5 Hot-WIMP+: an ACEE-based Software Framework

As the title of this chapter points out, *Hot-WIMP+* is a software framework based on the *ACEE* architecture. It can provide *white box technique*, *black box technique*, and *visual tools* for programming a *WIMP+ user interface*. It is built upon VisualWorks Smalltalk 2.5, which, through generations of evolution, embraces not only an object-oriented language but also a very attractive application framework. By use of the application framework of VisualWorks Smalltalk as the cornerstone for its construction, it is easier to achieve a quality framework prototype.

5.1 Application Framework of VisualWorks Smalltalk

The application framework of VisualWorks Smalltalk [111] [113] [114] is a canonical product for supporting the development of WIMP user interfaces. Its marrow is to impel separating the user interface part of an application from its domain model by use of the MVC architecture, which has been well-implemented and used by the framework itself. The framework provides also different arts and sciences, i.e., the white box technique, the black box technique, and visual tools for building a WIMP user interface, too. For example, a WIMP user interface, which consists of a set of *WIMP user interface objects*, i.e., windows and *WIMP*

widgets[30], can be built either with the help of its user interface builder or by coding manually with its System Browser.

In order to help integrate these WIMP user interface objects together with their separated domain models for providing an entire system functionality, the framework provides the class *ApplicationModel*. *ApplicationModel* is used to create application models for controlling the interaction among the related WIMP widgets and establishing the connections between these widgets and the aspects of their domain models. Furthermore, the framework provides different *ValueModel*, e.g., *ValueHolder*, *PluggableAdaptor*, in order to further help define diverse relationships between the related WIMP widgets and their application models or domain models.

The whole picture of the application framework of VisualWorks Smalltalk is shown in Fig. 20.

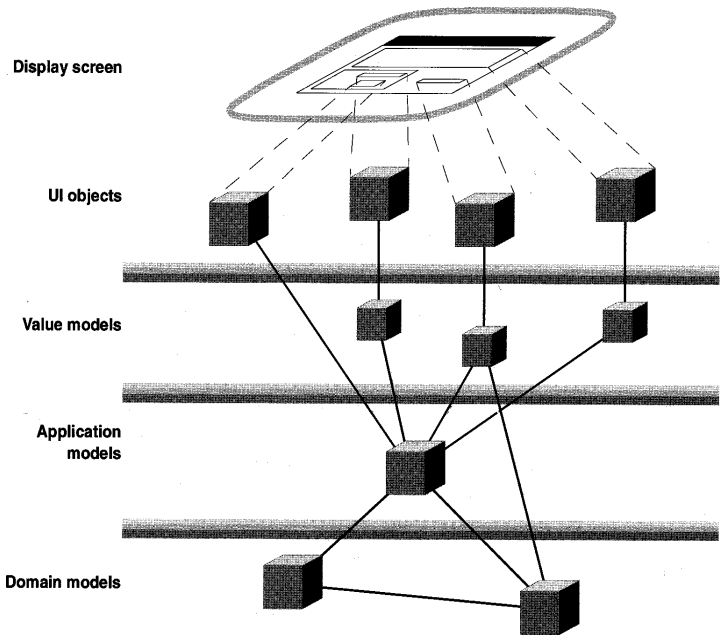


Fig. 20: The layer structure of the application framework of VisualWorks Smalltalk[14]

[30] WIMP Widgets are those widgets based on the desktop metaphor. In VisualWorks Smalltalk they can be divided into two categories: data widgets and action widget. Data widgets are used to display some aspect of a domain model and/or collect it from user, e.g., input field, list, and text editor. Action widgets are used to enable a user to invoke an application's actions, e.g., button and menu.

5.2 The Programming Layers of Hot-WIMP+

The goal of *Hot-WIMP+* is to provide a software framework similar to the application framework of VisualWorks Smalltalk for supporting the development of *WIMP+ user interfaces*. Based upon the application framework of VisualWorks Smalltalk, *Hot-WIMP+* embraces the components of *ACEE* and a set of *ACEE* visual tools. With *Hot-WIMP+*, developers can build up their *WIMP+ user interfaces* from different layers, as shown in Fig. 21. The higher layer is used, the easier to learn and use for developers.

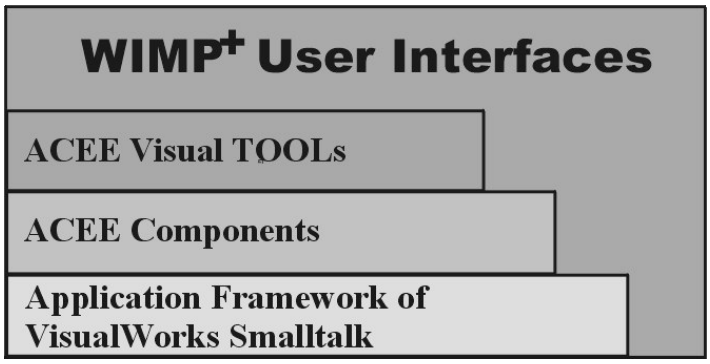


Fig. 21: The multi programming layers of Hot-WIMP+

5.3 Components of Hot-WIMP+

From the object-oriented programming point of view, a component consists of a set of well-defined collaborating classes for providing a specific functionality. It can provide larger granularity reuse than a class [60]. To reuse a component, developers just need to know the syntax and semantics of its external programming interface; they do not need to learn its internal construction in detail.

As described in chapter 4, each of the four *ACEE* components: *Acquisition*, *Computation*, *Expression*, and *Execution* has its well-defined functionality. Two Broadcaster/Listeners mechanisms are the backbone to bridge these four components together. The general strategy for building up the *ACEE* architecture has already been introduced in chapter 4. Here the concrete components of *ACEE* of *Hot-WIMP+* will be elaborated. And to be concise, in the following description, only the programming interfaces of the components will be introduced.

Most of their trivial variables and methods will be omitted. And since the Broadcaster/Listeners mechanism, which is used as the cornerstone of *Hot-WIMP+*, has been well provided as the dependency mechanism by the application framework of VisualWorks Smalltalk[31], it will be treated as a default functionality of *Hot-WIMP+*.

5.3.1 Fundamental Classes

5.3.1.1 Classes for Establishing Low Level Connection with Sensor and Effector

As described in section 2.2.2 of this thesis, a *WIMP+ user interface* deals with a *controlled process* and captures data from *sensors* and leverage the *controlled process* via *effectors*. Hence, it needs design and code for establishing connection with these *sensors* and *effectors*. In *Hot-WIMP+*, *AcceptorConnectorAM*, *Acceptor* and *Connector* are three classes which can be used to meet the demands.

5.3.1.1.1 AcceptorConnectorAM

AcceptorConnectorAM is a concrete class acting as a Facade [36] to provide a unified programming interface for creating a connectStream, a readConnectStream, or a writeConnectStream in order to connect an application with a *sensor* or an *effector*. A connectStream is an externalStream; a readConnectStream is an externalReadStream; and a writeConnectStream is an externalWriteStream [112]. These three streams are actually created by a concrete *Acceptor* or *Connector* (see following).

The *AcceptorConnectorAM* includes mainly four variables: readConnectStream, writeConnectStream, connectStream, and selector. The selector is a symbol used to specify if *Acceptor* or *Connector* is used to establish connection with peripheral devices.

---

[31] In VisualWorks Smalltalk, the dependency mechanism has been well defined in the class *Object*. Since the class *Object* is the superclass of every class, each class inherits the dependency mechanism automatically as it is created.

The method for creating an instance of *AcceptorConnectorAM* is:

```
AcceptorConnectorAM new.
```

Besides, within the initialize method of an *AcceptorConnectorAM*, the value of selector should be assigned:

```
"use Connector to establish connection with peripheral devices"
selector:= #selectConnector asValue.
```

or

```
"use Acceptor to establish connection with peripheral devices "
selector:= #selectAcceptor asValue.
```

5.3.1.1.2 Acceptor and Connector

Acceptor and *Connector* are two abstract classes which include two sets of concrete subclasses, each of which encapsulate different connection strategies for connecting with a `sensor` or an `effector`: *Acceptor* is used for passively connecting while *Connector* is used for actively connecting. Passively connecting means a server can only passively waiting for the connection from its clients. For example, if an *Acquisition* uses *Acceptor* as the connection strategy connecting with a `sensor`, the *Acquisition* can only passively wait for the connection from this `sensor`. On the contrary, actively connecting means a client can actively request for connecting with its server. In terms of the application demands, the client-server pair of *Acceptor* or *Connector* can be mounted on the same computer or two computers.

Since *Acceptor* and *Connector* are only two abstract classes used to group concrete *Acceptor* and *Connector*, for a concrete connection strategy, a concrete class of *Acceptor* or *Connector* must be created individually. For example, `Hot-WIMP+` includes two concrete classes of *Acceptor* and *Connector*, *TCPAcceptor* and *TCPConnector* which use the TCP communication protocol, for establishing passive and active connection with a `sensor` or an `effector`, respectively.

To create an instance of *TCPAcceptor*, aTCPAcceptor, the following message is necessary:

```
aTCPAcceptor:=TCPAcceptor with: aPortNumber.
```

And the message:

```
aTCPConnector:=TCPConnector with: aHostName with: aPortNumber.
```

is used to create an instance of *TCPConnector*, aTCPConnector.

Here aHostName and aPortNumber are used to hold the host name and the TCP port number.

By sending messages of connectStream, readConnectStream, or writeConnectStream to aTCPAcceptor or aTCPConnector, a connectStream, a readConnectStream, or a writeConnectStream can be created. For example,

```
aTCPConnector connectStream.
aTCPConnector readConnectStream.
aTCPConnector writeConnectStream.
```

5.3.1.2 Handlers

A *WIMP+ user interface* deals with traditional manipulation devices, *sensors*, and *effectors*. For reacting to traditional manipulation devices, the *Controller* class of VisualWorks Smalltalk can directly be applied. But for reacting to *sensors* and *effectors*, special handlers, *ACEE_AcquisitionHandler* and *ACEE_ExecutionHandler* which are used to process the service requests from *sensors* and create commands for instructing *effectors*, are required.

5.3.1.2.1 ACEE_AcquisitionHandler

ACEE_AcquisitionHandler is an abstract class. It has solely one instance variable, event, and two methods for accessing and assigning the variable, event and event:. It is used to group concrete subclasses, each of which encapsulates a set of handler methods for reacting to the events queued in a readConnectStream. This readConnectStream, as described in section 5.3.1.1.1, is used to connect an application with a *sensor*. For example, it contains a concrete class *RobotAcHandler* used to react to the events occurring in robot interface (see chapter 6), which includes four handlers: armHandler:, gripperHandler:, turntableHandler:, and pushbuttonHandler:.

5.3.1.2.2 ACEE_ExecutionHandler

ACEE_ExecutionHandler is an abstract class. It has two instance variables, event and driveCommandQueue. It provides four methods for accessing or assigning their values: event, event:, driveCommandQueue, and driveCommandQueue:. Like *ACEE_AcquisitionHandler*, it is also used to group concrete subclasses. Each of these subclasses encapsulates a set of handler methods for reacting to the change events occurring in a *Computation* component and producing commands. These commands are firstly put into a driveCommandQueue and then transmitted via a writeConnectStream to an *effector* of a *controlled process*.

5.3.1.3 ACEE_HandlerManager

Although handlers in a subclass of *ACEE_AcquisitionHandler* or *ACEE_ExecutionHandler* have encapsulated different strategies for processing different service requests, it is not clear which handler should be allocated to which request. This is the obligation of *ACEE_HandlerManager*.

ACEE_HandlerManager is a concrete class. Two of its major instance variables are eventHandlerDict and selectDispatch. An eventHandlerDict is an instance of *Dictionary* and records an *event-handler map*, i.e., which handler corresponds to which event. This implies that to dispatch a handler to an event appropriately, the eventHandlerDict must be assigned. A selectDispatch is a symbol which specifies which pattern, Reactor or Proactor, is used to manage the operations of all the dispatched handlers. The Reactor pattern supports synchronous operations while the Proactor pattern supports asynchronous operations [92].

To create an *ACEE_HandlerManager* with the Reactor pattern, the following message can be used to:

```
ACEE_HandlerManager with: #reactor.
```

5.3.2 ACEE_Acquisition

As already described, the *Acquisition* component of *ACEE* is responsible for capturing data from *sensors* and manipulation devices for the status of a *controlled process* and user's intention. For reacting to the traditional manipulation devices such as the mouse and

keyboard, the *Controller* class of VisualWorks Smalltalk can directly be used. For reacting to other continuous and non-standard input from a `sensor`, *Acquisition* should provide necessary reusable design and code.

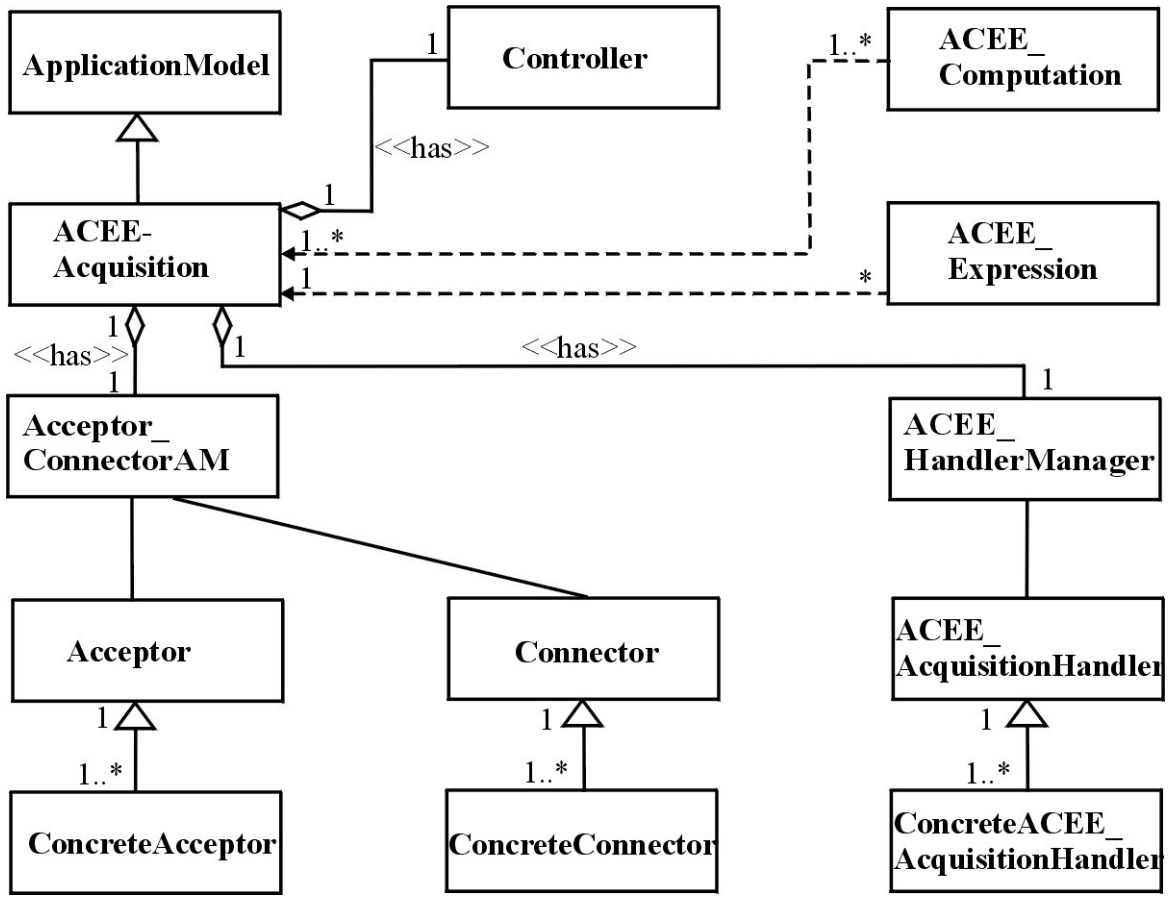


Fig. 22: The class diagram of ACEE_Acquisition

In `Hot-WIMP⁺`, *Acquisition* of `ACEE` is built up as *ACEE_Acquisition*. This *ACEE_Acquisition* is a concrete subclass of *ApplicationModel*. In addition to the *Controller* class of VisualWorks Smalltalk, it is also aggregated with classes of *AcceptorConnectorAM* and *ACEE_HandlerManager*, as shown in Fig. 22. Note that in Fig. 22, *ACEE_Computation* and *ACEE_Expression* which are the Listeners of *ACEE_Acquisition* are other two `ACEE` components of `Hot-WIMP⁺`.

With *AcceptorConnectorAM*, an ACEE_Acquisition can create a read stream, acStream, to record the incoming data from a `sensor`. With *ACEE_HandlerManager*, it can dispatch related handlers of a *ConcreteACEE_AcquisitionHandler* to process these data. The main

instance variables of *ACEE_Acquisition* is depicted in Table 3. To instantiate an *ACEE_Acquisition*, these instance variables should be specified[32].

Table 3: The main instance variables of ACEE_Acquisition

anACEE_Connect	an instance of *AcceptorConnectorAM*.
acStream	a read stream which records the incoming data from a `sensor`.
selectorAcMechanism	a symbol specifies which mechanism is used to capture the data in an acStream. In principle, two symbols which refer to sampling mechanism and event_driven mechanism are available: #selectSampling and #selectEvent_driven.
selectAcDispatch	a symbol specifies which pattern, Reactor or Proactor, is used to manage the operations of the dispatched handlers. There are two defined symbols, #selectReactor and #selectProactor which are used to specify the Reactor pattern and the Proactor pattern, respectively.
anAcHandlerManager	an instance of *ACEE_HandlerManager*.
handlerClass	specifies which *ConcreteACEE_AcquisitionHandler* is used to react to an acStream.
acTime	records the time interval for sampling.
beginAt	specifies the beginning time for reacting to an acStream.
acquireProcess	a process for capturing data from an acStream. If selectorAcMechanism is #selectSampling, the process will sample the data in terms of the time defined in acTime and beginAt. If selectorAcMechanism is #selectEvent_driven, the process will capture the data once the defined event occurs.

Following is a set of messages for creating an ACEE_Acquisition, anAcquisition.

```
anAcquisition:=ACEE_Acquisition new.
```

[32] In this case, *Controller* is not considered.

```
anAcquisition selectorAcMechanism:#selectSampling.
anAcquisition anACEE_Connect: TCPConnector with:'Peisu' with:60000.
anAcquisition acStream: anAcquisition anACEE_Connect readConnectStream.
anAcquisition anAcHandlerManager:(ACEE_HandlerManager with:#selectReactor ).
anAcquisition handlerClass:(Smalltalk at:#HandlerClass) new.
anAcquisition anAcHandlerManager aHandlerClass:anAcquisition handlerClass.
anAcquisition acTime:20.
anAcquisition beginAt: Time now asValue.
anAcquisition acquire.
```

This anAcquisition uses sampling mechanism to capture data and put them in acStream which in this example is created as a TCP externalReadStream and connected with host Peisu through port 60000. It uses the Reactor pattern to manage the operations of the dispatched handlers. These handlers are encompassed within a class named HandlerClass. The sampling time interval, acTime, which is usually determined by Sampling Theorem according to the signal frequency of the `sensor` it connects, in this case is 20 milliseconds. The sampling action will begin instantly. Note that the last message is used to activate acquireProcess. Of course, as described in section 5.3.1.3, before anAcHandlerManager can work, its instance variable eventHandlerDict must be assigned.

5.3.3 ACEE_Computation

In `Hot-WIMP+`, *Computation* of `ACEE` is implemented as *ACEE_Computation*. The *ACEE_Computation* is an abstract class and is defined as a subclass of *Model*. In addition to the instance and class variables inherited from its superclasses, *Model* and *Object*, *ACEE_Computation* does not include additional variable.

Since *ACEE_Computation* is a subclass of *Model*, an *ACEE_Computation* will automatically occupy all the variables and methods defined by classes *Model* and *Object*. This implies that an *ACEE_Computation* can react to the change messages sending by any

component in which it has registered its interesting[33]. This reaction is fulfilled by implementation of a corresponding update message within itself. It can also notify its changes to its Listeners by sending change messages to itself, too. All the change and update messages are inherited from *Object*.

With substantial classes of VisualWorks Smalltalk, an *ACEE_Computation* can be created relative easily. Generally, an *ACEE_Computation* is application domain specific. Its variables and methods operating on them are decided by the knowledge base or algorithms specific to the application domain. Therefore, the variables and methods of an *ACEE_Computation* vary from that of other *ACEE_Computation* by a great deal. For example, an *ACEE_Computation* of `LLDemo` and `Hot-Demo` (see chapter 6 of this thesis for details), *RobotComputation*, is defined to encapsulate a robot kinematical model. It includes a set of instance variables, e.g., armLength, bodyRadius, height, etc., and methods operating on them. Another example is *Checkbook*, which can be seen as an *ACEE_Computation*, of the "*Checkbook application*" demonstrated by [114]. It is defined to embrace registers of written checks and balance of checking account of a custom. It includes variables such as balance and register and methods operate on them which are totally different from that of *RobotComputation*.

5.3.4 ACEE_Expression

As already described, in `ACEE`, the aim of the *Expression* component is to represent the system states and provide manipulation possibilities to users. Since the *View* class of VisualWorks Smalltalk includes substantial WIMP user interface objects and the well-defined relationship among *Model* and *Controller*, in `Hot-WIMP+`, Expression of `ACEE` is defined as a subclass of *View*, *ACEE_Expression*. In addition to the variables it inherits from its superclasses, *DependentPart*, *VisualPart*, *VisualComponent*, and *Object*, *ACEE_Expression* has no

[33] Interesting can be registered by adding an *ACEE_Computation* as a Listener of a component.

additional variables. As *View* in VisualWorks Smalltalk, *ACEE_Expression* is also an abstract class.

A concrete subclass of *ACEE_Expression* will automatically inherit the ability of *View*. For example, it can update itself as its model[34] changes; it can talk with its controller[35] if necessary. Furthermore, it occupies all the variables and methods defined by the superclasses of *View*, *DependentPart*, *VisualPart*, *VisualComponent*, and *Object*. Hence, in addition to reacting to the change methods of its model, it can also react to the change messages sending by any component in which it has registered its interesting, e.g., an ACEE_Acquisition.

An *Expression* of a *WIMP+ user interface* consists of both traditional WIMP user interface objects and *Non-WIMP user interface objects* which are *not* based on the desktop metaphor. To create an *ACEE_Expression*, in addition to using of the well-defined WIMP user interface objects of VisualWorks Smalltalk, developers still need to create related *Non-WIMP user interface objects* according to the applications' demands. The layouts and contents of these *Non-WIMP user interface objects* can be created in terms of the static attributes of the real world objects they represent with the geometric classes of VisualWorks Smalltalk, e.g., *Lines*, *Circles*, etc. Their dynamic behaviors can be defined with the substantial classes of VisualWorks Smalltalk according to the applications' demands. The well-defined layouts and contents of *Non-WIMP user interface objects* should be embraced in display methods of an *ACEE_Expression*. These display methods can be invoked as their dynamic parts change.

For example, in *LLDemo* and *Hot-Demo* (see chapter 6 for details), a update message, update: anAS with: aParm, includes message for displaying a *Non-WIMP user interface object*, The Gripper, which reflects the target position of a robot gripper:

```
update: anAS with: aParm
```

---

[34] In this case, the model is an ACEE_Computation.

[35] In this case, controller is an Acquisition.

```
"if the goal is changed, invoke its display method"
| aGC |
aGC:=self graphicsContext.
anAS==#goalMove ifTrue:[ self displayGoal:aGC].
```

The layout and content of The Gripper is defined in displayGoal:aGC:

```
"display the goal on the desirable position"
|aRadius newPoint|
newPoint:=self initialX:model goal tXPosition andY:model goal tYPosition.
aRadius:=(model gripperRadius*self scale) asFloat.
aGC paint:ColorValue red.
(Circle center: newPoint radius: aRadius) asFiller displayOn:aGC.
self goalOldXY:newPoint.
```

Additionally, as a *View* in VisualWorks Smalltalk, each *ACEE_Expression* should overwrite the displayOn:aGraphicsContext method accordingly in order to respond to the messages sent by its window when the window opens, is resized, or is recovering from a damage [53] [112].

5.3.5 ACEE_Execution

As already described, the *Execution* component of `ACEE` is used to react to the change events occurring in *Computation* and produce relevant commands to drive `effectors` to leverage a `controlled process`. To fulfill this requirement, in `Hot-WIMP+`, *Execution* of `ACEE` is implemented as *ACEE_Execution*. The *ACEE_Execution* is built as a concrete subclass of *ApplicationModel*. It is aggregated with classes of *AcceptorConnectorAM* and *ACEE_HandlerManager*, as shown in Fig. 23.

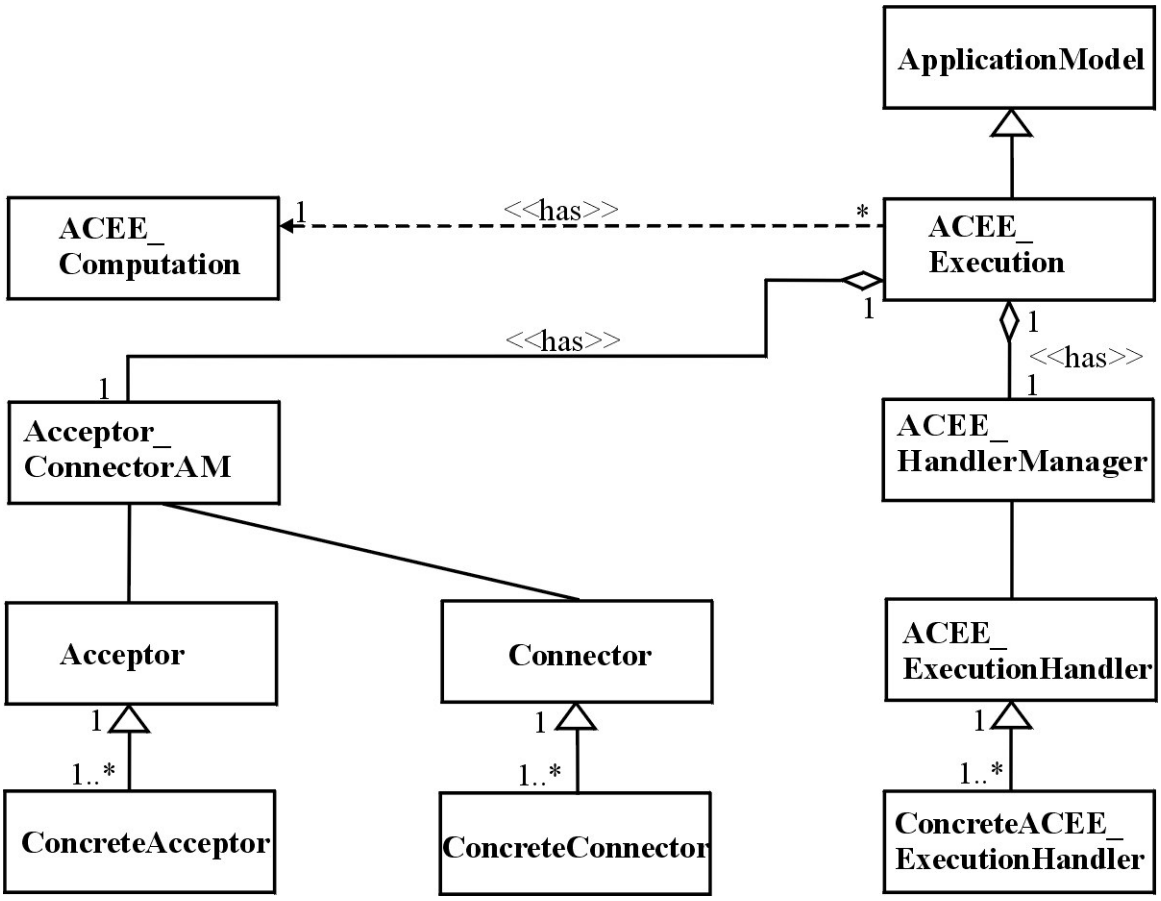


Fig. 23: The class diagram of ACEE_Execution

With *AcceptorConnectorAM*, an ACEE_Execution can create a write stream, ecStream, to connect with an `effector`. With *ACEE_HandlerManager*, it can dispatch relevant handlers of a C*oncreteACEE_ExecutionHandler* to process the change events occurring in its *ACEE_Computation*. These handlers respond to produce commands which are used to initiate the actions of an `effector` to leverage a `controlled process`. The main instance variables of *ACEE_Execution* are listed in Table 4. To instantiate an ACEE_Execution, these variables should be specified.

Table 4: The main instance variables of ACEE_Execution

anACEE_Connect	an instance of *AcceptorConnectorAM*.
ecStream	a write stream which is used to transfer commands to initiate actions of an `effector`.
selectEcDispatch	a symbol specifies which pattern, Reactor or Proactor, is used to manage the operations of the dispatched handlers. There are two defined symbols, #selectReactor and #selectProactor, which are used to specify the Reactor pattern and the Proactor pattern, respectively.
anEcHandlerManager	an instance of *ACEE_HandlerManager*.
handlerClass	specifies which *ConcreteACEE_ExecutionHandler* is used to react to the change events occurring in an *ACEE_Computation*.
executeProcess	a process for putting the produced commands into ecStream.

Following is a set of messages for creating an instance of *ACEE_Execution*, anExecution.

```
anExecution:=ACEE_Execution new.
anExecution anACEE_Connect: TCPConnector with:'Peisu' with:60000.
anExecution ecStream: anExecution anACEE_Connect writeConnectStream.
anExecution selectEcDispatch: #selectReactor asValue.
anExecution anEcHandlerManager:(ACEE_HandlerManager with:#selectReactor ).
anExecution handlerClass: (Smalltalk at: #HandlerClass) new.
anExecution anEcHandlerManager aHandlerClass:anExecution handlerClass.
anExecution drive.
```

This anExecution uses the Reactor pattern to manage the operations of the dispatched handlers which are encompassed within a class named HandlerClass, then puts the commands they produce in ecStream which in this example is created as a TCP externalWriteStream that connects with host Peisu through port 60000. Note that the last message is used to activate executeProcess. As instantiating an *ACEE_Acquisition*, before anEcHandlerManager can work, its instance variable eventHandlerDict must be assigned at first.

5.4 Visual Tools of Hot-WIMP+

Although components of *ACEE_Acquisition*, *ACEE_Computation*, *ACEE_Expression*, and *ACEE_Execution* can ease the burdens of creation of *WIMP+ user interfaces* a great deal by providing well-defined white-box technique and black-box technique, they are still difficult to those use interface designers who do not want to touch with the code details presented in the form of classes and methods. The following visual tools of *Hot-WIMP+* can further help to hide the textual programming details.

5.4.1 Handler Manager

As described in section 5.3.1.3, in *ACEE_HandlerManager*, instance variable eventHandlerDict records an event-handler map which is used to adequately dispatch handlers to process events.

Handler Manager, as shown in Fig. 24, is a visual tool to help establish an event-handler map. To specify an event-handler map with *Handler Manager*, the following steps should be followed:

- Give the handlers' class name which includes the related handlers via a dialogue window. As the class name is given, all its handlers are listed in a handler list located on the upper right of the tool.
- Input all possible events via a dialogue window. Each new inputted event will appear in an event list located on the upper left of the tool.
- Map a handler to an event by selecting an event in the event list and a handler in the handler list, then accept the map by selecting the Accept the Map button. As the Accept the Map button is pushed down, a map occurs in an Event-Handler Map pane. Repeat this step again and again, until all the events are mapped to their corresponding handlers.
- Accept the map shown in the Event-Handler Map pane by selecting the Accept button. Otherwise, use the Cancel button to discard the map.

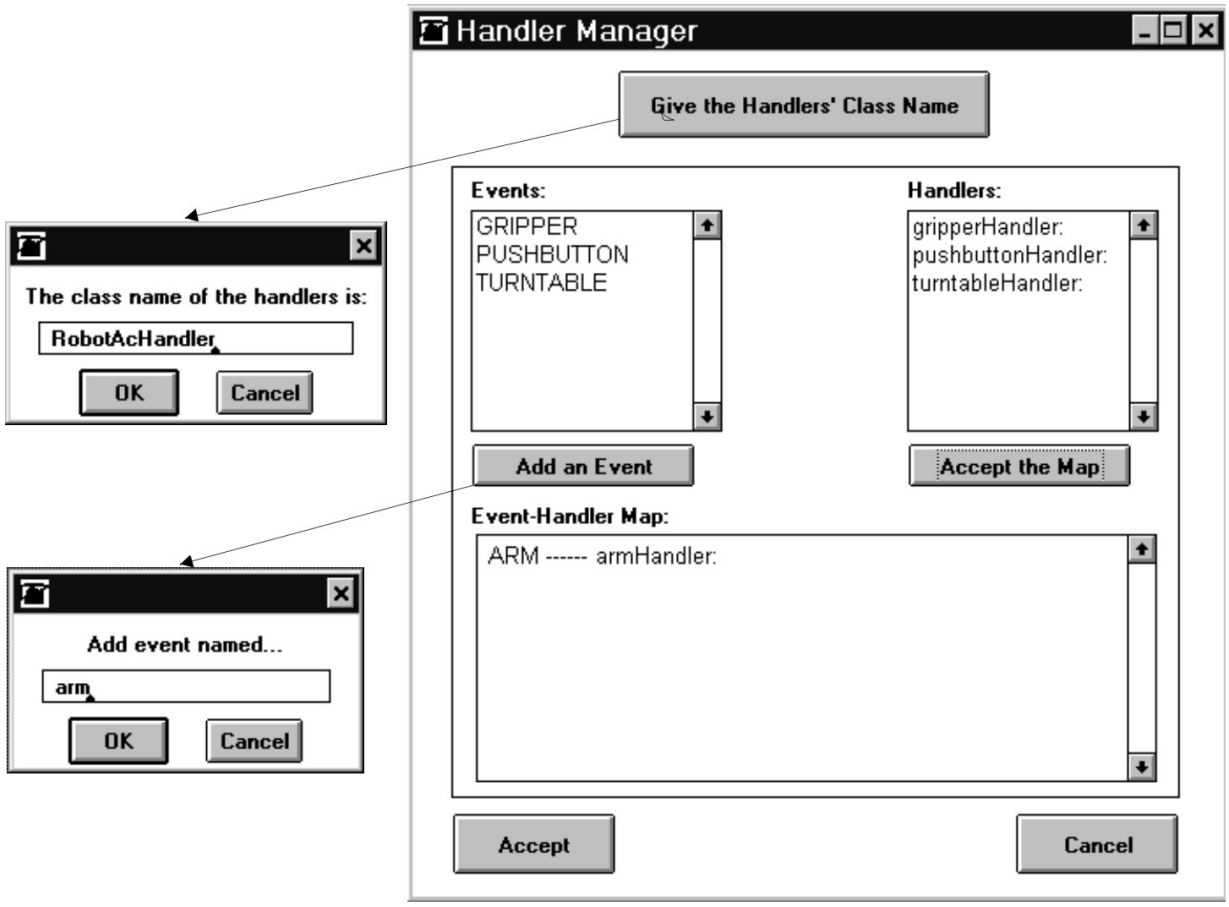


Fig. 24: Handler Manager

5.4.2 Connection Definer

`Connection Definer`, as shown in Fig. 25, is a tool for visually specifying a connection stream with a `sensor` or an `effector`. By selecting a connection strategy, Acceptor or Connector, all the relevant concrete connection strategies will occur in the upper right list of the tool. After selecting a concrete connection strategy and specifying the demanded parameters, a connection stream which connects with a `sensor` or an `effector` will be created. With it, developers can avoid touching the methods and parameters of the programming interfaces of *AcceptorConnectorAM*, *Acceptor*, and *Connector*.

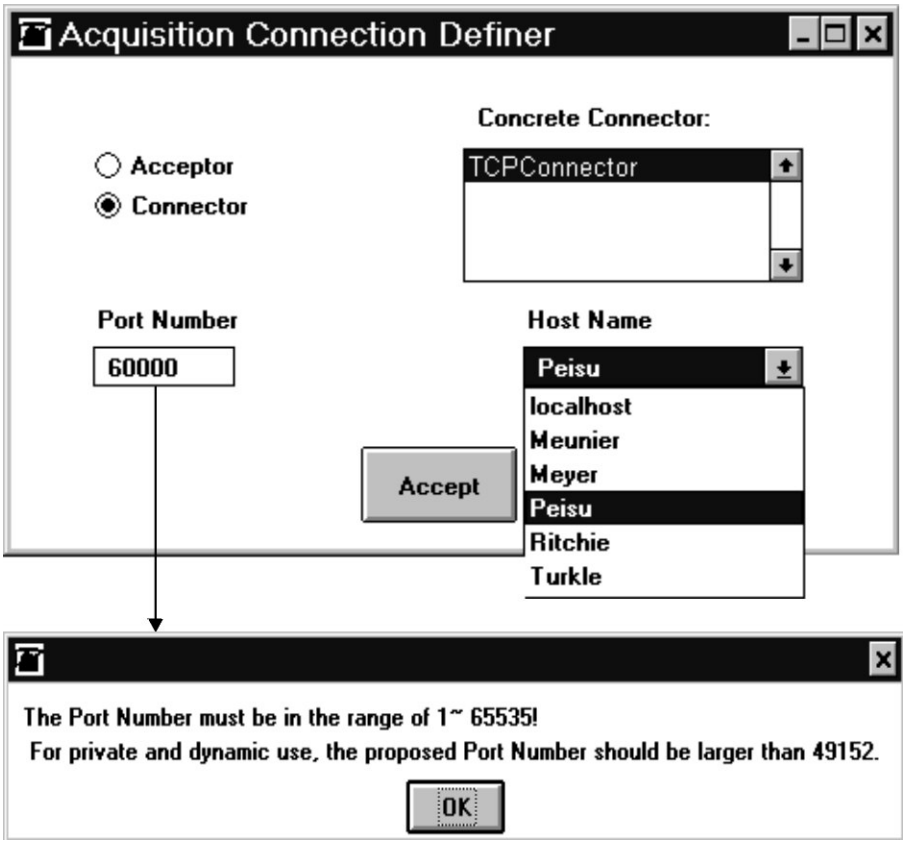


Fig. 25: Connection Definer

Fig. 25 demonstrates the use of `Connection Definer` to specify a connection stream with a `sensor` via *TCPConnector* for an *Acquisition*. In this case, the concrete connection strategy *TCPConnector* is selected[36]. With *TCPConnector*, two parameters, port number and host name, must be specified. The host name is the name of the computer which mounts a `sensor` or an `effector`. The port number is the TCP communication port[37] used to transfer the connection stream between the client and the server.

---

[36] Since in the prototype of `Hot-WIMP+` only *TCPAcceptor* and *TCPConnector* are implemented, therefore only *TCPConnector* is listed here.

[37] According to Internet Assigned Numbers Authority, for the private and/or dynamic use, the port number should range from 49151 to 65535 [87]. For security reason, when the inputted port number is less than 49151 or larger than 65535, the connection data stream can not be created with the Connection Definer.

5.4.3 Acquisition Definer

`Acquisition Definer` is a tool for visually programming a concrete *Acquisition*. As Fig. 26 shows, in addition to `Acquisition Handler Manager`[38] and `Connection Definer` which have been described above, it consists of other two visual specification tools, `Acquisition Mechanism` and `Strategy for Dispatching Handlers`.

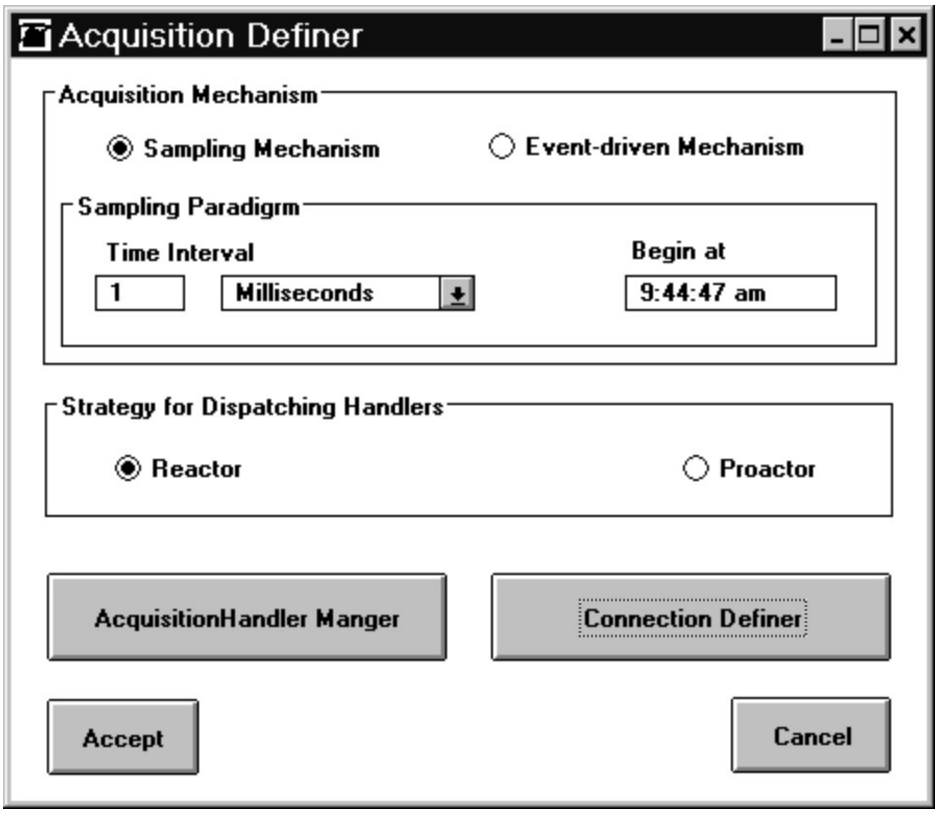


Fig. 26: Acquisition Definer

`Acquisition Mechanism` is a tool for specifying which mechanism, sampling mechanism or event-driven mechanism, is used to capture data and put them into an acStream. If the Sampling Mechanism radio button is pressed, the sampling mechanism will be selected and

[38] Acquisition Handler Manager is a `HandlerManager`, which is used to establish an eventHandlerDict for anAcHandlerManager of an *Acquisition*.

Sampling Paradigm specification tool will occur for further specifying the detailed sampling parameters. These sampling parameters are determined in terms of the characteristics of the *sensor* and the requirements of the application domain. For example, Time Interval and Begin at, are two parameters used to specify the sampling frequency[39] and the time for beginning the sampling action.

Strategy for Dispatching Handlers is used to specify using what kind of strategy to manage the operations of the dispatched acquisitionhandlers, Reactor or Proactor.

After finishing the specification of *Acquisition Mechanism*, *Strategy for Dispatching Handlers*, *Acquisition Handler Manager*, and *Connection Definer*, the creation of an ACEE_Acquisition is finished.

With *Acquisition Definer*, to define an ACEE_Acquisition is quite simple. Developers do not need to touch with the textual programming details which involve a set of variables and methods as described in section 5.3.2.

5.4.4 Expression Specification Tools

Tools of the user interface builder of VisualWorks Smalltalk, as shown in Fig. 27, can support visually programming traditional WIMP user interfaces. By opening a canvas with the Canvas tool, developers can select corresponding WIMP widgets in the Palette tool, put and arrange them in the Canvas, then use the Properties tool to specify their static attributes and the methods used to determine their dynamic behaviors.

[39] As already described, in most cases, sampling frequency is determined by Sampling Theorem [95].

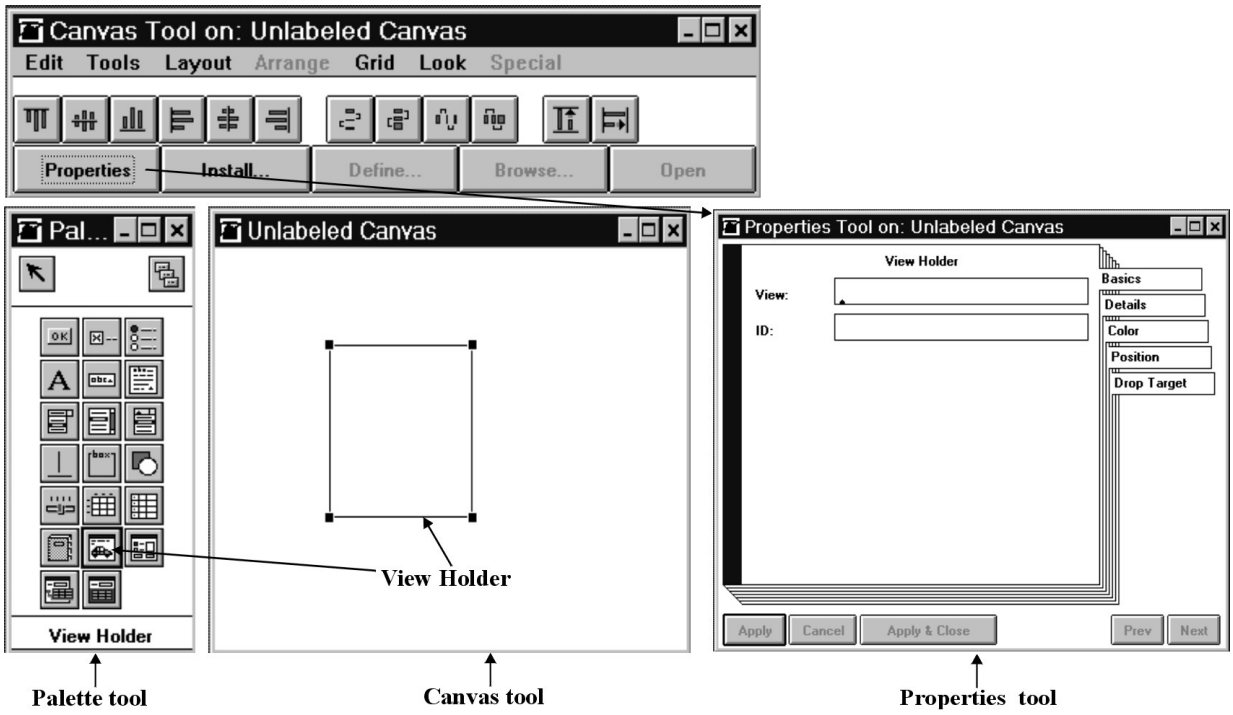


Fig. 27: Three major tools of the user interface builder of VisualWorks Smalltalk

Fortunately, these tools can be used to partially support visually programming the layout and content of an *Expression* of a *WIMP+ user interface*, too. An *Expression* consists of *WIMP+ user interface objects* which contain both traditional WIMP user interface objects and *Non-WIMP user interface objects*. For programming traditional WIMP user interface objects, e.g., window, radio button, label, etc., it is obvious that these tools can still be applied. For programming *Non-WIMP user interface objects*, the Canvas tool, the Palette tool, and the Properties tool can still be partially used.

To program *Non-WIMP user interface objects* with the Canvas tool, the Palette tool, and the Properties tool, the skill of programming View Holder widget of the Palette tool [111] [114] is required. Since there are substantial *Non-WIMP user interface objects* and each of which reflects an aspect of an application domain, to program such an object, manually programming is required. The View Holder and the Properties tool can help specify the location and size of a *Non-WIMP user interface object*, and further specify an Expression to determine its content, appearance, and dynamic behavior. This Expression is an instance of an *ACEE_Expression*.

5.4.5 Execution Definer

Execution Definer, as shown in Fig. 28, is a tool for visually programming a concrete *Execution*. In addition to *Execution Handler Manager*[40] and *Connection Definer*, it includes a *Strategy for Dispatching Handlers* tool.

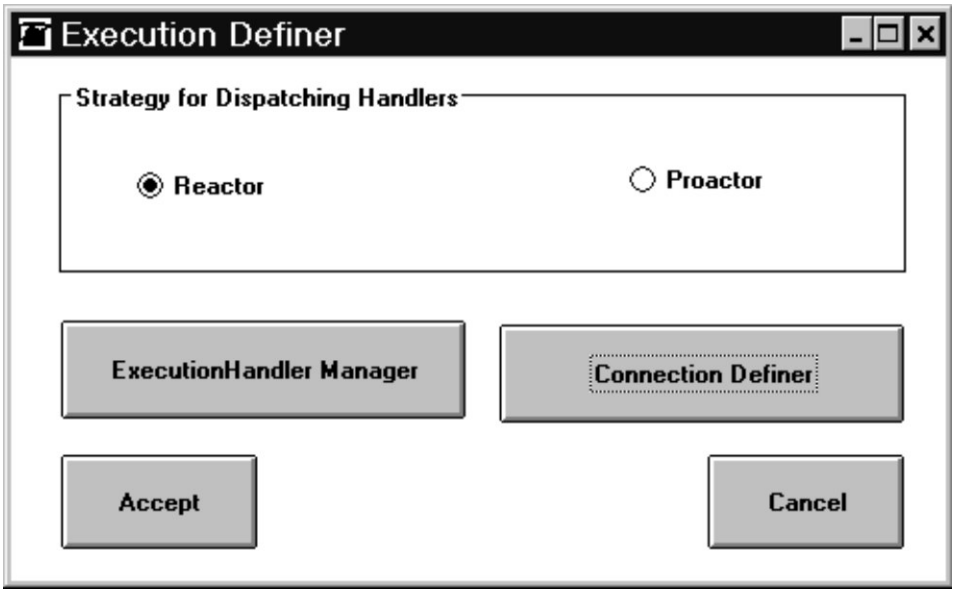


Fig. 28: Execution Definer

As *Strategy for Dispatching Handlers* of *Acquisition Definer*, *Strategy for Dispatching Handlers* here is also used to specify the strategy for managing the operations of the dispatched executionhandlers, Reactor or Proactor.

After the specifications of *Strategy for Dispatching Handlers*, *Execution Handler Manager*, and *Connection Definer* are finished (the operation details of *Handler Manager* and *Connection Definer* can be found in above sections), the creation of an ACEE_Execution is finished.

---

[40] *Execution Handler Manager*, is a *HandlerManager*, which is used to establish an eventHandlerDict for anEcHandlerManager of an *Execution*.

Obviously, creating an ACEE_Execution with *Execution Definer* is quite similar to creating an ACEE_Acquisition with *Acquisition Definer*: most specifications can be done only by mouse selection.

5.5 Put ACEE_Acquisition, ACEE_Computation, ACEE_Expression, and ACEE_Execution Together

With *Hot-WIMP+*, by use of the white-box technique, the black-box technique, and the visual tools described above, components of *ACEE_Acquisition*, *ACEE_Computation*, *ACEE_Expression*, and *ACEE_Execution* can be created relative easily. To construct a workable *WIMP+ user interface*, methods which are used to put all these components together in a reasonable way are still required. On the contrary, if a collaboration relationship among an *ACEE* quadruple does not need, methods for separating the relationship are required, too.

In VisualWorks Smalltalk, there are two generally used methods to fulfill the above requirements:

addDependent:

and

removeDependent:

As the name implies, the addDependent: method is used for adding a component as a Listener of its receiver. On the contrary, the removeDependent: method is used for removing the dependency relationship. And since *ACEE_Computation* and *ACEE_Expression* are two subclasses of *Model* and *View* of VisualWorks Smalltalk, the model: method, which is inherited from *ACEE_Expression*'s superclass *View*, *DependentPart*, can be used by an *ACEE_Expression* to specify the dependency relationship between an *ACEE_Computation* and an *ACEE_Expression*.

Following are two examples to show how to establish and separate the collaboration relationship. Example one, connecting, is used by *Hot-Demo* (see chapter 6 for details) to establish the collaboration relationship among a set of *ACEE_Acquisition*,

ACEE_Computation, *ACEE_Expression*, and *ACEE_Execution*. On the contrary, example two, separating, is used to remove the relationship.

connecting

"specify a Computation, aRobot, as a Listener of anAcquisition"

anAcquisition handlerClass addDependent:aRobot.

"specify an Expression, aRobotExpression, as a Listener of anAcquisition"

anAcquisition handlerClass addDependent:aRobotExpression.

"specify an Expression, robotMessageCollector, as a Listener of anAcquisition"

anAcquisition handlerClass addDependent:robotMessageCollector.

"specify an Expression, gripperExpression, as a Listener of anAcquisition"

gripperExpression model:anAcquisition handlerClass .

"specify an Expression, pushButtonExpression, as a Listener of anAcquisition"

pushButtonExpression model:anAcquisition handlerClass .

"specify an Expression, pushButtonExpression, as a Listener of a Computation, aRobot"

aRobot addDependent:pushButtonExpression.

"specify an Execution, anExecution, as a Listener of a Computation, aRobot "

aRobot addDependent:anExecution.

separating

"remove the dependency relationship between a Computation, aRobot, and anAcquisition"

anAcquisition handlerClass removeDependent:aRobot.

"remove the dependency relationship between an Expression, aRobotExpression, and anAcquisition"

anAcquisition handlerClass removeDependent:aRobotExpression.

"remove the dependency relationship between an Expression, robotMessageCollector, and anAcquisition"

anAcquisition handlerClass removeDependent:robotMessageCollector.

"remove the dependency relationship between an Expression, aRobotExpression, and its Broadcaster"

```
gripperExpression model:nil.
```

"remove the dependency relationship between an Expression, pushButtonExpression, and its Broadcaster"

```
pushButtonExpression model:nil.
```

"remove the dependency relationship between an Expression, pushButtonExpression, and its Computation, aRobot"

```
aRobot removeDependent:pushButtonExpression.
```

"remove the dependency relationship between an Execution, anExecution, and a Computation, aRobot"

```
aRobot removeDependent:anExecution.
```

5.6 Summary

This chapter elaborated on *Hot-WIMP+*, an *ACEE*-based software framework prototype designed and implemented by the thesis author herself. Since *Hot-WIMP+* uses the application framework of VisualWorks Smalltalk as its cornerstone, it inherits automatically all the design and code of the application framework of VisualWorks Smalltalk.

With *Hot-WIMP+*, user interface developers can build their *WIMP+ user interfaces* from different programming layers, they can reuse the well-defined components of *ACEE_Acquisition*, *ACEE_Computation*, *ACEE_Expression*, *ACEE_Execution* through class inheritances or/and their programming interfaces as described in section 5.3; or they can create directly the relevant components by use of visual tools, *Acquisition Definer*, *Expression Specification Tools*, or *Execution Definer*, as described in section 5.4.

Additionally, necessary examples were also given to demonstrate how to establish these components with the white-box technique, the black-box technique, and the visual tools of *Hot-WIMP+* and how to put the well-created *ACEE* quadruple together in order to form a workable *WIMP+ user interface*.

In the next chapter, in addition to *Hot-WIMP+*, the feasibility of all the patterns and frameworks that this work provides will be elaborated through *WIMP+ user interface* development practice.

6 WIMP+ Patterns and Frameworks in Practice

6.1 The Context of this Chapter

This chapter is a cocktail that tries to exhibit all the research results of the pervious chapters. Through developing *WIMP+ user interfaces* for controlling a model robot, this chapter demonstrates not only the practicability of *Hot-UCDP* in support of user-centered design development process but also the robustness, pluggability, flexibility, reusability, maintainability, and efficiency of the *ACEE* architecture and *Hot-WIMP+*.

According to *Hot-UCDP*, to develop a task-oriented user interface, the very first step is to seek application domain patterns which embrace abstract user interface models related to the users' tasks; then, apply HCI patterns to design the relevant user interface prototypes in terms of the abstract user interface models; finally, exploit software patterns and tools to implement these user interface prototypes as executable user interfaces. Consequently, this chapter will begin with a robot application domain pattern *SelectMe-ConveyMe-SettleMe*, which is revised from *Select Me, Convey Me, Settle Me* in [133], a position paper written by the thesis author and appeared at the Workshop of Pattern Languages for Interaction Design: Building Momentum at CHI2000. Then three executable *WIMP+ user interfaces* for fulfilling the task described in the robot application domain pattern will be introduced, respectively. Accompanying with each *WIMP+ user interface*, the related HCI patterns will be mentioned and a HCI pattern *Two-handed Manipulation* which is revised from *Two-handed Manipulation* in [133] will be described in detail.

To investigate the effect and efficiency of these three *WIMP+ user interfaces* for support of fulfilling the task for controlling the robot, a case study will be given. The features of these

WIMP+ user interfaces, design patterns, and frameworks will also be evaluated. Finally, several possible application domains are also mentioned.

6.2 SelectMe-ConveyMe-SettleMe: a Robot Application Domain Pattern

Name

SelectMe-ConveyMe-SettleMe

Intent

This robot application domain pattern tries to give a solution to support end users, especially those who are not interested in the hardware details of the robot, to interactively telecontrol the end-effector[41] of a 3-D robot arm to carry an object from a place, convey and put it into a target place with constraints.

Context

To telecontrol the end-effector of a 3-D robot arm, usually, there are three programming techniques available: TeachIn Programming, Textual Programming, and Visual Programming with Imperative Diagram [101] [117] [118]. But all these techniques require that end users know the hardware details of the robot in which many of them are not interested. Additionally, these techniques are short of intuitivity and flexibility for programming a new task. This pattern gives a solution to overcome the pitfalls of these techniques.

Example

As Fig. 29 shows, the Fischertechnik model robot [31] is a 3-D robot arm with an end-effector, a gripper. The robot arm is mounted on a turntable. Its gripper can be controlled by three

[41] In robotics, end-effector refers to the tip of a robot linkage used to manipulate a real world object. And in many robots, only the end-effector can be used to do manipulation.

motors: M1, M2, and M3. Its position can be detected by six pulse switches: E1, E2, E3, E4, E5, and E6. The functions of the motors and the pulse switches are depicted in Table 5.

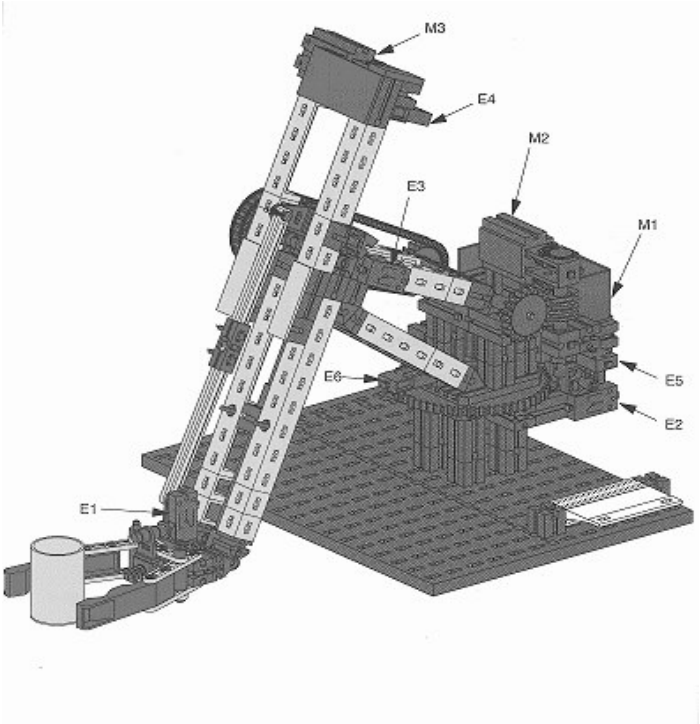


Fig. 29: The Fischertechnik model robot

Table 5: Motors and pulse switches of the Fischertechnik model robot

M1	is used to drive the turntable moving around which in turn drives the robot arm moving forward horizontally.
M2	is used to drive the arm raising or lowering.
M3	is used to drive the gripper opening or closing.
E1	is used to detect the gripper's full-opened position.
E2	is used to count the degree to which the arm has moved horizontally.
E3	is used to detect the limit vertical position of the arm.
E4	is used to count the width in which the gripper has been opened.
E5	is used to count the height that the arm has reached.
E6	is used to detect the limit horizontal position of the arm.

The constraints of the robot arm are: it can move maximally about 225° horizontally, its fully raised position is about 90 mm, and its gripper can open maximally about 70 mm in width.

The robot gripper can be controlled to carry a small object (less than 70 mm in width) from a place, convey and put it into a place with the above constraints.

Problem

How to effectively and efficiently support the end users who are not interested in the hardware details of the robot to telecontrol the end-effector of a 3-D robot arm to select an object, convey and put the object into a target place with constraints?

Forces

Traditionally, in robotics, to control the end-effector of a 3-D robot arm as the one in above example, end users can use TeachIn Programming technique or Textual Programming technique [117] [118]. But TeachIn Programming is very inflexible and requires that the end users know the low level hardware details of the robot, e.g., the mechanical and electric functions of the robot. Although Textual Programming is more flexible, it requires that the end users to be professional at a Textual Programming Language like C or C^{++}. It is very complex to most of the end-users. Moreover, using a Textual Programming can not help avoid the requirement of knowing the hardware details. Presently, Visually Programming with Imperative Diagrams, e.g., LLWin [68], which by providing the end users with a set of graphical commands, can improve greatly the efficient of programming a task for control the end-effector of a 3-D robot arm. But Visually Programming with Imperative Diagrams still requires that the end-users have some knowledge of the robot hardware. Moreover, to program a new task requires the end users to establish a new diagram. It could be not so interesting to some of the end users, e.g., children under 10 years old.

Additionally, the above techniques can not provide adequate and intuitive feedback, e.g., the current status of the robot arm, to the end users if the robot is put at a remote site. Although video camera can help establish intuitive feedback, using video camera is expensive. Additionally, transmitting large amount of real time pictures for telecontrol requires high speed communication channel. If a low bandwidth public communication channel such as the Internet

is used for telecontrol, it should avoid to transmit large amount of real time pictures for telecontrol since:

- A limited bandwidth and communication channel such as the Internet usually possesses communication delay while picture transmission requires high speed and bandwidth. And the delayed arrived pictures can not help the user much for further programming a task at the most time. Even worse, the user could spend lot of time to wait for the arriving of the pictures which could result in low work efficiency.

- An open public shared communication channel such as the Internet is accessible to millions of users. An application should not obstruct the communication channel too long.

Solution

Create a *WIMP+ user interface* for *Visually Programming by Direct Manipulation* [99]:

- Use *WIMP+ user interface objects* to reflect the relevant real world objects of the robot, e.g., the current status of the robot, the target position of the end-effector, etc.

- Provide the end users with the manipulation devices like the mouse to do direct-manipulation.

- Let the end users program a task for telecontrolling the end-effector of the robot by Selecting the relevant *WIMP+ user interface objects*, Conveying it to the target position, and then Settling it in that position within the *WIMP+ user interface*.

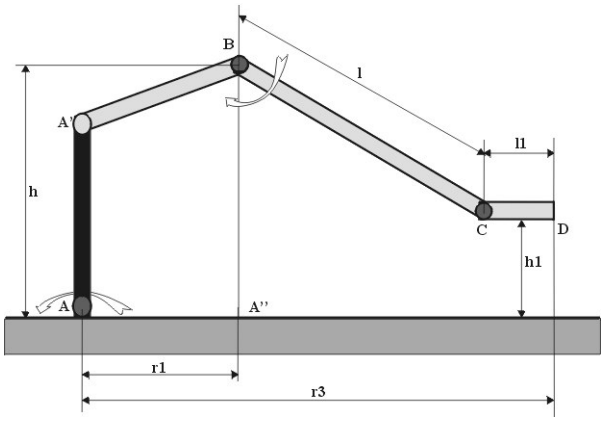
Domain Model

This domain model includes the kinematical structure and the equations to describe the example robot.

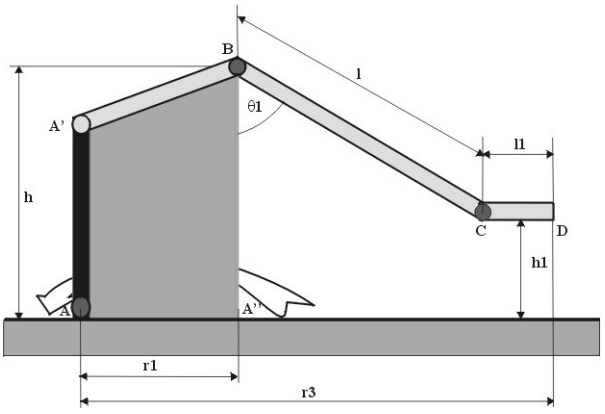
The Kinematical Structure of the Robot

Due to intuitive and natural, in robotics Cartesian space control is widely used [91] [117]. With a *WIMP+ user interface*, to control the end-effector of a robot arm, users just need to specify the relevant Cartesian coordinates via the user interface. They do not need to touch directly with the robot joint space which is not interesting and very complex to most of them. The difficulty of specifying each robot joint position is taken over by the *WIMP+ user interface* which integrates the related robot kinematical model to convert the Cartesian coordinates into the concrete positions for each robot joint and link.

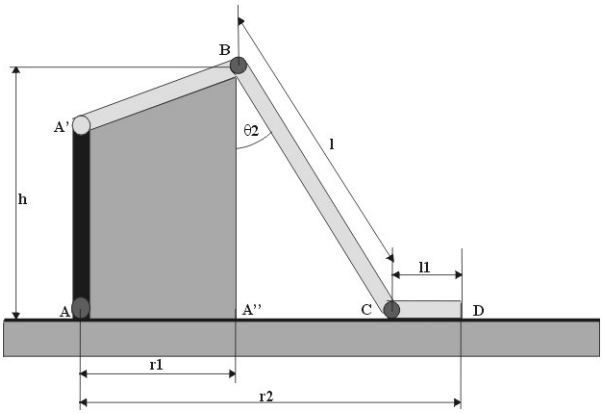
Usually, the complexity of a kinematical model of a robot depends on its degrees of freedom. The example robot can be seen as with three degrees of freedom, each degree is controlled by a motor. Suppose the robot is put at a table, its kinematical structure can be depicted as Fig. 30 (a)-(e). The variables used in Fig. 30 (a)-(e) as well as the following equations are listed in Table 6.



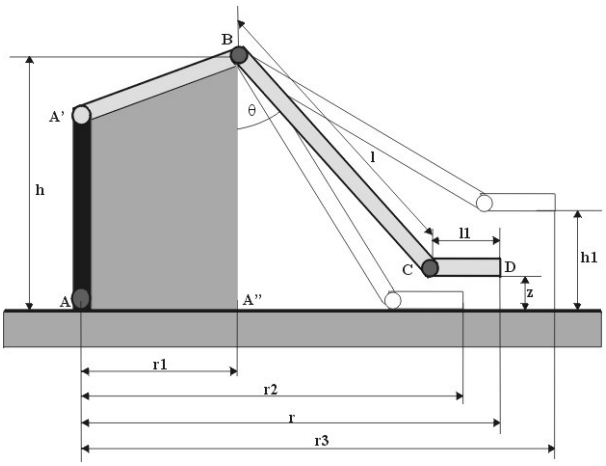
(a)



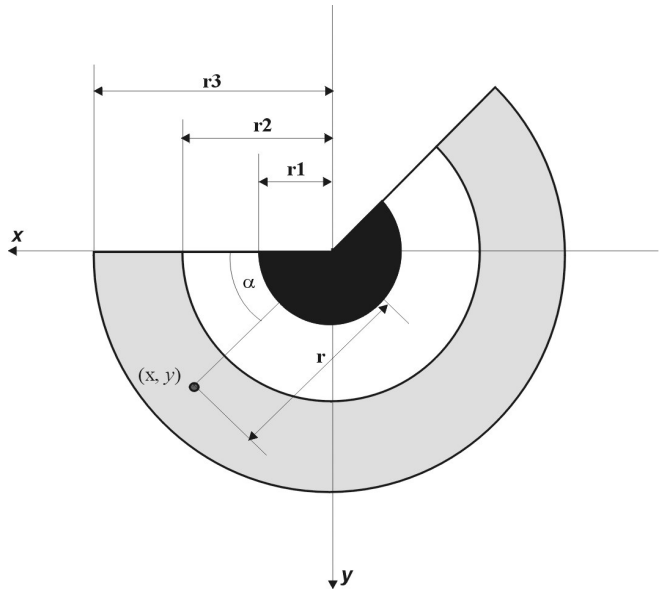
(b)



(c)



(d)



(e)

Fig. 30: The kinematical structure of the Fischertechnik model robot

Table 6: Variables used to depict a kinematical model of the Fischertechnik model robot

r1	the radius of the robot body, i.e., the distance between A and A'', as shown in Fig. 30 (a)-(e). It is a constant.
r2	the minimum distance between the gripper's tip and axis AA', as shown in Fig. 30 (c), (d), and (e).
r3	the maximum distance between the gripper's tip and axis AA', as shown in Fig. 30 (a), (b), (d), and (e).
r	the distance between the gripper's tip and axis AA' as the arm reaches an arbitrary vertical position, as shown in Fig. 30 (d) and (e).
l	the length of link BC, as shown in Fig. 30 (a)-(d). It is a constant.
l1	the length of the robot gripper, i.e., the length of link CD, as shown in Fig. 30 (a)-(d). It is a constant.
h	the height of the robot, i.e., the distance between B to A'', as shown in Fig. 30 (a)-(d). It is a constant.
h1	the maximum height that the gripper can reach, as shown in Fig. 30 (a), (b), and (d). It is a constant.
θ1	the maximum vertical angle that link BC can reach, as shown in Fig. 30 (b).
θ2	the minimum vertical angle that link BC can reach, as shown in Fig. 30 (c).
θ	An arbitrary vertical angle of link BC, as shown in Fig. 30 (d).
x	the x coordinate of the gripper's tip, as shown in Fig. 30 (e).
y	the y coordinate of the gripper's tip, as shown in Fig. 30 (e).
z	the z coordinate of the gripper's tip, as shown in Fig. 30 (d).
α	the degree to which the robot arm has moved horizontally, as shown in Fig. 30 (e).

Fig. 30 (a) shows the original vertical kinematical structure of the robot as its arm in its fullly raised position. Since the robot arm, which consists of link A'B and link BC, can rotate around

link AA' and since link A'B is unmovable, link BC can be seen as rotating around a cylinder whose radius is r1 and axis is AA', as shown in Fig. 30 (b). Fig. 30 (c) shows another vertical kinematical structure of the robot as its arm reaches the surface of the table. Fig. 30 (d) is a synthesis of (b) and (c) which depicts the arm in an arbitrary vertical position. The horizontal kinematical structure of the robot is shown in Fig. 30 (e).

The Equations of a Kinematical Model of the Robot

Since the *WIMP+ user interfaces* will use a 2-D display to express information to end users and the mouse as the manipulation device, the robot kinematical model should be able to convert a 2-D input into the 3-D position of the robot gripper.

As Fig. 30 (b) shows, the geometric relationship among r1, r3, θ1, and l1 can be depicted as:

$$r3 = r1 + l * \sin\theta 1 + l1, \text{ where } \theta 1 = \arccos(\frac{h - h1}{l}) \quad \text{(i)}$$

From Fig. 30 (c),

$$r2 = r1 + l * \sin\theta 2 + l1, \text{ where } \theta 2 = \arccos(\frac{h}{l}) \quad \text{(ii)}$$

As shown in Fig. 30 (d),

$$r = r1 + l * \sin\theta + l1, \text{ where } \theta 2 \le \theta \le \theta 1 \text{ and } r2 \le r \le r3 \quad \text{(iii)}$$

$$z = h - l * \cos\theta, \text{ where } 0 \le z \le h1 \quad \text{(iv)}$$

According to the Pythagorean Theorem and Fig. 30 (e),

$$x^2 + y^2 = r^2 \quad \text{(v)}$$

The triangular relationship among x, y, and α can be depicted as,

$$\alpha = \arctan(\frac{y}{x}), \text{ when x>0, y>0} \quad \text{(vi)}$$

$$\alpha = \frac{\pi}{2}, \text{ when x=0} \quad \text{(vii)}$$

$$\alpha = \frac{\pi}{2} + \arctan(\left|\frac{y}{x}\right|), \text{ when x<0, y>0} \quad \text{(viii)}$$

$$\alpha = \pi + \arctan(\frac{y}{x}) \text{, when x<0, y<0} \quad \text{(ix)}$$

And since the turntable can move maximally about 225° horizontally, the moveable area of the robot arm is limited within a five eighth circle in plane, as shown in Fig. 30 (e). Therefore, α should meet the constraint of

$$0 \le \alpha \le 225^{\circ} \quad \text{(x)}$$

If a user specifies a 2-D Cartesian coordinate (x, y) which represent the gripper's tip via a *WIMP+ user interface*, the α to which the arm should move horizontally can be calculated out with the equation (vi)-(ix) and the constraint (x). And with the equation (v), r, which should meet the constraint of $r2 \le r \le r3$, can be calculated out. Then according to the following equation (xi), a variant of equation (iii), if r is known, the angle θ can be calculated out,

$$\theta = \arcsin\left[\frac{r-(r1+l1)}{l}\right] \text{, where r1 and l1 are constants} \quad \text{(xi)}$$

By use of equation (iv), z can be calculated out.

In summary, the equations of this kinematical model of the robot embrace equations (i)-(xi).

Known Uses

LLDemo, *Hot-WebRobi*, and *Hot-Demo* which will be introduced in the following sections are *WIMP+ user interfaces* for telecontrolling the robot gripper via Visually Programming by Direct Manipulation.

MARCO [16], as described in section 2.3.1, also involves a *WIMP+ user interface* for telecontrolling a space robot via Visually Programming by Direct Manipulation.

6.3 The Experiment Environment

To construct corresponding *WIMP+ user interfaces* for controlling the model robot as described in *SelectMe-ConveyMe-SettleMe*, an experiment environment which uses the

TCP/IP communication protocol to link a *robot client* and a *robot server*[42] together was established by the thesis author. There are three major reasons to use the TCP/IP protocol:

- It is the most widely used communication protocol nowadays. For example, the Internet, the largest network in the world and accessible to millions of users, uses it as its communication protocol.
- It has been integrated in many modern operating systems, e.g., Windows family, Unix, Linux, etc.
- It can provide an economic and flexible way, e.g., local control or telecontrol, for programming a control task in robotics due to the above two reasons.

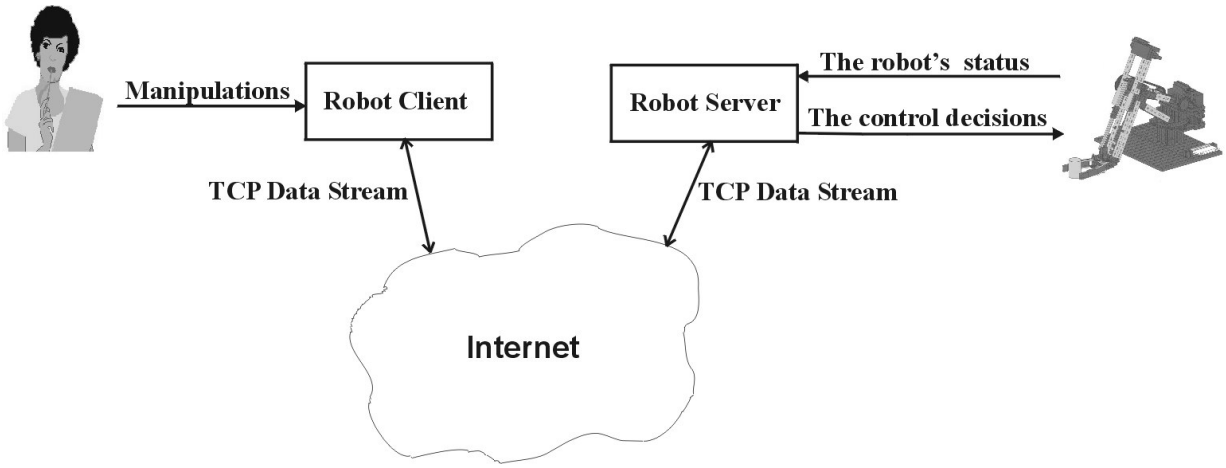


Fig. 31: The experiment environment

The robot client is used to run *WIMP+ user interfaces* while the robot server is used to fulfill the low level robot control activity. More concretely, the robot client and the robot server can be mounted either on the same computer (for local control) or two computers (for telecontrol) that can communicate with TCP/IP. For example, during the development process, the thesis author mounted the robot client on a SUN workstation and the robot server on a

[42] The robot server is implemented by Martin Friedmann and Wolfgang Hess, two students at the Darmstadt University of Technology, during their working at a student research project under the guidance of the thesis author.

personal computer. Both computers are connected with the Internet, as shown in Fig. 31. The communication between the robot client and the robot server depends on *the robot client and the robot server communication languages* (see Appendix C of this thesis).

6.4 LLDemo: a WIMP+ user interface

LLDemo is a *WIMP+ user interface* developed by the thesis author in the period of developing the *ACEE* architecture. It was implemented in VisualWorks Smalltalk. And now part of its design and code are used to form the components of *Hot-WIMP+*.

According to the abstract user interface model described in *SelectMe-ConveyMe-SettleMe*, a *WIMP+ user interface* prototype was designed by use of the HCI patterns, e.g., Step-by-Step Instruction, Status Display, Control Panel, Giving a Warning, Interaction Feedback, and Move in an Area [102] [107] [108]. Then according to *ACEE* and the interface prototype, an executable *WIMP+ user interface*, *LLDemo*, was created with VisualWorks Smalltalk.

In *LLDemo*, ***Acquisition***, which is used to capture a user's commands, inherits directly the design and code from the ***Controller*** class of VisualWorks Smalltalk. The ***Computation*** component, by use of the ***Model*** class of VisualWorks Smalltalk, is built upon the robot kinematical model as described in *SelectMe-ConveyMe-SettleMe*. By use of traditional WIMP user interface objects and creating the related *Non-WIMP user interface objects* according to the application requirements, ***Expression***, which is used to provide the system information and manipulation possibility to the user, was created. The ***Execution*** component is responsible for producing commands for leveraging the location and width of the robot gripper.

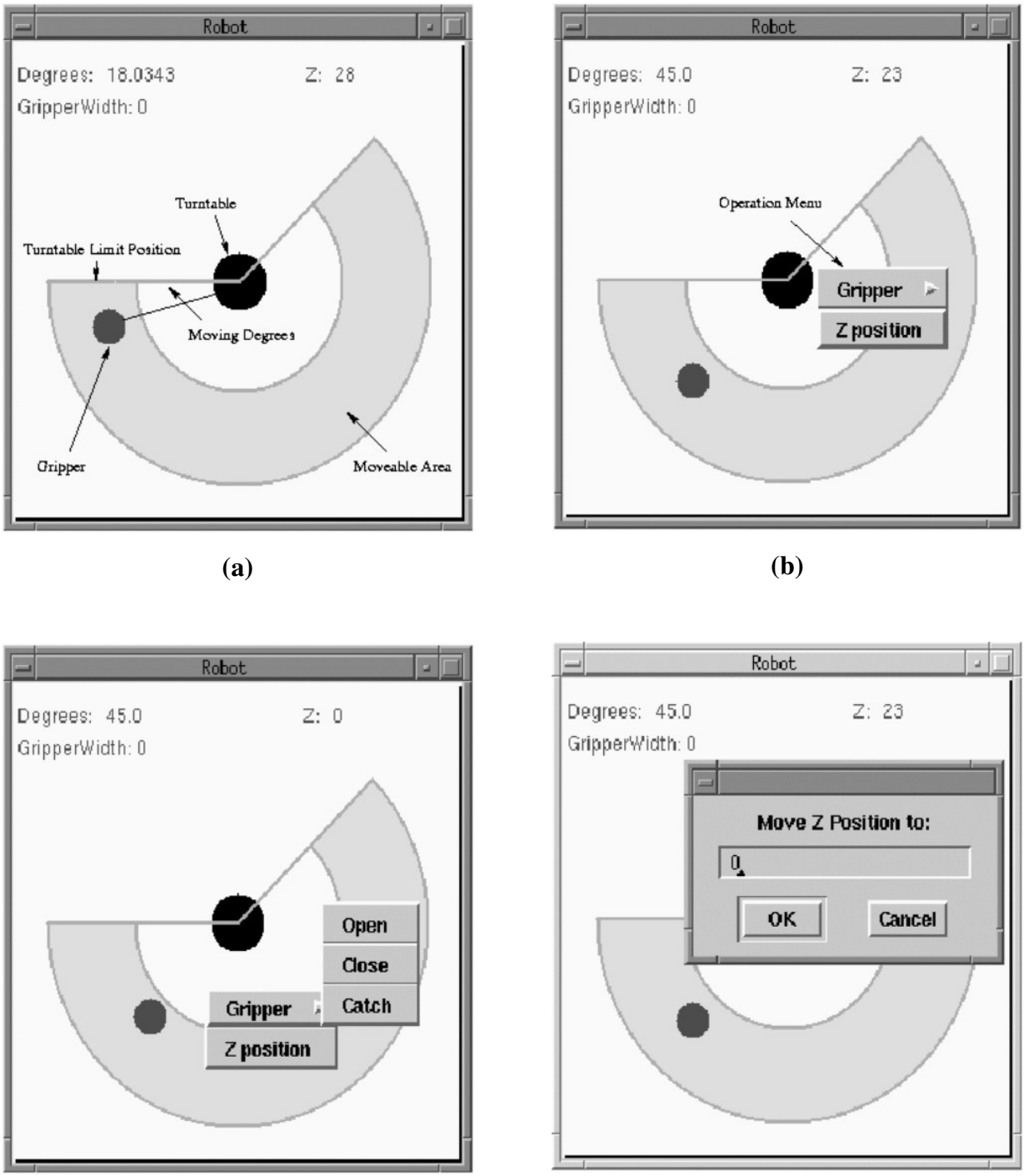


Fig. 32: LLDemo, a WIMP+ user interface for controlling the Fischertechnik model robot

The `Non-WIMP user interface objects` used in the *Expression* of `LLDemo`, as Fig. 32 (a) shown, are:

- Moveable Area: the shaded semicircle used to represent the movable area of the robot.
- The Gripper: the dark grey circle in Moveable Area represents the robot gripper. It is not allowed to leave Moveable Area.

Besides, the actual z coordinate is indicated by a number in the upper right corner; the turning angle and the opening width of the gripper are also given as numbers in the upper left corner.

LLDemo can support Visually Programming by Direct Manipulation technique. To control the robot gripper with it, a user just needs to manipulate The Gripper with the control button, i.e., the select button[43] of the mouse. When the user clicks and drags the mouse within Moveable Area, the robot arm will move accordingly. To control the width of the gripper and its vertical position, the Operation Menu, which can be activated by the operation button[44], can be used, as shown in Fig. 32 (b). With the Operation Menu activating, the user can further select to control the gripper or the z coordinate, as shown in Fig. 32 (c) and (d).

LLDemo hides the programming and hardware details. By use of the mouse and keyboard, a user can control the robot directly and interactively. But in *LLDemo*, the robot's current status is not reflected.

6.5 Hot-WebRobi: a WIMP+ user interface

Hot-WebRobi, another *WIMP+ user interface* for controlling the robot gripper as described in *SelectMe-ConveyMe-SettleMe*, was implemented by Erik Hansen during the time he worked at a student research project under the guidance of the thesis author. It is implemented as a Java applet which can run in a Web Browser, e.g., Netscape Communicator or Internet Explorer. It was developed according to the *ACEE* architecture. Its *Computation* and

---

[43] According to VisualWorks Smalltalk, the select button is the left button of a two-button or a three-button mouse [114].

[44] According to VisualWorks Smalltalk, if a two-button mouse is used, the right button is defined as the operation button; and if a three-button mouse is used, the middle button is the operation button [114].

Execution components are similar to that of `LLDemo`. Its *Acquisition* can communicate with the robot server and its *Expression* is designed and implemented as Fig. 33 shows.

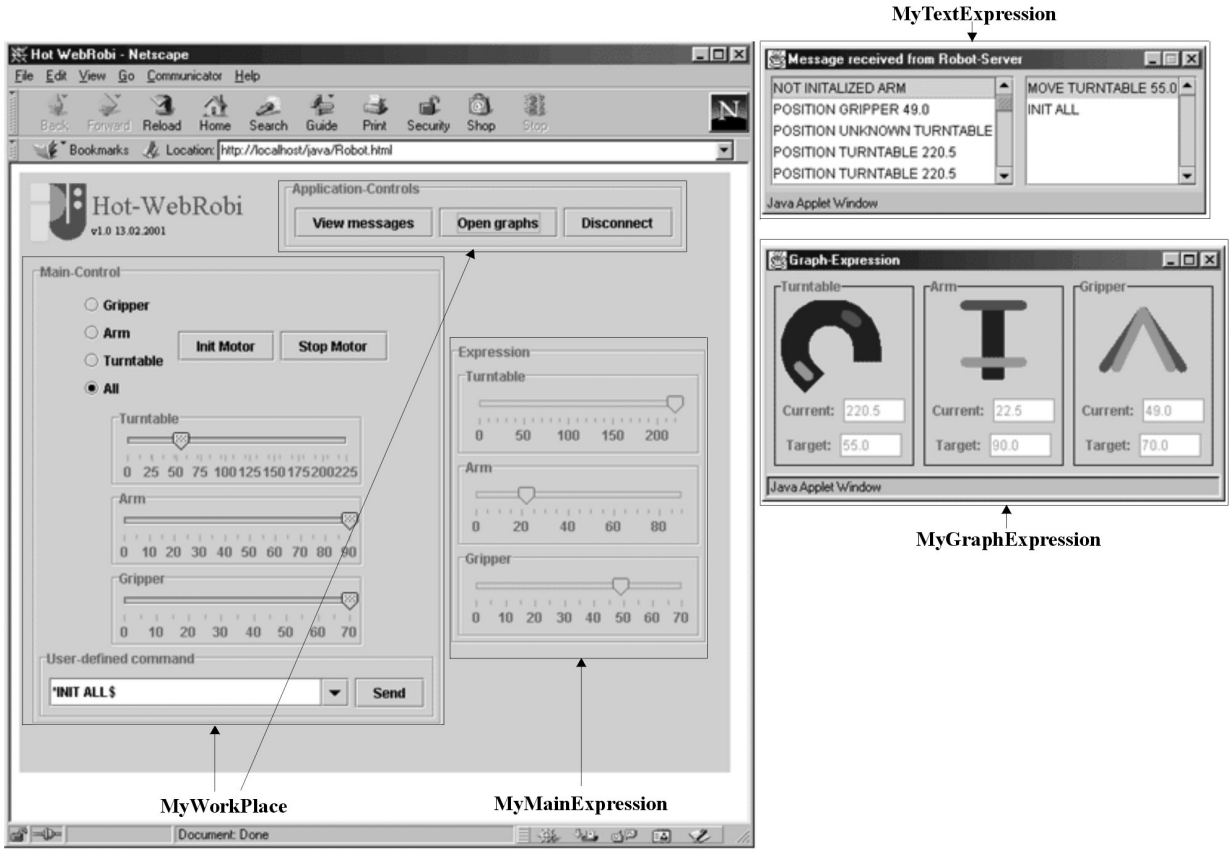


Fig. 33: Hot-WebRobi: a WIMP+ user interface running in a Web browser for controlling the Fischertechnik model robot [45]

The *Expression* is divided into two parts: MyWorkPlace and MyMainExpression. MyWorkPlace, which is further split into Main Control and Application Controls, is used to capture user's commands. MyMainExpression is used to express the robot's current status.

Within the Main Control area and the MyMainExpression area, there are two group of sliders, each consists of three sliders which are used to represent turntable: the arm's horizontal moving degrees, arm: the z coordinate, and gripper: the gripper's width, respectively. By moving the indicator of a slider within the Main Control area with the mouse, a user can control the robot remotely. And the executing situations of the robot at the remote side will be reflected in the corresponding sliders of MyMainExpression (the indicators of the corresponding sliders will move accordingly). A user can also directly use commands to control the robot via the User-

defined command input field (some commonly used commands are listed as radio buttons within the Main Control area).

Additionally, Application Controls can be used to activate MyTextExpression, MyGraphExpression or disconnect the robot client with its server. As shown in Fig. 33, MyTextExpression reflects every textual messages which reflect the robot's current status sent by the robot server. MyGraphExpression consists of three graphic representations which reflect the current and the target positions of the gripper.

The class diagram of *Hot-WebRobi* is shown in Fig. 34.

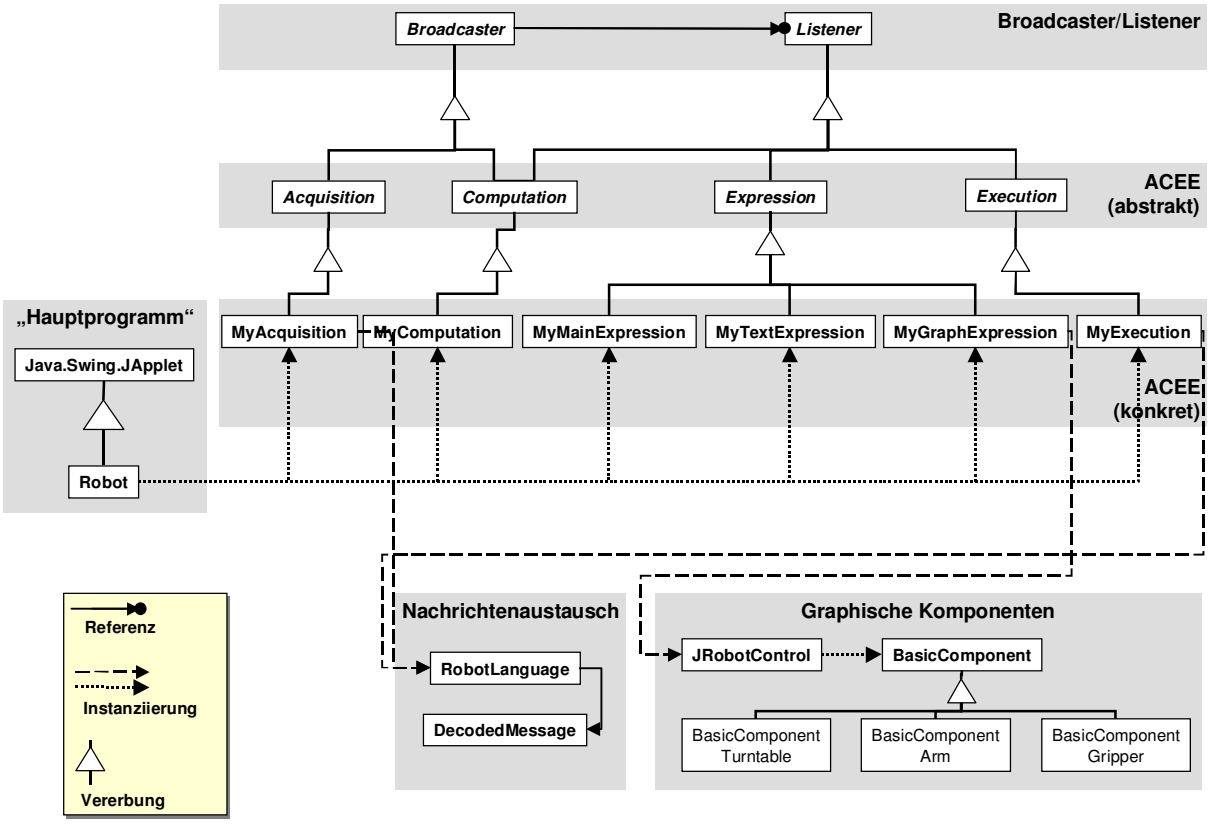


Fig. 34: The class diagram of Hot-WebRobi [45]

6.6 Hot-Demo: a WIMP+ user interface

Hot-Demo is a new version of *LLDemo*. In addition to all functions of *LLDemo*, it can reflect the robot's current status and provides a two-handed manipulation interaction technique for

HCI. This two-handed manipulation interaction technique was created according to the *Two-handed Manipulation* HCI pattern [133]. Before going to the details of *Hot-Demo*, the revised *Two-handed Manipulation* HCI pattern will be presented. It is reconstructed firmly according to the HCI pattern form proposed at CHI2000 (as described in section 3.1.2.4 of this thesis). Besides, since this thesis is not solely on research of HCI patterns, other related HCI patterns such as the smaller patterns for constructing this pattern will not be mentioned. Therefore, the References element of the HCI pattern form will be omitted here.

6.6.1 Two-handed Manipulation HCI Pattern

Name

*Two-handed Manipulation***

Example

In robotics, many systems equip their end users with two-handed manipulation interaction technique. For example, in MASCOT [4], a nuclear industry robot system, as shown in Fig. 35, end users can control the two robot hands to manipulate real world objects by mapping the actions of their own hands.

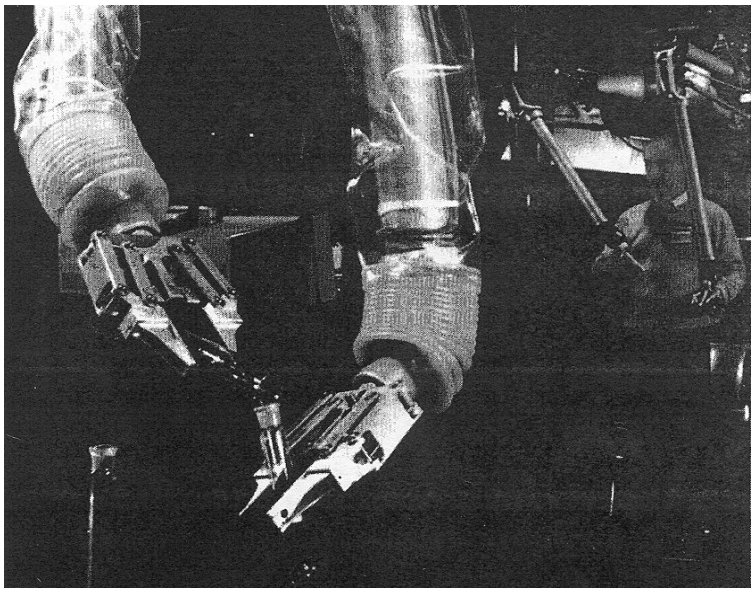


Fig. 35: A user of MASCOT is controlling a nuclear industry robot with two-handed manipulation technique [4]

Context

Traditional WIMP user interfaces have not appropriately exploited bimanual interaction of human beings. *WIMP+* and Non-WIMP user interfaces should utilize the two-handed manipulation interaction technique adequately in order to support their end users interacting with computers more efficiently.

Problem

Two-handed manipulation is an instinct of human beings. People could feel unbalance if they just use one hand in their every day life. In robotics, two handed-manipulation is also used in many electromechanical systems. In principle, two-handed manipulation should be more efficient for users interacting with computers than one-handed manipulation. However, in a user interface, if the subtasks for the two hands are not appropriately allocated, two-handed manipulation could be worse than one-handed. For example, it may increase the cognitive load of the users for coordinating the subtasks of the hands.

How could *WIMP+* and Non-WIMP user interfaces appropriately utilize the non-dominant hand of end users interacting with computers?

Evidence

Traditional WIMP user interfaces are one-handed oriented. For example, the user interface of Microsoft office, GSview, Acrobat Reader, MATLAB, etc. only utilizes the dominant hand of a user to interact with a computer. Although with these traditional WIMP user interfaces people use two hands in typing, it does not reflect the continuous nature of two-handed manipulation since during typing the hands are only perform discrete tasks sequentially. In robotics, two handed-manipulation is used in many electromechanical systems (e.g., [4] [97]). As the robot control is intervened by traditional WIMP user interfaces, the two handed-manipulation interaction technique in robotics has been at a great deal taken over by the one-handed mouse and keyboard interaction technique.

However, people are used to bimanual interaction. Many studies also show that two-handed manipulation interaction technique significantly outperforms the commonly used one-handed

manipulation technique of traditional WIMP user interfaces for fulfilling a compound task [7] [18] [38] [46] [47] [66] due to the following reasons:

- Two-handed manipulation is more natural to users since it is an instinct of human beings.
- Two-handed manipulation can improve the speed of a task fulfillment since it can provide more information simultaneously to a computer.

Therefore, *WIMP+* and Non-WIMP user interfaces should pick up two-handed manipulation interaction technique to support their end users fulfilling the compound tasks.

But engaging both hands might increase the cognitive load of the task in terms of carrying out cognitive processes such as monitoring, decision making, and task coordination [61].

Solution

Let the non-dominant hand participate in fulfilling a compound task in *WIMP+* and Non-WIMP user interfaces according to the research results in [44]:

- Allocate coarser action to the non-dominant hand while the dominant hand is used to fulfill finer action.
- Assign the non-dominant hand to set the frame of the reference for the action of the dominant hand.
- Let the dominant hand involve in interaction at first. The non-dominant hand follows the action of the dominant hand.

Sketch

Fig. 36 shows a picture of a two-handed manipulation interaction technique. The non-domain hand is allocated to do the coarser action: select a button of a two pushbutton switch. The domain hand is assigned to manipulate the mouse, a more dexterous action.

Fig. 36: A two-handed manipulation interaction technique

6.6.2 Equip Hot-Demo with Two-handed Manipulation

In addition to the HCI patterns used in *LLDemo*, *Hot-Demo* is designed with the *Two-handed Manipulation* HCI pattern.

According to the *Two-handed Manipulation* pattern, in design of *Hot-Demo*, the compound interaction task: moving the robot gripper then open or close it can be split into two subtasks: moving the robot gripper; open or close the gripper. And in terms of the solution of the pattern, the dominant hand is allocated to take the first subtask while the non-dominant hand is used to fulfill the second since the first needs higher skill and dexterity.

Fig. 37: Two-handed manipulation devices of Hot-Demo: the left one is a pushbutton switch for non-dominant hand while the right one is the traditional mouse for dominant hand.

Accordingly, *Hot-Demo* is equipped with two manipulation devices: the traditional mouse and a pushbutton switch, as shown in Fig. 37. The mouse is used for the dominant hand to fulfill the subtask moving the robot gripper, while the pushbutton switch which consists of two pushbuttons is allocated for the non-dominant hand to initiate the subtask open or close the gripper. Fig. 38 shows that the thesis author is testing the two-handed manipulation interaction technique of *Hot-Demo*.

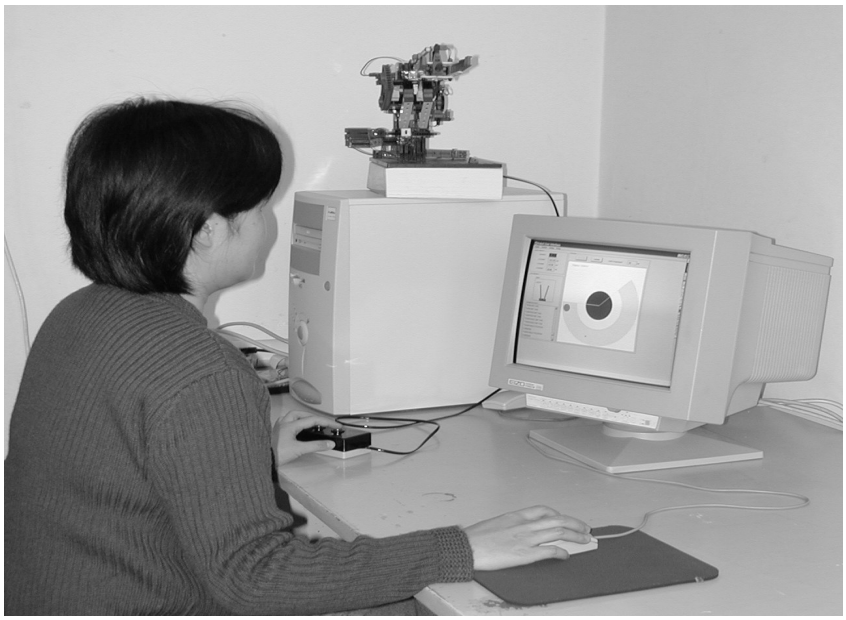


Fig. 38: The thesis author is testing the two-handed manipulation interaction technique of Hot-Demo.

6.6.3 Hot-Demo Outline

The *Expression* of *Hot-Demo* is shown in Fig. 39 (a)-(c). In addition to all functions of *LLDemo*, *Hot-Demo* provides:

- A more natural interaction technique. For example, two-handed manipulation interaction technique is exploited.

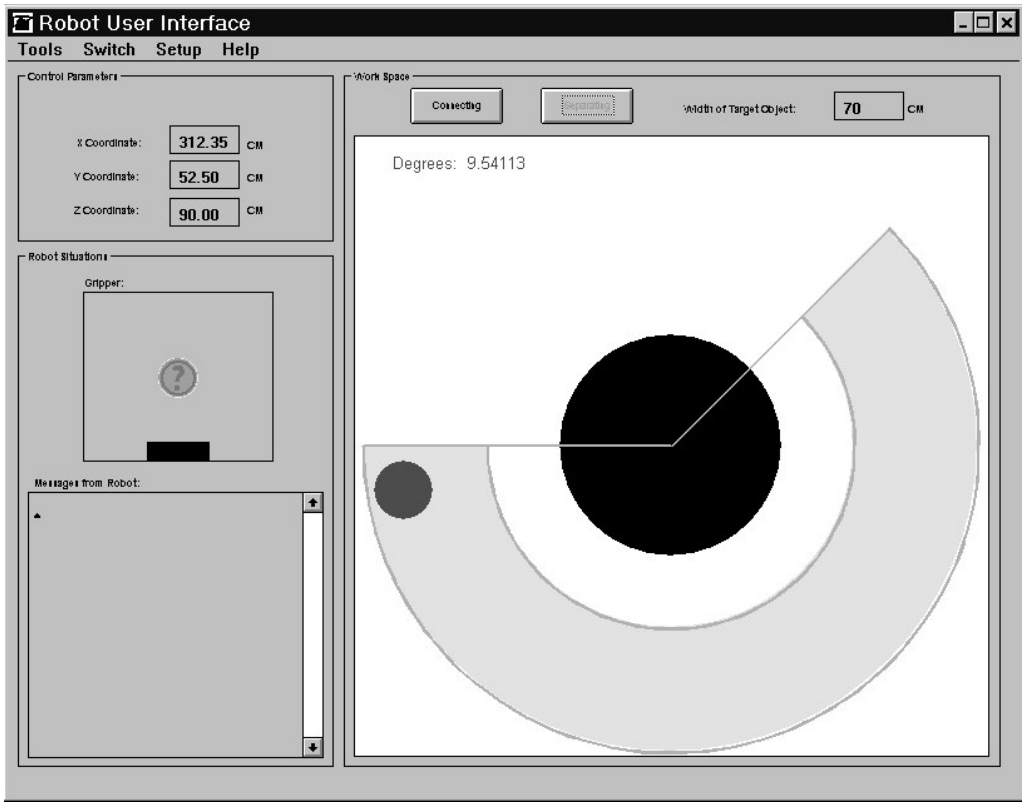
- Adequate and intuitive feedback. For example, not only the target position of the gripper (represented by a *Non-WIMP user interface object*: The Gripper, i.e., the dark grey circle) but also its moving track (represented by a *Non-WIMP user*

interface object: Moving Track, i.e., the grey circle) and other objects are clearly reflected, as Fig. 39 (a)-(c) show.

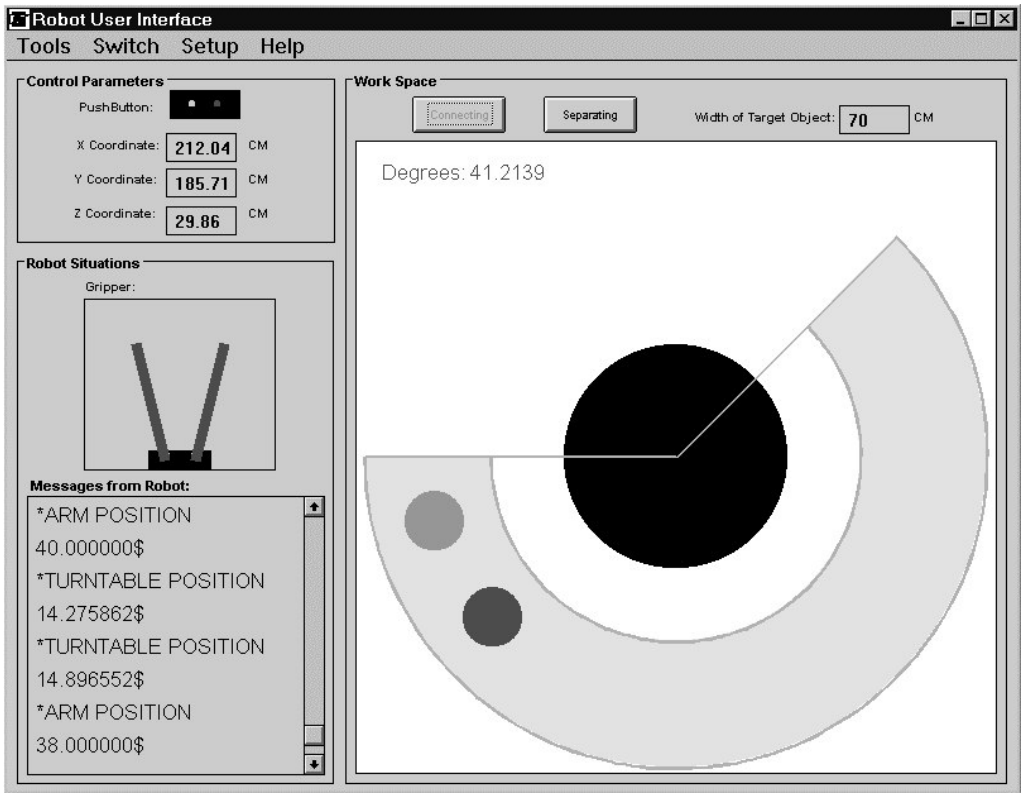
The *Expression* of Hot-Demo is divided into two parts: Work Space and Information Area, as Fig. 39 (a)-(c) shown.

Work Space, which is located on the right side of the *Expression*, is the corn of *Hot-Demo*. It is used to support its end user to program a task for controlling the robot. In addition to the two *Non-WIMP user interface objects* of *LLDemo*, Moveable Area (the shaded semicircle) and The Gripper (the dark grey circle), it includes another *Non-WIMP user interface object*, Moving Track (the grey circle), which reflects the current location of the robot gripper, as shown in Fig. 39 (b). When a user clicks and drags the mouse within Moveable Area, The Gripper will move to the position pointed by the mouse and *Hot-Demo* will produce commands for leveraging the robot arm. At the same time, if the robot client is connected with the robot server as described in section 6.3, the robot arm will be driven accordingly. And its executing status is sent back to *Hot-Demo* by the robot server and is reflected by Moving Track. Additionally, Work Space provides two control buttons: Connecting and Separating, for connecting or disconnecting the robot client with its robot server. The width of the target object can be specified with the Width of Target Object input field.

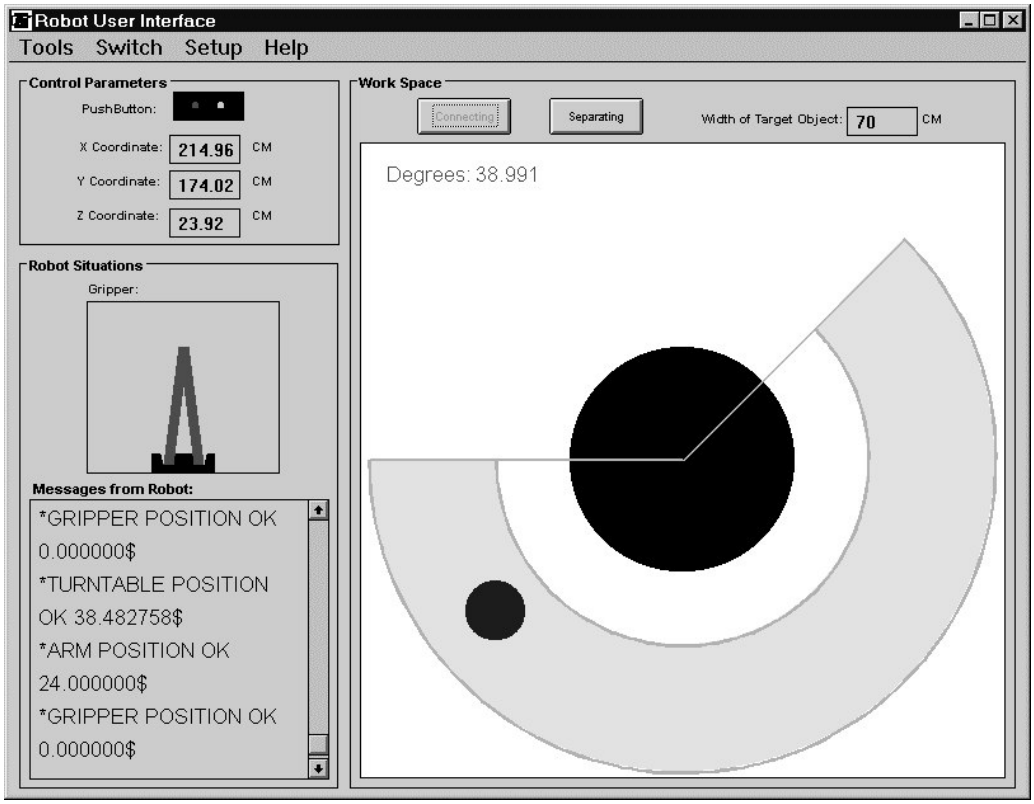
Information Area, which is located on the left side of the *Expression*, is used to display the system states. It consists of two groups: Control Parameters and Robot Situations. Control Parameters is used to reflect the 3-D coordinate of the target position of the gripper, as shown in Fig. 39 (a)-(c). If the pushbutton switch which is used for the non-dominant hand to fulfill manipulation as described in the above section is connected with the user interface, its operation status will be reflected by a *Non-WIMP user interface object*, Pushbutton, as shown in Fig. 39 (b)-(c). Robot Situations is used to reflect the robot's current status. It consists of Gripper Width, a *Non-WIMP user interface object*, and Messages from Robot. If robot client is connecting with the robot server, as shown in Fig. 39 (b)-(c), Gripper Width will reflect the gripper's opening/closing status and Messages from Robot will display the textual messages sent by the robot server to indicate more detailed executing information. If the robot client is disconnecting with the robot server, a question mark icon will appear in Gripper Width and Messages from Robot will display nothing, as Fig. 39 (a) shown.


Robot User Interface
Tools Switch Setup Help
Control Parameters
X Coordinate: 312.35 CM
Y Coordinate: 52.50 CM
Z Coordinate: 90.00 CM
Robot Situations
Gripper:
Messages from Robot:
Work Space
Connecting
Separating
Width of Target Object: 70 CM
Degrees: 9.54113

(a)


Robot User Interface
Tools Switch Setup Help
Control Parameters
PushButton:
X Coordinate: 212.04 CM
Y Coordinate: 185.71 CM
Z Coordinate: 29.86 CM
Robot Situations
Gripper:
Messages from Robot:
*ARM POSITION
40.000000$
*TURNTABLE POSITION
14.275862$
*TURNTABLE POSITION
14.896552$
*ARM POSITION
38.000000$
Work Space
Connecting
Separating
Width of Target Object: 70 CM
Degrees: 41.2139

(b)



(c)

Fig. 39: Hot-Demo, a WIMP+ user interface

6.6.4 A Brief Look at the Implementation

Hot-Demo was developed directly with *Hot-WIMP+*. Its *Acquisition* and *Execution* were directly created from the visual tools of *Hot-WIMP+*, *Acquisition Definer* and *Execution Definer*. Its *Computation* which integrates the kinematical model of the robot as described in *SelectMe-ConveyMe-SettleMe* was implemented as a subclass of *ACEE_Computation*. Its *Expression* which consists of several *Non-WIMP user interface objects*, were created with the help of *Expression Specification Tools* and well-defined *ACEE_Expression*. By linking the defined *Acquisition*, *Computation*, *Expression*, and *Execution* together with the related addDependent: methods which are encompassed in the connecting method as described in 5.5, *Hot-Demo* works!

As *LLDemo* and *Hot-WebRobi*, *Hot-Demo* can support Visually Programming by Direct Manipulation technique. It can be used to program a task to control the end-effector of a 3-D

robot as described in *SelectMe-ConveyMe-SettleMe* both locally (as shown in Fig. 38) or remotely (Fig. 40). Fig. 40 shows that a user is applying *Hot-Demo* to telecontrol the robot via the Internet.

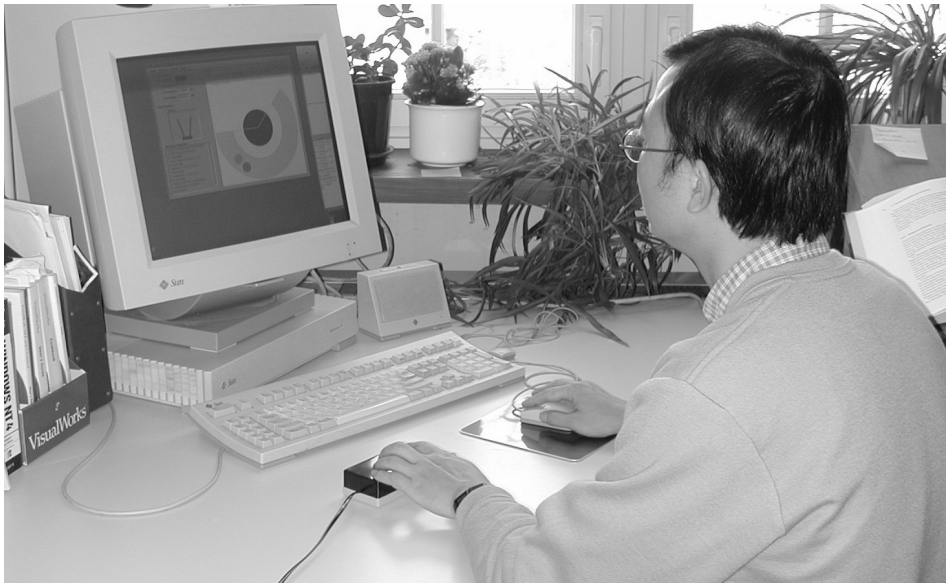


Fig. 40: With Hot-Demo running on a SUN workstation, a user is telecontrolling the robot via the Internet.

6.7 A Case Study

In order to investigate the effect and efficiency of the Visually Programming by Direct Manipulation technique that these *WIMP+ user interfaces* provide, Dr. Elke Siemon and the thesis author run a case study at the Darmstadt University of Technology [101]. This case study was on comparison of *LLDemo* with LLWin [68].

6.7.1 LLWin: a Visually Programming with Imperative Diagram Tool

LLWin is a Visually Programming with Imperative Diagram tool offered with the example robot as described in *SelectMe-ConveyMe-SettleMe*. It includes a set of graphical commands, as shown in Fig. 41, for end users to program a task for controlling the robot.

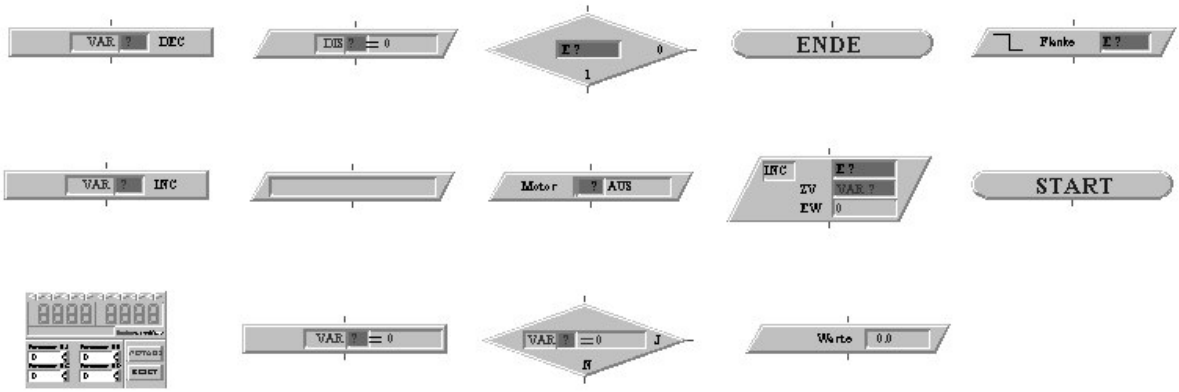


Fig. 41: The graphical commands of LLWin

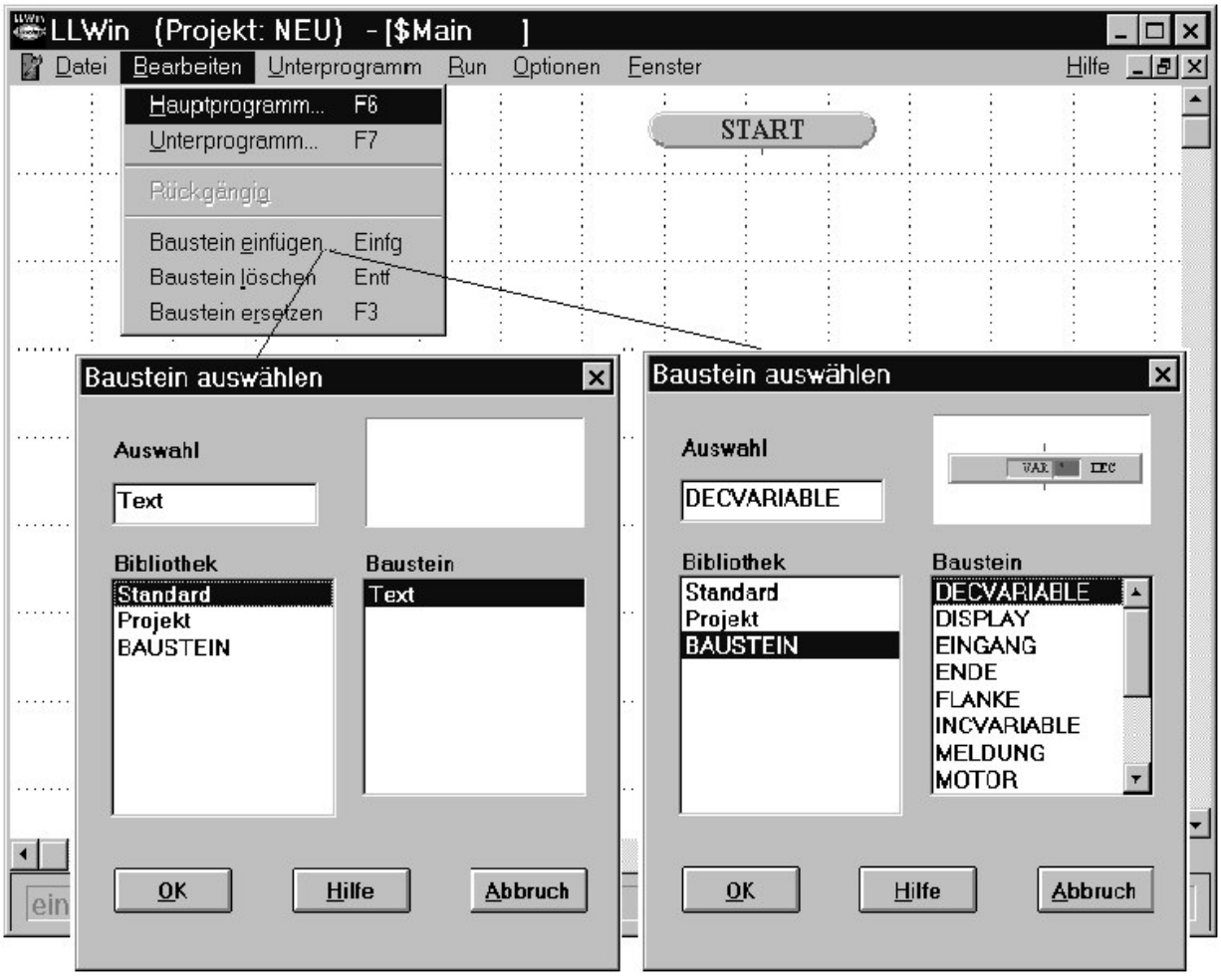


Fig. 42: The programming editor of LLWin

To create a program with LLWin, a user should choose the relevant graphical commands, put and edit them as an imperative diagram within the programming editor of LLWin, as shown in

Fig. 42. After the imperative diagram has been well established, a user can run the program directly with the run command which is integrated within the main menu of LLWin. Then the computer will follow the program indicated by the imperative diagram to control the robot. A detailed introduction to LLWin can be found in [68].

6.7.2 The Study

The Hypothesis

The hypothesis of this study was that *LLDemo* was more intuitive, easier to learn and to program. LLWin, on the other hand, was appropriate for learning the hardware details of the robot.

The Experiment Task

Let the subjects use *LLDemo* and LLWin to program respectively a task which consists of the following five steps:

Step 1: Move the arm of the robot 45° from the turntable limit position.

Step 2: Lower the arm 60 mm.

Step 3: Catch the target object about 10 mm in radius.

Step 4: Raise the arm and move it further 45 °.

Step 5: Open the gripper, i.e., release the target object onto its destination.

The Experiment Variables

- Simplicity of programming

  Can be measured by the learning time of each tool and the time for developing the solution to the task.

- Intuitivity

Depends on subjective answers to the questions concerning the features of the tools, like the visuality, needed mental skills, overview, and comfortability.

- Exactness of specification

  Can be measured by the exactness of the solutions to the task and if the tools could solve the task.

- Easiness of modifications and changes

  Can be measured by changing the task and then see how much time needed for both tools for programming the change.

- Consistency of terminology

  Depends on subjective answers to the questions concerning about tools for learning about the hardware details of the robot.

The Subjects

Eight students at the Computer Science Department of the Darmstadt University of Technology were part of this study. Three of them attended a class on visual support for programming. Two of them are female, six are male.

The Experiment

The experiment was curried out in the following steps:

Step 1: Introduce the hardware details of the robot, i.e., the motors and switches of the robot.

Step 2: Introduce LLWin with an example. Let the subjects experience the tool with a short example.

Step 3: Let the subjects program the task with LLWin. Let them modify the step 4 of the task as they finished programming the task.

Step 4: Introduce `LLDemo` with a short example.

Step 5: Let the subjects program a solution for the task with `LLDemo`.

Step 6: Let the subjects fill a questionnaire.

In two trails, the sequence of introducing LLWin and `LLDemo` were changed. But the change had no effect on the results.

The Solutions

To program a task for controlling the robot with LLWin, as above section described, the end user must use its graphical commands (as shown in Fig. 41) to edit an imperative diagram within its programming editor (as shown in Fig. 42) in terms of the needs of the task. Fig. 43 shows a LLWin solution to the experiment task.

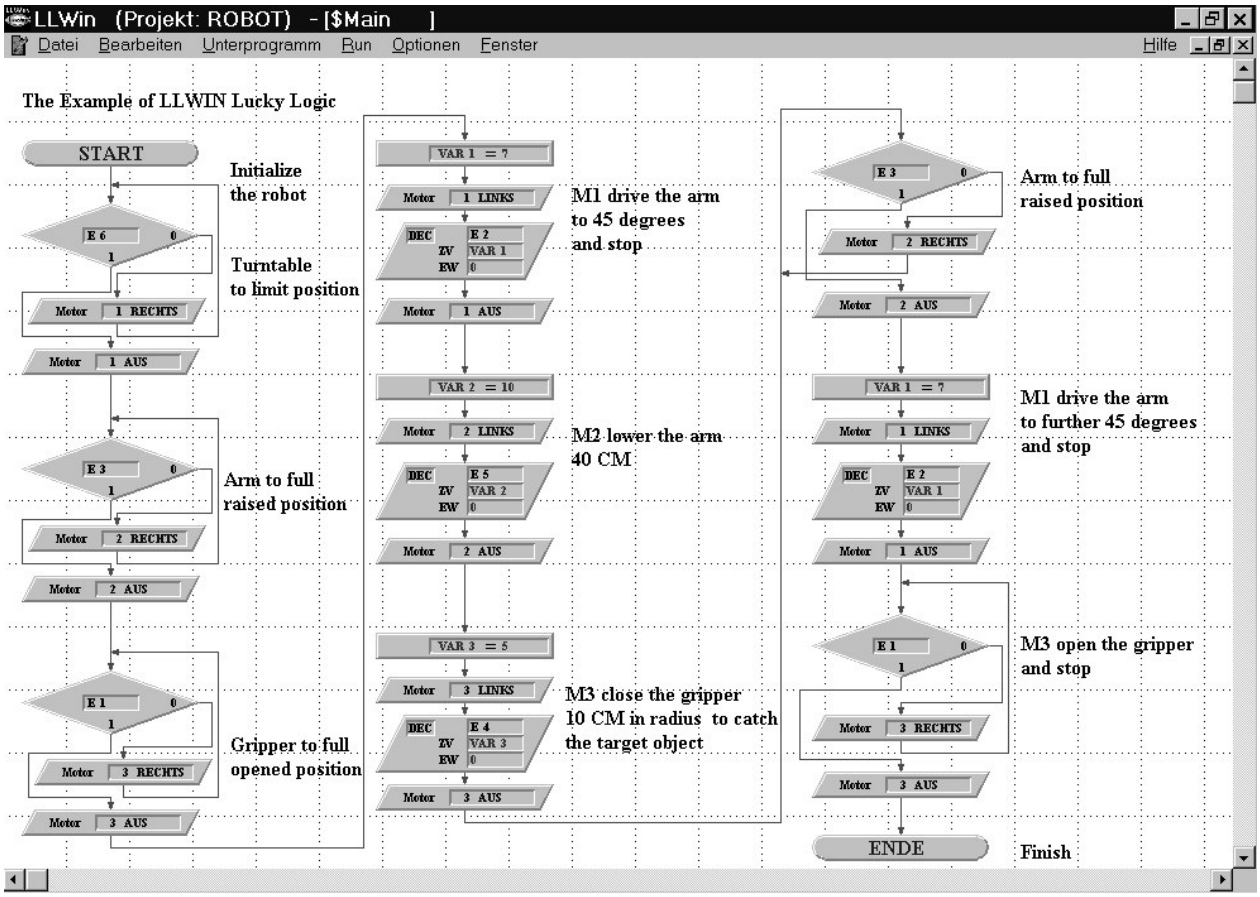


Fig. 43: A LLWin solution to the experiment task

To program a task for controlling the robot with *LLDemo*, as described in section 6.4, the end user needs to use the mouse to move The Gripper to the relevant position within Moveable Area and if necessary, with the help of its Operation Menu. In order to program the experiment task, the end user needs to follow these steps:

Step 1: Move the robot arm to 45° from the turntable limit position and lower it 60mm.

There are two methods with *LLDemo* to fulfil the subtask:

On The Gripper, press the control button. Then drag The Gripper with the mouse. As the mouse moves, the corresponding Degrees that the robot has turned and also the Z coordinate are displayed. When Degrees is 45.0, release the control button, then press the operation button. Operation Menu occurs as in Fig. 32 (b). Select Z position, then at the emerging dialog box as shown in Fig. 32 (d), type the intended value of the z coordinate, then click OK.

Alternatively, press the control button on The Gripper and drag it to the intended position, release the control button when the intended turning Degrees and Z coordinate are displayed in the interface.

Step 2: Control the gripper to catch the target object.

Press the operation button. As Operation Menu occurs, select Gripper, then select Catch, as shown in Fig. 32 (c).

Step 3: Raise the arm and move the robot another 45°, then open the gripper.

On The Gripper, press the control button and drag it. When Degrees is 90.0, release the control button. Actuate Operation Menu, type the intended Z position in, as shown in Fig. 32 (d). Then actuate Operation Menu again to control the gripper. Select Open, as shown in Fig. 32 (c).

The Results

Central to the experiment variables, the results of this case study are summarized as follows:

- Simplicity of programming

  From the time recorded during the experiment, as shown in Table 7, it is obvious that programming with `LLDemo` is not only faster to learn, but also easier to program since the development of the task takes less time.

Table 7: The time used to learn and develop the experiment task with LLDemo and LLWin

	Average	Best	Worst
Introduction Robot	17 min	21 min	10 min
Introduction LLWin	14 min	13 min	20 min
Task LLWin	59 min	51 min	83 min
Modification	60 sec	30 sec	94 sec
Introduction LLDemo	11,5 min	5 min	17 min
Task LLDemo	4 min	2 min	7 min

- Intuitivity

  The answers to question concerning the features of both tools confirm the interpretation of the results of time. The students mostly gave statements like "*LLDemo is more visual, since the robot can be programmed just by using the mouse. With LLWin one first has to develop an imperative diagram.*" Statements like these show that students see `LLDemo` as more direct, as no explicit programming language is needed, compared to LLWin. They think that mental skills for LLWin have to be much higher than that for `LLDemo`. For LLWin the code needs to be planned in advance. Moreover, the use of program variables in LLWin seems to be harder.

  Most students (6 of 7) prefer using `LLDemo` to program the robot, "*because they do not have to write code*." One student prefers LLWin because it allows programming in more detail. For intuitivity, `LLDemo` gets an average rate of 8.2 (on a scale from 0 to 10)

where LLWin gets an average rate of 5.1. Similarly, the comfortability is rated better for *LLDemo* than that of LLWin because of its more direct programming.

Besides, for programming the example task, both tools can provide a clear presentation and a good overview.

- Exactness of specification

  Generally, all students could solve the task with LLWin and *LLDemo*. Although in reality, LLWin allows control sequences to be specified more exact than *LLDemo*, the result of the study shows that some students have the feeling, that *LLDemo* is more exact. The reason for this result is probably due to the fact that LLWin does not hide all the hardware details. For example, to specify a turn of 45°, the degree has to be translated into turns of a pulse switch which leads to two errors: the translation has to be done by the students and is therefore error prone and there may be truncation and rounding errors. On the other hand, the direct manipulation of turn 45° with *LLDemo* gives the students a feeling of an exact specification and is much more task-oriented.

- Easiness of modification and change

  The experiment shows, that minor modifications, as changing a parameter (like the degree to which the robot turns) are very easy and fast in LLWin. Since in *LLDemo* modification costs at least a new manipulation step, this will be less comfortable. On the other hand, if a change deals with bigger parts of the program which makes the time equivalent to the development of a new program, probably *LLDemo* will be faster here.

- Consistency of terminology

  Different answers such as the rating of LLWin as a useful tool for users who want to learn about the hardware details of the robot was expected. Surprisingly, the students just looked at the skills needed to use the tools. They did not care the functionality of the robot hardware details provided by LLWin. They saw the tools just for programming.

From the experiment results, it is clear that to program a task for controlling the robot, the Visually Programming by Direct Manipulation technique that *LLDemo* provides outperforms

the Visually Programming by Imperative Diagram technique provided by LLWin. Also, *Hot-WebRobi*, and *Hot-Demo* should possess the same advantages of *LLDemo*.

6.8 Evaluation

6.8.1 Evaluation on LLDemo, Hot-WebRobi, and Hot-Demo

These *WIMP+ user interfaces*, *LLDemo*, *Hot-WebRobi*, and especially *Hot-Demo* were developed in order to achieve as many functions of Non-WIMP user interfaces (as described in section 2.1.3) as possible. They can present clearly "*Sites, Modes, and Trails*" [82] to their end users. They occupy the following characteristics compared to traditional WIMP user interfaces:

- *Supervisory control capability*

  Obviously, *LLDemo*, *Hot-WebRobi*, and *Hot-Demo* can observe and/or leverage a *controlled process*: the robot.

- *More powerful and natural languages for users interacting with computers*

  The case study in the section of 6.7 shows that *LLDemo* could bring better performance than LLWin for programming the same task for controlling the robot. Also, programming a task with *LLDemo* outperforms greatly than programming with traditional TeachIn Programming technique and Textual Programming technique. For the same reason, *Hot-WebRobi* and *Hot-Demo* provide their users with more powerful and natural languages to control the robot, too.

- *Richer interaction devices and techniques*

  In addition to the commonly used one-handed manipulation interaction technique, *Hot-Demo* provide its users with the two-handed manipulation interaction technique. It was developed in terms of the HCI pattern *Two-handed Manipulation*: an easy to use pushbutton switch was designed for the non-dominant hand to do coarser direct

demonstration, open or close the gripper; the finer interaction, move the gripper to a target position, is assigned to the dominant hand.

Additionally,

- *These WIMP+ user interfaces have explored a flexible, economic, and promising way for telerobotics.*

  Since TCP/IP is used as the communication protocol to link the robot client and the robot server together, these `WIMP+ user interfaces` can use the Internet or other TCP/IP based networks as the communication channel. The combination of the Internet technology and telerobotics should possess a very attractive prospect. The reason is, on the one hand, the Internet provides an open public shared communication channel to the people all over the world. And the applications which can run on it are still very limited. New applications have been strongly appealed to serve manifold needs of its world wide millions of users for years. On the other hand, telerobotics [97] which combines human telecontrol with the robot technology can relieve people from tedious and dangerous work. But at present, telerobotics is mostly limited to a handful specialists in very special fields, e.g., military, industry, aviation and space technology. One of the reasons is due to the expensive communication channel. The Internet provides a well-established world wide communication channel that links every computer system together, e.g., household robots, computers in offices or at public places, or mobile phones (via WAP technology). These Internet linked systems can extend the applications of telerobotics greatly by involving millions of the Internet users acting as specialists to interact with their controllable facilities far away from them. For example, users who are on vacations in Germany could control their house-hold air conditions in China via the Internet. Users could instruct their household robots to cook tea for them via WAP technology when they are on their home ways. `LLDemo`, `Hot-WebRobi`, `Hot-Demo`, together with other sample examples, e.g., Interactive Model Railroad [55], and New Operator [105], no doubt, could enhance the confident of further research and development work at the combination of the Internet and robotics.

According to the hypotheses described in the section of 2.1.3, no doubt, all the analyses above show that *LLDemo*, *Hot-WebRobi*, and *Hot-Demo*, are towards a kind of Non-WIMP user interfaces!

6.8.2 Evaluation on Hot-UCDP, ACEE, and Hot-WIMP+

The successful application of *Hot-UCDP*, *ACEE*, and *Hot-WIMP+* in the development of *LLDemo*, *Hot-WebRobi*, and *Hot-Demo* shows their suitability and availability to support the whole development process of user interfaces [132], especially *WIMP+ user interfaces*. Their merits and potential problems are evaluated in a subjective view of the thesis author as follows:

- *Hot-UCDP could be a promising framework for user-centered design methodology*

  Hot-UCDP provides a concrete framework for developing a take-oriented user interface. But up till now, there are very few application domain patterns available. To propagate the design pattern modeling methodology in application domain needs time. Furthermore, it is uncertain that if all domain experts will accept to use the same pattern form to describe their abstract user interface models related to diverse tasks. So it is unrealistic to hope that all application domain patterns could stick to one pattern form. Instead, different pattern forms can be used to capture application domain knowledge. Also, as described in section 3.1.2.4, HCI pattern research are confronted with lots of problems. Software patterns are still needed to be further researched in order to meet the manifold requirements.

- *ACEE can provide a relative robust, flexible, pluggable, and reusable architecture for creating WIMP+ user interfaces.*

  Although the *ACEE* pattern is quite mature, it needs to embrace more element patterns in order to extend its ability. For example, the Strategy pattern [36] could be embeded into *Computation* for switching among alternative algorithms.

- *Hot-WIMP+ can effectively support the implementation of WIMP+ user interfaces from diverse programming levels.*

As described in chapter 5, *Hot-WIMP+* provides the white box technique, the black box technique, and visual tools for implementing *WIMP+ user interfaces*. The higher programming layer (see Fig. 21) is used, the easier for reuse and programming. But at present, the visual tools of *Hot-WIMP+* cannot hide all the textual programming details. Further research and implementation are still required.

6.9 Other Applications

Hot-UCDP provides a concrete framework for support of the development process of user-centered design methodology. In principle, it could be used to guide all task-oriented user interfaces design.

In principle, *ACEE* and/or *Hot-WIMP+* can effectively support the *WIMP+ user interface* development. In addition to *LLDemo*, *Hot-WebRobi*, and *Hot-Demo*, in the Known Uses section of *ACEE* (see the chapter 4), there are two applications which have successfully used the *ACEE* pattern. One is Hot Rolling Mills, an application for process automation managed by Siemens AG. The other is SCUT-Voice, a telecommunication application developed by the thesis author and her colleagues and students at the South China University of Technology. Without doubt, these two industrial applications together with the academic projects *LLDemo*, *Hot-WebRobi*, and *Hot-Demo* have demonstrated the practical aspects of the *ACEE* architecture. And there will be many applications can be developed with the *ACEE* architecture or directly from *Hot-WIMP+*, too. For example, user interfaces in WAP and WWW applications, user interfaces in car driver assistant systems, user interfaces in email filter applications, user interfaces in truck logistics and maintenance, user interfaces in distributed multi-user database systems, user interfaces in automation rail systems, and user interfaces in aviation.

6.10 Summary

This chapter demonstrated the utility of *Hot-UCDP*, *ACEE*, and *Hot-WIMP+* through developing *WIMP+ user interface* practice. From extracting abstract user interface model to implementing executable user interfaces, corresponding application domain pattern, HCI patterns, and software patterns were used throughout.

The robust, pluggable, flexible, reusable, maintainable, and efficient aspects of *ACEE* architecture and *Hot-WIMP+* for implementing *WIMP+ user interfaces* were also illustrated. However, a good software architecture does not mean a good user interface. It can only support the implementation level. User interface design deals with half science and half art. The usability aspect of a user interface depends heavily on the cognitive model of its designers. To develop user interfaces with the same software architecture and the same application domain pattern, different designers could exploit different interaction techniques which could result in quite different user interfaces. This is implied by the diversity of *LLDemo*, *Hot-WebRobi*, and *Hot-Demo*. Therefore, application domain patterns, HCI patterns, and software patterns are equal important to user interface development.

According to *Hot-UCDP*, in this chapter, one application domain pattern *SelectMe-ConveyMe-SettleMe* and one HCI pattern *Two-handed Manipulation* were also researched, documented, and used.

In addition to unfolding each *WIMP+ user interface*, *LLDemo*, *Hot-WebRobi*, and *Hot-Demo*, which was developed with the design patterns and frameworks of this thesis, this chapter elaborated also on a case study in order to investigate the effect and efficiency of the task programming technique that these *WIMP+ user interfaces* provide. Evaluation on these *WIMP+ user interfaces*, *Hot-UCDP*, *ACEE*, and *Hot-WIMP+* were also given. Further application domains of these design patterns and frameworks were also mentioned.

The coming chapter will conclude this work and discuss future research.

7 Conclusions and Future Work

7.1 Conclusions

User interfaces are vita factors to contemporary and future applications. What are future user interfaces? What models and tools are effective and efficient to user interface, especially to future user interface development? This doctoral work has done deep and extensive research central to these questions. Based on the research results and the development experience in user interfaces as well as other parts of applications of the thesis author, in this work, corresponding models and tools, i.e., design patterns and frameworks, which have been demonstrated by a robot control application, have been developed. The major contributions of this work could be unfolded as follows:

- *Define WIMP$^+$ user interfaces based on investigation of the evolution of user interfaces*

  This work deduces the characteristics and functions of future user interfaces from extensive study on literatures and the research results on the evolution of user interfaces. It comes out that `supervisory control` should be one of the most important innovative functions of future user interfaces. Central to supervisory control, `WIMP+ user interfaces` are defined. That is, `WIMP+ user interfaces` are a kind of future user interfaces, which are partially built upon the WIMP interaction devices and techniques and can provide the function of observing and leveraging at least one `controlled process` under the supervision of their users. They deal with continuous and real-time response and feedback, ubiquitous computing, and manifold interaction devices and techniques which can bring high bandwidth and parallel input and output.

The similarity, diversity, and relationships among the traditional user interfaces, *WIMP⁺ user interfaces*, and future user interfaces have also been studied.

- *Pick up the attractive universal modeling methodology, design patterns, to structure interdisciplinary user interface models. Develop a framework: Hot-UCDP to support the development process of user interface engineering.*

  From the user interface engineering point of view, an investigation on the state-of-the-art high level models for user interface realization is conducted within this work. It comes out that user-centered design methodology is the most promising one that could bring higher usability user interfaces. But since it is a new model, at present, it is still short of concrete formal or semiformal rules to support the development process.

  User-centered design methodology calls for close cooperation among individuals of an interdisciplinary development team which consists of end users (application domain experts), HCI experts, software engineers, and other experts with different skills and expertise. Therefore, this work picks up the universal modeling methodologies, design patterns, to structure interdisciplinary user interface models. It is one of the pioneer works that uses design pattern methodology to structure end users' tasks. And for the rational reason, the root of design patterns, design patterns in software engineering, and design patterns in HCI are also overviewed.

  Based on the user interface and software development experience of the thesis author and research results on the literatures, it is clear that task-oriented user interface development approach could act as one of the candidates to support the development process of user-centered design methodology. Supplying the task-oriented approach with a users' task oriented interdisciplinary pattern system, this work proposes a framework, *Hot-UCDP*, to support the development process of user-centered design methodology. The content, hierarchy, and workflow of *Hot-UCDP* are clearly defined.

- *Discover and document a WIMP⁺ user interface software architectural pattern: ACEE*

  With *Hot-UCDP*, this work puts its most effort on elaborating on user interface software architectural patterns; and discovers that the state-of-the-art user interface

architectural patterns such as MVC and PAC do not cover the details of processing the spatial and time constraints of *WIMP⁺ user interfaces*.

Based on the intensive research on user interfaces and the software development experience in control technology (e.g., [127]) and telecommunication (e.g., [123]) of the thesis author, *Acquisition-Computation-Expression-Execution* (*ACEE*), a software architectural pattern was discovered during this work. The *ACEE* pattern was workshoped at PLoP1999 and won widely valuable comments and feedback from the pattern experts in both software and HCI communities.

ACEE pattern shows its great capability to guide *WIMP⁺ user interface* development due to the following reasons: it clearly divides a complex *WIMP⁺ user interface* into four components according to functionality; it defines a simple but efficient mechanism for coordinating the communications among this quadruple; it can effectively and efficiently support change, plug, or remove any component of this quadruple without influencing others; it makes the configuration of new input devices easier; it supports real-time and simultaneous interaction design; it supports of capturing both continuous signals and discrete events from *sensors* and manipulation devices according to their characteristics; it supports synchronous and asynchronous operations; and it supports distributed computing.

- *Develop an ACEE-based software framework prototype, Hot-WIMP⁺, for implementing WIMP⁺ user interfaces*

  According to the research results about tools for user interface implementation, it comes out that most traditional user interface tools are specific to help develop traditional WIMP user interfaces; tools for aiding the creation of *WIMP⁺ user interfaces* should be implemented. Since pattern-based software framework is regarded as one of the most promising tools in both user interface and software engineering nowadays, a pattern-based software framework prototype, *Hot-WIMP⁺*, was created by the thesis author herself during this work.

  Based on *ACEE*, *Hot-WIMP⁺*, which is built upon the application framework of VisualWorks Smalltalk, integrates the most present advantages of software framework.

It provides white box technique, black box technique, and visual tools for programming *WIMP⁺ user interfaces*. That is, with *Hot-WIMP⁺*, user interface developers can build their *WIMP⁺ user interfaces* from different programming layers: they can use either the relevant variables, methods, and programming interfaces provided in the components of *ACEE_Acquisition*, *ACEE_Computation*, *ACEE_Expression*, and *ACEE_Execution* or its visual tools, *Handler Manager*, *Connection Definer*, *Acquisition Definer*, *Expression Specification Tools*, and *Execution Definer*. Since *Hot-WIMP⁺* is built upon *ACEE*, it keeps all the characteristics of *ACEE*. The well-defined *ACEE* components and visual tools of *Hot-WIMP⁺* can greatly release user interface developers from the complexity of developing and maintaining *ACEE* components.

- *Investigate the practicability of Hot-UCDP, ACEE, and Hot-WIMP⁺ by applying them to create WIMP⁺ user interfaces for controlling a model robot*

  According to *Hot-UCDP*, to create a task-oriented user interface, domain experts should outline their requirements, their abstract user interface models, as application domain patterns. Accordingly, during this work, a robot application domain pattern, *SelectMe-ConveyMe-SettleMe*, was documented. It extracts and gives a solution to release the problem existing within the state-of-the-art techniques used to support end users programming a task to telecontrol the end-effector of a 3-D robot arm. It is the first abstract user interface model which is structured in the form of design patterns in robotics. This pattern can be used as a starting point to design user interface prototypes for controlling the end-effector of a 3-D robot arm with constraints.

  During the design phase of *WIMP⁺ user interface* prototypes, this work also researched bimanual interaction, discovered and documented it as a HCI pattern *Two-handed Manipulation*.

  Three executable *WIMP⁺ user interfaces*, *LLDemo*, *Hot-WebRobi*, and *Hot-Demo* have been successfully created with *ACEE* and/or *Hot-WIMP⁺*.

  The implementation practice shows the high availability of *Hot-UCDP*, *ACEE* and the simplicity of *Hot-WIMP⁺* in support of developing of *WIMP⁺ user interfaces*.

- *Explore using the Internet as the communication channel for telerobotics*

  This work (although it is not the first work) explores using the Internet as the communication cannel for telerobotics. It demonstrates again the attractive prospects of the combination of the Internet technology and telerobotics. It shows also the fact that the necessity of creating *WIMP⁺ user interfaces* for the Internet-based telerobotics since telerobotics calls for intuitive feedback from the remote site while transmission of large amount of real time video images via the Internet for telecontrol is unrealistic.

7.2 Future work

The design patterns and frameworks developed by this work can provide effective support on both the user interfaces, especially *WIMP⁺ user interfaces* development process and phases. Future work can be conducted in the following directions:

- *Establish substantial software architectural patterns, HCI Patterns, and application domain patterns. Categorize them appropriately*

  Hot-UCDP could be a promising framework for support of user-centered design development process. For a specific task, it calls for a relevant interdisciplinary pattern system. This requires large amount of adequate application domain patterns, HCI patterns, and software patterns available and easy to be found. Therefore, future work could go on the direction of developing application domain patterns, HCI patterns, and software patterns, categorizing and storing them appropriately.

- *Further implement Hot-WIMP⁺*

  Firstly, the visual tools of *Hot-WIMP⁺* should be further researched and implemented in order to support of automatic generation of *WIMP⁺ user interfaces*. Although at present *Hot-WIMP⁺* includes several visual tools, to create a *WIMP⁺ user interface* with it, manual textual programming is still required. Future work should go on this direction to further release user interface developers from textual

programming, e.g., integrating the new research results of visual specification techniques [69] [102] into *Hot-WIMP*$^+$.

Secondly, *Hot-WIMP*$^+$ should be equipped with enough handlers (device drivers) to react with all possible input and output devices; and enough communication protocols to establish connection with them. It should be no difficult to integrate handlers and communication protocols into *Hot-WIMP*$^+$ with its visual tools *Handler Manager* and *Connection Definer*.

Thirdly, use other programming languages to code *Hot-WIMP*$^+$ and integrate it into commercial user interface tools.

- *Further implement LLDemo, Hot-WebRobi, and Hot-Demo as Programming by Demonstration tools*

  At present, with *LLDemo*, *Hot-WebRobi*, and *Hot-Demo*, the end user must pay high attention during programming a task since every mouse movement will be interpreted as the real action of the robot. In a real control situation, this technique should be combined with other safety measures. For example, these user interfaces can be further implemented as Programming by Demonstration tools [28]. Let the end user do simulation with the user interfaces at first and save the simulation result at the same time. Provide intuitive user interfaces for modifying the simulation results. The end user can repeat the simulation and modification several times until a satisfied program is created, then uses the optimal program to control the robot.

- *Propagate the design patterns and frameworks presented in this work in future user interface development practice*

  The patterns and frameworks presented in this work can be used in many application domains. They should be propagated widely.

Design pattern methodology has shown its power in architecture. The thesis author also hopes that design patterns and pattern-based frameworks presented in this work can take their effect in future user interface development practice and help user interface developers, to some extent, achieve "*The Quality Without A Name*" [3].

Appendix A

Glossary

Abstract user interface models An abstract user interface model depicts an envisioned user interface of an application domain.

ACEE ACEE is the acronym of Acquisition-Computation-Expression-Execution. It refers to a software architectural pattern discovered by the thesis author for developing WIMP$^+$ User Interfaces.

Alexanderian patterns Alexanderian pattern refers to the design patterns written by architect Alexander and his colleagues in architecture, e.g., [2].

Application domain patterns An application domain pattern is a users' task model, which involves an abstract user interface model, depicted in a pattern form.

Applications An application is a software used to help solve a problem of an application domain. An interactive application is an application needs at least one user to interact with in order to run properly. In this thesis an application is used as the synonym of an interactive application.

Black box technique Black box technique refers to a software component reuse technique with which developers can reuse the components of a software framework without the requirement to know their implementation details by providing well-defined programming interfaces.

Components A component of a software refers to a set of well-defined collaborating classes or procedures which can provide a specific functionality.

Controlled process (task environment) In this thesis, it refers to any entity which possesses dynamic status that can be detected by *sensors* and leveraged by *effectors*.

Design pattern modeling methodology. It refers to using the design pattern forms to structure different models.

Design patterns Design patterns, in this thesis, refer to three things: the design pattern modeling methodology; models structured by the design pattern modeling methodology; and the medium-scale level software patterns, i.e., element patterns, which are used to support the software architectural patterns.

Effector In this thesis, an effector refers to either a hardware component or a programming interface of a software component which can be used to initiate an action of other objects.

End users End users refer to the people who exploit the applications to fulfil their tasks. End users are equal to users.

Frameworks A framework contains a set of components or subsystems and rules to relate them together in order to provide solutions to a family of applications.

HCI models A HCI model depicts a solution to a usability problem in an interactive system.

HCI patterns A HCI pattern conveys a well-proven HCI model in a pattern form.

Hot-UCDP It is a task-oriented interdisciplinary pattern-based framework which not only includes a set of interdisciplinary patterns but also defines the context and hierarchy of them for support of user-centered design development process.

Hot-WIMP^{+} It is an *ACEE*-based software framework prototype developed by the thesis author, which provides white box technique, black box technique, and visual tools for support of developing *WIMP*$^{+}$ *user interfaces*.

Interaction devices Interaction devices are the peripheral hardware devices of computer systems used for human and computer interaction.

Interaction tasks An interaction task is a meaningful step to perform an end users' task.

Interaction techniques An interaction technique is a way of using an interaction device to perform an interaction task.

Models A model is a representation of a design or the idea of a design.

Non-WIMP user interface objects Non-WIMP user interface objects refer to those user interface objects that go beyond the traditional WIMP metaphor.

Non-WIMP user interfaces It is an alias of the future generation user interfaces.

Sensors In this thesis, a sensor refers to either a hardware component or a programming interface of a software component which can be used to capture data generated by other objects.

Software architectural patterns A software architectural pattern captures the system structure of a family of applications. It locates on the highest level of software patterns. It consists of a set of element patterns (design patterns) and includes rules for organizing them together.

Software frameworks A software framework is a tool that provides reusable design and code for development of a family of user interfaces and other parts of applications.

Software patterns A software pattern is a model which uses a pattern form to describe objects, classes, and their communications that are customized to solve a general design problem in a particular context.

Toolkits A toolkit is a collection of widgets that are provided in the form of classes or procedures.

User-centered design methodology The state-of-the-art most attractive high level model for user interface planning and realization.

User interface builders A user interface builder is a user interface design system which can at least partially support arranging and editing interface layouts and contents graphically.

User interface design phase User interface design phase is a phase of user interface development process used to design user interface prototypes.

User interface design system A user interface design system is a tool for developing user interfaces. In addition to widgets, it encompasses other mechanisms for user interface development, too, e.g., the mechanism for creating and managing an application window.

User interface design User interface design refers to how user interface designers create an unexecutable user interface prototype according to an abstract user interface model.

User interface designers User interface designers refer to the people who design user interfaces for human and computer interaction.

User interface developers User interfaces developers refer to the people who design and implement user interfaces. They refer to both user interface designers and programmers.

User interface development phases User interface development phases include all the phases involved in a user interface development process, e.g., analysis phase, design phase, implementation phase, and test phase.

User interface development process User interface development process refers to the whole process that a user interface is planned, designed, implemented, tested, and delivered.

User interface development User interface development refers to both user interface design and implementation.

User interface implementation phase User interface implementation phase refers to the procedure that an executable user interface is created.

User interface implementation User interface implementation refers to that programmers create executable user interfaces according to user interface prototypes delivered by user interface design phase.

User interface prototypes A user interface prototype refers to a user interface delivered by the user interface design phase. It encompasses detailed specification of a user interface but it is still unexecutable.

User interface researchers User interface researchers refer to HCI experts who investigate techniques and tools for human and computer interaction.

User interfaces User interfaces are that portions of interactive computer systems that communicate with the users [58].

Users (See end users.)

Users' tasks Users' tasks refer to the work that end users have to carry out.

Visual tools A visual tool can help developers create user interface or other part of an application visually.

White box technique White box technique refers to a software component reuse technique that requires the developers to know the implementation details of the components in order to reuse them. For example, to create a subclass to overlap several methods of a class, the implementation details of the class must be well understood.

Widgets A kind of user interface objects.

WIMP user interface objects WIMP user interface objects refer to windows, WIMP user interface widgets, and other graphical elements used to create WIMP user interfaces.

WIMP user interfaces WIMP are the acronym of Windows, Icons, Menus, and Pointing devices: the mouse. WIMP user interfaces refer to the traditional graphical user interfaces.

WIMP widgets A WIMP widget is a traditional WIMP user interface object that is used to represent a data of an application and/or let its users exploit the traditional WIMP interaction techniques and devices interacting with this data.

WIMP+ user interface objects WIMP+ user interface objects include traditional WIMP user interface objects and Non-WIMP user interface objects.

WIMP+ user interfaces WIMP+ user interfaces are a kind of future user interfaces. In addition to traditional user interface interaction techniques and devices, they can observe and leverage at least one *controlled process* under the supervision of their users and may involve other innovative interaction techniques and devices for HCI.

Appendix B

Typographic Convention

There are several special fonts used in this thesis, as described in Table 8.

Table 8: Typographic conventions

Example	Description
ACEE	indicates an important term, a new term where it is defined, or a tool of this work.
Supervisory Control	indicates quotes and emphasized words or sentences.
Acquisition	indicates a class or a component name.
*GOTO 30 24 40$	indicates an instance, a message, a method, a command, a filename, a parameter, a label, a menu, a task, or an object.
Table	indicates the title of a table, a figure, or a UML diagram.

Appendix C

The Robot Client and the Robot Server Communication Languages

The robot client and the robot server communication languages presented below are used for the robot client and the robot server to exchange information. As described in section 6.3, the robot client is used to run three *WIMP+ user interfaces*, *LLDemo*, *Hot-WebRobi*, and *Hot-Demo*. The robot server is used to mange the low level robot control activity.

1. Message Format and Necessary Abbreviations

The message format and abbreviations below are used in the communication languages.

Message Format

*XXXXXXXXXXX$

- * indicates the beginning of a command or message.
- $ indicates the end of a command or message.

Abbreviations

- MOTOR

It refers to one of the three motors of the robot, i.e., Turntable, Arm, or Gripper[45]. In real communication, instead of using MOTOR, a concrete motor's name, i.e., TURNTABLE, ARM, or GRIPPER, must be given.

- NUMBER

  It refers to a decimal number. In real communication, a concrete decimal number, e.g., 123, 24, etc., must be given.

2. Language for the Robot Client Communicating with its Server

The language for the robot client communicating with its server includes a set of commands which are used to initialize the motors of the robot, move them, stop their movements, or get their positions.

Commands for Initializing Motors

Before a motor works, it should be initialized so that its movement can be measured correctly.

- *INIT ALL$

  This command is used to drive all motors to their initial positions.

- *INIT MOTOR$

  This command is used to initialize a given motor, e.g., *INIT TURNTABLE$ is used to drive the motor of the turntable, into its initial position.

Commands for Stopping the Movement of Motors

Following commands are used to stop the movement of the motors:

- *STOP ALL$

[45] Turntable, Arm, and Gripper here are used to refer the motors attached to them.

It is used to stop the movement of all motors.

- *STOP$

  The function of this command is identical to that of the above command.

- *STOP MOTOR$

  It is used to stop the movement of a given motor, e.g., *STOP TURNTABLE$ is used to stop the movement of the motor of the turntable.

Commands for Moving Motors

There are three commands to move the motors:

- *MOVE MOTOR NUMBER$

  It makes a given motor move to a defined position, e.g., *MOVE TURNTABLE 30$ will move the motor of the turntable for a while until it arrives 30 radian horizontally.

- *GOTO NUMBER1 NUMBER2 NUMBER3$

  It makes all three motors of the robot move to the defined positions. NUMBER1, NUMBER2, and NUMBER3 are used to specify the target position of Turntable, Arm, and Gripper, respectively.

Commands for Getting the Positions of Motors

The following two kinds of commands can be used by the client to get the current positions of the motors:

- *GET POSITION MOTOR$

  It informs the server to send the current position of a given motor back to the client.

- *GET POSITIOM ALL$

  It informs the server to send the current positions of all three motors back.

3. Language for the Robot Server Communicating with its Client

The goal of the language for the robot server communicating with its client is to inform the robot's status and respond the requests from its client. The language consists of a set of messages for describing the position of the robot and the status of each motor.

Messages for Describing the Position of the Robot

- *MOTOR POSITION NUMBER$

  It is sent every time when a motor has changed its position or when the client calls for its position.

- *MOTOR POSITION UNKNOWN$

  It will be sent to the client if the client calls for a motor's position and the position is not known (e.g. if the motor is not yet initialized).

- *MOTOR POSITION OK NUMBER$

  It is sent, if a motor has reached its target position (given in NUMBER) and has stopped there.

- *MOTOR TARGET BEYOND BOUNDARY$

  It is sent when the client requests a motor to move out of its moveable boundary (e.g., if the client requests the motor to drive the turntable move to 290° which is larger than its limited position, 225°).

- *MOTOR ON BOUNDARY$

  It is sent, if the motor is on the maximum (or minimum) position and the client is still calling for incremental (or descent) movement.

Messages for Describing the Status of Motors

- *MOTOR INIT OK$

It is sent, if the initialization action of a motor is completed.

- *MOTOR DEFECTIVE$

  It is sent when the server detects that a motor is defective or when the client calls for moving a defective motor.

- *MOTOR NOT INITIALIZED$

  It is sent if the client calls for moving a motor which has not been initialized.

Appendix D

List of the Figures

Appendix E

List of the Tables

References

[1] ACM Special Interest Group on Computer-Human Interaction Curriculum Development Group. *Curricula for Human-Computer Interaction* [online]. Available from: http://www.acm.org/sigchi/cdg/index.html. Accessed 2000 Aug. 25.

[2] Alexander C, Ishikawa S, Silverstein M, Jacobson M, Fiksdahl-King I, Angel S. *A Pattern Language*. New York: Oxford University Press; 1977.

[3] Alexander C. *The Timeless Way of Building*. New York: Oxford University Press; 1979.

[4] Bailey E. *MASCOT: Technical Description.* Consortium Telerobot. Genoa, Italy; 1992.

[5] Bayle E, Bellamy R, et al. *Putting It All Together: Towards a Pattern Language for Interaction Design*. SIGCHI Bulletin. New York: ACM; 1998; 30 (1): 17-23.

[6] Beck K, Cunningham W. *Using Pattern Languages for Object-Oriented Programs*. Workshop on the Specification and Design for Object-Oriented Programming. OOPSLA87. 1987.

[7] Bier EA, Stone MC, Pier K, Buxton W, DeRose TD. *Toolglass and Magic Lenses: The See-Through Interface*. Proceedings of SIGGRAPH'93. 73-80.

[8] Birrer A, Bischofberger W, Eggenschwiler T. *Widerverwendung durch Framework Technik vom Mythos zur Realität*. OBJECTSpektrum. 1995; 5:18-26.

[9] Boehm BW. *A Spiral Model of Software Development and Enhancement*. IEEE Computer. 1988, May: 61-72.

[10] Booch G, Rumbaugh J, Jacobson I. *The Unified Modeling Language User's Guide*. Addison Wesley Longman, Inc.; 1999.

[11] Booch G. *Object oriented design with applications*. California: The Benjamin/Cummings Publishing Company, Inc.; 1991

[12] Borchers JO. *A Pattern Approach to Interaction Design*. PHD thesis. Darmstadt University of Technology; 2000, May.

[13] Boshra M, Zhang H. *Localizing a Polyhedral Object in a Robot Hand by Integrating Visual and Tactile Data*. Pattern Recognition. 2000, March; Vol. 33, Issue 3: 483-501.

[14] Braess HH, Seiffert U. *Vieweg Handbuch Kraftfahrzeugtechnik*. Vieweg, 2001.

[15] Brown PL. *A Design Controversy Goes Cozy.com* [online]. Available from: http://www.nytimes.com/2000/11/23/living/23Alex.html. Accessed 2000 Nov. 30.

[16] Brunner B, Landzettel K, Schreiber G, Steinmetz BM, Hirzinger G. *A Universal Task-level Ground Control and Programming System for Space Robot Applications*. i-SAIRAS 5th International Symposium on Artificial Intelligence, Robotics and Automation in Space. ESTEC, Noordwijk, The Netherlands; 1999, June 1-3.

[17] Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M. *Pattern-Oriented Software Architecture: A System of Pattern*. John Wiley & Sons; 1997.

[18] Buxton W, Myers BA. *A study in two-handed input*. CHI1986 Conference Proceedings: Conference on Human Factors in Computing Systems. 321-326.

[19] *Conference about Patterns* [online]. Patterns Home Page. Available from: http://hillside.net/patterns/conferences/. Accessed 1998 Apr. 22.

[20] Coplien JO, Schmidt DC. *Pattern Languages of Program Design*. Addison-Wesley Publishing Company; 1995.

[21] Coplien JO. *Advanced C++ Programming Styles and Idioms*. Addison-Wesley Publishing Company; 1992.

[22] Coplien JO. *Software Patterns*. SIGS Management Briefings. NewYork: SIGS Books& Multimedia; 1996.

[23] Copper JW. *The Design Patterns Java Companion* [online]. Available from: http://www.patterndepot.com/put/8/JavaPatterns.htm. Accessed 2001 Aug. 17.

[24] Coutaz J. *PAC, an Object Oriented Model for Dialog Design*. In H.-J. Bullinger and B. Shackel (Eds.), Human-Computer Interaction: INTERACT'87. Elsevier Science Publishers B.V.; 1987: 431-436.

[25] Coutaz J. *PAC-Based Software Architecture Modeling for Interactive Systems*. Softwaretechnik'96. Koblenz, Germany; 4-11.

[26] Cunningham W. *The Vision of the Pattern Languages of Programs*. The Workshop on the Specification and Design for Object-Oriented Programming. OOPSLA1988. Ann Arbor, Michigan, USA; 1988; Oct. 16-19.

[27] Curtis B, Hefley B. *A WIMP No More: The Maturing of User Interface Engineering*. Interactions. 1994, Jan.: 22-34.

[28] Cypher A. *Watch What I do, Programming by Demonstration*. ed. Cambridge, Ma., USA: MIT press; 1993.

[29] Dam A. *Post-WIMP User Interface*. Communication of the ACM. 1997, Feb.; Vol.40, No.2: 63-67.

[30] Erickson T. *Lingua Franca for Design: Sacred Places and Pattern Language*. Available from: http://www.pliant.org/personal/Tom_Erickson/index.html. Accessed 2000 Sept. 16.

[31] *Experimentierbuch: Profi COMPUTING*. Fischerwerke Artur Fischer GmbH & Co.

[32] Fichman RG, Kemerer RG. *Object Technology and Reuse: Lessons from Early Adopters*. IEEE Computer. 1997, October; Vol.30, No.10: 47-59.

[33] Fincher S, Windsor P. *Why patterns are not enough: some suggestions concerning an organising principle for patterns of UI design*. The Workshop of Pattern Languages for Interaction Design: Building Momentum. CHI2000. The Hague, The Netherlands; 2000, Apr. 2-3.

[34] Fisher AS. *CASE: using software development tools*. 2nd ed. John Wiley & Sons, Inc.; 1991

[35] Foley JD, Dam A, Feiner S, Hughes J. *Computer Graphics Principles and Practice*. 2nd ed. Addition-Wesley Publishing Company; 1990.

[36] Gamma E, Helm R, Johnson R, Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company; 1995.

[37] Gentner D, Nielsen J. *The Anti-Mac Interface*. Communication of the ACM. 1996, Aug.; Vol.39, No.8: 70-82.

[38] Goble J, Hinckley K, Snell J, Pausch R, Kassell N. *Two-handed spatial interface tools for neurosurgical planning*. IEEE Computer. 1995, July; Vol.28, No.7: 20-26.

[39] Granlund Å, Lafrenière D. *PSA: A Pattern-Supported Approach to the User Interface Design Process* [online]. Available from: http://www.gespro.com/lafrenid/PSA.pdf. Accessed 2000 Aug. 12.

[40] Green M, Jacob RJK, et al. *Software Architecture and Metaphors for Non-WIMP User Interfaces*. SIGGRAPH'90 Workshop Report.

[41] Green M. *A Survey of Three Dialogue Models*. ACM Transactions on Graphics. 1986, July; Vol.5, No.3: 245-275.

[42] Griffiths R, Borchers J, Stork A. *Pattern Languages for Interaction Design: Building Momentum*. CHI2000 Extended Abstracts. New York: ACM; 2000, Apr.; 363.

[43] Griffiths R. *The Workshop of Usability Pattern Language: Creating a Community*. INTERACT'99. Edinburgh, Scotland; 1999, Aug. 30-31. Available from: http://www.it.bton.ac.uk/staff/rng/UPLworkshop99/index.html#Before. Accessed 1999 Dec. 02.

[44] Guiard Y. *Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a model*. Journal of Motor Behavior. 1987; Vol.19, No.4: 486-517.

[45] Hansen E. *Hot-WebRobi-Applet: Entwicklung einer Robotersteuerung über das Internet*. Technical Report. Darmstadt University of Technology; 2001, Feb. 22.

[46] Hinckley K, Czerwinski M, Sinclair M. *Interaction and Modeling Techniques for Desktop Two-Handed Input*. Proceedings of the ACM Symposium on User Interface Software and Technology. UIST1998. 49-57.

[47] Hinckley K, Pausch R, Proffitt D, Kassell F. *Two-Handed Virtual Manipulation*. ACM Transaction on Computer-Human Interaction; 1998, Sept.; Vol.5, No.3: 260-302.

[48] *Historic Photos - Doug Engelbart* [online]. Available from: http://www.bootstrap.org/history/photos/. Accessed 2000 Aug. 20.

[49] Hoffmann HJ, Handl D. *Document exchange as a basis for business-to-business co-operation*. In J.-Y. Roger et al. (ed.), Business and Work in the Information Society. Amsterdam: IOS Press; 1999: 325-331.

[50] Hoffmann HJ. *Design models and object-oriented framework for users interfaces in computer-supported automation and control technology*. Research Proposal. Darmstadt University of Technology; 1998.

[51] Hoffmann HJ. *How to improve overall system quality by a sensible design of man-machine interaction: view of a software engineer*. In P. Kafka and J. Wolf Safety (ed.), ESREL'93. Amsterdam: Elsevier Science Publishers B.V.; 1993: 939-947.

[52] Hoffmann HJ. *Software Engineering in User Interface Design with Guidelines: from Traditional Application to the Web Sphere*. In J. Vanderdonckt, C. Farenc (ed.), Tools for Working with Guidelines. London, Springer-Verlag; 2001: 263-271.

[53] Hopkins T, Horan B. *Smalltalk: an introduction to application development using VisualWorks*. Prentice Hall; 1995.

[54] *IBM SanFrancisco* [online]. Available from: http://www.ibm.com/software/ad/sanfrancisco/. Accessed 2000 Feb. 01.

[55] *Interactive Model Railroad* [online]. Available from: http://rr-vs.informatik.uni-ulm.de/rr/. Accessed 2001 June 13.

[56] Jacob RJK, Deligiannidis L, Morrison S. *A Software Model and Specification Language for Non-WIMP User Interfaces*. ACM Transactions on Computer-Human Interaction. 1999, Mar.; Vol.6, No.1: 1-46.

[57] Jacob RJK. *A Specification Language for Direct-Manipulation User Interfaces*. ACM Transactions on Graphics. 1986, Oct.; Vol.5, No.4: 283-317.

[58] Jacob RJK. *User Interfaces*. In D. Hemmendinger, A. Ralston, and E. Reilly (4th ed.), Encyclopedia of Computer Science. Macmillan Reference Ltd.; 2000.

[59] *Jakob Nielsen Interview* [online]. Internet.com; 1999, Dec. Available from: http://webreference.com/new/nielsen.html. Accessed 2000 June 20.

[60] Johnson RE, Foote B. *Designing Reusable Classes*. Journal of Object-Oriented Programming. 1988, June/July.

[61] Kabbash P, Buxton W, Sellen A. *Two-Handed Input in a Compound Task*. CHI1994 Conference Proceedings: Conference on Human Factors in Computing Systems. 1994, Apr. 24-28: 417-423.

[62] Kan SH. *Metrics and Models in Software Quality Engineering*. Massachusetts: Addison Wesley Longman, Inc.; 1995

[63] Koh YH. *Mutual Influence of Cultural and Environmental Changes and Information Technologies-Experiences in the People's Republic of China*. Proceedings of World Engineers' Convention [CD-ROM]. Expo2000. Germany, Hanover; 2000, Jun. 19-21.

[64] Krasner GE, Pope ST. *A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80*. JOOP. 1988, Mar.

[65] Larson JA. *Interactive Software: Tools for Building Interactive User Interfaces* . New Jersey: Yourdon Press Computing Series; 1992.

[66] Leganchuk A, Zhai S, Buxton W. *Manual and Cognitive Benefits of Two–Handed Input: An Experimental Study*. ACM Transaction on Computer-Human Interaction. 1998, Dec.; Vol.5, No.4: 326-359.

[67] Leveson NG, Turner GS. *An Investigation of the Thereac-25 Accidents*. IEEE Computer. 1993, July: 19-41.

[68] *LLWin - Lucky Logic für Windows* . Fischerwerke Artur Fischer GmbH & Co. KG: Handbuch fischertechnik. 1996.

[69] Martin Ludger. *Visualisierung von Komponenten für Benutzungsoberflächen*. Diplomarbeit. Darmstadt University of Technology; 2000, June.

[70] Martin R, Riehle D, Buschmann F. *Pattern Languages of Program Design 3*. Addison-Wesley Publishing Company; 1997.

[71] Meyer B. *Object-oriented Software Construction*. Prentice Hall; 1988.

[72] *Mobile Phone R380 Design Guidelines for WAP services* [online]. Ericsson Mobile Communication AB; 2000. Available from: http://www.ericsson.com/WAP/products/R380s_design_guidelines_B.pdf. Accessed 2000 Oct. 06.

[73] Myers BA, Hudson SE, Pausch R. *Past, Present, and Future of User Interface Software Tools*. ACM Transaction on Computer-Human Interaction. 2000, March; Vol.7, No.1: 3-28.

[74] Myers BA, Rosson MB. *Survey on User Interface Programming*. CHI1992 Conference Proceedings: Conference on Human Factors in Computing Systems. Monterey, CA; 1992, May: 195-202.

[75] Myers BA. *A Brief History of Human Computer Interaction Technology*. ACM Transaction on Computer-Human Interaction. 1998, Mar.; Vol.5, No.2: 44-54.

[76] Myers BA. *Challenges of HCI Design and Implementation*. ACM Transaction on Computer-Human Interaction. 1994; Jan.: 73-83.

[77] Myers BA. *UIMSs, Toolkits, Interface Builders* [online]. Available from: http://www.cs.cmu.edu/afs/cs.cmu.edu/user/bam/www/toolnames.html. Accessed 2000 Aug. 31.

[78] Myers BA. *User Interface Software Tools* [online]. Available from: http://www.cs.cmu.edu/afs/cs.cmu.edu/user/bam/www/toolnames.html. Accessed 2000 Aug. 31.

[79] Newburger EC. *Computer Use in the United States*. U.S Census Bureau: Current Population Survey (CPS) Reports [online]. 1997, Oct. Available from: http://www.census.gov/population/www/socdemo/computer.html. Accessed 2000 June 20.

[80] Nielsen J. *Non-Command User Interfaces*. Communications of the ACM. 1993, Apr.; Vol.36, No.4: 82-99.

[81] Nielsen J. *Usability Engineering*. Boston, MA: AP Professional; 1994.

[82] Nievergelt J, Weydert J. *Sites, Modes, And Trails: Telling The User Of An Interactive System Where He Is; What He Can Do; And How To Get Places*. Eidgenössische Technische Hochschule Zürich, Institut für Informatik; 1979, Jan.

[83] Norman DA. *The Invisible Computer*. Cambridge, MA: MIT Press; 1998, Oct.

[84] *Overview of ACE* [online]. Available from: http://www.cs.wustl.edu/~schmidt/ACE-overview.html. Accessed 1999 Dec. 08.

[85] *Patterns in Human Computer Interaction*. BCS HCI Group /IFIP Working Group 13.2 Workshop on HCI Patterns. 2000, Nov. 16-17. Available from: http://www.co.umist.ac.uk/hci_design/HCIpatswksp.htm. Accessed 2000 Nov.

[86] Pfaff GE. *User Interface Management Systems*. In Proceedings of Workshop on User Interface Management Systems. Springer-Verlag; 1985.

[87] Port Number. *Protocol Numbers and Assignment Services*. IANA. Available from: http://www.iana.org/assignments/port-numbers. Accessed 1999 Oct.

[88] Raines P. *Tcl/Tk Pocket Reference*. O'Reilly Associates; 1998.

[89] Raskin J. *Looking for a Human Interface: Will Computers Ever Become Easy to Use?* Communication of the ACM. 1997, Feb.; Vol.40, No.2: 98-101.

[90] Roberts D, Johnson RE. *Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks*. Available from: http://st-www.cs.uiuc.deu/~droberts/evolve.htn. Accessed 2000 Aug. 20.

[91] Sayers C. *Remote Control Robotics*. Springer-Verlag; 1998.

[92] Schmidt DC, Stal M, Rohnert H, Buschmann F. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. Wiley&Sons; 2000.

[93] Schmidt DC. *Pattern Languages of Program Design 3*. Addison-Wesley Publishing Company; 1998.

[94] Schwanninger C. *About Known uses of ACEE*. E-mail discussion during the shepherd process of ACEE; 1999, July.

[95] Shannon CE. *Communication in the Presence of Noise*. IRE Proceedings; 1949, Jan.; Vol.37: 10-21.

[96] Sheridan TB. *Human-Computer Interaction: Problems and Prospects*. Proceedings of World Engineers' Convention [CD-ROM]. Expo2000. Germany, Hanover; 2000, Jun. 19-21.

[97] Sheridan TB. *Telerobotics, Automation, and Human Supervisory Control*. The MIT Press; 1992.

[98] Shneiderman B. *Designing the User Interface*. 3rd ed. Addison-Wesley Publishing Company; 1998.

[99] Shneiderman B. *Direct Manipulation: A Step Beyond Programming Languages*. IEEE Computer. 1983, Aug.; Vol.16, No.8: 57-69.

[100] Sibert LE, Jacob RJK. *Evaluation of Eye Gaze Interaction*. CHI2000 Conference Proceedings: Conference on Human Factors in Computing Systems. Addison-Wesley Publishing Company; 2000, Apr.: 281-288.

[101] Siemon E, Wu Y. *A Comparison of Two Visual Programming Techniques for End Users*. A Workshop on Visual languages for End-user and Domain-specific Programming: The Visual End User. Seattle, WA, USA; 2000, Sept. 10.

[102] Siemon E. *Über den Entwurf von Benutungsschnittstellen technischer Anwendungen mit visuellen Spezifikationsmethoden und Werkzeugen*. Doctoral thesis. Darmstadt University of Technology; 2001, Jan.

[103] Sutherland IE. *Sketchpad: A Man-Machine Graphical Communication System.* In reading material for two-day seminar: Programming Techniques For Interactive Computer Graphics. National Engineering Laboratory, USA; 1968, Aug. 12-13: 329-345.

[104] Tanriverdi V, Jacob RJK. *Interacting with Eye Movements in Virtual Environments*. CHI2000 Conference Proceedings: Conference on Human Factors in Computing Systems. Addison-Wesley Publishing Company; 2000, Apr.: 265-271.

[105] *Telerobot* [online]. Available from: http://telerobot.mech.uma.edu.au. Accessed 2001 June 13.

[106] *The Amsterdam Collection of Patterns in User Interface Design* [online]. Available from: http://www.cs.vu.nl/~martijn/patterns/index.html. Accessed Apr. 2000.

[107] *The Brighton Usability Pattern Collection* [online]. Available from: http://www.it.bton.ac.uk/cil/usability/patterns. Accessed Sept. 1999.

[108] Tidwell J. *Interaction Design Patterns* [online]. Available from: http://www.mit.deu/~jtidwell/common-ground.html. Accessed 1999 Aug. 12.

[109] *User-Centered Design*. IBM. Available from: http://www-3.ibm.com/ibm/easy/eou_ext.nsf/Publish/570. Accessed 2000 Dec. 12.

[110] van Welie M. *Interaction Patterns in User Interfaces*. In the 7th Annual Pattern Languages of Programming Conference [Online Proceedings]. PLoP2000. Allerton Park, Illinois, USA; 2000, Aug. 13-16.

[111] *VisualWorks Cookbook*. ParcPlace Systems, Inc.; 1994.

[112] *VisualWorks Object Reference*. ParcPlace Systems, Inc.; 1994.

[113] *VisualWorks Tutorial*. ParcPlace Systems, Inc.; 1994.

[114] *VisualWorks User's Guide*. ParcPlace Systems, Inc.; 1994.

[115] Vlissides JM, Coplien JO, Kerth NL. *Pattern Languages of Program Design 2*. Addison-Wesley Publishing Company;1995.

[116] Vollmar F, Schäckermann F. *San Francisco: Application Business Process Components*. STJA'98 Tagungsband. Erfurt, Germany; 1998, Oct. 6-8: 299-309.

[117] von Stryk O. *Mobile and Sensor Guided Robotic Systems*. Lectures held at Darmstadt University of Technology. Material available from: http://www.sim.informatik.tu-darmstadt.de/. Accessed 2001 June 13.

[118] Wahl VFM. *ROBOTIK-QUO VADIS?* Schwerpunkthelf Informatik. Germany; 1999.

[119] *WAP Architecture Version 30-Apr-1998: Wireless Application Protocol Architecture Specification* [online]. Wireless Application Protocol Forum, Ltd.; 1988. Available from: http://www1.wapforum.org/tech/documents/WAP-100-WAPArch-19980430-a.pdf. Accessed 2000 Sept. 20.

[120] Weik MH. *The ENIAC Story* [online]. Available from: http://ftp.arl.army.mil/~mike/comphist/eniac-story.html. Accessed 2000 Aug. 31.

[121] Wilson S, Johnson P. *Bridging the Generation Gap: From Work Tasks to User Interface Designs.* In J. Vanderdonckt (ed.), Proceedings of CADUI'96. Belgium: Presses Universitaires de Namur; 1996: 77-94.

[122] Winterhagen J. *Fahrerassistenzsysteme für Nutzfahrzeuge*. Automobiltechnische Zeitschrift. 1999, Oct: 820-821.

[123] Wu Y, Ni Y, Li W. *A Large Radio Paging System Based on a Microcomputer Local Area Network*. The Journal of Computer Engineering and Application. Beijing; 1997.

[124] Wu Y, Ni Y. *The Software Implementation Technique of Telephone Voice Mailbox*. The Journal of South China University of Technology. 1997, Apr.;Vol.25, No.4: 96-100.

[125] Wu Y, Wen W, Zhu B, Ni Y. *The Implementation of a Radio Paging System Based on the Novell Btrieve*. The Journal of South China University of Technology. 1997, Mar;Vol.25, No.3: 40-46.

[126] Wu Y. *A Decision Support System for GuangZhou No.2 Cigarette Factory*. M.Sc. Thesis. Guangzhou: South China University of Technology; 1991, June.

[127] Wu Y. *A Multi-Route Data Application System*. Edited-collection of Annual Proceeding of Guangzhou AI Association. 1988.

[128] Wu Y. *Acquisition-Computing-Execution-Expression (ACEE): A Software Architecture Pattern for Computer-supported Automation and Control Systems*. In the 6th Annual Pattern Languages of Programming Conference [Online Proceedings]. PLoP1999. Allerton Park, Illinois, USA; 1999, Aug. 15-18. Available from: http://jerry.cs.uiuc.edu/~plop/plop99/proceedings/.

[129] Wu Y. *Design and Implementation of a Synthesis Radio Paging System*. Edited-collection of 95 Guangdong Youth Academic Proceeding in Automation and Computer. China: Electronic Industry Press; 1995:145-150.

[130] Wu Y. *Design Patterns and Framework for WIMP+ User Interface Design*. CHI2000 Extended Abstracts: Conference on Human Factors in Computing Systems. Addison-Wesley Publishing Company; 2000, Apr.: 359.

[131] Wu Y. *LLDemo: a graphical user interface for control a model robot via Internet*. Technical Report. Darmstadt University of Technology; 1999, Apr. 8.

[132] Wu Y. *Using Design Pattern Approach in the Development of User Interfaces for Controlling a Robot*. Patterns in Human Computer Interaction. BCS HCI Group /IFIP Working Group 13.2 Workshop on HCI Patterns. 2000, Nov. 16-17.

[133] Wu Y. *What else should a HCI Pattern Language include*? The Workshop of Pattern Languages for Interaction Design: Building Momentum. CHI2000. The Hague, The Netherlands; 2000, Apr. 2-3.

Curriculum Vitae

Yongmei Wu, MSc, is a doctoral candidate at the Chair Programming Languages and Compilers led by Prof. Dr. Hans-Jürgen Hoffmann at the Computer Science Department of the Darmstadt University of Technology. Before she joined the group in the early of 1998, she was a lecturer and a researcher at the Section Computer Application at the Department of Computer Science and Engineering at the South China University of Technology, where in addition to giving lectures she led and attended several industrial projects. For example, “SCUT-Voice: a Public Telephony Voice Service System”, “SRC Radio Paging System”, “MIS of Guangzhou No. 2 Cigarette Factory”, “A Distributed Information System of Automatically Transferring Navigational Telegrams”, “A Multi-Route Data Acquisition System”, etc. These systems have already taken their effects in relevant application domains.

Before this doctoral program, she won her BSc degree in Computer and Application at South China University of Technology in 1986 and MSc degree in Computer Organisation and Architecture at the same university in 1991, respectively.

She is an experienced software researcher and practitioner.