AwareWare User's Manual

Version 1.0.2

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

Dec. 30, 2008

1. Introduction

1.1 AwareWare System Overview

AwareWare consists of three major function layers, as depicted in Figure 1: an awareness measurement layer, an adaptation decision layer, and an adaptation execution layer.

The awareness measurement layer can be further separated into individual measurement tools, which measure context-awareness information about networks, devices, end-user preferences, application internal states, and physical environments, and awareness management, which organizes these tools and provides query and notification interfaces for the adaptation decision layer. Awareness information has to be distributed efficiently because an adaptive application may also need to know the awareness of its distributed peers for action coordination, in addition to its own awareness information, Therefore awareness needs to be distributed and accessible to any other applications that are interested in this awareness. There are two basic methods for distributing the awareness information: pull and push. Via pull method, an application can explicitly query awareness information. While via push method, an awareness information source pushes information to interested applications through event notification when pre-defined conditions meet. A detailed discussion of awareness information management in AwareWare can be found in our previous work.

In the adaptation decision layer, a decision engine makes the adaptation decisions based on the context-awareness and user-defined adaptation policies to satisfy the adaptation goals under the constraints of restricted resources. The adaptation policies are written in a high level declarative language by application developers in XML format. The script-driven policy has several advantages. First, it separates the adaptation rules from the rest of function codes, which makes adaptation policy easy to check and validate. Second, a user can change adaptation policies of a deployed system on the fly without recompiling the application or middleware, by downloading and applying a different adaptation policy file, which makes the application development and updates very flexible. The decision engine takes the adaptation policy file as the input, creates composite event sensors, and initiates adaptation actions based on these adaptation policies.

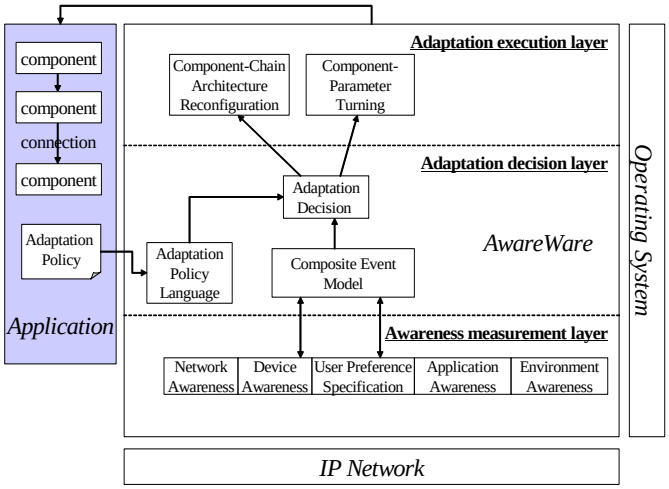

Fig. 1. System architecture of AwareWare.

The adaptation execution layer contains several modules to execute the actual behavior changes of the application and the middleware itself, including the dynamic reconfiguration and parameter tuning mechanisms to the application's components, and a feedback control loop to awareness manager, which controls the behaviors of measurement tools.
AwareWare views an adaptive application as a dynamic collection of reflective components and their connections. A component can be added to and deleted or removed from an adaptive application's architecture. Inter-component connections can be added, deleted, blocked, and changed dynamically. A component's parameters can also be changed at the execution time. It was realized by using component reflection and dynamic reconfiguration mechanisms implemented in AwareWare.
In summary, by "externalizing" adaptation mechanisms, AwareWare is responsible for monitoring environment conditions and changes that trigger adaptations, deciding when, where, and how to adapt application behaviors, and executing the adaptation policies specified by application developers in a running system.

1.2 Demonstration

This demonstration illustrates the major functionality of "AwareWare": 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 awarelets (*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* awarelet or using different compressing ratio according to three adaptation rules.

Firefighter training application: The firefighter training application is a peer-to-peer application that facilitates firefighters to collaboratively deal with emergent fire situations. Each firefighter program can show the positions of its holder and other firefighters in the layout, which are updated frequently, and nearby fire situations by communicate with environment sensors. The firefighter training application uses AwareWare to adaptively transmit audio and video information among firefighters so that they can be still connected in hash environments.

2. Installation Instructions

Requirements:

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

Provided file list for the demo applications:

DCEngine.dll	// the major DLL file of the AwareWare system
DCEngine.lib	
AwareVCTest.exe	// the executable file of the video-conferencing application
AwareFFTest.exe	// the executable file of the firefighter training application
ComponentManager.exe	// the executable file of the component manager application
Mods.exe	// the executable file of the directory service application
cm.inf	// the configuration file the component manager

Provided example components:

Measurement Tool Components:

AvailableBW.dll	// get the available bandwidth data
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

To use AwareWare to build an adaptive collaborative application, developer(s) need to provide a script file that can be divided into a declaration part and an adaptation-rule part as shown in Fig. 2. The declaration part declares the CM and all components used in the local program and middleware agent. Based on the declaration, the AwareWare agent loads and instantiates the components from the CM, 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 in the collaborative application.

```
<ComponentManager>
 <ip> localhost </ip>
 <port> 5501 </port>
</ComponentManager>

<Awarelets>
 <component name="Grab" version="1.0" type="Proactive">
  <alias> GRAB </alias>
  <param name="CaptureWidth"> 160 </param>
  <param name="CaptureHeight"> 120 </param>
 </component>
 …
</Awarelets>

<AwareTools>
 <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>
 <component name="Average" version="2.0" type="Function">
  <alias> Ave </alias>
 </component>
</AwareTools>

<Rules>
 <rule>
  <sensor> Ave(AVI_BW, 5) > 10 && Min(AVI_CPU, 10) < 1.0 </sensor>
  <proActuator>
   <marchlet name="GRAB">  </marchlet>
  </proActuator>
  <reActuator>
   <marchlet name="DISPLAY"> </marchlet>
  </reActuator>
 </rule>
 …
</Rules>
```

Figure 2. An example of the adaptation rule script

Modify the script file based on your local setting and user preference.

3.2 Start the application.

Video-conferencing application:

Platform: Desktops or Laptops
Operating System: Windows XP
Pre-Installed Software: Microsoft Visual Studio 2005, MSXML 4.0
File list:

Application: AwareVCTest.exe, ComponentManager.exe
Marchlets: Grab.dll, Compress.dll, Decompress.dll, Display.dll
AwareTools: AvailableBW.dll
Adaptation rule script file: dcRules.xml
Component Manager Configuration file: cm.inf

Application Usage:

Step 1: Install all required software in a machine called host1. And put all the files in the same folder of host1 except the marchlets and awaretools which can be put in any folders.

Step 2: Modify the adaptation rule script file as the user requirements. And modify the component manager configuration file to setup the search directory for the marchlets and awaretools.

Step 3: Double click the "ComponentManager.exe" to start the component manager. You will see the following Fig. 3.

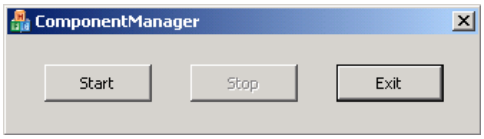

Figure 3. The GUI of component manager application

Step 4: Click the "Start" button to start the component manager. It will search and register all the components in the directories configured in the "cm.inf" file.

Step 5: Double click "AwareVCTest.exe " file to start the video conference application that is called application1. You will see the following Fig. 4. 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 will show the current actuator and its component connections.

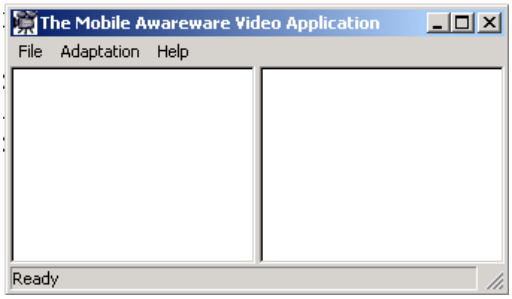

Figure 4. The GUI of the video conference application

Step 6: Repeat step 1 ~ 5 in another machine called host2, and start application 2.

Step 7: Click the "file" menu in application1 to connect to application 2 as Fig. 5.

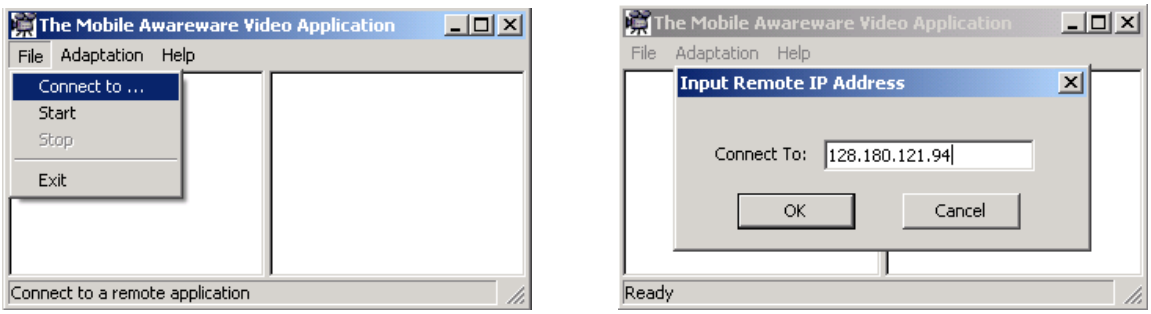

Figure 5. Connect to another application

Step 7: After connecting to host2 (or connecting to itself in this example), 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, shown in Fig. 6.

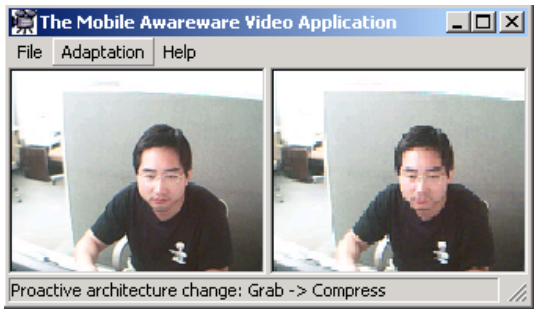

Figure 6. Running status of the video conference application

Firefighter training application:
Platform: PC and PDA
Operating System: Windows XP and Windows Mobile 5
Pre-Installed Software: Microsoft Visual Studio 2005, MSXML 4.0, .Net 2.0
File list:

Application: AwareFFTest.exe, mods.exe
Marchlets: Grab.dll, Compress.dll, Decompress.dll, Display.dll
AwareTools: AvailableBW.dll
Adaptation rule script file: dcRules.xml
Component Manager Configuration file: cm.inf
Other support files: PtoI.dll, mdosClient.dll, mods.ini, and layout.bmp

Application Usage:
Step 1: Double click mods.exe to start the directory server. You would see the following Fig. 7.

Figure 7. Directory server application

Step 2: Modify the adaptation rule script file and mods.ini file based on local settings and user preferences.
Step 3: Double click the "AwareFFTest.exe" to start the firefighter training application component manager. You will see the following Fig. 8.

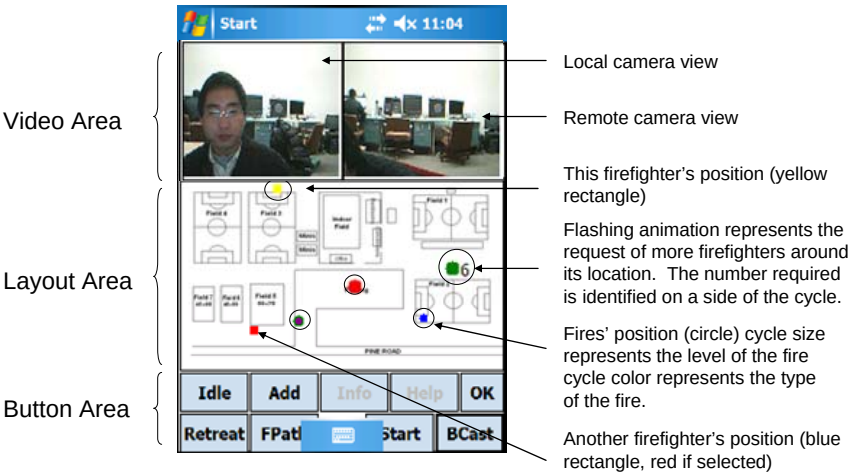

Figure 8. The graphic user interface of our application

There are three parts in the user interface. The top area is video area. Left video is local camera view. It is blank because there is no camera attached in this PDA currently. The right video is remote camera view. When we select another firefighter from the layout by stylus, the camera view of that firefighter would be transmitted to this node, and displayed on right hand side video area. If we select a third firefighter, the previous session would be ended automatically. And a new transmission session would be constructed to the third firefighter.

There are two kinds of display information in the layout area: fires and firefighters. Fires are displayed as cycles: Cycle color represents the type of the fire: red is normal, green is chemical, blue is oil, and purple is other. And cycle size represents the level of the fire. Firefighters are displayed as rectangles. When a new node joins in the network, the firefighters' information would be updated automatically.

Step 4: Add a new fire in the layout.

In fig. 3, the first row of buttons is for fire manipulation. In the initial state, there are three active buttons: "Idle", "Add", and "OK". Push "Add", the caption of this button would be changed to "Comp" (Complete), and Info, and Help buttons become active. First select the fire position in the layout using stylus, this position can be modified at any time before you touch "Comp" button. Touch "Info" button to set the information of the fire, including fire level and fire type, see Fig. 9. If more firefighters needed, we can set the number of firefighters needed for this fire. Touch the "Help" button; we would see the Help dialog, see Fig. 10. The radio button is used to select the number area, and the edit box with spin control is used to set the accurate number.

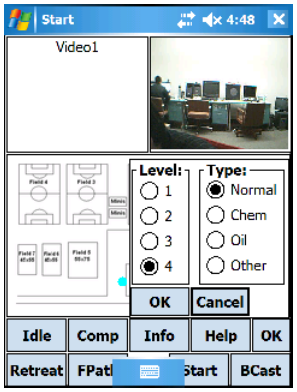

Figure 9. The fire information dialog

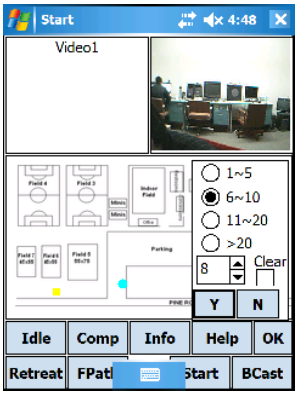

Figure 10. The number of firefighters dialog

Step 5: Edit a fire in the layout.

If a fire is selected, this fire would be displayed highlight, and the "Add" button would be changed to "Edit" button, see Fig. 11. Then touch "Edit" button, "Info" and "Help" buttons would be activated, so that we can edit the selected fire as previous

step. If the selected fire needs more firefighters, and this firefighter want to go there to help, then touch “OK” button in the Edit mode, and the number of firefighters required by that fire would be decreased by 1 (8→7) and this information would be synchronized in the whole network so that anybody know how many firefighters are still needed for this fire.

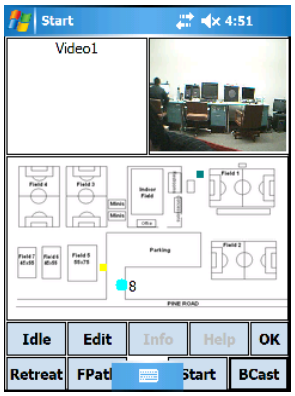

Figure 11. The fire edit mode

Step 6: Delete a fire from the layout.

Select the fire, and push the “Idle” button, the fire would be removed from the layout. If no fire is selected, touching “Idle” button means this firefighter is currently in idle state and prepares to go anywhere else.

Step 7: Retreat from the dangerous area.

If this area is to dangerous, Push “Retreat” button, then the alarm would be broadcast to the whole network, all the firefighters close to this dangerous area should retreat immediately.

Step 8: Find a path in the layout for retreat.

The firefighter can draw a path for safe retreat. Push the “FPath” (Find Path) button in the second row, the caption would be changed to “FComp” (Find Complete). Then, we can draw a path in the layout by using stylus. After the path is complete, push “FComp” button, there would be a dialog, see Fig. 12. Select “Complete”, and push “Y”, then the new path is added, the firefighter can broadcast this path information out or unicast it to the help center.

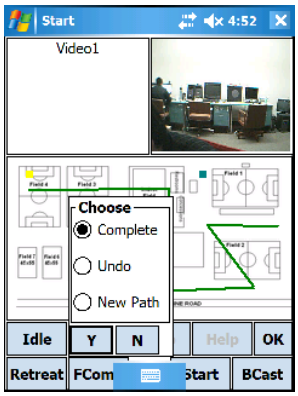

Figure 12. Add a path

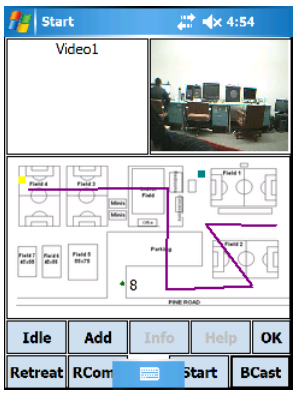

Figure 13. Select a Path

Step 9: Edit a path.

When a firefighter receives a path, he can ignore it if he has no idea if this path is safe, or it can modify it if he knows some part of the path is not safe. To modify a path, first select a path, the color of the path would be changed from green to purple, and the caption of the “FPath” button would be changed to “RPath” (Revise Path). Push “RPath” button, the caption would be changed to “RComp” (Revise complete), see Fig. 13. Select the line you think it is not safe, and it would be changed to red color, see Fig. 14. Then push the “RComp” button, you can select ignore this path, delete selected line or delete the whole path, see Fig. 15.

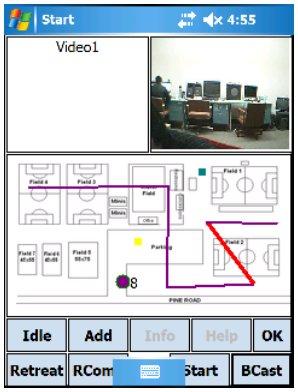

Figure 14. Select a line from the selected line

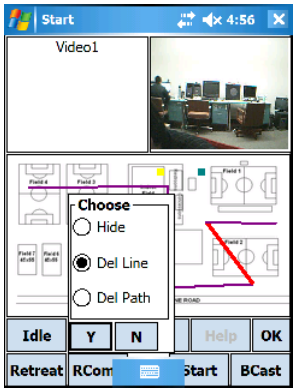

Figure 15. Delete the selected line

Fig. 16 shows the path after one line is deleted. We can also delete another line from the path by using the same method. Then, we can add a new path as the previous step and merge it to the existing path. The completed path is shown in figure 17.

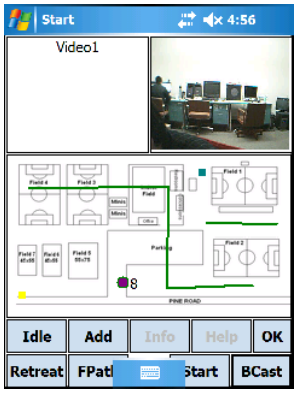

Figure 16. The path deleting one line

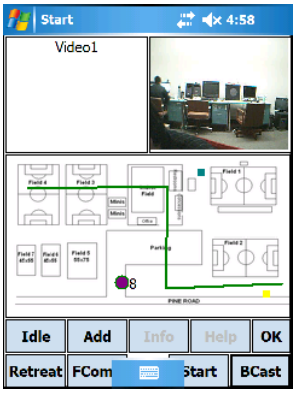

Figure 17. The revised path

4. Development Instructions

4.1 The AwareWare system UML diagram

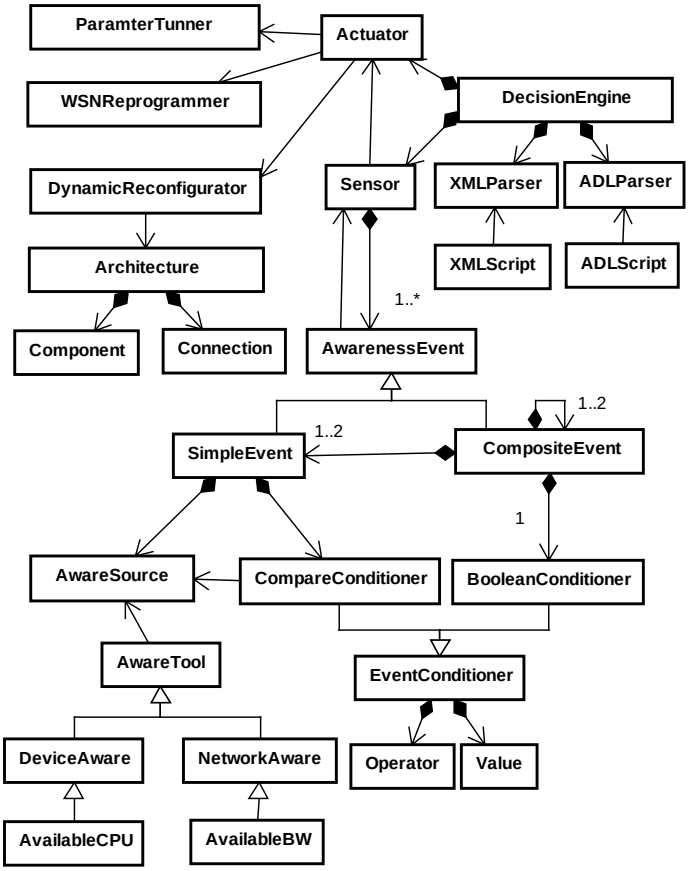

Figure. 18. The UML Diagram of Decision Engine

As shown in Fig. 18, the adaptation decision layer contains four major function modules: a decision engine, a composite event model, a XML based script parser, and an ADL parser, which are discussed in details in the following sections.

AwareWare decision engine has two major classes: *Parser* and *Sensor*. A *parser* takes script files in XML or ADL formats as the input to the decision engine, which generates a set of *Sensor* objects and adaptation-execution-layer *Actuator* objects. There are two parser types called the *XML Parser* and the *ADL Parser*. The XML parser parses the condition part of the adaptation policies or decision logics described in the XML file to construct a set of Sensor objects. The ADL parser parses the ADL specification of the software architecture and its dynamic architectural reconfiguration to create a set of *Actuator* objects. *Actuator* objects and *Sensor* objects form a subscription and notification paradigm: *Actuators* subscribe to their interested *Sensor* objects and *Sensor* objects notify *Actuators* when their interested events trigger.

The *Sensor* class contains a set of events. The event could be a simple event, which only has a “comparison” operation, or a composite event, which contains “Boolean” operations. A simple event is specified as a triple in the XML script file: the name of awareness source, a comparison operator, and a condition value. For example, “CPU_Loading > 10” is a simple event. A composite event is a combination of multiple simple events connected by Boolean operators. For example, “(CPU_Loading) > 10 and (Avi-bw < 10)” is a composite event.

To use AwareWare to build an adaptive application, developer(s) need to provide a XML based script file and an ADL file. A XML script parser is proposed in the decision layer to parse the XML file that can be divided into a declaration part and an adaptation-rule part. The declaration part declares all application level functional components and measurement tools used in the local program and middleware agent. Based on the declaration, the decision engine loads and instantiates the components, and initializes them with the provided parameters. In <component> section, the XML script defines the name, location, and parameters of the component. The name of the component can be any string as the component identifier and it is also used in the decision rules describing adaptation policies. The <location> is the file directory and filename of the component so that the decision engine can find and load it dynamically. <param> supplies a list of default parameters to initialize the component. The example below shows that a video compression component, named as "*compress*", is located at "C:\demoApp\ compress.dll" of local host. An initial parameter, called CompressionRatio, is set as 100 when the decision engine loads the *Compress* component.

4.2 AwareWare Classes

```
/*********************************************************
Decision Engine.
This is the main class of decision engine.
First, it loads the xml file and parses it as following:
1. create the awareness tools
2. load the reconfigurable components
3. create the compositional event and actuator for each rule.
4. start the actuator and event.
*********************************************************/
class CDCEngine

/*********************************************************
Component.
This is the parent class of all reconfigurable components.
Any one can implement his(her) own components by inheriting
this class.
The implemented components could be a DLL, so that our
middleware can load the dll dynamically, just like awareness
measurement tools
There is only one instance of each component in our middleware.
All the actuators or metaobject just get a pointer pointing to
the component.
*********************************************************/
class CComponent

/*********************************************************
Awareness.
This is the parent class of all awareness measurement tools.
Any one can implement his(her) own awareness by inheriting
this class.
The implemented awareness tools could be a DLL, so that our
```

```
middleware can load the dll dynamically, just like components
There is only one instance of each awareness tool in our
middleware. However, this measurement tool can notify multiple
conditioner.
*********************************************************/
class CAware

/*********************************************************
AwareEvent
This is a virtual class that is the parent of SimpleEvent
and CompositionalEvent.
AwareEvent includes event source, event conditioner, and
event listener, event source provides the data (from the
awareness tool or from the lower layer event) to conditioner,
and the conditioner notify the listener if the condition
is sufficed. Upper compositional event or acutual would
registers itself to the event as listener.
AwareEvent itself is also a subclass of EventSource, because
it is a eventsource of the upper layer compositional event.
*********************************************************/
class CAwareEvent

/*********************************************************
Actuator
Actuator is the action of the application when one rule
is sufficed and notified by the compositional event.
There is one actuator in each rule, corresponding to one
compositional event.
When the rule is sufficed, the corresponding actuator will
update the architecture (metaobject)
*********************************************************/
class CAwareActuator

/*********************************************************
CAwareMetaObject
This class is the architecture of the configured components.
Because there is only one architecture, it is a global variable.
It gets the components data from acutuator that is notified by
the compositional event.
To form the architecture, first, all the parameters of each
component in the acutuator must be reset because the components
are also global variables and only has one copy of each component
Currently, there is no synchronization part in this architecture.
*********************************************************/
class CAwareMetaObject
```

```
/*******************************************************
Event Source
This class describe the source in aware event. event source
provides data to conditioner. It is the parent class of
aware source and event.
event is the source used in the compositional event,
and aware source is used in the simple event.
*******************************************************/
class CEventSource
/*******************************************************
CEvent Conditioner
This class describe the conditioner in aware event. It is
the parent class of boolean conditioner and comapre conditioner.
boolean conditioner is used in the compositional event,
and compare conditioner is used in the simple event.
*******************************************************/
class CEventConditioner

/*******************************************************
CEventLister
This class describe the listener in aware event. It could
be the actuator or the upper layer compositional event.
If the listener registers itself to the event, the conditioner
of the event would notify the listener if the condition
is suffied.
*******************************************************/
class CEventListener

/*******************************************************
Compositional Event.
This is a kind of aware event that has two event sources,
called left hand side source and right hand side source,
so that it is inherited from aware event.
Compsitional event contains a boolean conditioner and two
event sources. the event sources could be simple events or
compositional events, but could not be aware sources.
The listener of the compositional event could be acutator
or upper layer conditioner.
*******************************************************/
class CCompositionalEvent : public CAwareEvent

/*******************************************************
Simple Event.
This is a kind of aware event that has one event source,
which is a aware source that get the data from awareness
tools directly, and it is inherited from aware event.
```

```
Compsitional event contains a compare conditioner and only
one source that is a aware source.
The listener of the compositional event could be acutator
or upper layer conditioner.
*************************************************************/
class CSimpleEvent : public CAwareEvent

/*************************************************************
AwareSource
This is a kind of event source, so that it is inherited from
CEventSource. The data of awaresource is from awareness tool
instead of awareevent.
*************************************************************/
class CAwareSource : public CEventSource

/*************************************************************
Boolean Conditioner
This is a kind of conditioner that compares "AND" or "OR"
operations, so that it is inherited from event conditioner.
There are two sources of boolean conditioner, called left
source and right source.
the event sources of boolean conditioner must be awareevent,
instead of awaresource.
*************************************************************/
class CBooleanConditioner : public CEventConditioner

/*************************************************************
Compare Conditioner
This is a kind of conditioner that compares "BIGGER","SMALLER",
"BETWEEN", or "BESIDES" operations, so that it is inherited
from event conditioner.
There is only one event source of compare conditioner.
the event sources of boolean conditioner must be awaresource,
instead of awareevent. so that the data is directly from
awareness measurement tools
*************************************************************/
class CCompareConditioner : public CEventConditioner

/*************************************************
CInifile
Load and analyze configuration file
*************************************************/
class CIniFile

/*************************************************
directory service client
```

```
This class is the client side directory server. It is
used to communicate with directory server.
*****************************************************/
class CModsClient
```

4.3 AwareWare application development

All the source code are implemented in Microsoft Visual Studio 2005

1. Set the environment:
   Add the additional dependencies in the environment: DCEngine.lib

2. Define the architecture change messages and response functions:

```
int gnArchChangeEventApp= RegisterWindowMessageA(szArchChangeEvent);

ON_REGISTERED_MESSAGE(gnArchChangeEventApp, OnReceiveArchChange)
```

You can also define the local data process done message and received data process done messages:

```
int gnActionDoneEvent = RegisterWindowMessageA(szActionDoneEvent);
int gnReceivDoneEvent = RegisterWindowMessageA(szReceivDoneEvent);
```

3. Start Engine:

```
// create a new decision engine instance
pdcEngine = new CDCEngine();
// set the xml file directory
pdcEngine->SetXMLFile("dcSync.xml"); //"dc.xml" is the default xml file;
// Initialize the decision engine: load and parse the XML file, construct the architectures
pdcEngine->Initialize();
// set the application instance, decision engine would send messages to this window
pdcEngine->SetHWnd(m_pLocaview);

// set the components parameters:
pdcEngine->SetAppParam("Grab", GRAB_APP_PARA_MAINWND, (void *)&hlocalwnd);
pdcEngine->SetAppParam("Grab", GRAB_APP_PARA_DISPWIDTH, (void *)&localwidth);
pdcEngine->SetAppParam("Grab", GRAB_APP_PARA_DISPHEIGHT, (void *)&localheight);
pdcEngine->SetAppParam("Grab", GRAB_APP_PARA_AUTOCAPTURE, (void *)&bAutoCapture);
pdcEngine->SetAppParam("Grab", GRAB_APP_PARA_SAVEMODE, (void *)&bLocalShow);

pdcEngine->SetAppParam("Display", DISP_APP_PARA_MAINWND, (void *)&hRemotewnd);
pdcEngine->SetAppParam("Display", DISP_APP_PARA_DISPWIDTH, (void *)&remotewidth);
pdcEngine->SetAppParam("Display", DISP_APP_PARA_DISPHEIGHT, (void *)&remoteheight);

// start the decision engine, start the measurement tools and adjust the architecture.
pdcEngine->Start();
```

4. Implement the message response functions

```
LRESULT CLocalView::OnReceiveArchChange(WPARAM wParam, LPARAM lPara)
{
    char buf[1024];
    CAwareDllTestDoc * pDoc = (CAwareDllTestDoc *)GetDocument();
    pDoc->pdcEngine->GetFormatValue(buf);

    SetStatusBar(buf);

    return 0;
}
```

5. Input data if needed and start to process the data. Processed the data can also be achieved from the output interface.

```
// input the data, and data size
GetDocument()->pdcEngine->SetMetaObjectInput(in, inlen);
// start to process the data
GetDocument()->pdcEngine->StartMetaProcess();
// get the output data, and processed data size
GetDocument()->pdcEngine->GetMetaObjectOutput(&out, &outlen);
```

6. Stop the decision engine

```
if (pdcEngine != NULL) {
    // stop the decision engine
    pdcEngine->Stop();
    // release the memory
    delete pdcEngine;
    pdcEngine = NULL;
}
```

5. Source File List:

Video-conferencing application:

Table I. The classes and files in the video-conferencing application

Classes	Files	Code lines
CDCEngine	DCEngine.h	77
	DCEngine.cpp	6628
	ParserComm.h	14
	ParserComm.cpp	7062
CDCFormat	DCFormat.h	46
	DCFormat.cpp	150
CAware	Aware.h	65
	Aware.cpp	73
CAvailableCPU	AvailableCPU.h	31
	AvailableCPU.cpp	786
CAvailableBW	AvailableBW.h	31
	AvailableBW.cpp	585
CAwareEvent	AwareEvent.h	47
	AwareEvent.cpp	37
CSimpleEvent	SimpleEvent.h	33
	SimpleEvent.cpp	267
CCompositionalEvent	CompositionalEvent.h	35
	CompositionalEvent.cpp	688
CEventSource	EventSource.h	39
	EventSource.cpp	11
CAwareSource	AwareSource.h	59
	AwareSource.cpp	805
CEventConditioner	EventConditioner.h	40
	EventConditioner.cpp	19
CCompareConditioner	CompareConditioner.h	55
	CompareConditioner.cpp	1151
CBooleanConditioner	BooleanConditioner.h	51
	BooleanConditioner.cpp	1184
CEventListener	EventListener.h	33
	EventListener.cpp	13
CAwareActuator	AwareActuator.h	49
	AwareActuator.cpp	1121
CAwareMetaObject	AwareMetaObject.h	45
	AwareMetaObject.cpp	3299

Firefighter training application:

Table II. The classes and files in the firefight training application

Classes	Files	Code lines
CModsServer	modsServer.h	110
	modsServer.cpp	574
	modsCommon.h	122
	modsCommon.cpp	476
CModsClient	modsClient.h	71
	modsClient.cpp	272
CPosToIdMapping	PosToIdMapping.h	158
	PosToIdMapping.cpp	663
CFire	fire.h	56
	fire.cpp	200
CFirefighter	Firefighter.h	53
	Firefighter.cpp	145
CLine	Line.h	40
	Line.cpp	102
CPath	Path.h	40
	Path.cpp	102
CGUIDlg	GUIDlg.h	156
	GUIDlg.cpp	1172
	Other files	1686

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).