Voice Browser Working Group (VBWG)

Input on application backplane topics

Scott McGlashan (HP)
Rafah Hosn (IBM)

W3C Backplane Meeting,
Amsterdam, November 2006

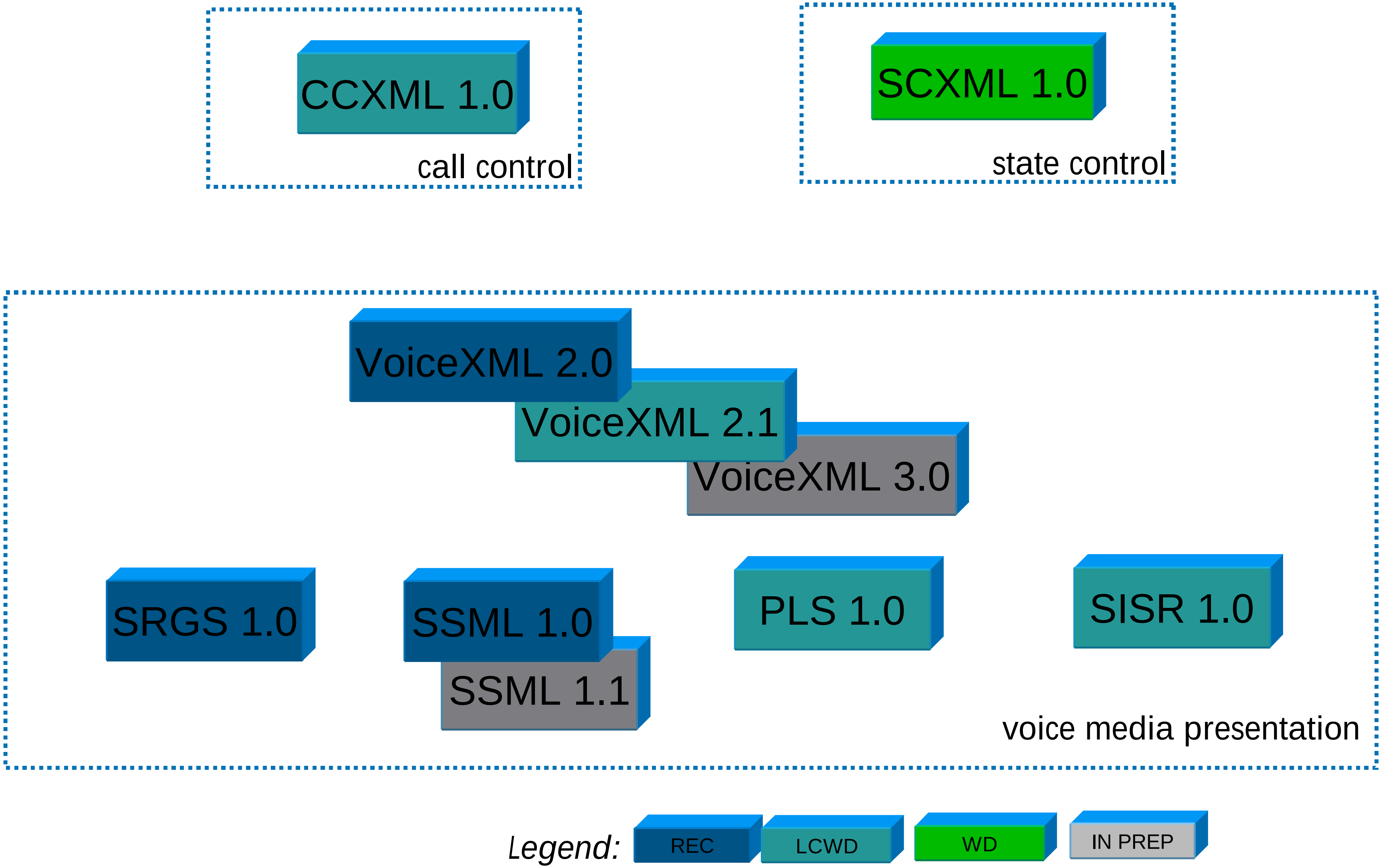
Agenda

- Key points
- VBWG specifications
- Data-Flow-Presentation (DFP) framework
- VoiceXML 2.0 / 2.1 / 3.0
- SCXML 1.0
- Key points

Key points

- VBWG specifications demonstrate loose coupling between (distributed) application components
- Application components have independent data models
  - Multiple data models for a given application
  - Relationship between data models is under developer control
- Data model binding and submission is under developer control
  - Data binding allows validation
  - Data submission is separate from document transition and may be asynchronous
- SCXML is a generic state machine language
  - a backplane mechanism to coordinate and synchronize application components

VBWG specifications



CCXML 1.0

call control

SCXML 1.0

state control

VoiceXML 2.0

VoiceXML 2.1

VoiceXML 3.0

SRGS 1.0

SSML 1.0

SSML 1.1

PLS 1.0

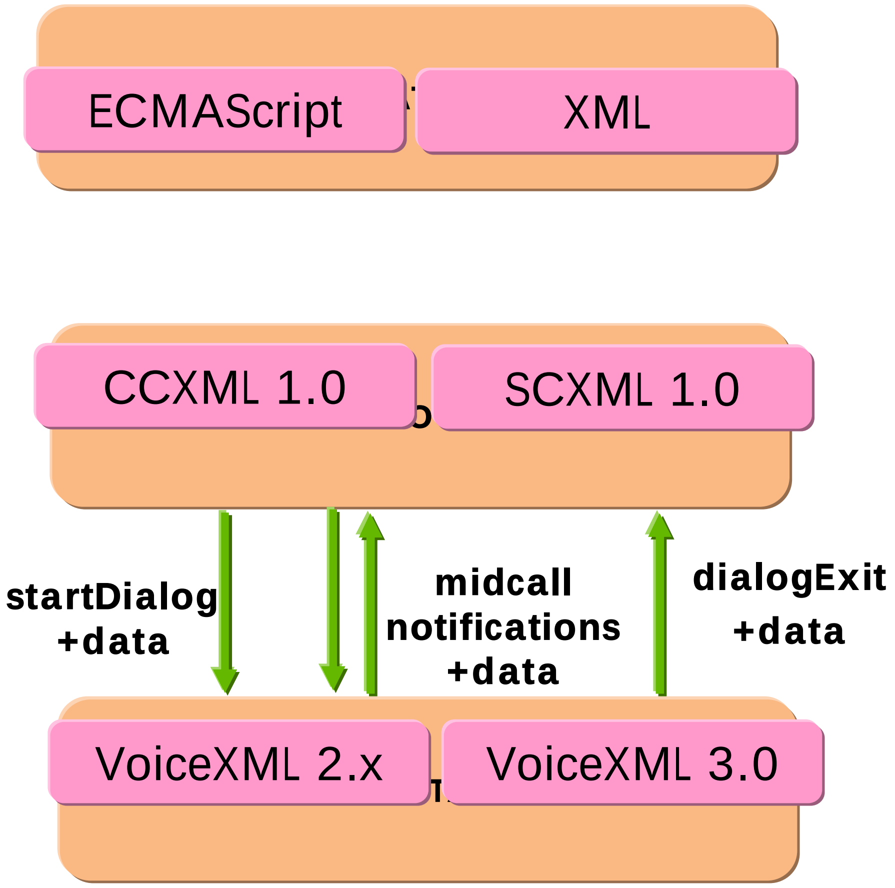
SISR 1.0

voice media presentation

Legend: REC LCWD WD IN PREP

Data Flow Presentation (DFP) application framework

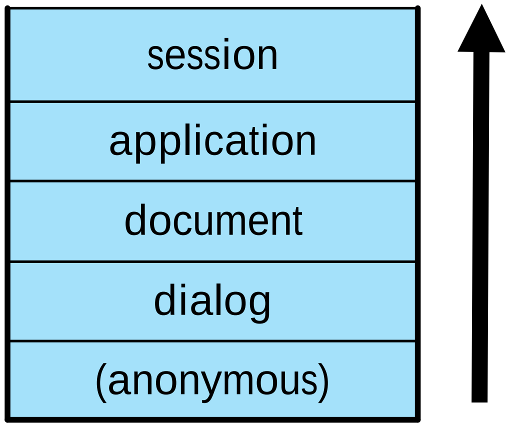
- Logical framework for modular voice-centric application development (cf. MVC)
- Data: application data representation
- Flow: controls application flow
  - no interaction with user
- Presentation: input/output dialog
  - interaction with user
- Benefits:
  - Simplifies code reuse
  - Improves intelligibility
  - Extensible to multimodality; e.g. presentations may be VoiceXML, SVG, and/or XHTML



Data models

- Some VBWG languages are executed in containers on flow layer, others in containers on presentation layer
- Flow and presentation containers have their own independent data models
  - Data models are private to a container
  - Container can send parts of its data model to other containers or resources
  - Container can update its data model using information received from other containers or resources
- Policy for updating data models is under developer control
  - How data received by a container is used to update model; none, automatic, filtered, translated, …
  - How to resolve conflicts when incompatible data received from the same or multiple resources
  - Generally how one data model affects another
- These data model properties are embodied in both current and emerging VBWG languages
  - Expect continuation for mashups where document contains multiple namespace

VoiceXML 2.0 – data



- Data model
  - ECMAScript
  - Scoped
- Data binding
  - Automatic binding; e.g. user input
    - <field name="var2" type="digits">
  - Manual binding
    - <assign name="var2" expr="filter(var2)"/>
    - <assign name="application.myresult" expr="var2"/>
- Data submission
  - Synchronous with page transition
  - Automatic: <submit next="http://example.com/page.vxml"/>
  - Manual: <submit next="http://example.com/page.vxml" namelist="var1 var2"/>

VoiceXML 2.1 – data submission

- synchronous without page transition
- 1. Automatic binding with ECMAScript
  - <script srcexpr="'http://example.com/service?param=var1'"/>
  - 'srcexpr' evaluated when element executed – contents of the script can be added to the data model at the current scope
- 2. Manual binding with XML
  - <data name="myxmldata" srcexpr="'http://example.com/service?param1=var1'"/>
  - where 'myxmldata' is an ECMAScript variable which exposes the received XML data as read-only DOM2 subset; ECMAScript then used to access the XML data and bind it to data model

VoiceXML 3.0 – data submission

- Synchronously or asynchronously without page transition: send data in event to external resource

```
<send async=“true” target=“http://www.example.com/app”
   event=“myevent” namelist=“param1”/>
```

- Catch handler receives event asynchronously

```
<catch event=“externalevent”>
     <log>
      event name: <value expr=“application.lastmessage$.event”/>
     </log>
</catch>
```

- Receive handler receive events synchronously

```
<receive maxtime=“10s” fetchaudio=“liftmusic.wav”>
     <log>
      received event <value expr=“application.lastmessage$.event”>
     </log>
</receive>
```

VoiceXML 2.0 – event model

- VoiceXML 2.0 has its own event model
  - <catch> event handlers and event propagation
- Event model is not compatible with DOM2, but almost compatible with DOM3
- DOM3 addresses
  - Partial name matching using event categories
  - Document order selection using listener groups
- DOM3 restrictions required
  - Only bubble phase supported
  - Selected event handler always stops propagation

VoiceXML 3.0 – event model

- Key issue for DOM3 compatibility
  - Events have a count (times the same event fired within FIA cycle)
  - Propagate event to find best qualified handler; e.g. ‘nomatch’ event with count = 2
- Possible solution: allow ‘count range’ syntax
  - Compatible with DOM3, incompatible with VoiceXML 2.x

```
<form>
    <catch event=“nomatch” count=“4”>
        …
    </catch>
    <field>
        <catch event=“nomatch” count=“1”>
            …
        </catch>
    </field>
</form>
```

VoiceXML 3.0 – event model

- Key issue for DOM3 compatibility
  - Events have a count (times the same event fired within FIA cycle)
  - Propagate event to find best qualified handler; e.g. ‘nomatch’ event with count = 2
- Possible solution: allow ‘count range’ syntax
  - Compatible with DOM3, incompatible with VoiceXML 2.x

```
<form>
    <catch event=“nomatch” count=“4”>
        …
    </catch>
    <field>
        <catch event=“nomatch” count=“1-3”>
            …
        </catch>
    </field>
</form>
```

This handler now matches event count – best qualified handler can be determined locally

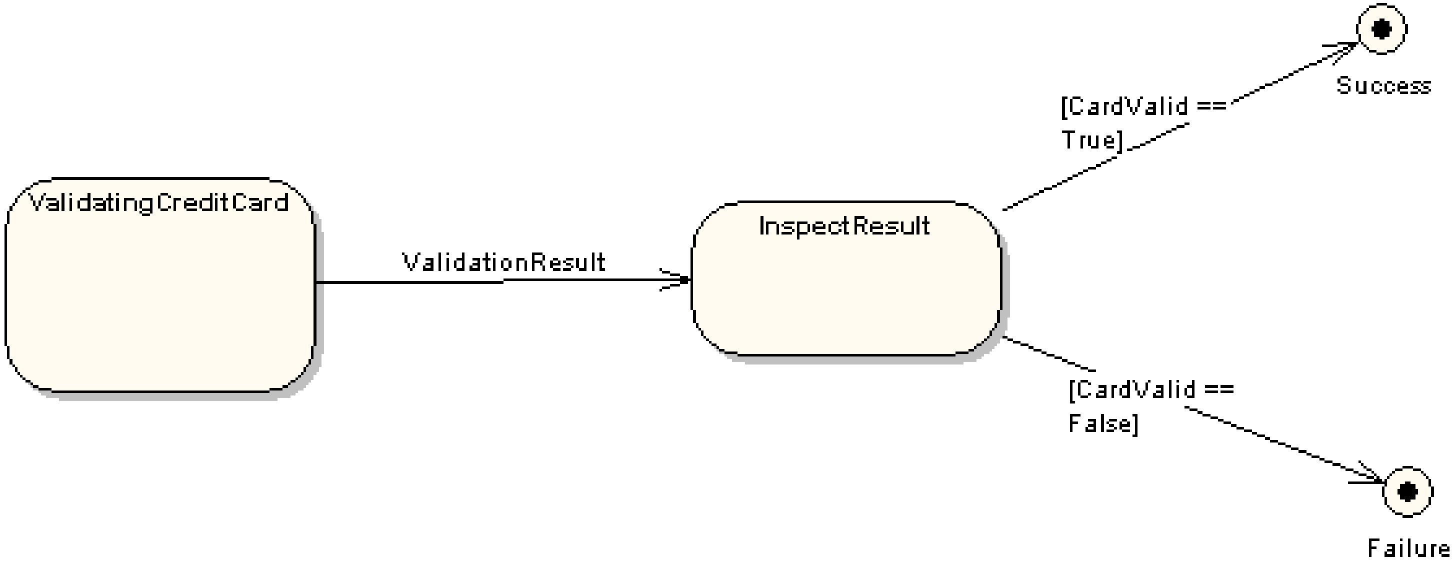
SCXML 1.0

- Design Goals
  - Flow control container in the VBWG DFP architecture
  - Interaction manager in the MMIWG Multimodal architecture
- SCXML is a **generic** state machine language
  - Can be used to manage flow between application components (presentation or otherwise)
  - Backplane mechanism to coordinate and synchronize application components
- SCXML is based on Harel state charts:
  - a mathematical representation of state machines
  - the underpinning of UML state semantics
  - Provide powerful, compact control abstractions
- SCXML re-uses CCXML concepts:
  - CCXML: an event-driven language for managing flow between telephony connections, conferences and dialogs (e.g. VoiceXML)
  - SCXML inherits non-DOM event model, asynchronous data submission, action handlers, (dialog) invocation, etc

SCXML 1.0 – state chart semantics

- State charts have all the traditional state machine semantics:
  - States – status of machine
  - Transitions – move between states
  - Events and conditions – transition triggers
- As well as advanced features:
  - hierarchical states – state decomposed into child states
  - parallel states – multiple active child states
  - action handlers – executable behavior
  - history states – checkpointed version of a state
- And SCXML has some state chart extensions:
  - invocation of external resources

SCXML 1.0 – state chart in UML



SCXML 1.0 – state chart in XML

```
<scxml initialstate=“ValidatingCreditCard”>

<state id=“ValidatingCreditCard”>
  <transition event=“ValidationResult” target=“InspectResult”/>
</state>

<state id=“InspectResult”>
  <transition cond=“CardValid==true”  target=“Success”/>
  <transition cond=“CardValid==false”  target=“Failure”/>
 </state>

<state id=“Success” final=“true”/>
<state id=“Failure” final=“true”/>

</scxml>
```

SCXML 1.0 – data model

- XML data model rooted at <datamodel>
    - Contains 0 or more <data> elements
- <data> element has a name and an XML value
    - Data value can be specified inline or by reference
    - <datamodel>
        - <data name="mycds" src="http://example.com/cds.xml"/>
        - <data name="mydvds">
            - <dvds>
            -     <dvd artist="alabama3" …/>
            - </dvds>
        - </data>
    - <datamodel>

SCXML 1.0 – data binding

- XPath to specify location in data model
  - Other languages may be supported
- ECMAScript to specify value in data model
  - Other languages may be supported
- The data model is updated using <assign>; e.g. with information in external event

```
<transition event=“incomingevent”>
  <assign location=“/data[@name=‘mydvds’]/dvds”
   expr=“_eventdata.update”/>
</transition>
```

SCXML 1.0 – data validation

- Data model may be validated on loading
  - Optional ‘schema’ attribute on <datamodel>
- Developer can control validation on data binding
  - Optional <validate> child of <assign>
  - <validate> element has two attributes:
    - optional ‘location’ to specify data model portion to validate
    - optional ‘schema’ to specify schema to use for validation (alternative to using data model’s schema)

```
<assign location=“/data[@name=‘mydvds’]/dvds”
   expr=“_eventdata.update”>
    <validate location=“.” schema=“mydvds.xsd”/>
</assign>
```

SCXML 1.0 – data submission

- Fragments of the data model may be sent asynchronously to external resources

```
<send event=“myevent” target=“…” namelist=“mycds
  mydvds”/>

<invoke targettype=“vxml” src=“myscript.vxml”>
    <param name=“cds” expr=“mycds”/>
    <param name=“dvds” expr=“mydvds”/>
</invoke>
```

Key points

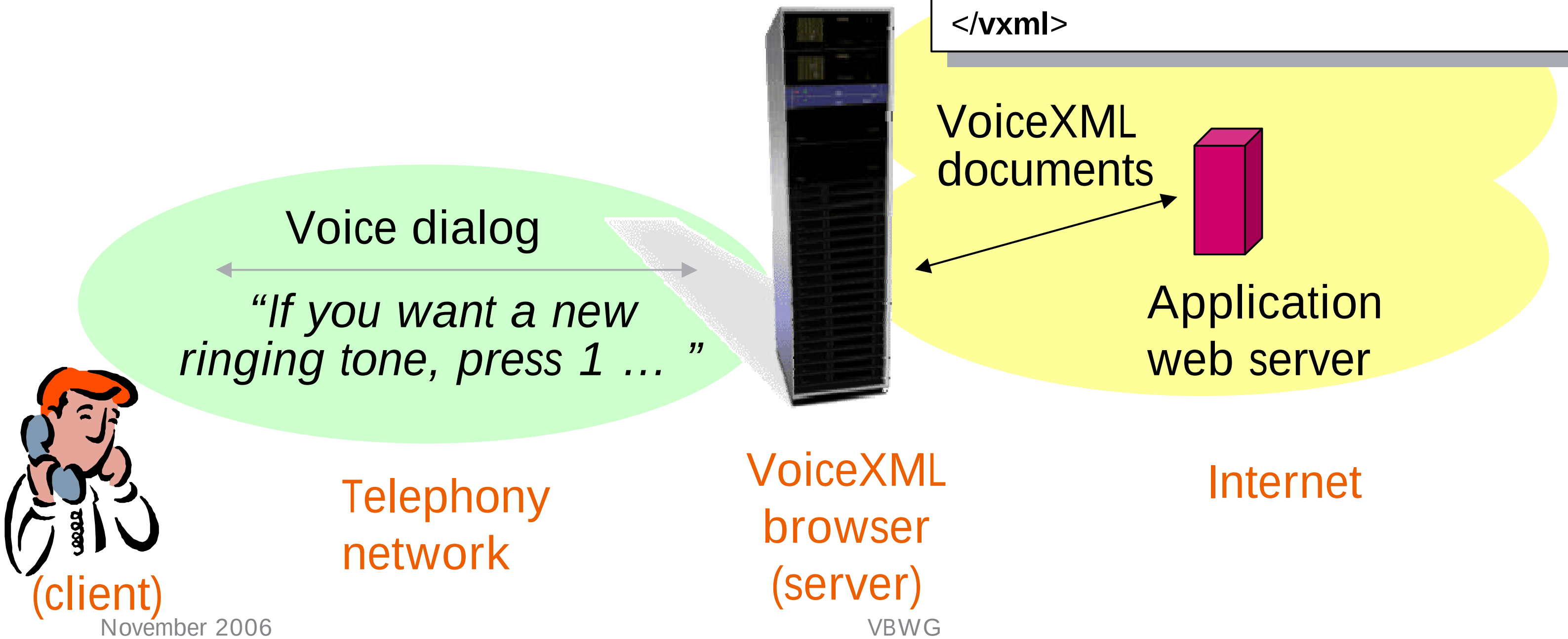
- VBWG specifications demonstrate loose coupling between (distributed) application components
- Application components have independent data models
  - Multiple data models for a given application
  - Relationship between data models is under developer control
- Data model binding and submission is under developer control
  - Data binding allows validation
  - Data submission is separate from document transition and may be asynchronous
- SCXML is a generic state machine language
  - a backplane mechanism to coordinate and synchronize application components

BACKUP

Typical VoiceXML deployment today

- Network browser
  - Browser in network server
  - Client (telephone) with basic media capabilities
  - Browser performance is critical

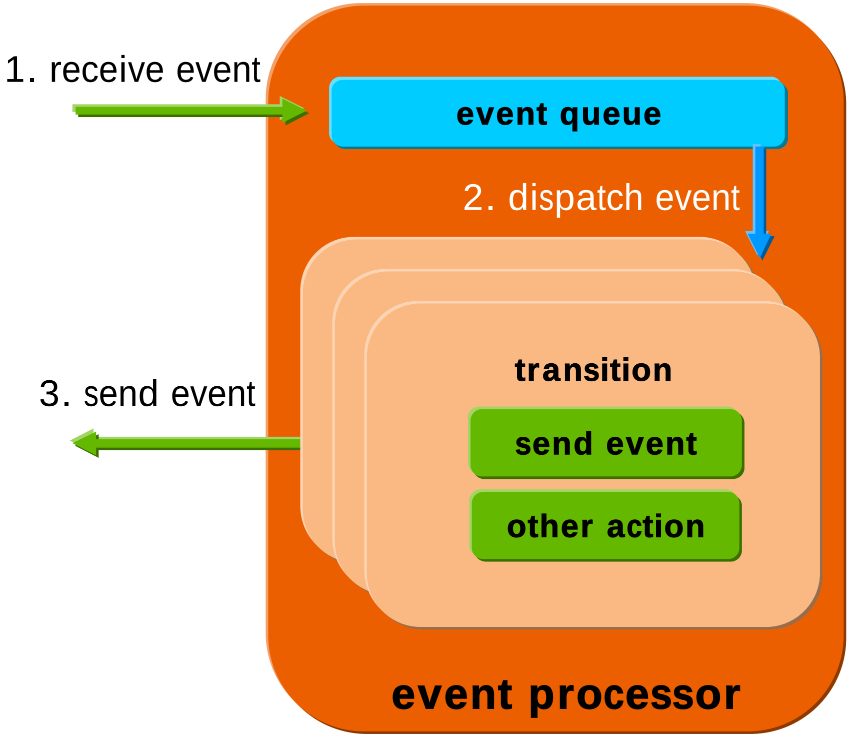
```
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">
<form>
        <field name="choice1">
                <prompt>If you want a new ringing tone,
                press 1 …  </prompt>
                <grammar src="dtmf-choices.grxml"/>
        </field>
        <submit next="result.jsp"/>
</form>
</vxml>
```



CCXML 1.0 - event processor

1. Receives events (internal or external) and stores them in event queue
2. If no events in queue, wait; otherwise, dispatch head event to event processor; if a transition matches the event, it processes the event:
   - Sends another event
   - Performs another action

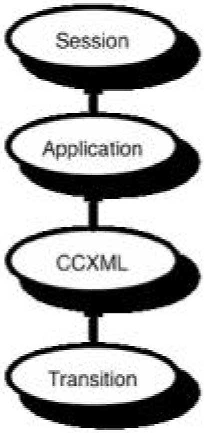
- Event processing is asynchronous



```
<ccxml version="1.0“ >
  <eventprocessor>
    <transition event="connection.alerting">
      <send target=“’...’"/>
    </transition>
  </eventprocessor>
</ccxml>
```

CCXML 1.0 – data model

- An event-driven language for managing flow between telephony connections, conferences and dialogs (e.g. VoiceXML)



- Data model
  - ECMAScript
  - Scoped
- Data binding
  - <assign name=“application.myvar” expr=“’astring’” />
- Data submission
  - Asynchronous without page transition; e.g.
  - <send target="'http://example.com/service'“ targettype=“’basichttp’” namelist=“param1 param2” />

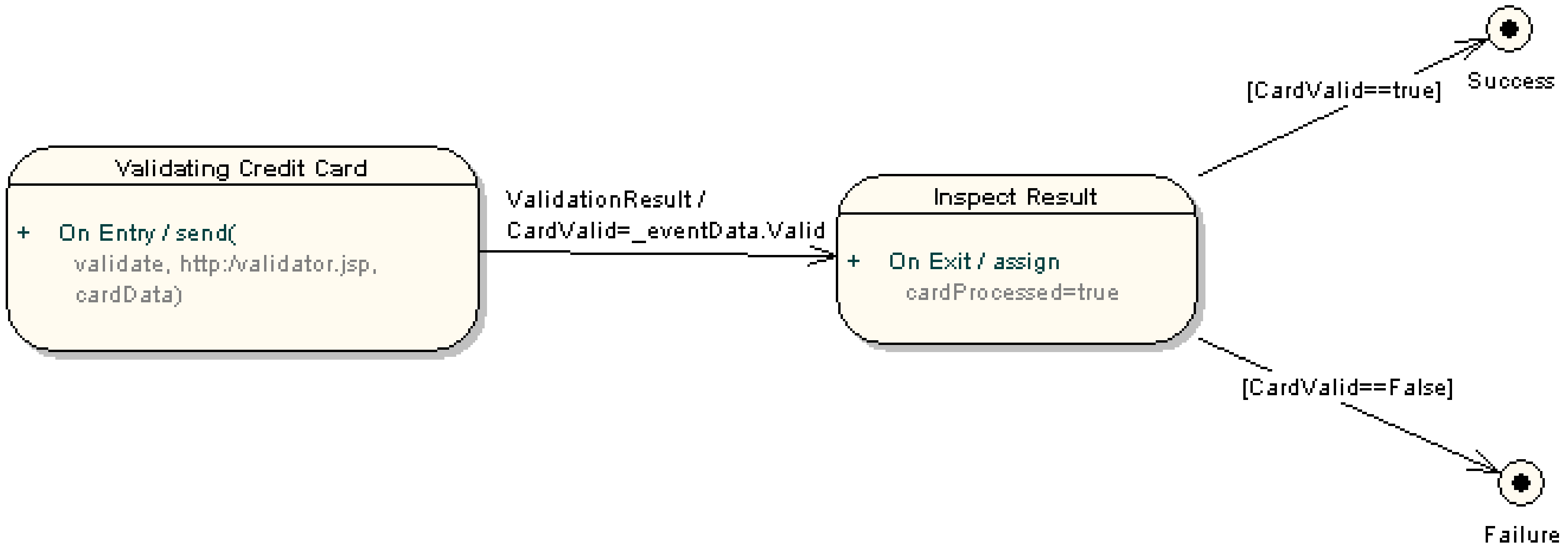
CCXML 1.0 – event model

- Event model
  - Not DOM based – own event processor
- CCXML can invoke external dialogs using <dialogstart>
  - <dialogstart> as shortcuts for <send …> event with data payload
  - Dialog can <send> events back to CCXML

```
<transition event="connection.connected">
   <dialogstart src="'http://example.com/page.vxml'"
   type="'application/voicexml+xml'" data="param1 param2"/>
</transition>

<transition event="dialog.exit">
     <assign name="dm" expr="event$.values.input"/>
</transition>
```

SCXML 1.0 – state chart in UML



SCXML 1.0 – state chart in XML

```
<scxml initialstate=“ValidatingCreditCard”>
<state id=“ValidatingCreditCard”>
       <onentry>
            <send event=“validate” target=“http:/card-validator.jsp” namelist=“cardData”/>
       </onentry>
      <transition event=“ValidationResult” target=“InspectResult”>
            <assign location=“CardValid” expr=“_eventData.valid?”/>
      </transition>
 </state>
<state id=“InspectResult”>
      <transition cond=“CardValid==true”  target=“Success”/>
      <transition cond=“CardValid==false”  target=“Failure”/>
      <onexit>
           <assign location=“CardProcessed” expr=“true”/>
      </onexit>
 </state>
<state id=“Success” final=“true”/>
<state id=“Failure” final=“true”/>
</scxml>
```