This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2008 proceedings.

Active Message Oriented Adaptation Middleware for Collaborative Applications in Heterogeneous Environments

Shengpu Liu
Computer Science and Engineering
Lehigh University
Bethlehem, USA
shl204@lehigh.edu

Liang Cheng
Computer Science and Engineering
Lehigh University
Bethlehem, USA
cheng@cse.lehigh.edu

***Abstract*—Adaptation middleware is becoming widely used to build adaptive collaborative applications. However, collaborative applications requiring real-time services are intolerant of the long reconfiguration time of the existing adaptation middleware, which is in a range of seconds or even tens of seconds. In this paper, we present MARCHES, which is active message oriented adaptation middleware that reduces the reconfiguration time. Different from the traditional middleware that supports the single component-chain based application architecture, MARCHES maintains multiple component chains or actuators. Then the process of architecture reconfiguration is done by a new method of switching active and inactive actuators, which replaces the traditional method of modifying the single-chain architecture. An active message based synchronization protocol is proposed according to the new method to reduce the communication overhead and reconfiguration time. Experiment results demonstrate that MARCHES improves the packet delivery ratio and throughput of collaborative applications. Results also show that the reconfiguration time achieved by MARCHES is in a range of hundreds of microseconds and the extra costs introduced by the multi-actuator architecture are extremely low.**

Keywords- Middleware; adaptation; active messages

I. INTRODUCTION

A distributed collaborative application is a set of programs that help human beings, software, or hardware work together to fulfill certain tasks in networked collaboration environments. Recently, there is a need to migrate traditional collaborative applications, which previously run in homogenous computing platforms and network infrastructure, to heterogeneous environments due to the popularity of portable devices and advances of wireless communication techniques (e.g. Wi-Fi). Such a heterogeneous environment with mobile devices and wireless links challenges the performance of the applications because of its dynamic feature of resource availability.

One approach to addressing the performance issue is to make collaborative applications adaptive by using adaptation middleware, which offers the following benefits. Firstly, middleware can be used to facilitate the implementation of complex applications so that developers can pay more attention to the application logic and architecture design. Secondly, because middleware abstracts the low-level details of network operations and interfaces, it supports development of generic distributed applications by providing connections among distributed software components.

Adaptation middleware uses the component-based metamodel to build adaptive collaborative applications. The applications consist of a set of function-independent interacting components, which form a component chain. For each distributed program of a collaborative application, existing adaptation middleware supports only one component chain, which is also named as an actuator in this paper (Fig. 1a). The actuator can be dynamically reconfigured via chain-structure modifications by the middleware corresponding to the run-time contextual information of the heterogeneous environments so that the application can be adaptive to environment changes. However, collaborative applications with real-time data are intolerant of long reconfiguration time of the existing adaptation middleware since each reconfiguration process includes operation suspension, buffer clearance, and chain-structure modifications that take seconds or more in total [1].

In this research, we propose MARCHES (Middleware for Adaptive Robust Collaborations across Heterogeneous Environments and Systems), which solves the critical issue of the reconfiguration time by using active messages and supports context-aware application-layer adaptation. Different from any existing middleware, MARCHES supports multiple actuators in each program of a collaborative application (Fig. 1b). Thus modifying the actuator in traditional middleware is replaced by switching active and inactive actuators in MARCHES based on the active messages. This results in a dramatic reduction of the reconfiguration time by eliminating the operation suspension time and buffer clearance time. Furthermore, the robustness of the distributed application is improved since there is no communication and system halting in the distributed actuator-synchronization process by using the active messages. The costs introduced such as extra resource consumption and active message overhead are negligible to the computing platforms including mobile devices as validated by our experiments.

In the rest of this paper, Section II presents the details of MARCHES, Section III describes the experiments used for evaluating MARCHES, Section IV covers the related work, and Section V concludes this paper.

This research is supported by the U.S. National Science Foundation (Award# 0438300). The project website is http://marches.cse.lehigh.edu/.

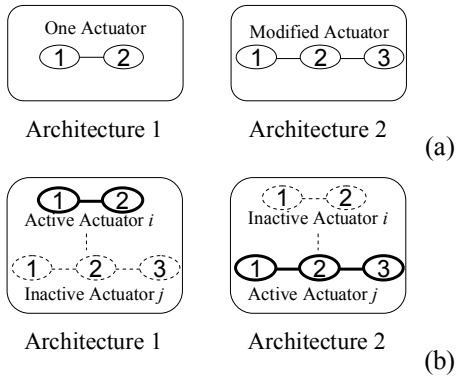

Figure 1. Dynamic reconfiguration: (a) single-actuator architecture in existing middleware, (b) multiple-actuator architecture in MARCHES.

II. System Architecture of MARCHES

As shown in Fig. 2, MARCHES is located between the lower hardware and network layer and the upper application layer to monitor environments and support application adaptation. It is peer-to-peer middleware and there is one middleware agent per application in each host. There are five parts in each MARCHES agent: measurement tools, event sensors based on a hierarchical event model, a script parser based on XML, a decision engine, and a dynamic reconfigurator. Measurement tools monitor the heterogeneous environments and report the context awareness results; and the contextual information will be processed by the event sensors. The sensors and actuators, in addition to adaptation rules, are defined by application developers in a XML script file. There are two types of actuators, proactive and reactive ones. The XML script parser parses the script file and constructs the sensors and proactive actuators to process local data. The reactive actuators are constructed through a synchronization process with peer agents to process the received data. Once a context triggers an event sensor, a corresponding proactive actuator will be activated.

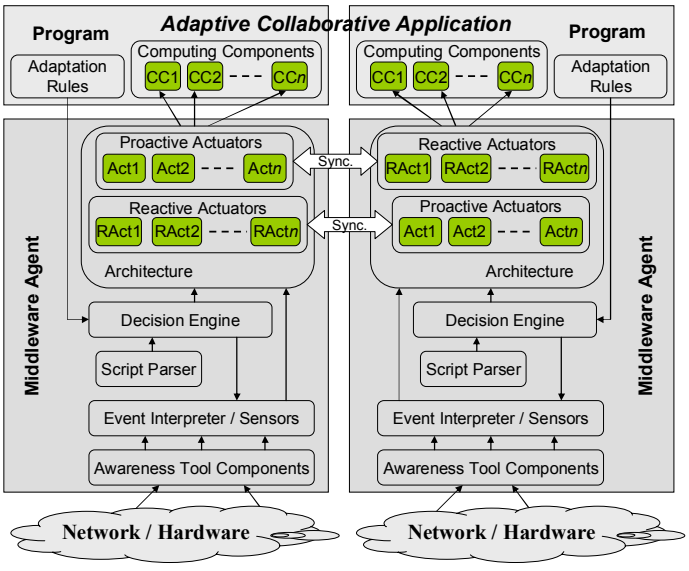

Figure 2. System architecture of MARCHES.

A. MARCHES Components

Components in MARCHES are function independent computing units that implement and provide some interfaces. Each component has a component interface that fetches a unique ID (e.g. the name and version pair) from its attributes file described in IDL (Interface Description Language) so that the component can be identified by MARCHES.

Reconfigurable computing components (named as marchlets) are the basic units to construct MARCHES actuators. Each marchlet has a *comm* interface for communication purpose. The *comm* interface provides both message based and function based communication modes for the marchlet to interact with other marchlets. In the message based mode, the output interface of a marchlet notifies the subscribed input interfaces of other marchlets through messages after the input data are processed. This mode facilitates the parallel processing in marchlets and it is suitable for the actuators that contain virtual components. In the function based mode, all the data processing functions are connected and invoked one by one by the actuator in the same thread. If all the marchlets are in a local host, the function based mode serves the actuator better with smaller communication overhead than the message based mode.

Context awareness for adaptive applications has been studied in our previous work [3]. MARCHES facilitates the reuse and extension of existing measurement tools and integrates them via an awaretool component metamodel that accepts the registration of event sensors as listeners and notifies them through an awaretool interface. Thus measurement tools can be implemented as independent components (awaretools) that can be used and extended in MARCHES.

B. Adaptation Rule Script

To use MARCHES to build an adaptive application, developer(s) need to provide a script file that divided into a declaration part and an adaptation-rule part as shown in Fig. 3.

```
<Marchlets>
 <component name="Grab" version="1.0" type="Proactive">
  <alias> GRAB </alias>
  <param name="CaptureWidth"> 160 </param>
  <param name="CaptureHeight"> 120 </param>
 </component>
 ......
</Marchlets>
<MarchTools>
 <component name="AvailableBW" version="2.0" type="Tool">
  <alias> AVAILABLEBW </alias>
  <param name="packetSize"> 64 </param>
  <param name="packetNum"> 2 </param>
  <param name="Interval"> 300 </param>
 </component>
 ......
</MarchTools>
<Rules>
 <rule>
  <sensor> AvailableBW GT 10 && AvailableCPU LT 1.0 </sensor>
  <proActuator>
   <marchlet name="GRAB">  </marchlet>
  </proActuator>
  <reActuator>
   <marchlet name="DISPLAY"> </marchlet>
  </reActuator>
 </rule>
 ......
</Rules>
```

Figure 3. An example of the adaptation rule script.

The declaration part declares all components used in the local program and middleware agent. Based on the declaration, the MARCHES agent loads and instantiates the components, and initializes them with the provided parameters. The adaptation rule part contains adaptation rules and each rule can be further separated into three sections: a sensor, a proactive actuator, and an optional reactive actuator. The sensor section can be parsed by the event interpreter to build an event sensor that accepts the subscription of the proactive actuator declared in the proactive actuator section. Each proactive actuator consists of a list of marchlets that have been declared in the marchlets segment with a parameter list for each marchlet. The reactive actuator section describes the corresponding actuator of a peer agent that processes the received data from the proactive actuator of the peer agent, so that the actions of the proactive and the reactive actuators can be synchronized.

C. *Composite Event Model*

Each sensor in MARCHES is build upon a hierarchical event notification model where several contextual events can be integrated as a binary event tree to form a sensor. In the constructed event tree, there are two types of event nodes as shown as dashed frames in Fig. 4: leaf nodes called simple events and branch nodes called composite events. A simple event has one *event source* that reports the awareness results spawned from awaretools / awarefuns to a *compare conditioner* located in the same event node; and the comparison result will be sent to its subscribers. A composite event has two *event sources*, which are lower layer simple or composite events, and a *Boolean conditioner*, which does the Boolean operation of the event source reports. An upper layer *Boolean conditioner* or an actuator can subscribe to an event. Thus the sensor can monitor and process the awareness results based on the event tree, and report interested events to its subscribed actuator and trigger the application architecture reconfiguration at run time.

D. *Component Reconfiguration and Synchronization*

Conventional component-based adaptation middleware uses one component chain or actuator per application at each host. Thus the operation suspension and buffer clearance in the reconfiguration process introduce large overhead and delay.

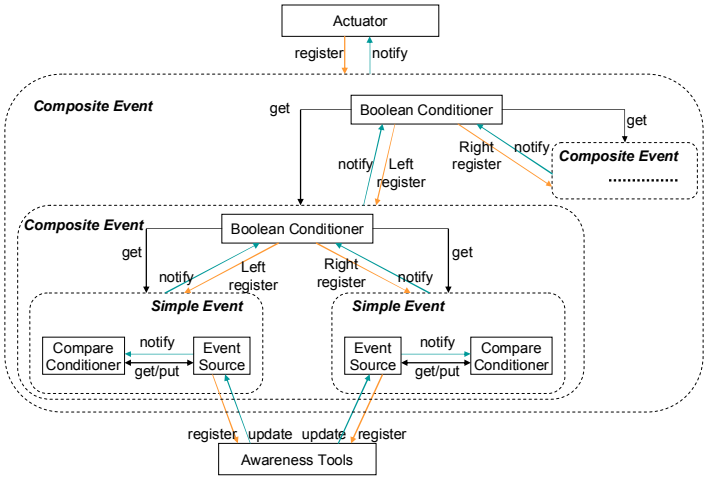

Figure 4. The binary tree based hierarchical event notification model.

By contrast, our middleware uses multi-actuator architecture with shared components as shown in Fig. 1. Proactive actuators are constructed when the adaptation rules are parsed by connecting the references of the marchlets, which are instantiated when the marchlets segment is parsed. To reduce the resource consumption by the multiple actuators, each actuator only consists of a list of pointers that point to the marchlets instances and maintains a customized parameter list for each marchlet reference. Thus all actuators share the marchlets and only one of the actuators is active at any time to process the application data. And the actuator modification process in the conventional single-actuator architecture is replaced by the switching process of the active and inactive actuators in the MARCHES architecture. When the context changes and the condition of a new sensor is met, the current active actuator is either stopped immediately after the component states are stored to its parameter list or deactivated after its currently task is completed. And the new actuator subscribing to the new sensor will be reinitialized by its parameter list and activated to process the application data.

Since each distributed program of a collaborative application has its own component chain or actuator, architecture synchronization is a crucial service provided by the middleware for dynamic reconfiguration to achieve behavior consistency among the distributed component chains. We have designed an efficient synchronization protocol in MARCHES using active messages with the following initialization steps.

- In the initialization phase of a middleware agent, proactive actuators are constructed based on the script file. Each proactive actuator is associated with a middleware-assigned unique index and the architecture information of an optional reactive actuator.
- The middleware agent of the proactive actuators sends a *synchronization request* packet to each collaborative peer agent. It contains the indices of the proactive actuators and the architecture information of the reactive actuators.
- After receiving the *synchronization request* packet, the peer agent constructs the reactive actuators, each of which is associated with the IP address of the packet sender and a middleware-assigned unique index carried by the packet.
- The receiver or the peer agent returns the sender a *synchronization response* packet that includes related index pairs, each of which contains an index of the proactive actuator and the index of the reactive actuator.
- The sender agent replaces the architecture information of each reactive actuator with the corresponding index received from the synchronization response packet.

The initialization is a one-time process for each peer agent. Then the middleware agent of the proactive actuators appends the index of the reactive actuator, which corresponds to the current active actuator, to the payload of each data packet. The peer agent receiving the data packet activates the reactive actuator indexed by the received index to process the data.

The active message based synchronization protocol has four advantages: low overhead, short delay, high efficiency, and better robustness. Only the index of the reactive actuator needs

to be stored in the active message header of each data packet. By using the actuator switching method, the system does not need to be paused in the reconfiguration process, which dramatically reduces the reconfiguration time. Based on the information in the active message header, a peer agent can process the received packets by choosing the correct reactive actuator. Therefore no buffered clearance is needed for reconfiguration. Moreover, once the reactive actuators are constructed, the packet receiver agent does not need to be re-synchronized with the sender agent when the architecture of the sender agent is reconfigured. Thus the application's robustness is improved and communication overhead is reduced.

III. Implementation and Performance Evaluation

MARCHES aims at improving the performance of collaborative applications under heterogeneous environments with vigorously dynamic features by facilitating application adaptation through architecture reconfigurations. Since the reconfiguration process introduces some performance cost such as extra resource consumptions to maintain the multiple actuators and application response time to accommodate reconfiguration delays, it is important to check the feasibility of using MARCHES by studying its performance cost in terms of the reconfiguration time and resource consumption, and the benefits of using MARCHES by evaluating the performance gain in terms of package delivery ratios and throughputs.

A. Testbed

In our experiments, we setup a small testbed by using two routers (Cisco 3200), two switches (Cisco Catalyst 2900XL), two laptops (Thinkpad-X60: Intel T2300 1.66GHz, 512MB PC2-5300, and Windows XP), and two PDAs (Dell x51v: Intel XScale 624MHz, 64MB, and WM5) to simulate heterogeneous environments. The routers are connected back-to-back through their serial interfaces with a maximum bandwidth of 1300Kbps using a DCE/DTE cable and the serial link as the bottleneck can have its bandwidth changed on-the-fly manually.

We implement a video-conferencing application based on MARCHES where proactive actuators prepare and send video frames and reactive actuators receive and display the frames.

There are four marchlets (*Grab*, *Compress*, *Decompress*, and *Display*), two awaretools that measure the available bandwidth between the laptops and the available CPU resource respectively. The application architecture can be dynamically reconfigured by using or not using the *Compress* marchlet, or set different compression ratio according to three adaptation rules. For example, when the available bandwidth is less than 10Mbps and the available CPU resource is larger than 1.0GHz, the event sensor will activate the proactive actuator for reconfiguration. The video rate is fixed as 2fps. The maximum frame size is 36910 bytes (128×96 pixels) so that a frame can be sent in one packet to avoid application-layer segmentation.

B. Dynamic Reconfiguration Time

The reconfiguration time or delay indicates the level of middleware's responsiveness to environments. Table I describes the notation used in the analysis of the dynamic reconfiguration time T_{dr}, which is expressed as:

$$T_{dr} = T_{event} + T_{arch} + T_{sync} . \quad (1)$$

where $T_{sync} = T_{reco} + T_{init}$ for the first reconfiguration process and $T_{sync} = T_{reco}$ for later reconfigurations.

TABLE I. Parameters for Reconfiguration Time Analysis

Notation	Parameter
T_{event}	**Composite event notification time**: the time period between the time when the context changes and the time when the architecture is notified for reconfiguration
T_{arch}	**Architecture change time**: the actuator switch time in MARCHES
T_{sync}	**Synchronization time**: the time period that the reactive architecture is adapted to the proactive architecture change
T_{dr}	**Dynamic reconfiguration time**
T_{init}	**Initialization time:** one-time initialization time at the beginning of system construction
T_{reco}	**Reactive architecture reconfiguration time**

TABLE II. The Dynamic Reconfiguration Time of MARCHES

Scenarios		Average Time (μs)	Standard Dev. (μs)	Confidence Int. (α = 0.5)
T_{event}	one rule and one level	161.7	8.895067	1.89725
	one rule and two level	315.3	10.44616	2.228086
	two rules and one level	322.7	7.242621	1.544796
	two rules and two level	423.8	52.38702	11.17375
T_{arch}		36.2	0.421637	0.089932
T_{sync}	T_{init} (one time init.)	825.9031	144.2641	30.77042
	T_{reco} (reactive reconf.)	33.7	0.948683	0.202347
T_{dr}	one rule and one level	231.6	8.771165	1.870823
	two rules and two level	493.7	52.60344	11.21991

To better evaluate the performance of the hierarchical event model, four types of event sensors are tested: one adaptation rule with a one-level event tree (a simple event), one adaptation rule with a two-level event tree (a composite event), two adaptation rules with a one-level event tree for each rule, and two adaptation rules with a two-level event tree for each rule. The experiment results, which are calculated based on 10 measurements, are listed in Table II.

We observe that the reconfiguration time of MARCHES in our test bed is only in a range of hundreds of microseconds. Comparing to the reconfiguration time of seconds or tens of seconds in other adaptation middleware [1], the responsiveness of our middleware is significantly better. Moreover, as a major contributor of the reconfiguration time, the event notification time is directly proportional to the number of the adaptation rules and the complexities of the event sensors.

C. Resource Consumption and Active Message Overhead

The resource consumption R can be expressed as follows where P_{ijk} is the size of parameter k for marchlet j in acutator i (10 bytes as default), l_{ij} is the reference and name size of marchlet j in actuator i (12 bytes as default), and a_i is the index size of actuator i (8 bytes as default).

$$R = \sum_i \left(\sum_j \left(\sum_k p_{ijk} + l_{ij} \right) + a_i \right) . \quad (2)$$

For the MARCHES agent in our experiments that contains 3 actuators as described in Fig. 3, the resource consumption is 164 bytes. For a more complicated agent that contains 5 actuators, 10 marchlets for each actuator, and 10 parameters for each marchlets, the resource consumption is 5640 bytes (≈ 5.5 KB) that is still fairly small for most mobile devices hosting megabits or gigabits memories. The overhead induced in the one-time synchronization initialization process includes the synchronization request and response packets that are much smaller than the payload size of a data packet. After the initialization, only a 1-byte active message header is appended to each data packet to store the index of a reactive actuator.

D. Throughput and Packet Delivery Ratio

Experimental results show that the adaptive application can always achieve higher throughput than the non-reconfigurable application with compression because it transmits high-quality videos in the high-bandwidth condition and adaptive-quality videos in the low-bandwidth condition while the non-reconfigurable application always transmits low-quality videos regardless of the network status. The non-reconfigurable application without compression achieves extremely low throughput in the low-bandwidth condition as most packets are dropped due to congestion; and it can achieve higher throughput than the adaptive application at the beginning of the bandwidth-increasing period since some packets buffered at the router in the previous congestion period are now delivered.

IV. Related Work

Based on the placement of the adaptation, the adaptation middleware can be divided into an application-transparent category [4], in which the adaptation occurs in the middleware, and an application-aware category [1], in which the adaptation occurs in the application. Application-transparent systems can reduce complexity of the applications because contextual information is hidden from the applications and the adaptation is completely controlled by the middleware. However, such middleware can only provide best-effort adaptation since the characteristics and objectives of applications are unknown to the middleware. MARCHES is a type of middleware enabling application-aware adaptation, which assists applications to make more efficient adaptation decisions. Since the application directly controls its adaptive behavior, the middleware is more easily to be reused to build different adaptive applications.

Based on the supported applications architecture, adaptation middleware can also be classified as grid-based [4], client/server-based [1], or peer-to-peer [7] middleware. Delphoi [4] is grid-based middleware that monitors environments and makes adaptation decisions for grid-aware applications in the GridLab platform. However, the existing grid-based adaptation middleware does not provide a component synchronization service, which is important for developing collaborative applications. MobiPADS [1] is client/server-based middleware that enables adaptation in both middleware layer and application layer for mobile client/server applications. It supports the single service-chain reconfiguration and synchronization. Its reconfiguration and service suspension time is in a range of seconds or tens of seconds [1]. Thus MobiPADS is not suitable for building real-time collaborative applications. MARCHES is peer-to-peer middleware and each agent functions as both a client and a server to other agents in distributed applications. Contrast to MobiPADS, MARCHES supports multi-actuator architecture and active messages, which largely reduces the overhead and reconfiguration time.

The concept of active messages was originally proposed for large-scale multiprocessors to minimize inter-processor communication overhead and allow communication to overlap computation [8]. This paper utilizes the active-message concept in the first time to address the architecture synchronization problem in the dynamic reconfiguration middleware.

V. Conclusions

In this paper, we have described a middleware-based adaptation approach called MARCHES to building adaptive collaborative applications. It supports adaptive application architecture with multiple component chains called actuators in each distributed program and uses an active message based synchronization protocol for the application reconfiguration. This novel architecture eliminates the operation suspension and buffer clearance delays in conventional adaptive application architecture that uses single-component-chain reconfigurations. Thus MARCHES offers reduced reconfiguration time and improved reconfiguration robustness to adaptive applications, which was supported by our experiments based on the complete implementation of MARCHES. Results also show that the extra costs introduced by the multi-actuator architecture in MARCHES are extremely low.

Acknowledgment

The authors would like to acknowledge the support by the U.S. National Science Foundation (Award# 0438300).

References

[1] A. T.S.Chan, S. N. Chuang, MobiPADS: a reflective middelware for context-aware mobile computing. *IEEE Trans. on Software Engineering*, 29(12), Dec. 2003.

[2] R. Litiu and A. Prakash, DACIA: a mobile component framework for building adaptive distributed applications. *Technical Report CSE-TR-416-99*, University of Michigan, EECS, Dec. 1999.

[3] Q. Wang and L. Cheng, AwareWare: an adaptation middleware for heterogeneous environments. In *Proceedings of* 2004 *IEEE International Conference on Communications*, Vol. 3, pp. 1406-1410, June 2004.

[4] J. Maassen, R. V. Nieuwpoort, and T. Kielman et. al., Middleware adaptation with the delphoi service. *Concurrency and Computation: Practice & Experience*, 2006.

[5] B. Li, Agilos: a middleware control architecture for application-aware quality of service adaptations. *PhD Thesis*, University of Illinois, 2000.

[6] B. Aziz and C. Jensen, Adaptability in CORBA: the mobile proxy approach. In *IEEE International Symposium on Distributed Objects and Applications*, pp. 295–304, 2000.

[7] L. Capra, W. Emmerich, and C. Mascolo, Carisma: context-aware reflective middleware system for mobile applications. *IEEE Trans. on Software Engineering*, 29(10), pp. 929–945, 2003.

[8] T. von Eicken, D. E. Culler, S. C. Goldstein, and K.E. Schauser, Active messages: a mechanism for integrated communication and computation. In *Proceedings of the* 19*th Annual International Symposium on Computer Architecture*, pp. 256–266, Gold Coast, Australia, May 1992.