MassWare User's Manual

Version 1.0.2

The MassWare Team
LONG Lab, Lehigh University
http://marches.cse.lehigh.edu/

Aug. 15, 2008

1. Introduction

1.1 MassWare

MassWare (Mobile Ad-hoc and Sensor Systems' Middleware) is an adaptive and reflective middleware framework to solve the critical issue of the reconfiguration time and engineer real-time distributed adaptive applications in the dynamic environments. Compared to the traditional middleware that supports the single component-chain based application architecture, MassWare maintains multiple component chains (Fig. 1b). Therefore, there is a new method proposed for the behavior reconfiguration that switches active and inactive chains. This new method replaces the traditional method of modifying the single-chain structure to reduce the local behavior change time. Further, according to the new method, an efficient active-message based asynchronous synchronization protocol is proposed for synchronizing the behaviors of distributed programs. This results in a dramatic reduction of the distributed behavior synchronization time by eliminating the operation suspension time and buffer clearance time. The robustness of the distributed application is improved since the use of active messages results in no synchronous communication and system halting in the synchronization process.

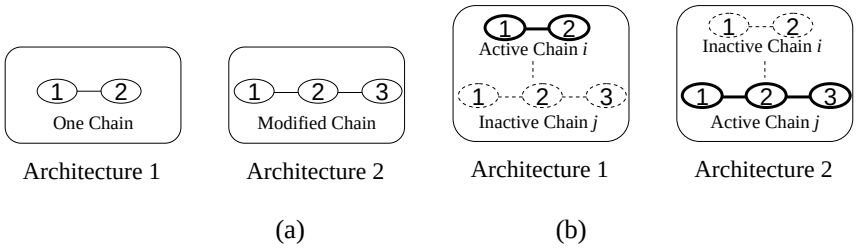

Fig. 1. Dynamic reconfiguration architecture: (a) single component-chain architecture in existing middleware, (b) multiple component-chain architecture in MassWare.

MassWare 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. Each MassWare agent can be separated into a core layer and an operation layer. The core layer consists of eight function modules, which construct the adaptive and reflective framework for accommodating MassWare components and efficiently reconfiguring its behaviors: Measurement tools, a decision engine, and an XML-based script parser constitute the basic reflection model; A dynamic reconfigurator and a hierarchical event interpreter are proposed to support the efficient context evaluation and reconfiguration; In addition, a communication channel for the inter-communication among peer agents, an information register/ requester for reporting/fetches awareness information to/from a centralized information manager, and a component manager for managing all types of MassWare components are designed as surrounding services.

Measurement tools are the lowest building blocks that monitor the dynamic environments and report the environment awareness results as the contextual information to be processed by event sensors. The sensors and actuators, in addition to adaptation policies, are defined by application developers or users in a XML script file. There are two types of actuators: proactive and reactive ones, each of which contains a component chain and performs reconfiguration actions. 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 the synchronization process with peer agents to process the received data. Once a context triggers an event sensor, a corresponding proactive actuator will be activated by the decision engine to

perform the reconfiguration actions.

In the operation layer, various services are offered by software components that implement specific algorithms and protocols. There are two types of components in the operation layer: computing components (called *masslets*) for processing application data, and awareness measurement tool components (called *awaretools*), for measuring contextual information. A MassWare supported application consists of a list of computing components and can be adaptively reconfigured at run-time according to context changes and adaptation policies.

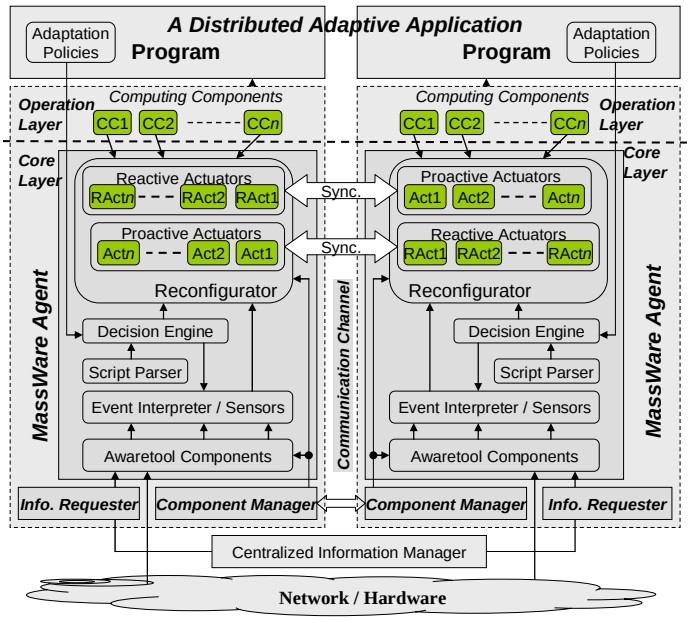

Fig. 2. System architecture of MARCHES.

1.2 Demonstration

This demonstration illustrates the major functionality of “MassWare”: an adaptive and reflective context-aware middleware framework that efficiently handles reconfiguration for distributed real-time and embedded (DRE) applications in a generic, consistent, and clearly-separated way. By providing a set of tools and underlying mechanisms, it achieves it primary goal of facilitating a software developer to handle adaptation inside their applications, therefore to let the software developer to be more focused on application’s own logic instead of dealing with the complexity of adaptation.

The Demonstration also validates the efficiency of the run-time reconfiguration for supported applications. There are two applications implemented and used in this demonstration:

Video conference application: the video-conferencing application is a peer-to-peer application and each distributed program contains a sender part and a receiver part. In the sender part, proactive actuators prepare and send video frames; and in the receiver part, reactive actuators receive and display the frames. There are four masslets (*Grab*, *Compress*, *Decompress*, and *Display*) and two awaretools that measure the available bandwidth between the hosts and the available CPU resource respectively. The application architecture can be dynamically reconfigured by using or not using the *Compress* masslet or using different compressing ratio according to three adaptation rules.

E-learning application: The E-learning application is a client-server application for distance-education. The teacher (server) program can prepare and disseminate video, audio, and text message, and share slides with connected students. Based on the network conditions, the server program can dynamically change the video and audio quality or shut down the video and audio components and only send the text messages.

2. Installation Instructions

Requirements:

Operating System: Windows XP & Windows Mobile 5, 6.
Memory: 256MB (minimum) or more
Runtime environment: .Net 2.0
Development kit: Microsoft Visual Studio 2005
Other required tools: MSXML 4.0

Provided file list for the demo applications:

MassWare.dll	// the major DLL file of the MassWare system
Masslet.dll	// the masslet base component
MassTool.dll	// the awaretool base component
VideoConferencing.exe	// the executable file of the video-conferencing application
Teacher.exe	// the executable file of the teacher program in the E-learning application
Student.exe	// the executable file of the student program in the E-learning application

Provided example components:

Measurement Tool Components:

ABWPTR.dll	// get the available bandwidth data based on IGI/PTR algorithm
AvailableCPU.dll	// get the available CPU data

* Users can use their own developed measurement tools by just replacing the tools description in the script file

Reconfigurable Components:

WebCam.dll	// grab the video from the camera
JPEGCompress.dll	// compress the video data by JPEG algorithm
JPEGDecompress.dll	// decompress the video by JPEG algorithm
Display.dll	// display the data in local view

* Users can also use their own components, and please refer the XML file description in the following pages.

Provided XML script files

dcRules.xsd	// XML schema file, any user developed XML file should follow this schema for grammar check.
vcRules.xml	// XML-based adaptation-policy script file for the video-conferencing application
elRules.xml	// XML-based adaptation-policy script file for the E-learning application

3. Usage Instructions

3.1 Modify the XML file

Set the components locations in the XML script file to your local components directory.

For example, if your measurement tool “ABWPTR.dll” is in D:\temp\, set your script file like:

```
<component cid="1001">
 <file> D:\temp\ABWPTR.dll </file>
 <name> MassTools.ABWPTR.ABWPTR</name>
 <vtype> Double </vtype>
 <alias> AVAILABLEBW </alias>
 <param pid="001">
  <name> SetPacketSize </name>
  <vtype> Int32 </vtype>
  <value> 32 </value>
 </param>
 <param pid="002">
  <name> SetPacketNum </name>
  <vtype> Int32 </vtype>
  <value> 2 </value>
 </param>
 <param pid="003">
  <name> SetInterval </name>
  <vtype> Int32 </vtype>
  <value> 30000 </value>
 </param>
</component>
```

Also modify the reconfiguration components locations in the XML script file.

3.2 Start the application.

Video-conferencing application:

1) Download all the files to two different machines.
2) Double click the VideoConferencing.exe file to start the application, and you will get the following GUI (Fig. 3).

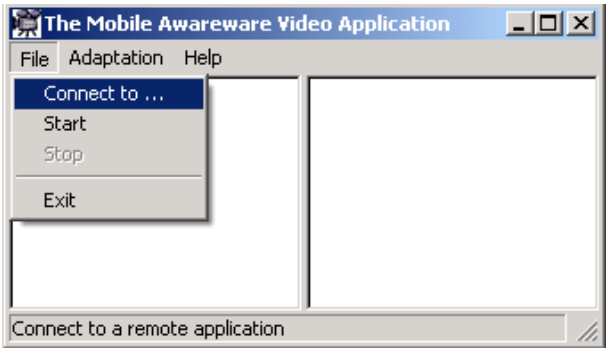

Figure 3. The Graphic User Interface of Video-conferencing Demo

The main frame is divided into two sub-frames. The left frame is view of local video; and the right frame is the view of remote video. The status bar shows the current action and internal stats.

3) Start the application in another machine

4) Connect to the remote machine by click the “Connect to …” submenu in the “File” menu as Fig. 3.
5) After connecting that machine (or connecting to itself), click “start” menu to start the data video capture and transmission. The left frame will show the local camera video, and the connected remote application will show your video view in its right window.

E-learning application:

1) Download all the files to two different machines.
2) Double click the teacher.exe file to start the teacher (server), and you will get the following GUI (Fig. 4).

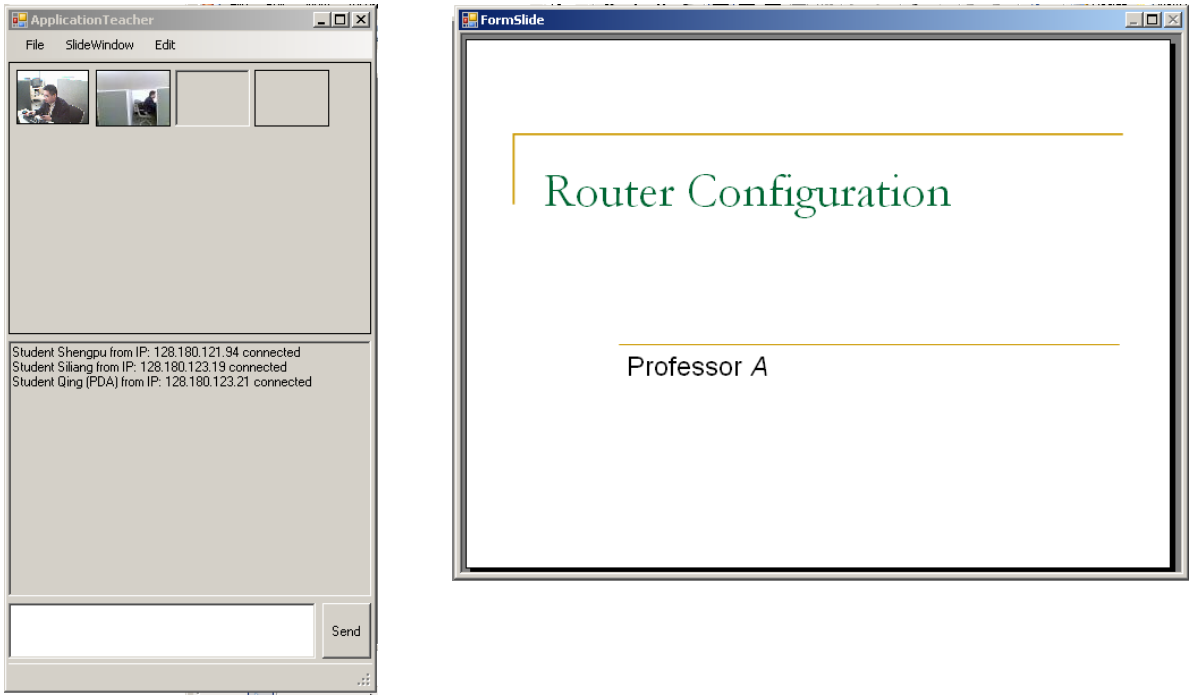

Figure 4. The Graphic User Interface of the Teacher program in the E-Learning Demo

There are two frames in the teacher side: the main frame (left side) contains the menu bar and two areas: the video area that show the local and connected student videos and the text area show for message chatting; The slide (right side) frame is adjustable to sharing the interest desktop image (e.g. slides) to connected remote students.

3) Start the student.exe in another machine, and you will get the following GUI (Fig. 5)

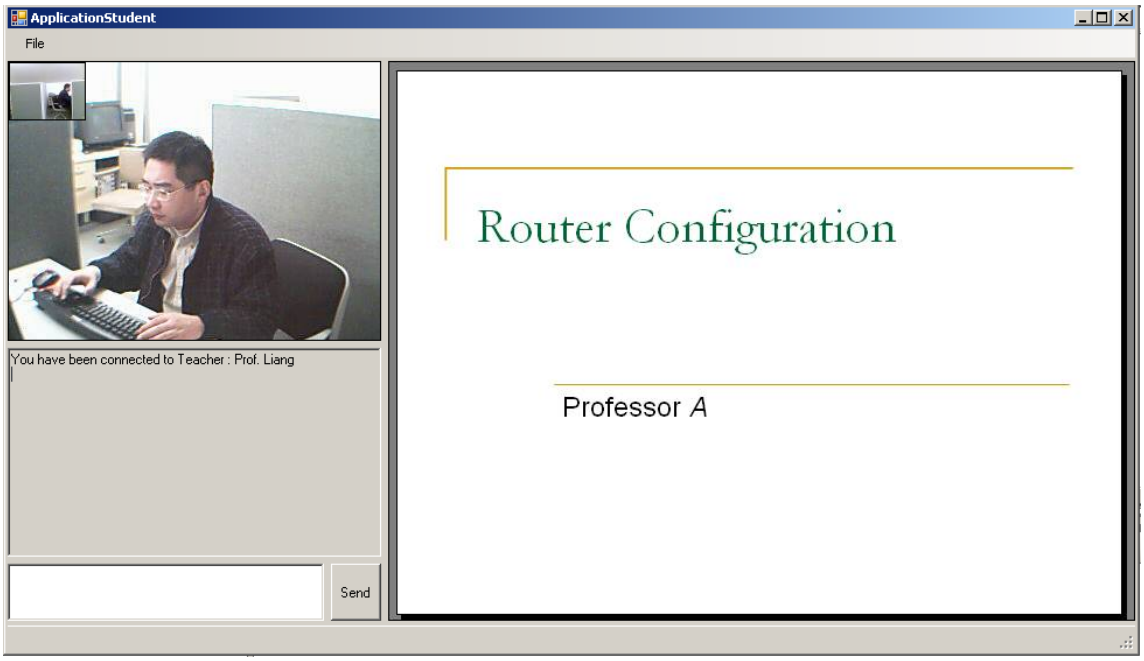

Figure 5. The Graphic User Interface of the Student Program in the E-Learning Demo

There are also three frames in the student side: the video (left top) area that show the local and remote teacher videos, the text (left bottom) area show for message chatting; and the slide area that shows the slides in connected teacher machine.

4) Connect to the teacher program from the student program by click the “Connect to …” submenu in the “File” menu. You will see the message in the text message area if the connection is established.
5) After connecting that machine (or connecting to itself), the student can retrieve the slides from the teacher side and communication with the teacher through video, audio, and text information.

4. Development Instructions

4.1 The MassWare system UML diagram

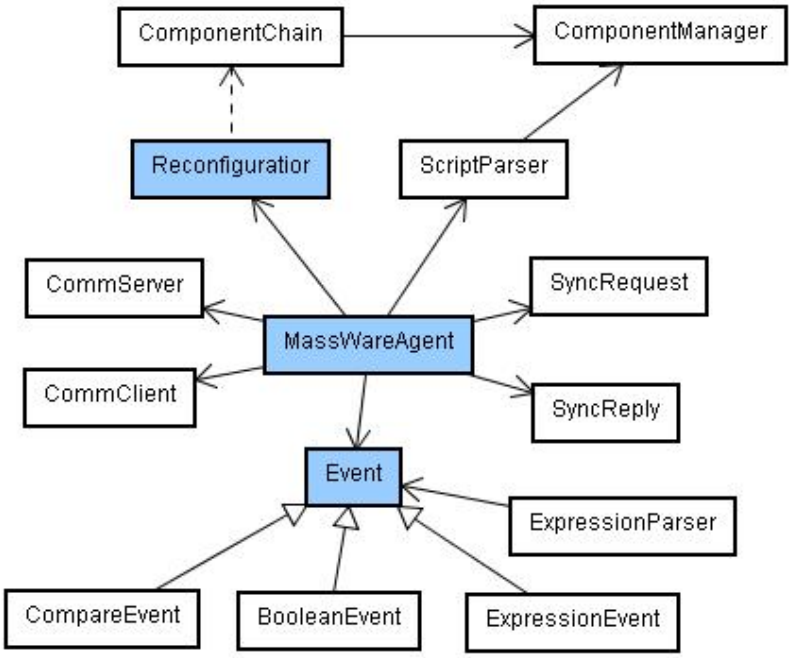

Figure 6. The UML diagram of MassWare Implementation

As shown in Fig. 6, MassWare contains three core classes (blue color): *Event*, *MassWareAgent*, and *Reconfigurator*. The *Event* class is the core class of MassWare sensor, which can be further inherited by *CompareEvent*, *BooleanEvent*, and *ExpressionEvent*. The *CompareEvent* contains two double-type event sources (left hand side and right hand side) and a compare conditioner for comparing the two event sources. The *BooleanEvent* contains two Boolean-type event sources and a Boolean conditioner. The *ExpressionEvent* is the class to build context expressions in a binary tree manner. It supports addition, subtract, multiply, and division operations and brackets. The *ExpressionParser* is used to parse the *ExpressionEvent* to build the expression tree.

The *MassWareAgent* class is the core class of the MassWare decision engine. It invokes the *ScriptParser* object to parse the user defined adaptation policy script, builds the hierarchical events, and subscribes the actuators to corresponding event sensors. It also communicates with peer agents for synchronization after reconfiguration.

The *Reconfigurator* is the core class of the MassWare reconfigurator. It contains multiple actuators and each actuator contains a component chain. The corresponding actuator notified by the context sensor will active its component chain to process the application data.

4.2 MassWare application development

All the source code are implemented in Microsoft Visual Studio 2005

1. Set the environment:

Add the middleware reference to the application project: MassWare.dll

2. Define the middleware object and instantiate it with the script file

```
MassWare.MassWare thisWare = null;
thisWare = new MassWare.MassWare(@"D:\temp\vcRules.xml");
```

3. Initialize the middleware and define the architecture change callback function:

```
thisWare.Initialize();
thisWare.AddArchSubscriber(new MassWare.Architecture.MassWareArchChangeEventHandler(MassWareArchChanged));
```

4. Set the application level component parameters

```
// set the parameter list
object[] paramlist = { localWnd.Handle, localWnd.Width, localWnd.Height };
// set the parameter for the component "Masslets.WebCam.WebCam"
thisWare.SetAppParameter("Masslets.WebCam.WebCam", "SetLocalWindowInf", paramlist);
paramlist = new object[3] { remoteWnd.Handle, remoteWnd.Width, remoteWnd.Height };
thisWare.SetAppParameter("Masslets.Display.Display", "SetDisplayInf", paramlist);
thisWare.SetAppParameter("MassTools.ABWPTR.ABWPTR", "SetRemoteAddr", new object[] { remoteIP });
```

5. Start Engine

```
// start the middleware and run-time reconfigure the application behavior accroding to the adaptation plicies.
thisWare.Start();
```

6. Implement the behavior change call back function.

```
private void MassWareArchChanged(object sender, MassWare.MassWareArchEventArgs e)
{
    if (e.mType == MassWare.MassWareArchEventArgs.PROACTIVE_CHANGE)
    {
        toolStripStatusLabel.Text = "Proactive Architecture Changed: " + e.mMessage;
    }
    else if (e.mType == MassWare.MassWareArchEventArgs.REACTIVE_CHANGE)
    {
        toolStripStatusLabel.Text = "Reactive Architecture Changed: " + e.mMessage;
    }
}
```

7. Stop and release the middleware object

```
if (thisWare!= null)
{
    thisWare.Stop();
    thisWare.Dispose();
    GC.Collect();
}
```

5. Source File List:

MassWare.cs	// Decision Engine main class. Contain a XML parser and a reconifigurator
XMLParser.cs	// the XML Parser to parse the XML file, create the event sensors and actuators
Architecture.cs	// the MassWare reconfigurator, contains multiple actuators
Actuator.cs	// the MassWare actuator, contains a component chain
EventBC.cs	// the MassWare event
EventComp.cs	// the MassWare compare event
EventBool.cs	// the MassWare Boolean event
EventExpr.cs	// the MassWare expression event
ExprParser.cs	// the expression parser to parse the expression event
SyncRequest.cs	// the synchronization request package
SyncReply.cs	// the synchronization reply package
UDPServer.cs	// the receiver side of the communication channel
UDPClient.cs	// the sender side of the MassWare communication channel
Masslet.cs	// the Masslet base class, it defines the message type for masslet communication
WebCam.cs	// the WebCam component that grab the image from the camera
JPEGCompress.cs	// the JPEG Compress component that compress the input image
JPEGDecompress.cs	// the JPEG decompress component
Display.cs	// the display component that display the received image in local window
MassTool.cs	// the MassTool base class, it defines the message type for masstool communication
AvailableCPU.cs	// the measurement tool component that measure the available CPU
ABWPTR.cs	// the measurement tool component that measure the available bandwidth
Common.cs	// the common definition for ABWPTR server and client
PTRClient.cs	// the client side of the PTR bandwidth measurement tool
PTRServer.cs	// the server side of the PTR bandwidth measurement tool

6. Source Code Copyright

This source code is a part of the MARCHES project.

The MARCHES project is supported by the National Science Foundation, and carried out at the Laboratory Of Networking Group (LONGLAB).