etc.

Basing on the mechanism given out above, the QoS monitor can monitor the occurrence of the failure. As long as this event is detected, the fault report should be provided. Then, the QoS controller will do system adjustment. We propose to provide a model-based self-configuration method to archive this purpose.

The model-based self-configuration method can provide the automatic graceful degradation in response to the fault detected by the fault-tolerance mechanism. Self-configuration will select degradation strategy according to the detected error and the dependability requirement of service invocation. The main degradation strategies are components reassignment, component re-connection and component killing. Any kind of faults among the three can result in component reassignment and component killing. And only crash fault will result in component re-connection.

Component reassignment means relocating the faulty component to another node. This can be archived simply by the heuristic searching for the most suitable component to relocate the component [4]. However, this method will results in NP-complexity. Thus, we plan to develop a model-based components reassignment method. In this method, we will firstly study a modeling approach to describe the load of the nodes in the system. State chart can be considered as a modeling approach. Furthermore, we should also provide a searching algorithm to search the state chart of the system component load whereas to find the most suitable component to relocate the component.

For the component re-connection policy, when the crash fault of the component is detected, in additional to component reassignment, the component re-connection should be processed. Thus, we can choose "the second suitable component " to fulfill the indicated service invocation. Moreover, component killing means killing the suspended component replica according to the fault report sent by QoS monitor.

In second case, since the embedded systems are featured by heavy interaction with physical environment, the change of environment will result in the change of system constraints. For example, consider a sensor processing application with sampling operations and processing operations on different end-systems. Depending on the environment and application state, sensor sampling operations may run at any one of a set of rates. The change of sampling rate will result in the change of timing constraints on accordingly tasks. Fail to response to the changed constraints may incur the system failure. Thus, in this proposed work, QoS management will also provide the capability to adjust the end-system to maintain the changed constraints. In response to the changed timing constrains, we plan to adjust the local scheduling method on the end-system to ensure the scheduling feasibility. And according to the component model, we can capture the relation between the components whereas to negotiate with related end-systems to adjust the local scheduling methods for ensuring the changed end-to-end timing constraints.

C2.4 Component Connection

When the applications make service invocation, the middleware should search "the most suitable" component to fulfill this service, i.e., component connection should be made at this point. For this purpose, we will make use of the component model, service invocation model and the observed state of system QoS to make decision about which one is the most suitable component to fulfill the invoked service.

In general speaking, following factors should be considered when choosing component to fulfill the invoked service:

(a) Fault rate during the last certain period of time for each component
(b) Estimated worst execution time for each component

From the specification of component model, we can capture the worst execution time of related components' methods while by the service invocation model and observed state of system QoS, the fault rate during the last certain period of time for the component can be captured. Thus, we propose to involve the above factors in a heuristic function and develop a heuristic method to search "the most suitable component to fulfill the component connection.

C2.5 Dependency Management

In distributed embedded system, different programmers create components, often working in different groups with different methodologies. It is hard to create dependable applications if they do not understand the dynamic dependencies between components. And, at the same time, dynamically reconfigurable software can be used effectively in the design of embedded systems to provide many major advantages over conventional software development techniques. The benefits include rapid development which reduces development time and cost, ability to easily upgrade and maintain embedded software, and both analytic and practical tools for automated analysis and systematic fine tuning of the embedded system. However, since the dynamic dependencies among components exist in the system, reconfiguration of single components (i.e., creation of new component, destruction of a component, modification of a component and replacement of a component implementation) will have side-effect on the dependability of embedded applications if it can not be managed properly.

Thus, in this proposed work, we will provide a method to manage the dynamic dependencies between components in distributed embedded systems. The key technology that should be studied in this topic is how to do the run-time monitoring for the dynamic dependency. For this purpose, a mechanism called dependency run-time monitor will be developed.

Dependency run-time monitor includes event recognizer, impact pattern and dependency checker. According to the component model, dynamic dependency of can be reveal by the ports. Thus, basing on this, each reconfiguration of components generates an event, and event recognizer will indicate which components will be impacted by this reconfiguration according to component model. After that, an impact pattern can be generated according to the results obtained by event recognizer. This impact pattern reveals the potential impacts on components incurred due to each component reconfiguration. Thus, at the run-time, dependency checker will check the impact pattern when specific service invocations are made and give information about the status of components that can be used during component connection.

C2.6 Other Related Issues

C2.6.1 Request Scheduling Method

In distributed embedded systems, several processes can make service invocation concurrently. To ensure the timing constraints during the interaction of components, we will provide a scheduling method to schedule these service invocation requests. This can be taken as a typical dynamic real-time scheduling problem. So far, many algorithms are presented to dynamically schedule tasks in real-time embedded system. We can exploit some priority-based scheduling methods to fulfill the schedule of service invocation requests. There are two points should

be considered in this topic. One is how to decide the priority of each service invocation request, another is which algorithm should be taken to do the schedule decision for these service invocation requests.

The priority of the service invocation request can be simply inherited from the task which sends this request. And regarding to the scheduling algorithm, several typical dynamic real-time scheduling algorithm can be considered, such as EDF (Earliest Deadline First) and RM (Rate Monimonic).

C2.6.2 Communication Services/protocol

We will develop several communication services/protocols to support the communication of each infrastructure in the proposed architecture, such as communication between the replica of each component, communication between the fault tolerance mechanism and self-configuration mechanism etc. These protocols will support the translations of exchanged information involved in the communication of infrastructures whereas to provide a uniform interface to every infrastructure in this proposed middleware.

C.3 General Plan of Work

C3.1 Plan Overview

We plan to make use of some communication protocols such as IIOP in CORBA and take it as a start point to do additional research. Thus, we should investigate the current technologies in CORBA and study the non-functional properties of software component and the QoS requirement of the remote invocation in distributed embedded systems.

Considering the non-functional properties of software component in distributed embedded systems, component model should be firstly developed. Furthermore, paying attention to the QoS requirements such as real-time performance and dependability in the remote invocation, we will develop a service invocation model. Basing on the component model and service invocation model, QoS management is presented. This mechanism will be implemented in two layers. In the first layer which is to monitor the state of system QoS and adjust the system behavior in the case of the desired QoS requirement of service invocation can not be satisfied, we plan to give the framework of QoS management. And also, we will provide the fault-tolerance method and self-configuration mechanism according to this framework. In the second layer of QoS management, we will develop a mechanism to adjust the end-system to maintain the changed constraints such as end-to-end timing constraints.

Since we use the service invocation instead of method call during the remote invocation, the component connection method should be studied in order to choose "the most suitable component" to fulfill the invoked service. Moreover, to manage the dynamic dependency among components in distributed embedded systems, we should give out method to do dependency run-time monitoring. And some other related issues such as scheduling method for the service invocation requests and communication service/protocol will be also developed.

To use the test-bed, we can use this proposed middleware in the real distributed component-based embedded systems. By doing some experiments, some metrics such as reliability metrics, availability metrics, timely response can be evaluated.

C3.2 Broad Design Activities

The steps to develop the adaptive middleware framework for building dependable component-based applications in distributed embedded systems are as followings:

(1) Study the current technology in the CORBA.

(2) Investigate the non-functional features of component-based applications in distributed embedded system and the QoS requirements for the service invocation
(3) Study the component model and create the "component modeling language.
(4) Study the service invocation model and create the quality description language to specify service invocation model.
(5) Develop the QoS management mechanism which includes two layers mentioned above.
(6) Study heuristic method to fulfill the component connection.
(7) Develop the method to do dependency run-time monitoring, including the construction of event recognizer and the approach to do dependency checking.
(8) Develop the real-time dynamic scheduling algorithm to schedule the service invocation request sent by client components
(9) Develop several communication protocols for the interaction between different infrastructures in the architecture.

C3.3 Deliverables

(1) A component model written by "the modeling language".
(2) A service invocation model specified by a set of quality description language.
(3) A QoS management framework to adjust the system behavior when the desired QoS requirement of service invocation can not be satisfied
(4) An adjustment mechanism in response to the changed non-functional constraints on embedded applications.
(5) Fault-tolerance methods for each component which can tolerate three kind of faults, i.e., crash faults, value faults and timing faults
(6) Model-based self-configuration methods
(7) Component connection method which makes use of heuristic searching
(8) Dependency run-time monitoring mechanism to manage the dynamic dependency among the components
(9) A real-time dynamic algorithm to schedule the service invocation request
(10) Several basic communication services/protocols for the communication among infrastructure in the middleware framework

C3.4 Description of Procedure

Based on the strengths in the proposed research work, we will perform the followings:

(1) Provide a specification language for the programmer to specify the QoS requirement of remote service invocation in the applications
(2) Provide a QoS management mechanism to adjust system behavior in case of the desired QoS requirement can not be satisfied and non-functional constraints on applications are changed
(3) Construct a mechanism to take charge of component connection whereas to fulfill the service invocation
(4) Give a mechanism to support the dynamic reconfiguration without affecting the dependability of the applications and modifying the application code

To achieve the work described above, we will take the following steps:

(1) Further study the non-functional constraints on the component-based applications in the distributed embedded systems and the QoS requirement of service invocation
(2) Study the component model in distributed embedded systems and its specification language
(3) Study the service invocation model and its quality description language Develop component dependency model and run-time dependency monitoring approach.
(4) Study the QoS management framework to adjust the system behavior when the desired QoS requirement can not be satisfied

(5) Develop the fault-tolerance approach to tolerant three kind of fault, i.e. crash fault, value faults and timing faults.
(6) Construct the mechanism to fulfill the model-based self-configuration.
(7) Study the method to adjust the end-system when the timing constraints on the applications are changed
(8) Develop a heuristic method to fulfill the component connection
(9) Study the approach for dependency run-time monitor to support the dynamic dependency management
(10) Use this middleware to the test-bed ----- the distributed component-based embedded system to evaluate its performance, such as measuring the effect on the dependability of embedded application resulted by this proposed middleware.

C3.5 Evaluation Factors

C3.5.1 Reducing the development cost and cycle-time

As the technology of object-oriented and the distributed system is more and more popular, how to improve the robustness and dependability of the component-based applications in the distributed embedded systems becomes an important issue. Providing run-time support for dealing with QoS requirements of service invocation is an efficient way to building the dependable applications. However, this needs a lot of work to be done. The proposed middleware resides between client and service applications and services in complex software systems. Most of the tedious and error-prone work is encapsulated in this proposed middleware. Only thing the applications should do is to specify a service invocation model while most of the work is transparent to them. This dramatically reduces the complexity of the applications and the burden on the programmer. Accordingly, the cycle-time and effort required to develop high-quality real-time and embedded applications and service is highly reduced.

C3.5.2 Improving the Flexibility of Applications

Components in distributed embedded systems are highly autonomous. Component reconfiguration is a typical activity in distributed systems because of its benefit for the rapid development. Since the proposed middleware provide a service invocation style instead of the method call to do the remote invocation, it implicitly indicates the component which fulfills the invoked components rather than explicit indications. Thus, it is no necessary to modify the code of applications during the component reconfiguration. This makes component reconfiguration transparent to applications whereas to improve the flexibility of applications.

However, since dynamic dependency exists in the system, component reconfiguration may result in the undependability of applications if there is lack of proper dependency management. To deal with this problem, the proposed middleware provides a dependency management mechanism basing. on dependency run-time monitoring. This mechanism ensures the change of component due to reconfiguration can be captured by applications whereas to maintain the correctness of component connection. Thus, it provides an efficient way to improve the flexibility of applications without resulting in the side-effect on the dependability of embedded applications.

C3.5.3 Making Applications Portable

The embedded systems are featured by its intensive interaction with physical environment. It is possible that the embedded systems are ported to a different physical environment. In this case, some non-functional constraints on the embedded applications such as timing constraints will be changed due to the change of physical environment. Fail to response to this change will result in the system failure. In the proposed middleware, the QoS management mechanism is responsible to adjust the end-system according to the change of timing constrains. Through this mechanism, embedded applications can run in different physical environment without causing the system failure. This will make embedded applications easier to port to different physical environment whereas to reduce the development cost and improve the reuse of the embedded applications.

C3.6 Schedule

(1) Study the current technology in the CORBA. (1 month)
(2) Investigate the non-functional properties of component-based applications in distributed embedded system and QoS requirement of service invocation (1 month)
(3) Study the component model and its specification language. (3 months)
(4) Study the service invocation model (5 months)
(5) Develop the quality description model for service invocation model. (2 months)
(6) Develop QoS management framework to adjust the system behavior in case of the desired QoS requirement can not be satisfied (5 months)
(7) Develop the method to adjust the end-system when the timing constraints on applications are changed (5 months)
(8) Study the approaches to tolerant three kind of fault, i.e. crash fault, value faults and timing faults (4 months)
(9) Study approach to fulfill model-based self-configuration. This will include the error description language, modeling the component load and searching algorithm. (4 months)
(10) Develop run-time dependency monitoring mechanism, including the construction of event recognizer and the approach to do dependency checking (3 months)
(11) Develop heuristic method to fulfill the component connection (3 months)
(12) Develop service invocation request scheduling algorithm (3 months)
(13) Develop several communication protocols for the interaction between different infrastructures in the architecture. (6 months)
(14) Do experiments on the test-bed (3 months)

C3.7 Comparison with Other Research

Work related to the topics discussed in this paper includes research in the areas of QoS management, component dependency management, fault detection and fault recovery, scheduling algorithm and networking protocols.

There are several typical distributed object middleware have been presented. The Common Object Request Broker Architecture (CORBA)[23] is a standard for distributed object computing and is the broadest distributed object middleware available in terms of scope. It offers heterogeneity across programming language and vendor implementations. DCOM[24] is another distributed object middleware. Its distributed object abstraction is augmented by other Microsoft technologies, including Microsoft Transaction Server and Active Directory. DCOM provides heterogeneity across language but not across operating system or tool vendor. COM+ is the next-generation DCOM that greatly simplified the programming of DCOM. SOAP is a distributed object framework from Microsoft that is based on XML and HyperText Transfer Protocols (HTTP). Its specification is public, and it provides heterogeneity across both language and vendor. Microsoft's distributed object framework .NET also has heterogeneity across language and vendor among its stated goals. Java has a facility called Remote Method Invocation (RMI) that is similar to the distributed object abstraction of CORBA and DCOM. RMI provides heterogeneity across operating system and Java vendor, but not across language.

Quality Objects (QuO) was the first middleware framework to provide general-purpose and extensible quality of service for distributed objects[36,37]. TAO was the first major CORBA system to provide quality of service , namely real-time performance, directly in the ORB[38]. Furthermore, there are several other researches related to provide QoS management in the middleware. [27] applies the QuO to the problem of distributed real-time embedded applications. Basing on the QuO, [1,2] present a method to make the remote object with dependability requirement, and introduce the fault-tolerance into the QuO. [22] provides a formal support for dynamic QoS management in the middleware including the specifying and verifying QoS management subsystems. [31] also presents a QoS management method in the middleware for embedded computing systems. It provides an open-source integrated middleware framework that supports adaptive distributed admission control strategies and mechanisms for end-to-end QoS management in real-timing embedded middleware.

However, among above researches, some focus on the object abstraction instead of component abstraction while others only focus on general distributed systems rather than real-time embedded systems. Thus, in this proposed work, we present a middleware which can capture the non-functional features of components in distributed embedded systems. Basing on this, we also introduce the QoS management into the middleware. Compared to previous researches, the QoS management in this proposed middleware has multi-layer functions. It integratedly considers the change of system QoS itself and the change of the interaction physical environment instead of considering only one factor. This can make embedded systems more robust and more dependable.

Moreover, most of current middleware use method call to do remote invocation. This will result in the inflexibility of applications because it has to modify the application code during the component reconfiguration. If the component reconfiguration is very frequent in the system, the situation is even worse. Furthermore, since this kind of invocation lack of knowledge about the state of the component which is indicated to be invoked, using this style also will incur the uncertain factor to the dependability of applications. However, our proposed middleware plans to use service invocation as the style of remote invocation. Since this style does not explicitly indicate which component or method should be involved in the remote invocation, no matter how the component will be reconfigured, the application codes are no necessary to be modified. And also, by the heuristic searching during the component connection, a benefit for building the dependable component-based applications can be derived.

In the distributed environment, dynamic dependency should be dealt with carefully, otherwise it will have side effect on the dependability of applications because of the frequent component reconfiguration. Some researches such as [16] solved this problem by providing a separate dependency management mechanism. In our research, instead of separating it, the dependency management is encapsulated in the middleware. Furthermore, unlike the previous research [16] which provides the specific class to deal with the dynamic dependency, our research fulfill dependency management basing on component model and dependency run-time monitoring so that it gives a more efficient way to prevent the side-effect of dynamic dependency on the dependability of applications.

In the QoS management of our middleware, one important issue is the fault-tolerance method. Since a lot of work has been done in this topic, we will study some "suitable" method for the distributed embedded applications after the deep investigation of previous methods. The provided fault-tolerance method should consider the non-functional constraints of applications. Furthermore, our proposed research plans to provide different methods to tolerant different kinds of faults and associates the fault-tolerance method to the dependability requirement of service invocation whereas to improve the adaptability of embedded applications.

Self-configuration in our proposed research plays an important role in improving the system robustness, some previous researches such as [4][5] provide an efficient way to fulfill self-configuration. However, this method is NP-complexity since its simply component reassignment policy based on heuristic searching. Our proposed research plans to provide a model-based self-configuration method to reduce the complexity.

C.4 Broader Impacts

If this proposed architecture can be established successfully, the following can take place:

(1) Allow the application to make remote service invocation with QoS requirements whereas to provide a run-time support for building dependable component-based applications in the distributed embedded systems.
(2) Improve the robustness of distributed embedded systems by the adjustment in case of QoS requirements of service invocation can not be satisfied and the change of non-functional constraints on the embedded applications.
(3) Allow transparently dynamical components reconfiguration in the distributed embedded systems without modification of applications and eliminate the side-effect on the dependability of applications resulting from the component reconfiguration.

(4) Encapsulate several mechanism which is used to ensure the dependability of applications at run-time in the middleware whereas to reduce the complexity of embedded applications

C4.1 Transition of Technology

Technology transfer will be addressed by integrating the proposed new capabilities with existing middleware such as CORBA. Publishing results in OMG, ACM, and IEEE sponsored conferences and making toolkits readily available can facilitate acceptance.

The software Engineering Group at the Naval Postgraduate School offers M.S. and Ph.D. degrees. The students at NPS will contribute to this research and develop effort. We also plan to integrate emerging technologies into the courses we teach.

C4.2 Experimentation and Integration Plan

The work will be performed by the faculty of the Software Engineering Group at the Naval Postgraduate School and their Ph.D. and M.S. Students. The principle investigators will be responsible for coordination of the following plan previously stated in section C.3.6 for schedule:

(1) Study the current technology in the CORBA.
(2) Investigate the non-functional properties of component-based applications in distributed embedded system and QoS requirement of service invocation
(3) Study the component model and its specification language.
(4) Study the service invocation model
(5) Develop the quality description model for service invocation model.
(6) Develop QoS management framework to adjust the system behavior in case of the desired QoS requirement can not be satisfied
(7) Develop the method to adjust the end-system when the timing constraints on applications are changed
(8) Study the approaches to tolerant three kind of fault, i.e. crash fault, value faults and timing faults.
(9) Study approach to fulfill model-based self-configuration. This will include the error description language, modeling the component load and searching algorithm.
(10) Develop run-time dependency monitoring mechanism, including the construction of event recognizer and the approach to do dependency checking
(11) Develop heuristic method to fulfill the component connection
(12) Develop service invocation request scheduling algorithm
(13) Develop several communication protocols for the interaction between different infrastructures in the architecture.
(14) Do experiments on the test-bed

REFFERENCES CITED

[1] Michel Cukier, Jennifer Ren, Chetan Sabnis, David Henke, etc, AQuA: An Adaptive Architecture that provides dependable distributed objects, Proceedings of the 17th IEEE symposium on reliable distributed systems, West Lafayette, Indiana, USA, October 20-23, 1998.pp.245-253.

[2] Jennifer Ren, Michel Cukier, etc.,Building Dependable Distributed Applications Using AQuA, Proceedings of SCTF'2001 - IX Brazilian Symposium on Fault-Tolerant Computing, Florianópolis, Santa Catarina,Brazil,March5-7,2001.

[3] J.P. Loyall, R.E. Schantz, J.A. Zinky, D.E. Bakken, Specifying and Measuring Quality of Service in Distributed Object Systems, Proc. of ISORC'98, Kyoto, Japan, April 1998.

[4] William Nace, Philip Koopman, A graceful degradation framework for distributed embedded system. *Workshop on Reliability in Embedded Systems* (in conjunction with Symposium on Reliable Distributed Systems/SRDS-2001), October 2001, New Orleans, LA.

[5] David B. Stewart and G. Arora, Dynamically Reconfigurable Embedded Software---Does it make sense?, Proc. Of IEEE Intl. Conf. On Engineering of Complex Computer Systems and Real-time Applications Workshop, Montreal, Canada, oct 21-25, 1996. pp.217-220.

[6] David B. Stewart, Richard A. Volpe and Pradeep K. Khosal, Design of Dynamically Reconfigurable Real-time Software Using Port-Based Objects, IEEE Transactions on Software Engineering, vol.23, no.12, Dec. 1997.

[7] David B. Stewart and Robert A. Brown, Grand Challenges in Mission-Critical Systems: Dynamically reconfigurable real-time software for flight control systems,

[8] Andrea Bondavalli, Alessandro Fantechi, Diego Latella, Luca Simoncini, Design Validation of Embedded Dependable Systems,IEEE Micro, Sep---Oct, 2001.

[9] Sdavid B. Stewart, Designing Software Components for Real-time Applications, 2001 Embedded Systems Conference, San Francisco, CA, April 2001.

[10] John A. Stankovic, VEST A Toolset For Constructing and Analyzing Component Based Embedded Systems, Lecture Notes in Computer Science, http://citeseer.nj.nec.com/stankovic00vest.html.

[11] Anders Wall and Christer Norstrom, A Component Model for Embedded Real-time Software Product-Lines, http://www.mrtc.mdh.se/publications/0330.pdf.

[12] Uwe Rastofer, Frank Bellosa, An Approach to Component-Based Software Enginnering for Distributed Embedded Real-time Systems, http://www4.informatik.uni-erlangen.de/Publications/pdf/Rastofer-Bellosa-SCI2000-RT-Components.pdf

[13] Jarmo Kalaoja, Eila Niemela and Harri Perunka, Feature Modeling of Component-based Embedded Software, 8th International Workshop on Software Technology and Engineering Practice (STEP '97) (including CASE '97), July 14-18, London, UK.

[14] Edward A. Lee, Embedded Software---- An Agenda for Research, ERL Technical Report UCB/ERL No. M99/63, University of California, Berkeley, CA, USA 94720, December 15, 1999.

[15] Thais Batista and Noemi Rodriguez, Dynamic Reconfiguration of Component-based Applications, International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE 2000), 10-11, June, 2000, Limerick, Ireland

REFFERENCES CITED

[16] Fabio Kon and Roy H. Campbell, Dependency Management in Component-based Distributed Systems, IEEE Concurrency, January---March 2000.

[17] Philip T Cox, Baoming Song, A Formal Model for Component-based Software, http://www.cs.dal.ca/~pcox/publications/Components-HCC01.pdf

[18] Edward A. Lee, Embedded Software, ACM SIG\-PLAN Notices, http:// citesseer.nj.nec.com/497408.html

[19] David Urting, Yolande Berbers, Stefan Van Baelen, Tom Holvoet, Yves Vandewoude and Peter Rigole, A Tool for Component Based Design of Embedded Software, 40th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Pacific 2002), Sydney, Australia.

[20] Dieter K.H ammer, Component-based Architecting for Distributed Real-time Systems, http://www.win.tue.nl/oas/architecting/papers/ component-based-drts-architecting.pdf.

[21] Shige Wang and Kang G. Shin, An Architecture for Embedded Software Integration Using Reusable Components, Proceedings of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, San Jose, CA. 2000.

[22] L.Blair, G.S. Blair, A. Andersen and T. Jones, Formal Support for Dynamic QoS Management in the development of Open Component-based Distributed Systems, IEE Proceedings of Aspect-oriented and Component-based Software Engineering, 2001

[23] Siegel J., OMG Overview: CORBA and OMA in Enterprise Computing, CACM, Vol.41, No.10,1998.

[24] Microsoft Corporation, Distributed Component Object Model Protocol, Version 1.0. Redmond, Washington,1998

[25] Arnold K., O'Sullivan B., Scheifler R.W., Waldo J., and Wollarath A., The Jini Specification. Addison-wesley, 1999.

[26] Douglas C. Schmidt, Middleware Techniques and Optimizations for Real-time, Embedded systems, Proceedings 12th International Symposium on System Synthesis, 2000.

[27] Joseph P. Loyall, Richard Schantz, partha Pal, John Zinky, Michael Atighetchi, Emerging Patterns in Adaptive, Distributed Real-time, Embedded Middleware, http://www.dist-systems.bbn.com/papers/2001/ OOPSLAPatterns/bbnloyall.doc

[28] A. David Mckinnon, Olav Haugan, Tarana R. Damania, David E. Bakken, John C. Shovic, MicroQoSCORBA: A QoS-Enabled, Reflective, and Configurable Middleware Framework for Embedded systems, http:// microqoscorba.eecs.wsu.edu/ MicroQoSCORBA-November2001.pdf.

[29] Divid E. Bakken, Middeleware, Chapter in *Encyclopedia of Distributed Computing,* Kluwer Academic Publishers, 2001

[30] Echhardt Holz, Specification and Design of Distributed Embedded Middleware Applications with SDL , http:// www.omg.org/news/meetings/embedded2001/ abstracts/Sanjose.pdf

[31] Christopher D. Gill, David L. Levine and Douglas C. Schmidt, Towards Real-time Adaptive QoS Management in Middleware for Embedded Computing Systems, http:// www.cs.wustl.edu/~cdgill/PDF/ HPEC.pdf

REFFERENCES CITED

[32] Venkita Dubramonian and Christopher Gill, Experiences with Middleware for a Networked Embedded Software Technology Open Experimental Platform, http://www.omg.org/news/meetings/rtembedded2002/abstracts/nestomg_jan02.pdf

[33] Greg Eisenhauer, Fabian E. Bustamante, Karsten Schwan, A Middleware Toolkit for Client-Initiated Service Specialization, Principles of Distributed Computing (PODC 2000) Middleware Symposium, July 18-20, 2000.

[34] Joseph K. Cross, Quality Connectors: A Declarative Meta-Programming Technique to Optimize Distributed Real-time and Embedded Middleware, IEEE Distributed Systems Online, http://dsonline.compouter.org/0107/features/cro0107.htm

[35] Cristian Borcea, Deepa Iyer, Porlin Kang, Akhilesh Saxena, and Liviu Iftode, Middleware for Cooperative Computing in Large Ad Hoc Networks of Embedded Systems, IEEE Distributed Systems Online, http://dsonline.computer.org/0107/features/bor0107.htm

[36] J.Zinky, D. Bakkenk, and R. Schanta, "Architectural Support for Quality of Service for CORBA Objexts", Theory and Practice of Object Systems, 3:1, April 1997.

[37] Joseph P. Loyall, Richard E. Schantz, John A. Zinky, David E. Bakken, Specifying and Measuring Quality of Service in Distributed Object Systems, Proceedings of ISORC'98, 20-22 April 1998, Kyoto, Japan.

[38] D. Schmidt, D. Levine, and S. Mungee, "The Design of the TAO Real-time Object Request Broker." Computer Communications, Elsevier Science, 21:4, April, 1998.