Extensible Interaction Sheets Language
XISL

version: 1.0
January 30, 2002

Abstract

This document specifies XISL, the Extensible Interaction Sheet Language. XISL is designed for describing interaction using multimodal inputs and outputs. Its goal is to realize advanced interaction using multi-modalities.

1. Introduction

XISL is designed for describing interaction between an element in an XML document (XML element) and a user. Interaction, in this document, is composed of a user's operation (e.g. click, speech input) for an XML element (e.g. displayed on a web browser) and an action (e.g. screen update, speech output) according to the operation. XISL can deal with multimodal interaction that uses input/output modalities cooperatively. For this purpose, XISL is designed to control parallel inputs/outputs, sequential inputs/outputs, and alternative input. Moreover, to control dialog (a set of interaction) flow, some commands, such as dialog transition, bargein control, and so on, are prepared.

Figure 1 shows an example of XISL and XML to output "Hello World". In this example, "Hello.xml" is an XML document to which an interaction is attached. The element <page> is assumed to be displayed on a web browser decorated by a XSL style sheet "hello.xsl". The XISL document "hello.xisl" contains a user's click operation and system's speech action.

NOTE : This document does not specify strict attributes and contents of <input> and <output> elements. That is, the authors have the flexibility to describe input/output modalities. This enables us to expand or modify modalities easily. The descriptive details of <input> and <output> should be specified by the platform developers. (It is intended that some standards should be specified for each type of terminal.)

hello.xml

```
<?xml version="1.0" encoding="Shift-JIS"?>
<?xml-stylesheet type="text/xsl" href="hello.xsl"?>
<!DOCTYPE hello SYSTEM "hello.dtd">
<page>
  Touch this page to hear"Hello World!"
</page>
```

hello.xisl

```
<?xml version="1.0" encoding="Shift-JIS"?>
<!DOCTYPE xisl SYSTEM "xisl.dtd">
<xisl version="1.0">
  <head> …… </head>
  <body>
    <dialog id="Hello World">
      <exchange>
        <operation target="hello.xml">
          <input type="pointing" event="l_click" match="/page" />
        </operation>
        <action>
          <output type="speech" event="play">
            <![CDATA[
              <param name="speech_text"> Hello World! </param>
            ]]>
          </output>
        </action>
      </exchange>
    </dialog>
  </body>
</xisl>
```

Figure 1: Sample of an XML and an XISL document

2. XISL execution system

XISL execution system shown in Figure 2 consists of the following three modules: a front-end module, a dialogue manager module, and a document server module. The document server is a general web server. It holds XISL, XML, and other documents (e.g. XISL stylesheet, speech grammar file, and so on), and sends them to the dialogue manager according to demands. The dialogue manager interprets XISL documents, manages dialog flow, and controls inputs and outputs. The front-end is a user interface terminal that contains microphone, speaker, screen, and so on. It is recommended that the dialog manager module is independent of the front-end module for modularity.

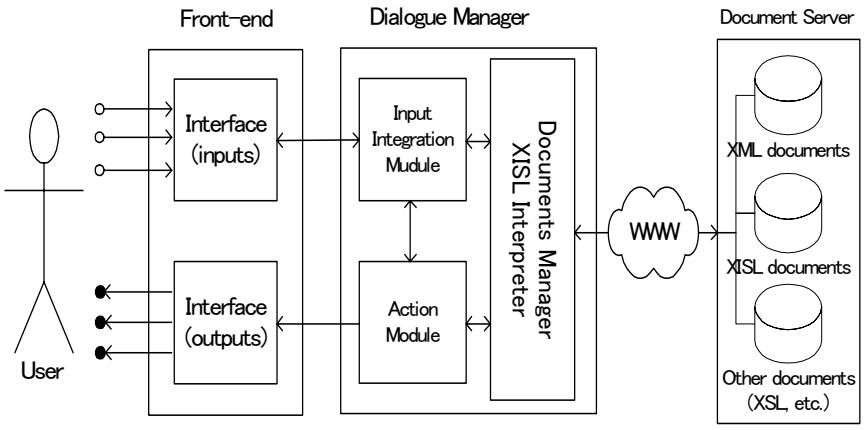

Figure 2: Architecture of XISL execution system

3. Goals of XISL

XISL is a language that:

1. Enables to use various type of front-ends (e.g. mobile phone, PC, digitai TV, …), and realizes seamless web services.
2. Is useful for describing complex multi-modal interaction.
3. Makes it easy to reuse XML contents and interaction by separating them.
4. Provides means for describing user initiative interaction, system initiative interaction, and mixed initiative interaction.

In order to bring these features, 1) XISL has the flexibility to expand input/output modalities arbitrary, 2) a lot of commands are prepared (e.g. parallel, sequential, and alternative dialog flow, bargein, conditional branch, iteration, calculation, editing of XML documents, and so on), 3) only interaction is described in XISL, 4) both prompting interaction and non-prompting interaction are able to be described.

4. Scope of XISL

XISL is a language to describe interaction between an arbitrary XML element and a user. Each XML element has to be presented to a user by some method (e.g. displayed in a web browser, provided as some hardware) in order to catch operations from the user. The front-end module may be a personal computer or a mobile phone, and so on, however, XISL does not restrict the type of terminals. (The front-end might be a robot which possesses a lot of sensors as input modes, and operates three-dimensional actions as outputs.)

5. Dialog levels

XISL constructs five layers of dialog levels: session, application, document, dialog, and exchange. A user always executes a dialog, and at the same time, the user executes the document, the application, and the session containing the dialog. A user is either executing an exchange or standing by to execute some exchanges. An XISL author can declare variables in all levels except for session level, and can describe transition of dialog to all levels except for session level and exchange level. Figure 3 shows the relation of the levels.

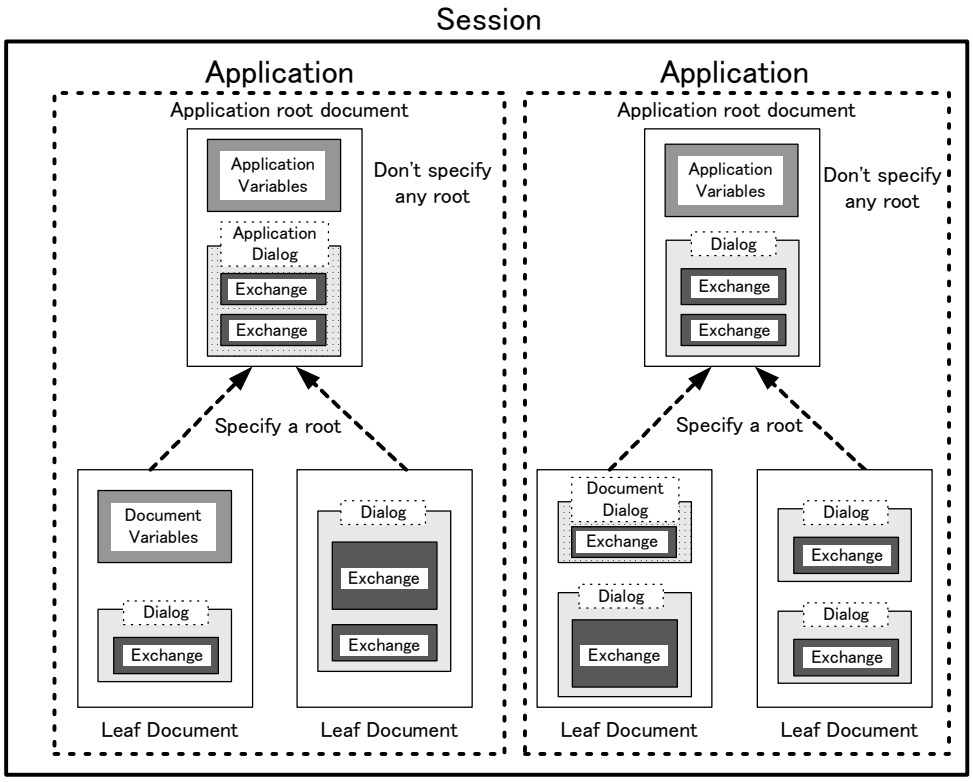

Figure 3: Relation of dialog levels

5.1. Exchange

Exchange is a unit of interaction between a user and a system. Two ways are prepared to denote exchange. One is the non-prompting description that only contains a user's operation and a system's action. This description is used for user initiative interaction without prompting. The other is the prompting description that contains a prompt, a user's operation, and a system's action. This description is used for system initiative interaction by the prompt. By mixing these descriptions, the author can

describe mixed initiative interaction.

5.2. Dialog

Dialog is composed of some exchanges. The user is always executing a dialog. The exchanges in a dialog are executed in sequential order unless any dialog control commands are described. An XISL author can indicate the subsequent dialog in the arbitrary dialog.

Each dialog belongs to dialog level, document level, or application level. A dialog belongs to the dialog level is executed only when the dialog is indicated to be the subsequent dialog. That is, it is not executed if it is not indicated to be the subsequent dialog.

5.3. Document

Document is an XISL file composed of some dialogs. In each XISL document, the author can declare document level variables and document level dialogs. Document level variables are available during the document is being executed. Document level dialogs are available during the document is being executed, that is, the user can transit to those dialogs at all times in the document. These variables and dialogs become unavailable when the dialog transits to the other document.

5.4. Application

Application, which constructs a task, is composed of a set of XISL documents. An application contains zero or more leaf documents and an application root document. In the application root document, the author can declare application level variables and application level dialogs.

When an application is executed, either a leaf document or an application root document is executed. In the case where a leaf document is executed, corresponding application root document is also loaded to enable use of application level variables and dialogs.

Application level variables are available during the application is being executed. Application level dialogs are available during the application is being executed, that is, the user can transit to the dialog at all times in the application. These variables and dialogs become unavailable when the dialog transit to the other application.

5.5. Session

Session begins when a user connects to the XISL execution system. It continues as

XISL documents are loaded and processed, and ends when requested by the user or the XISL execution system.

6. XISL elements

Element	Purpose
<action>	Execute the actions in the element
<alt_exchange>	<exchange>s in this element are executed alternatively.
<alt_input>	<input>s in this element are accepted alternatively.
<assign>	Assign a variable a value
<author>	The author of the document
<begin>	Initial processes in a <dialog>
<body>	A container of <dialog>s
<call>	Go to another dialog (Return to the original dialog)
<case>	Used in <switch> elements
<date>	The date of document update
<details>	Details about the document
<delete_element>	Delete an XML element from the indicated XML document or the XML element held in an indicated variable
<dialog>	A set of interactions
<dialog_var>	Declare dialog level variables
<document_var>	Declare document level variables
<else>	Used in <if> elements
<elseif>	Used in <if> elements
<end>	Terminal process in a <dialog>
<exchange>	A unit of interaction between a user and a system
<exit>	Terminate executing dialog, document, and application
<get_attribute>	Obtain an attribute of XML element from the indicated XML document or the XML element held in an indicated variable
<get_element>	Obtain an XML element from the indicated XML document or the XML element held in an indicated variable
<get_value>	Obtain the text content of an XML element from the indicated XML document or the XML element held in an indicated variable.
<goto>	Go to another dialog (Does not return to the original dialog)
<head>	Meta informations about the XISL document
<history>	Update history
<if>	Conditional branch used together with <else>, <then>, and <elseif>
<input>	A user input
<item>	An item of update history
<name>	Name of the dialog
<operation>	A set of user's input
<other>	Used in <switch> elements
<output>	A user output
<par_exchange>	<exchange>s in this element are executed parallel.
<par_input>	<input>s in this element are accepted parallel.
<par_output>	<output>s in this element are executed parallel.

<prompt>	Give prompt to the user
<reprompt>	Reprompt the <prompt> element
<return>	Return to the original dialog
<seq_exchange>	<exchange>s in this element are executed sequentially.
<seq_input>	<input>s in this elements are accepted sequentially.
<seq_output>	<output>s in this elements are executed sequentially.
<set_attribute>	Update the attribute of an XML element in the indicated XML document or the XML element held in an indicated variable
<set_element>	Update the XML element in the indicated XML document or the XML element held in an indicated variable
<set_value>	Update the content of an XML element in the indicated XML document or the XML element held in an indicated variable
<submit>	Send some values to the document server via HTTP GET or HTTP POST request, and receive its return value
<switch>	Conditional branch used together with <case> and <other>
<then>	Used in <if> elements
<var>	Declare a variable
<value>	Insert the value of an expression in an <output>
<while>	Itarate the process in this element while the condition is true
<xisl>	The root element of an XISL document

7. Structure and execution of XISL document

7.1. Structure of XISL document

XISL is an XML application. The authors are expected to be familiar with the concepts and notations of XML 1.0. The grammar of XISL is defined by the DTD shown in Appendix.1. Here is a basic structure of an XISL document.

```
<?xml version="1.0" encoding="Shift-JIS"?>
<!DOCTYPE xisl SYSTEM "./xisl.dtd">
<xisl version="1.0">
  <head>
        --title, author, history, etc. --
  </head>
  <body>
        --contents--
              :
              :
  </body>
</xisl>
```

The root element of an XISL document is <xisl>, which contains elements <head> and <body>. Attribute "version" of <xisl> is a required attribute that indicates the version of XISL. Current version is 1.0. Attribute "application" specifies the URI of application root document (See section 7.2). If the specified document does not exist,

XISL execution system generates an execution error.

The <head> element is a container of the document's meta informations such as author, update history, and so on. The <body> element is a main component of XISL document that contains dialogs.

element	<xisl>	
purpose	The root element of an XISL document	
attributes	version	Indicate the version of XISL. Current version is 1.0.
	application	Specify the URI of application root document

element	<head>
purpose	Meta informations about the XISL document
attributes	No attributes

element	<body>
purpose	A container of <dialog>s
attributes	No attributes

Here is an example of <head> description.

```
<head>
   <name> OnLine Shopping System </name>
   <author> Takanori Kobayashi </author>
   <date> Dec.26 2001 </date>
   <details> Version 1.0 </details>
   <history>
      <item>
         <date> Jun.18 2001 </date>
         <details> Version 0.0 </details>
      </item>
               :
               :
   </history>
</head>
```

The <head> element contains following elements.

element	<author>
purpose	The author of the document
attributes	No attributes

element	<date>
purpose	The date of document update
attributes	No attributes

element	<details>
purpose	Details about the document
attributes	No attributes

element	<history>
purpose	Update history
attributes	No attributes

element	<item>
purpose	An item of update history
attributes	No attributes

element	<name>
purpose	Name of the dialog
attributes	No attributes

The <body> element contains a set of user-system interactions, their control description, and the other commands. <body> is composed of a <document_var> element and some <dialog> elements. The first <dialog> of an XISL document is executed if no <dialog> is specified to be executed. If a <dialog> terminates without specifying the next <dialog>, current document, and application are also terminated.

```
<body>
   <document_var>
          --Declaration of document variables--
   </document_var>
   <dialog> --A unit of dialogue-- </dialog>
                              :
                              :
</body>
```

7.2. Application and its execution

The attribute “application” in <xisl> is used to distinguish between the application root document and the leaf document. If an <xisl> does not have an attribute “application”, the document itself is the application root document. On the other hand, if an <xisl> has an attribute “application”, and indicats a URI, the document indicated by the URI is the application root document. In this case the document itself is a leaf document. In order to construct an application by multiple documents, firstly the author selects a document as the application root document, and then indicates its URI in the other documents' “application” attributes.

Every time the XISL interpreter loads a leaf document, it checks whether its

application root document has already been loaded. If it has not been loaded, it loads at that occasion. The application root document remains loaded until the application changes. Thus one of the following two conditions always holds during interpretation.

1. An application root document is loaded and the user is executing it.
2. An application root document and a leaf document of the application are both loaded and the user is executing in the leaf document.

There are several benefits to multi-document applications. First, the application level variables are available for use by the other documents in the application, so that information can be shared and retained. Second, the application level dialog is available when the user is executing a leaf document, so that some common interrupt dialog can be described.

The following is an example of application composed of two documents.

app-root.xisl (root document)

```
<?xml version="1.0" encoding="Shift-JIS"?>
<!DOCTYPE xisl SYSTEM "xisl.dtd">
<xisl version="1.0">
  <head> …… </head>
  <body>
    <dialog id="Help" scope="document">
      <exchange>
        <operation target="help.grxml">
          <input type="speech" event="recognize"
                 match="/grammar/rule[@id=help]">
            <param name="mode">command</param>
          </input>
        </operation>
        <action>
          <output type="speech" event="play">
            <![CDATA[
              <param name="speech_text"> Click the Page! </param>
            ]]>
          </output>
        </action>
      </exchange>
    </dialog>
  </body>
</xisl>
```

hello.xisl (leaf document)

```
<?xml version="1.0" encoding="Shift-JIS"?>
<!DOCTYPE xisl SYSTEM "xisl.dtd">
<xisl version="1.0" application="app-root.xisl">
  <head> …… </head>
  <body>
    <dialog id="Hello World">
      <exchange>
        <operation target="hello.xml">
          <input type="pointing" event="l_click" match="/page" />
        </operation>
        <action>
          <output type="speech" event="play">
            <![CDATA[
              <param name="speech_text"> Hello World! </param>
            ]]>
          </output>
        </action>
      </exchange>
    </dialog>
  </body>
</xisl>
```

This example assumes that "hello.xisl" is executed as a leaf document. The attribute "application" of <xisl> in "hello.xisl" indicates "app-root.xisl" as the application root document. In the "app-root.xisl" document, <dialog>'s attribute "scope" is set to document. This means the dialog "Help" is declared as an application level dialog. Hence the dialog is available during the application is being executed.

7.3. Dialog transition and change of application

Dialog transition sometimes causes change of application and document. The followings are the relation between dialog transition and change of application, change of document. Figure 4 shows the relation.

- Dialog transition that does not change document

  If the executing dialog and the subsequent dialog are in the same document, the dialog transition does not change the document.
- Dialog transition that changes document

  When the executing dialog and the subsequent dialog are in the different document, the dialog transition changes the document.
- Dialog transition that does not change application

  This type of transition belongs to one of the following cases.

- The application root document of current dialog and that of the subsequent dialog are the same document.
- The current dialog is in the application root document D, and the subsequent dialog's application root document is also D.
- The application root document of current dialog is the document D, and the subsequent dialog is in the document D.
- The current dialog is in the application root document D, and the subsequent dialog is also in the document D.

- Dialog transition that changes application

  The application changes if a dialog transition does not satisfy the above conditions.

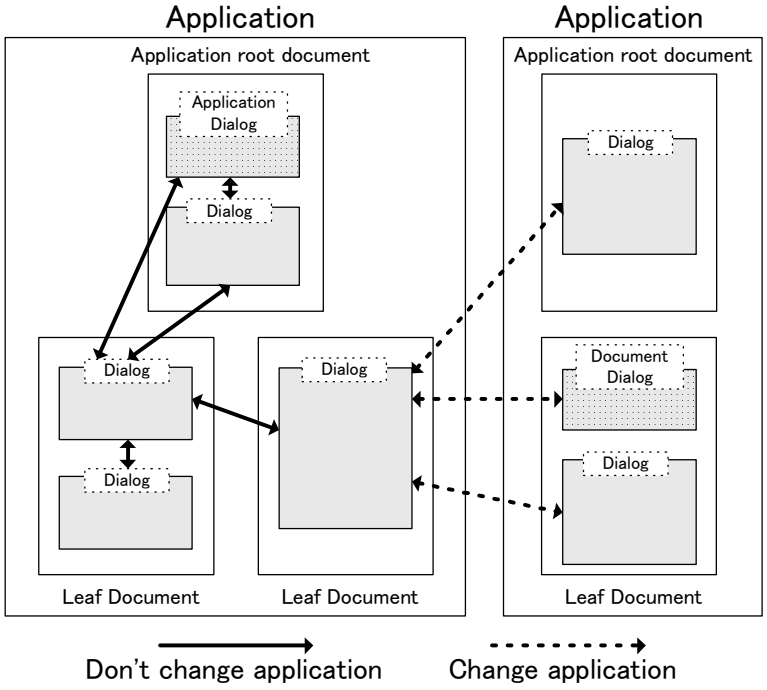

Figure 4: Transition of dialog and change of application

Appendix 1. DTD of XISL

```
<!--
    XISL 1.0 DTD
    version 2002-01-21
-->

<!ENTITY   % exchange.type " par_exchange | seq_exchange | alt_exchange |
exchange ">

<!ENTITY   % input.type " input | par_input | seq_input | alt_input ">

<!ENTITY   % output.type " output | seq_output | par_output ">

<!ENTITY   % begin.end.content " var | assign | if | switch | while |
                                 call | return | goto | submit |
                                 exit |
                                 seq_output | par_output | output |
                                 get_value | get_element | get_attribute |
                                 set_value | set_element | set_attribute |
                                 delete_element ">

<!ENTITY   % action.content " %begin.end.content; | reprompt ">

<!ENTITY % boolean     "( true | false )" >

<!ENTITY % duration "CDATA" >

<!ENTITY % conditional.logic "CDATA">

<!ENTITY % expression "CDATA" >

<!ENTITY % variable.name "CDATA" >

<!ENTITY % variable.names "CDATA" >

<!ENTITY % integer "CDATA" >

<!ENTITY % target.name "CDATA" >

<!ENTITY % uri "CDATA" >

<!ENTITY % path "CDATA" >

<!ENTITY   % get.set.attrs      " target   %target.name;       #REQUIRED
```

```
                                var       %variable.name;     #REQUIRED
                                match     %path;              #REQUIRED
                                place     (temp | master)     'temp' ">

<!--========================== Root ==========================-->

<!ELEMENT xisl (head?,body) >
<!ATTLIST xisl   version             CDATA   #REQUIRED
                 application         %uri;   #IMPLIED>

<!ELEMENT head      (name?,author?,date?,details?,history?)>

<!ELEMENT name      (#PCDATA)>

<!ELEMENT author    (#PCDATA)>

<!ELEMENT date      (#PCDATA)>

<!ELEMENT history (item+)>

<!ELEMENT item      (date,details)>

<!ELEMENT details (#PCDATA)>

<!ELEMENT body      (document_var?,dialog*)>

<!ELEMENT document_var (var+)>

<!--========================== Dialog ==========================-->

<!ELEMENT dialog   ( dialog_var?, begin?, (%exchange.type;)* , end?) >

<!ELEMENT dialog_var (var+) >
<!ATTLIST dialog    id          ID       #REQUIRED
                    comb        (par | seq | alt)     "seq"
                    repeat      CDATA                  "1"
                    scope       (dialog | document)    "dialog"
                    arg         %variable.names;       #IMPLIED>

<!--========================== Begin ==========================-->

<!ELEMENT begin     (%begin.end.content;)*>

<!--========================== End ==========================-->
```

```
<!ELEMENT   end       (%begin.end.content;)*>

<!--========================== Exchanges ==========================-->

<!ELEMENT par_exchange   (%exchange.type;)+>

<!ELEMENT seq_exchange   (%exchange.type;)+>

<!ELEMENT alt_exchange   (%exchange.type;)+>

<!ELEMENT exchange        (prompt?,operation,action)>

<!--========================== Prompt ==========================-->

<!ELEMENT prompt   (%output.type;)+   >
<!ATTLIST prompt    count         %integer;    "1"
                    timeout       %duration;   #IMPLIED
                    bargein       %boolean;    "true" >

<!--========================== Operation ==========================-->

<!ELEMENT operation    (%input.type;)+ >
<!ATTLIST operation    target %target.name;        #IMPLIED
                       comb ( par | seq | alt )   "par">

<!ELEMENT par_input (%input.type;)+>

<!ELEMENT seq_input (%input.type;)+>

<!ELEMENT alt_input (%input.type;)+>

<!ELEMENT input (param)*>
<!ATTLIST input   type        CDATA    #REQUIRED
                  event       CDATA    #REQUIRED
                  target      %target.name;       #IMPLIED
                  match       %path;      #REQUIRED
                  return      %variable.names;   #IMPLIED >

<!ELEMENT param (#PCDATA)>
<!ATTLIST param   name      CDATA    #REQUIRED >

<!--========================== Action ==========================-->

<!ELEMENT action    (%action.content;)*>
```

```
<!ELEMENT var EMPTY>
<!ATTLIST var    name   %variable.name;     #REQUIRED
                 expr   %expression;        #IMPLIED>

<!ELEMENT assign EMPTY>
<!ATTLIST assign   name   %variable.name;  #REQUIRED
                   expr   %expression;     #REQUIRED>

<!ELEMENT if (then, (else? | elseif*))>
<!ATTLIST if cond   %conditional.logic; #REQUIRED>

<!ELEMENT then   (%action.content;)*>

<!ELEMENT else   (%action.content;)*>

<!ELEMENT elseif   ( then, (else?|elseif*) )>
<!ATTLIST elseif   cond %conditional.logic;   #REQUIRED>

<!ELEMENT switch   ( case+, other? )>
<!ATTLIST switch   expr   %expression;   #REQUIRED>

<!ELEMENT case   (%action.content;)*>
<!ATTLIST case   value CDATA #REQUIRED>

<!ELEMENT other (%action.content;)*>

<!ELEMENT while (%action.content;)*>
<!ATTLIST while cond %conditional.logic; #REQUIRED>

<!ELEMENT call EMPTY>
<!ATTLIST call   next          %uri;   #REQUIRED
                 namelist      %variable.names;   #IMPLIED
                 return        %variable.names;   #IMPLIED>

<!ELEMENT return   EMPTY>
<!ATTLIST return   namelist   CDATA   #IMPLIED>

<!ELEMENT goto EMPTY>
<!ATTLIST goto   next        %uri;                #REQUIRED
                 namelist   %variable.names;   #IMPLIED>

<!ELEMENT submit EMPTY>
<!ATTLIST submit next       %uri;   #REQUIRED
                  method    (get | post) "get"
                  namelist %variable.names;    #IMPLIED
```

```
                    return     %variable.names;     #IMPLIED >

<!ELEMENT exit EMPTY>

<!ELEMENT reprompt EMPTY>

<!ELEMENT seq_output (%output.type;)*>

<!ELEMENT par_output (%output.type;)*>

<!-- The <value> elements appear in the CDATA section of <output> elements.
      <!ELEMENT value    EMPTY>
      <!ATTLIST value    expr    %expression;    #REQUIRED>
 -->

<!ELEMENT output    (#PCDATA) >
<!ATTLIST output    type       CDATA    #REQUIRED
                    event      CDATA    #REQUIRED>

<!ELEMENT get_value EMPTY>
<!ATTLIST get_value    %get.set.attrs;>

<!ELEMENT get_element EMPTY>
<!ATTLIST get_element %get.set.attrs;>

<!ELEMENT get_attribute    EMPTY>
<!ATTLIST get_attribute    %get.set.attrs;>

<!ELEMENT set_value    EMPTY>
<!ATTLIST set_value    %get.set.attrs;>

<!ELEMENT set_element    EMPTY>
<!ATTLIST set_element    %get.set.attrs;
                         type (insert | overset) #REQUIRED>

<!ELEMENT set_attribute     EMPTY>
<!ATTLIST set_attribute     %get.set.attrs;>

<!ELEMENT delete_element EMPTY>
<!ATTLIST delete_element target     %target.name;      #REQUIRED
                         match      %path;             #REQUIRED
                         place      (temp | master)    "temp">
```