Sneak Preview: VoiceXML 3.0

- Slides presented at SpeechTEK August 2006
- Presenters:
  - Jim Barnett, Aspect
  - Emily Candell, Comverse
  - Jerry Carter, Nuance
  - Rafah Hosn, IBM
  - Scott McGlashan, Hewlett-Packard
- The contents of these slides represent the opinions of the presenters

Agenda

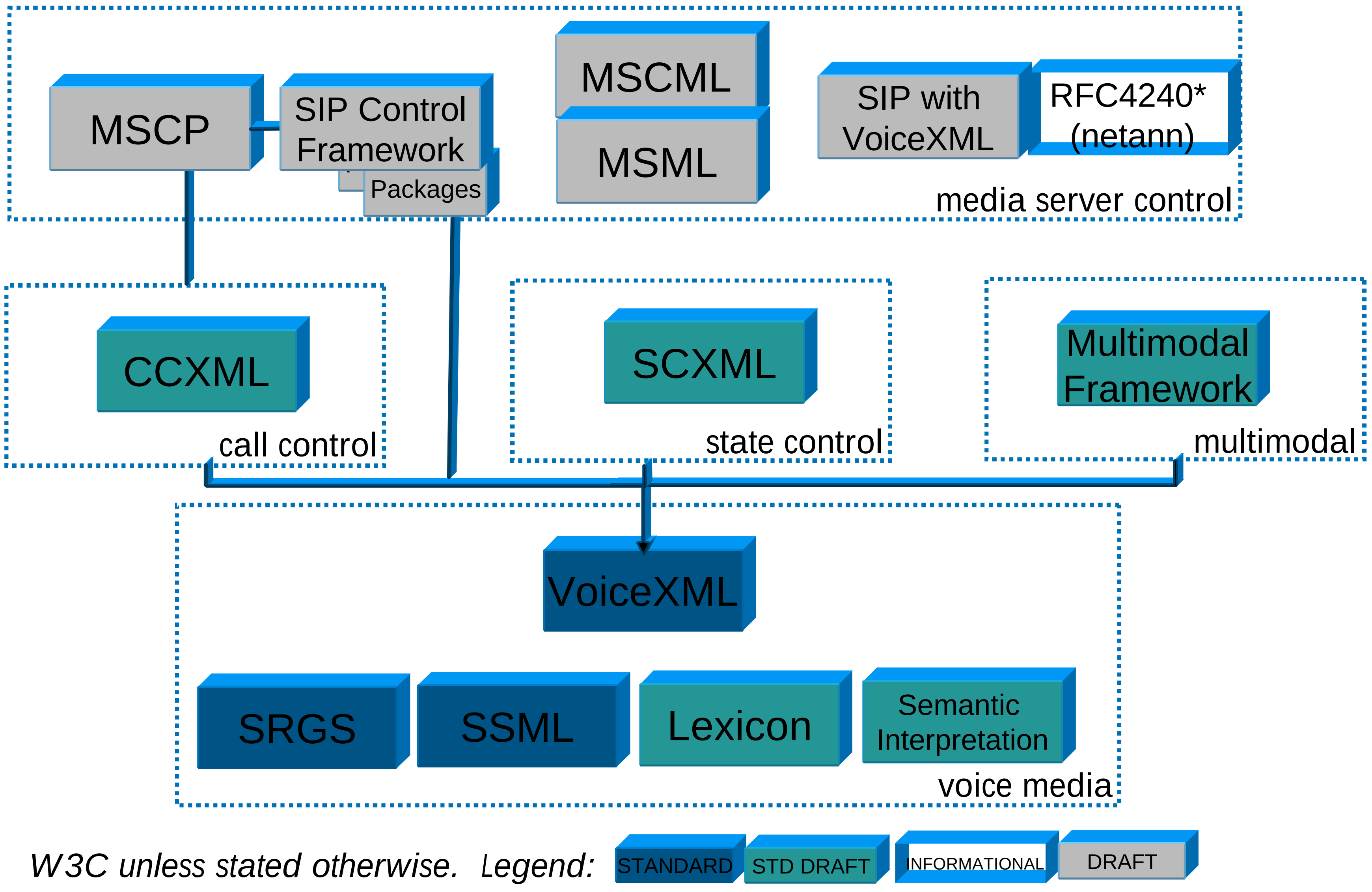
Agenda

- **Overview**
- DFP architecture
- Overview of Flow/SCXML
- Overview of Data Model
- VoiceXML 3 process
- Modularization
- New media features
- Recording and media control
- EMMA

VoiceXML 3.0

- Next major version of VoiceXML
  - Initial working draft expected 2007
- Driven by external and internal requirements
  - Addressing unresolved issues from VoiceXML 2.0/2.1
  - VoiceXML is being used in new ways (e.g. multimodal) and in other languages
- WARNING
  - VoiceXML 3.0 is still at an early stage of development
  - Its characteristics and features described in this tutorial are the *opinions* of the presenters
  - As an open standard driven by the W3C community process, the actual features of the language are determined when the specification is published
- We welcome your comments and opinions on the issues raised in this tutorial

XML voice media standards overview



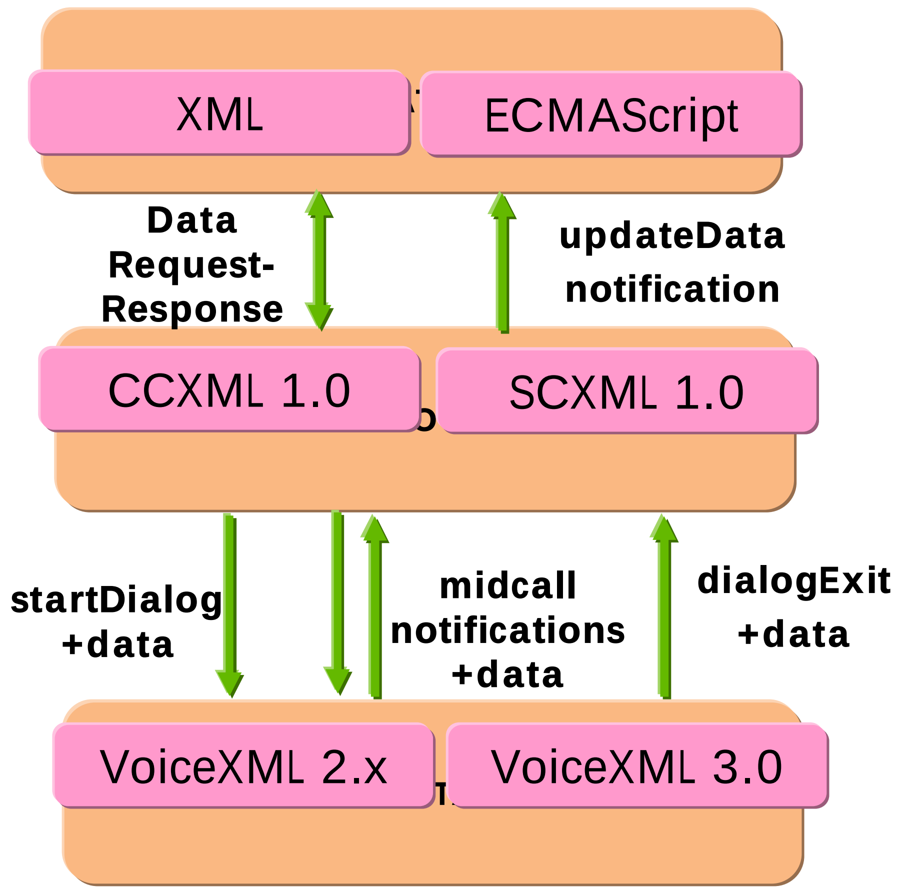
W3C unless stated otherwise. Legend:

* Not all RFCs are Standards

Agenda

Data Flow Presentation (DFP) Application Framework

- W3C framework for modular voice-centric application development
- Data: canonical data representation
- Flow: controls application flow
- Presentation: interaction with user
- Benefits: layered applications with ability to call VoiceXML 3.0 modules (e.g. prompt&CollectDTMF)



Separating Flow from Presentation

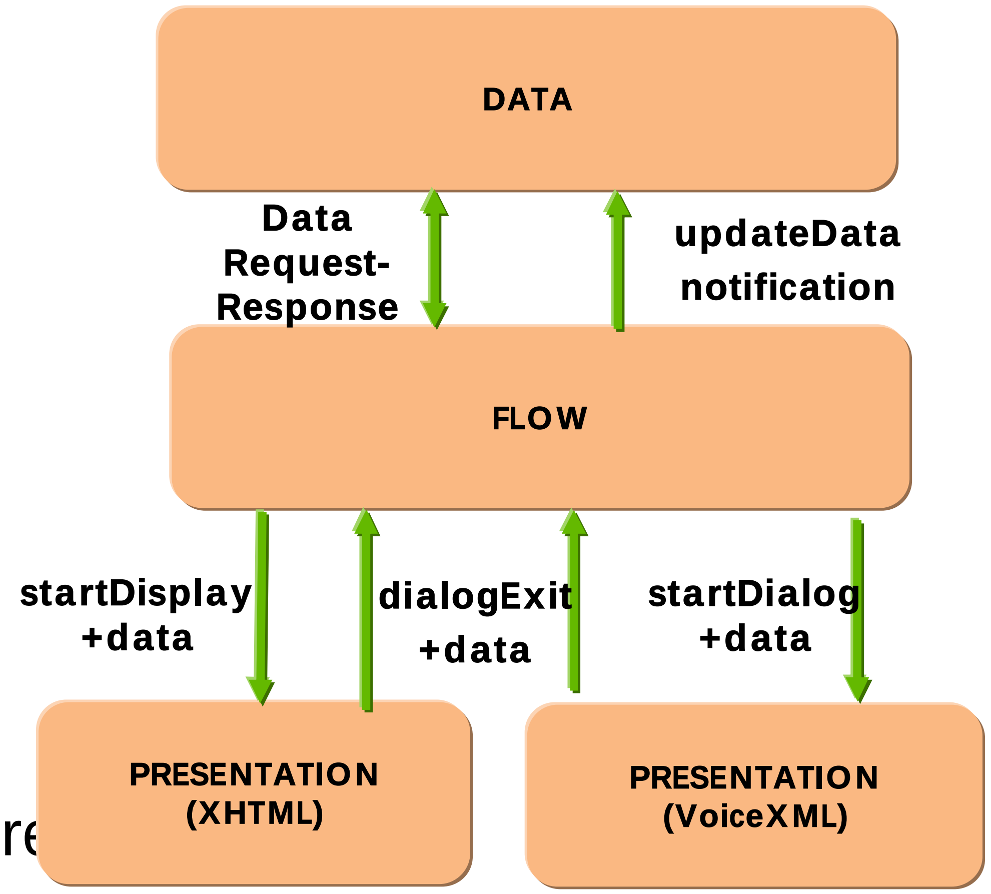
- Simplifies Code Reuse
  - Presentation layer not tangled with <goto> logic
  - Easy to insert and remove operations

- Improves Intelligibility
  - Flow description separate from presentation details

- Natural Extension to Multimodal Applications
  - Same flow layer, multiple presentation layers

Data Flow Presentation (DFP) *Multimodal* Framework

- W3C framework for modular multimodal application development
- Data: canonical data representation
- Flow: controls application flow and presentation coordination
- Presentation: *multiple* interactions with user
- Presentation components are mutually *independent*



Separation is the Key

- Multimodality: Enables Modality Plug-ins
  - In V2, need custom definition for each modality
    - e.g. H+V
- Code Reuse: Write Flow and Data
  - Reuse pluggable presentation components
- Separation Simplifies Complex Applications
  - Real world IVR applications are very complex
- Simple Applications May be Presentation Only

Agenda

- Overview
- DFP architecture
- **Overview of Flow/SCXML**
- Overview of Data Model
- VoiceXML 3 process
- Modularization
- New media features
- Recording and media control
- EMMA

Flow Layer

- Represents Application Logic
  - Does Not Interact with User
- Removes Control from Presentation Layer
- Various Languages Possible
  - CCXML, C, SCXML

SCXML

- Designed as Dialog Flow Language

- A State Machine Language
  - Based on Harel State Charts
  - A few dialog-specific extensions

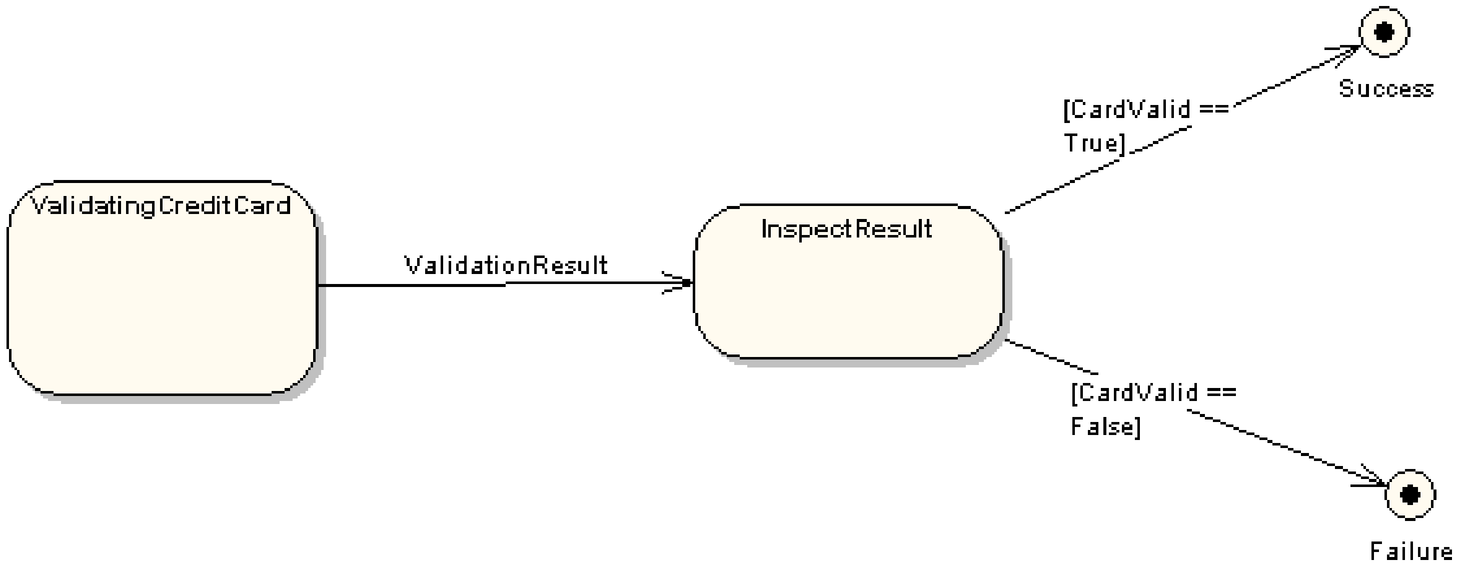
- Powerful, Compact Control Abstractions

State and Transition

- State Represents Status of System
  - Getting Order, Getting Credit Card, Validating Card

- Events are What Happens
  - Order Complete, Card Complete

Transitions Move Between States
  - Triggered by Events and Conditions
  - Getting Order to Getting CreditCard on OrderComplete


ValidatingCreditCard
ValidationResult
InspectResult
[CardValid == True]
Success
[CardValid == False]
Failure

```
<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” / >
```

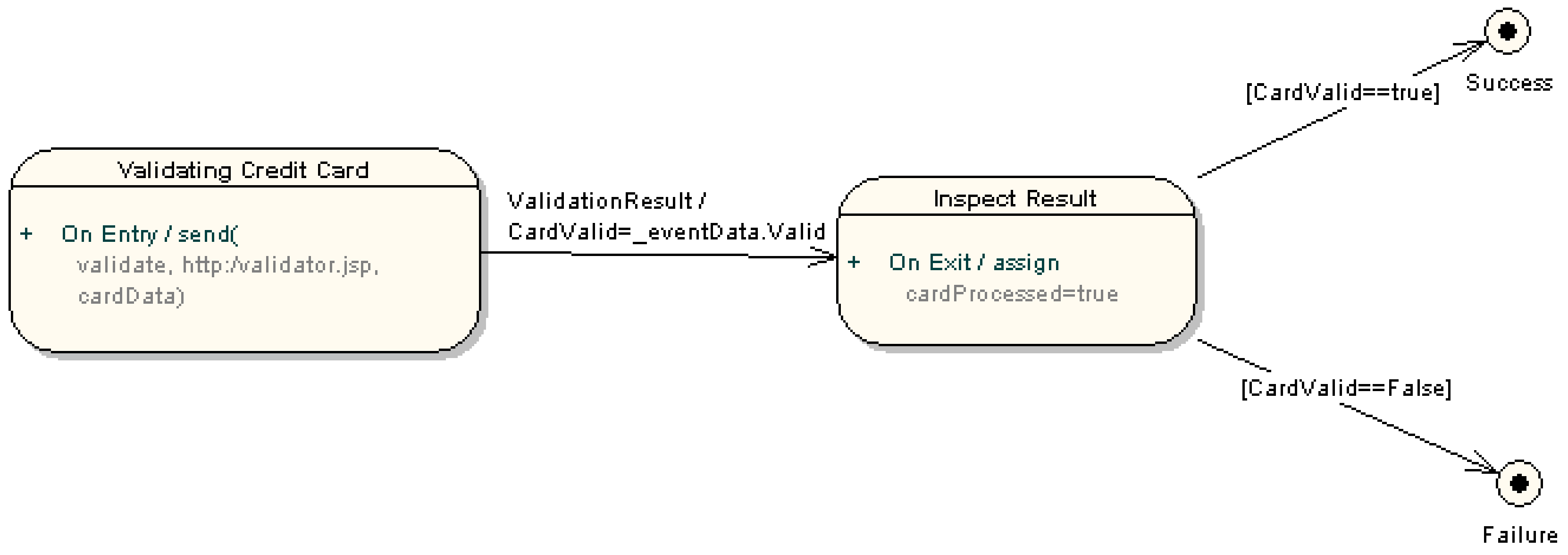
Executable Content

- Platform Operations
  - In <onentry>, <onexit> and <transition>

- Basic Operations
  - <send> asynchronous message to arbitrary URL
  - <assign> update data model
    - <datamodel> <data>

- Extensions
  - By profile or platform-specific

```
<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”/>
```


Validating Credit Card
+ On Entry / send(validate, http:/validator.jsp, cardData)
ValidationResult / CardValid=_eventData.Valid
Inspect Result
+ On Exit / assign cardProcessed=true
[CardValid==true]
Success
[CardValid==False]
Failure

Event Handlers

- Targetless Transitions
  - If <target> not specified, stay in same state
  - Executable content still invoked

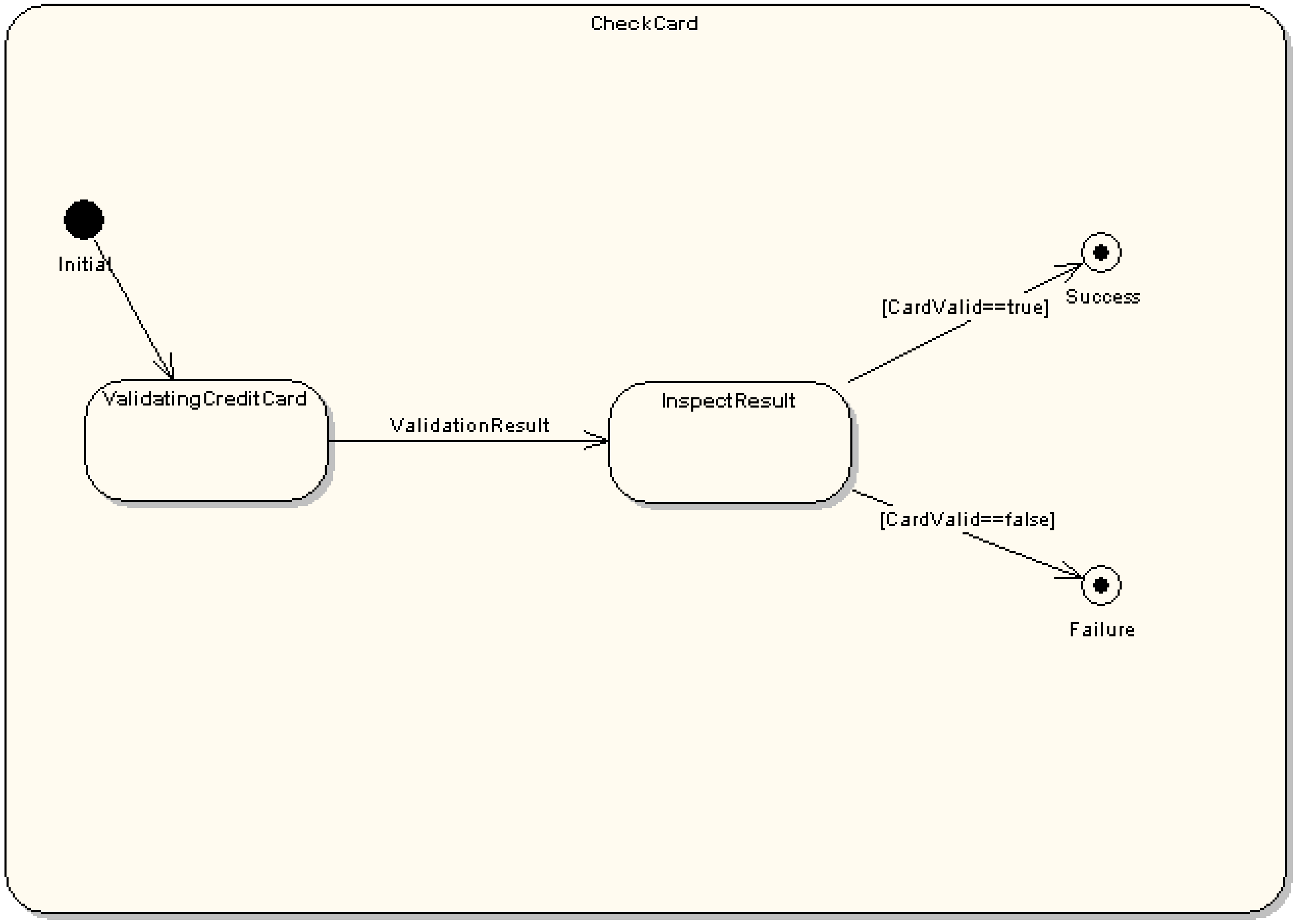
- In State S1, these are <u>not</u> the same:

  -
    ```
    <transition target="S1">
        <send…./>
     </transition>
    ```

  -
    ```
    <transition>
         <send…/>
       </transition>
    ```

Compound States

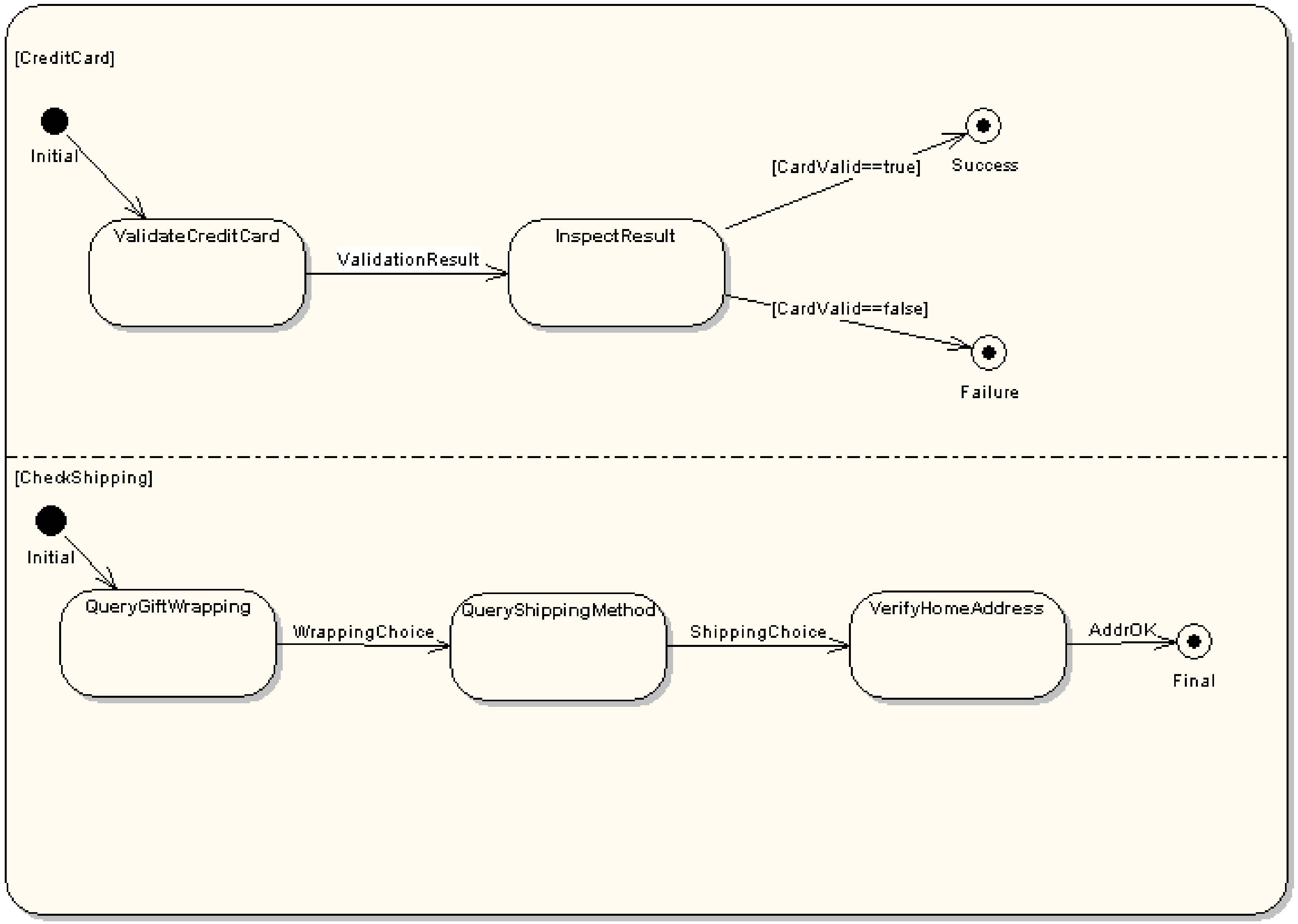
- Parent State Decomposed into Child States
- Represents Task Decomposition
  - Parent state always in a single child state
  - Transitions between child states show progress of the task
- <initial> tag gives default starting state


CheckCard
Initial
ValidatingCreditCard
ValidationResult
InspectResult
[CardValid==true]
Success
[CardValid==false]
Failure

```
<state id="CheckCard">
   <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"/>

   <initial id="ValidatingCreditCard"/>
</state>
```

Parallel States

- Parent State has Multiple Active Children
- Parallel Children Operate Independently
- Represents Fork/Join Logic
- Parent Finishes When Each Parallel Substate Reaches a Final State


[CreditCard]
Initial
ValidateCreditCard
ValidationResult
InspectResult
[CardValid==true]
Success
[CardValid==false]
Failure
[CheckShipping]
Initial
QueryGiftWrapping
WrappingChoice
QueryShippingMethod
ShippingChoice
VerifyHomeAddress
AddrOK
Final

```
<parallel id=“p1”>
<state id=“CheckCard”>
    …..
     </state>


<state id=“CheckShipping”>
    <state id=“QueryGiftWrapping”>
         <transition event=“WrappingChoice” target=“QueryShippingMethod”/>
    </state>

    <state id=“QueryShippingMethod”>
         <transition event=“ShippingChoice” target=“VerifyHomeAddress”/>
    </state>

   <state id=“VerifyHomeAddress”>
        <transition event=“AddrOK” target=“Final”/>
   </state>

  <state id=“Final” final=“true/>
</state>

</parallel>
```

<history> and <anchor>

- <history> Pseudo-State Allows Jumping Back
  - Re-enter complex state at point it was last exited
  - Pause and resume logic
  - Transition must specify ID of the <history> state
- <anchor> tag is More Dynamic
  - <transition anchor="foo" /> goes to last state to define "foo"
  - Do not need to know the name of the state
  - Can rollback data model as well
  - Designed for dialog applications
  - Not in Harel

```
<state name=“S1”>
   <anchor type=“foo” snapshot=“datamodel.subtree1” />
</state>

<state name=“S2”>
   <anchor type=“foo” />
</state>

…..
<state name=“somewhere_else”>
  <transition anchor=“foo” />
</state>
```

<invoke> and <finalize>

- <invoke> calls external service
- <finalize> preprocesses the results
  - Used to update the data model

- <state> with <invoke> represents activity of the external service

- Not in Harel

```
<state id=“ValidatingCreditCard”>

    <invoke type=“card-validator”>
       <param name=“cardData” expr=“cardData”/>
    </invoke>

   <finalize>
      <assign location=“CardValid” expr=“_eventData.valid?”/>
  </finalize>

    <transition event=“card-validator.Done” target=“InspectResult”/>

  </state>
```

SCXML Status

- Public Draft available:
  - http://www.w3.org/TR/2006/WD-scxml-20060124/
- New Draft this Fall
- Datamodel Less Stable than Rest
  - Executable Content May Also Change
- Core Harel Functionality Won’t Change

Agenda

- Overview
- DFP architecture
- Overview of Flow/SCXML
- **Overview of Data Model**
- VoiceXML 3 process
- Modularization
- New media features
- Recording and media control
- EMMA

Data Model

- Holds Application State
- Constitutes the “D” in the V3 DFP architecture
- Least designed part of V3 so far

Data Model – Some Requirements

- Accessible to presentation and flow layers in DFP
  - Allow for manipulation of application state
- Semantics & access mechanisms that can be used for:
  - V3 documents
  - SCXML documents
  - Mash ups of various namespaces including speech
- Allow some form of validation

Possible Representations

- XML
  - Inline with other W3C technologies
    - XHTML, CDF
  - Satisfies requirements for consistent representation across multiple namespaces
  - Various access mechanisms: XPath, ECMAScript
- As per V2.1
  - Not interoperable with other Web technologies
  - Restricted access to the DOM (read-only)
- But in *Any* representation
  - Try to have consistent semantics across various Web Technologies

Possible Candidates

- XForms data model as it exists today
  - Usable
  - Semantics & lifecycle may not be optimal for V3 & SCXML
- XML data model as joint taskforce across interested W3C WGs
  - Come up with semantics & lifecycle
  - Use same data model across multiple namespaces ex: XHTML, V3, CDF, MMI

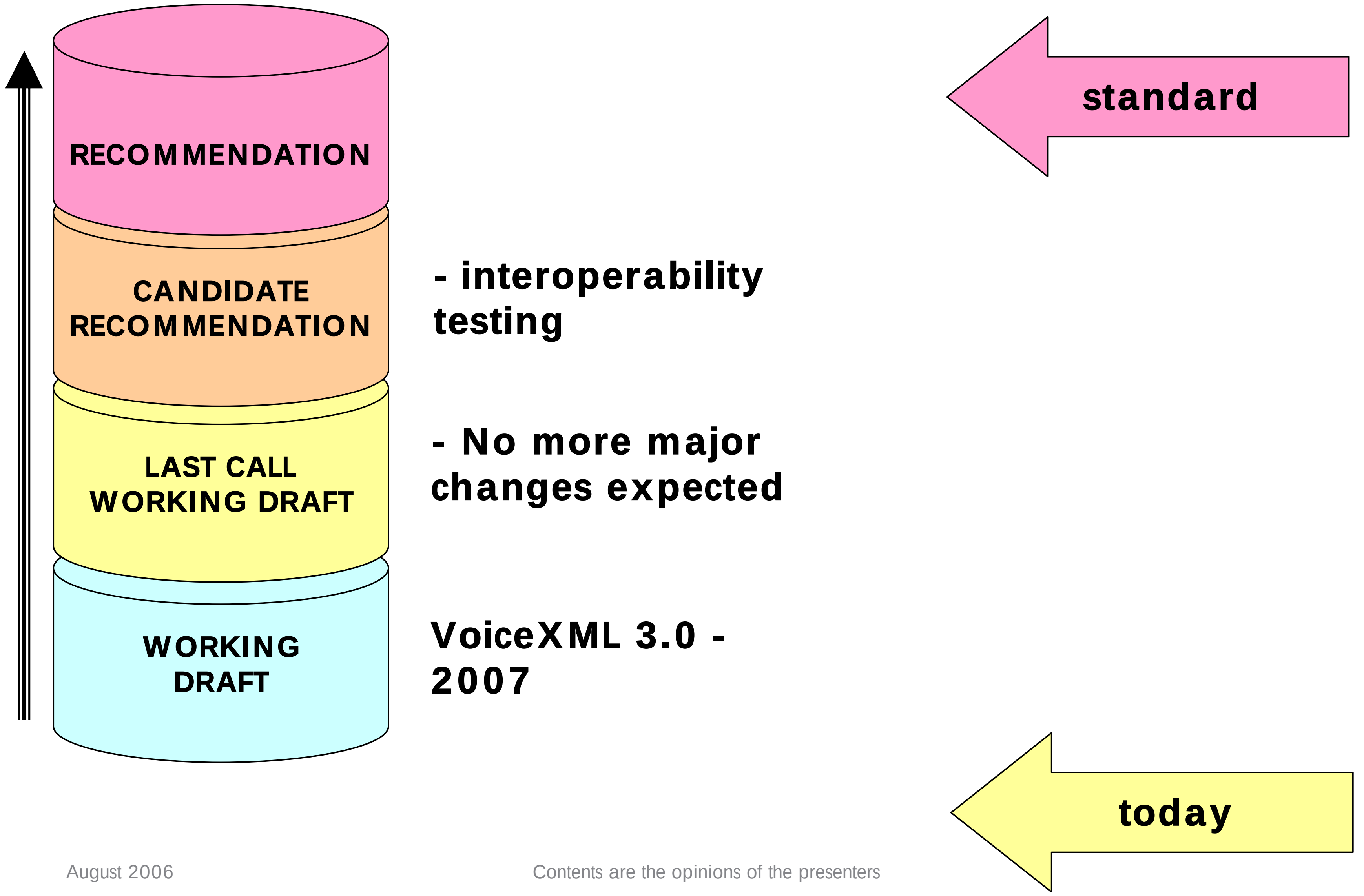
Agenda

- Overview
- DFP architecture
- Overview of Flow/SCXML
- Overview of Data Model
- **VoiceXML 3 process**
- Modularization
- New media features
- Recording and media control
- EMMA

VoiceXML 3.0 Process

- VoiceXML 3.0 is being developed by the W3C Voice Browser Working Group (VBWG)
  - http://www.w3.org/Voice
  - 3-4 F2F meetings per year, weekly teleconferences, …
- VBWG is one of the largest W3C groups
  - 95 group participants, 38 organizations
  - Join W3C to get more involved!
- VoiceXML 3.0 is being developed under VBWG charter and the W3C Patent Policy
- Public information and comments on VBWG activities, including VoiceXML 3.0 through mail list
  - www-voice@w3.org

W3C Standard Stages



- interoperability testing

- No more major changes expected

VoiceXML 3.0 - 2007

VoiceXML 3.0 Change Requests

- As with VoiceXML 2.0, a list of Change Requests (CRs) is maintained
  - Unresolved VoiceXML 2.0/2.1 issues
  - New uses, new requirements
  - Alignment with other W3C specifications
  - Etc
- Current CR status
  - 14 incoming
  - 26 pending
  - 66 active
  - 25 deferred

Issues actively being addressed

- Language modularization:
  - Language organized as set of modules
  - Profiles: core, multimodal, media server, …
- Functionality improvements:
  - Media VCR control: seek, pause / resume, volume / speed control
  - Speaker verification
  - Record with HTTP PUT upload
  - External events: send / receive info from external components
  - Extended media support (audio, video, …)
  - Dynamic <property>

Agenda

Why Modularize?

- VoiceXML is the preeminent language for speech applications but…
- Emerging markets and applications often require only subsets of VXML
  - Ex: embedding of VXML elements in a future multimodal language
- Breaks language into well defined ‘bite-sized’ chunks
  - Ex: call transfer could be moved into separate module
- Enables easy extensibility without breaking backward compatibility
  - Ex: new speaker verification module
- Promotes reuse of both authoring techniques and implementations to address unexpected niches

Barriers to Modularization

- Requirements have evolved and the next generation specification must address broader interests
  - VXML 1.0 was published in May 2000
  - Much experience gained in six years
- Conformance becomes more complex
  - VXML without telephony
  - VXML without ECMAScript
  - VXML using only menu and static prompts
- Being careful and ‘doing it right’ takes time

Examples of Modularized Specs

XHTML 1.1:

- http://www.w3.org/TR/xhtml-modularization/
- Element based modules: tables
- Attribute based modules: text directionality

SMIL 2.1:

- http://www.w3.org/TR/SMIL2/
- Language divided into several modules
- Individual modules may depend on other modules
- Profiles defined from collections of modules and conformance criteria specified

Roadmap to Modularization

- Clearly define the underlaying semantic model for VoiceXML 2.x
- Determine where extensions for VoiceXML 3.0 are necessary.
- Build syntax (elements and attributes) described in terms of semantic model
- Assemble authoring profiles to address key constituencies

Semantic Model

- Speech recognition is described in VoiceXML 2.0 in several sections
  - 3.1, 2.2, 2.3, 5.2, 6.3, and elsewhere

- A semantic model would explain, *in one place,* the underlying
  - Timings (e.g. input timeout, interdigit timeouts)
  - Events (e.g. start of input, partial result, partial match, complete state reached)
  - Result reporting
  - Etc.

Syntax and Profiles

- Modules are defined as blocks of syntax (elements and attributes) which expose portions of the logical functionality contained in the semantic model.
  - <vxml:field> might support only noinput and nomatch events
  - <advanced:recognition> might expose other functionality

- Profiles are assembled from groups of modules and may impose additional restrictions.
  - One profile might be build around <vxml:field> to provide substantial compatibility with VoiceXML 2.1.
  - A second profile might be based upon other elements.

Enabling The Future

Modularization and a clear semantic model result in this recipe for adding functionality:

1. Update the semantic model
2. Define a new or update an existing module
3. Define a new and / or update an existing profile
4. Define test assertions for new functionality

Which Profiles Matter To You?

W3C process encourages community feedback. Which of these profiles would you find beneficial?

- VoiceXML 2.x with additional functionality
- Multimodal language based in part on VoiceXML syntax
- Simplified media server
- Others?

Agenda

- Overview
- DFP architecture
- Overview of Flow/SCXML
- Overview of Data Model
- VoiceXML 3 process
- Modularization
- **New media features**
- Recording and media control
- EMMA

VoiceXML 3.0 New Media Features

- Motivation
- Information about connection’s media
- <media> element
  - Advanced control attributes
- Simultaneous media output
- <record> enhancements
  - VAD and output properties
  - Play prompt while recording
- <transfer> enhancements

Motivation

- Requests for more control over media playback; e.g. play audio file from offset
- VoiceXML being used for multimedia playback/record
  - 3G networks and VOIP/SIP: beyond audio
  - Container file formats (3gpp/mp4/avi) with audio and video tracks
  - VoiceXML 2.0 today:
    - <audio src="video/kylie.mp4"/>
    - <record name="msg" type="video/3gpp">
- Innovative applications demand more capability
  - playing a prompt while recording needed for telephony karaoke applications

Connection Media information

- **session.connection** is enhanced with information about the connection's media
- **media**: object describes the defined or negotiated media between the local and remote devices
  - **type**: type of the media associated with the stream (mandatory)
  - **direction**: directionality of the media (default: sendrecv)
  - **format**: details about the format, include a name

Connection media examples

- Connection with bi-directional audio:
  - session.connection.media[0].type = audio
  - session.connection.media[0].direction = sendrecv

- Connection with bi-directional Mulaw audio and H.263 video:
  - session.connection.media[0].type = audio
  - session.connection.media[0].direction = sendrecv
  - session.connection.media[0].format[0].name = audio/PCMU
  - session.connection.media[0].format[0].rate = 8000
  - session.connection.media[1].type = video
  - session.connection.media[1].direction = sendrecv
  - session.connection.media[1].format[1].name = H263-1998
  - session.connection.media[1].format[1].QCIF =1

Connection media issues

- With some protocols, the connection may be re-negotiated during the session resulting in a change in the connection's media properties
  - for example, SIP re-INVITE
- In this case, media values need to be updated *during* a VoiceXML session
- To alert the developer to this update, we probably need to inject an event into the VoiceXML session
  - the nature of this event and how injection takes place still to be defined

<media> element

- New <media> element which provides a general container for playing media
  - <audio> not intuitive for non-audio media
  - Want to avoid new elements for each new media
  - Semantics: extends SMIL 2.1<ref> element
- <media> basic attributes:
  - **src**, **srcexpr**: URI of media to play
  - **fetchtimeout**, **fetchhint**, **maxage**, **maxstale**: control fetching
  - **type**: preferred media type (NB: web server is authoritative)
  - **cond**: boolean determining whether media is queued

<media> element

- <media> has a content model which allows
  - <media>: fallback
  - <desc>: textual description
  - <property>: control
  - non-VoiceXML namespace elements (e.g. SSML)
  - CDATA for ‘simple’ inline SSML
- <media> is a child of <prompt>
- <media> related property: **outputmodes**
  - Value is a space-separated list of media; e.g. “audio video”
  - Default depends on connection media values
  - If outputmodes references a media value not supported by the connection or platform, then value ignored

<media> examples

```
<media type="audio/x-wav“ src="icon.wav"/>

<media type="video/3gpp" src="http://server/avatar.3gp"/>

<media type="video/3gpp" src="http://server/avatar.3gp">
  <property name=“outputmodes” value=“video”/>
</media>

<media type="application/ssml+xml"
       src="http://www.example.com/resource.ssml"/>

<media type="application/ssml+xml">
   <ssml:speak version="1.0"
       xmlns:ssml="http://www.w3.org/2001/10/synthesis">
     <ssml:emphasis>Hello there!</ssml:emphasis>
     <ssml:audio src="yourbalance.wav"/>
   </ssml:speak>
</media>
```

<media> examples

```
<media type="application/ssml+xml">
  Simple text SSML.
</media>

<media type="application/ssml+xml">
  <ssml:speak version="1.0" xmlns:ssml="…">
    <ssml:emphasis>Hello there <value expr="username"/>!
    </ssml:emphasis>
  </ssml:speak>
</media>

<media type="audio/x-wav" src="ichbineinberliner.wav">
   <media type="application/ssml+xml">
     <ssml:speak version="1.0" xmlns:ssml="…">
       Ich bin ein Berliner.
     </ssml:speak>
   </media>
   <desc xml:lang="en-gb">
    Kennedy's famous German language gaffe
   </desc>
</media>
```

Advanced <media> control attributes

- Attributes derived from SMIL 2.1
  - Semantics: SMIL 2.1 Timing Modules and others
- Clipping Module:
  - **clipBegin**: offset to play from
  - **clipEnd**: offset to play to
  - **`<media src="audio.wav“ clipBegin="2s“ clipEnd="9s"/>`**
  - *(starts 2s into media and lasts 7s)*
- Duration Timing Module:
  - **dur**: simple duration of media
  - **begin**: when media becomes active
  - **end**: when media no longer active
  - **`<media src="audio.wav" begin=“2s“ dur=“9s“/>`**
  - *(2s of silence then plays for 9s)*
  - **`<media src="audio.wav" clipBegin=“2s” clipEnd=“9s” dur=“9s“/>`**
  - *(starts 2s into media and last 9s – last 2s are silence)*

Advanced <media> control attributes

- Minmax Timing Module:
  - **min**: minimum time for active duration
  - **max**: maximum time for active duration
  - **`<media src=“holidays.3gp” min=“10s”/>`**
  - *(if media shorter than 10s , then last video frame frozen and audio is silent)*
- Repeat Timing Module:
  - **repeatCount**: number of iterations of simple duration
  - **repeatDuration**: total duration for repeat
  - **`<media src="music.mp3" dur="2.5s" repeatCount=“2"/>`**
  - *(2.5s simple duration repeated twice)*
  - **`<media src="music.mp3" dur="2.5s" repeatDur="7.5s"/>`**
  - *(media played for 7.5s – 3 x simple duration)*

<media> Issues

- Conformance requirements are not likely to change
  - G.711 only as required media types
  - Platform may support other media types – if not, <media> ignored (should media failures throw an error?)
- Relationship with <audio>
  - <audio> enhanced with “cond” and “type” attributes
  - Issue whether ‘type’ attribute handled in SSML 1.1
- Relationship between <media> and SSML elements
  - <prompt> in VoiceXML 2.1 is effectively SSML <speak>
  - How should the content model of <speak> change to accommodate <media>?
    - Separate models: a prompt is either <media> only or SSML element
    - Integrated model: mix and match <media> with SSML elements?
  - Currently investigating issue …

Simultaneous media output

- Container formats – mp4, 3gp – tightly synchronize playing of media streams
- Use cases with loose synchronization between separate media:
  - Videomail: add video avatar to an audio message
  - Enterprise: add TTS voiceover to security camera video
  - Education: add commentary in student's language to video showing medical procedure
- The intention is NOT to provide full multimodal output, but to cover basic use cases where audio or TTS output from one resource can be complemented with output from another resource as permitted by the connection

<par> element

- Play <media> in parallel
- Semantics: SMIL 2.1 <par> time container
- **endsync** attribute indicates whether <par> ends when first or last media ends

- Play SSML with video

```
<par endsync=“first”>
 <media type=“application/ssml+xml" src="intro.ssml"/>
 <media type="video/3gpp" src=“camera.3gp"/>
</par>
```

<seq> element

- Play <media> in sequence
- Semantics: SMIL 2.1 <seq> time container
- Child of <par> element

- Plays audio sequence (TTS then audio file) with video

```
<par>
  <seq>
    <media type=“application/ssml+xml“ src="intro.ssml"/>
    <media type="audio/x-wav“ src="main_commentary.wav"/>
  </seq>
  <media type="video/3gpp" src="avatar.3gp"/>
</par>
```

<par>/<seq> issues

- Definitely not a required feature – may be in module for ‘advanced’ media profile
- Investigate whether advanced media attributes (dur, repeatCount, etc) should be added
- Interaction with VoiceXML 2.1 mark and marktime
- Interaction with VCR media control; e.g. how to restart – or pause – parallel media
- …

<record> enhancements

- <record> already permits non-audio media to be record
  - **type**: if platform (or connection) doesn't support specified media-type, then error.unsupported.format event thrown
- New properties to control Voice Activity Detection (VAD) when recording
  - **record.vad.initial**: whether VAD can be used to initiate recording (*on* / off)
  - **record.vad.final**: whether VAD can be used to terminate recording (*on* / off)
- New **recordmodes** property to specify media modes used for recording:
  - Value is a space-separated list of media; e.g. "audio video"
  - Default depends on connection media values
  - If recordmodes references a media value not supported by the connection or platform, then value ignored

<record> examples

Video only recording with no VAD initially or finally:

```
<record name=“msg” type=“video/3gpp">
  <property name=“record.vad.initial” value=“off”/>
  <property name=“record.vad.final” value=“off”/>
  <property name=“recordmodes” value=“video”/>
</record>
```

Playing prompts during <record>

- Uses cases include:
  - audio: 'warning' feedback to indicate that recording is about to terminate. Also: ‘fun’ applications like karaoke.
  - video: present visual countdown to indicate recording duration
- <prompt> child of <record> with attribute to indicate that it is played at onset of recording
  - Attribute is **`playonrecord`** – but open to alternative suggestions (“background”?)!

```
<record name=“bbc_shout” type="video/3gpp" maxtime="30s">
  <prompt>
   <media src="prompt.3gp"/>
  </prompt>
  <prompt playonrecord="true">
    <media src=“countdown.3gp"/>
  </prompt>
</record>
```

Playing prompts during <record>

- Allow complex mixing of audio, TTS and video

```
<record name=“video_karaoke” type="video/3gpp" maxtime="30s">
    <prompt>
       <media src="prompt.3gp"/>
     </prompt>
     <prompt playonrecord="true">
       <par>
         <seq>
            <media type="application/ssml+xml">
             Start singing along John.
            </media>
•            <media type="audio/x-wav“ src=“track.wav"/>
            <media type="application/ssml+xml">
             John – you really can sing!
            </media>
         </seq>
         <media src=“track_lyrics_with_pointer.3gp"/>
       </par>
     </prompt>
</record>
```

<transfer> enhancements

- New property to control the media types used for transfer: **transfermodes**
  - Value is a space-separated list of media; e.g. “audio video”
  - Default depends on connection media values
  - If transfermodes references a media value not supported by the connections or platform, then value ignored
- If the current session is audio video, then we can force an outbound audio only call:

```
<form>
  <property name="transfermodes" value="audio"/>
  <transfer name="mycall" dest="tel:+1-555-123-4567“
       bridge="true"/>
</form>
```

Agenda

- Overview
- DFP architecture
- Overview of Flow/SCXML
- Overview of Data Model
- VoiceXML 3 process
- Modularization
- New media features
- **Recording and media control**
- EMMA

Media Enhancements

- Media Controls
  - VCR Controls
  - Speed Control
  - Volume Control
- Record Enhancements
  - Record Destination
  - Progressive Upload

VCR Controls for VoiceXML Media Playback

- Provides the ability to start media playback at a specific offset
  - Fast Forward
  - Rewind
  - Pause
  - Resume

- New element introduced: <reposition>

<reposition> element

- Element: <reposition>

- Attributes
  - time / timeexpr
    - Time offset into media stream at which to start media playback
  - mark / markexpr
    - <mark> in media stream at which media playback should begin
  - If both mark and time are specified, time offset is applied from the specified mark

<reposition> element (continued)

- <reposition> may be used anywhere a <prompt> element is used in a VoiceXML document
- <reposition> specified with no prompts will result in an implicit re-queuing of all prompts queued prior to the last waiting state of the FIA and the reposition will be applied to these prompts
- <reposition> preceded by <prompt> elements will result in a reposition in the specified prompts

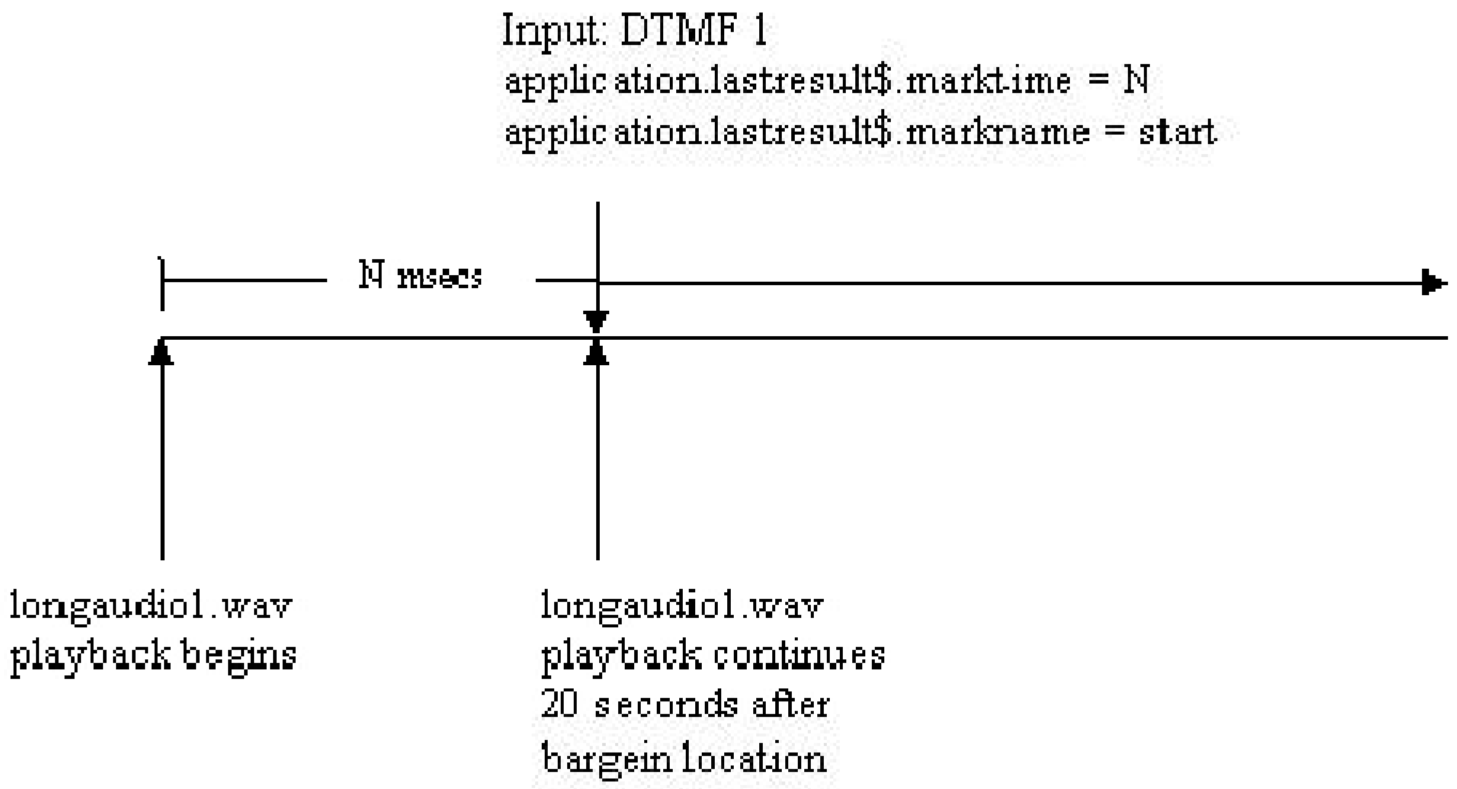
VCR Controls (continued)

- Makes use of VoiceXML 2.1 shadow variables:
  - application.lastresult$.marktime
  - application.lastresult$.markname
- Designed to have no impact on VoiceXML 2.0 FIA prompt queuing behavior

<reposition> example

```
<field>
   <grammar mode=”dtmf”> 1 </grammar>
   <prompt>
            <mark name=”start”/>
                        <audio src=”longaudio1.wav”/>
            <mark name=”end”/>
   </prompt>
   <filled>
            <!—fast forward 20 seconds from where prompt was   interrupted -->
            <reposition timeexpr=”application.lastresult$.marktime + 20000
                        + ‘ms’”/>
   </filled>
</field>
```

Reposition Timing



Speed and Volume Control within a VoiceXML document

- Provides the ability to adjust the speed or volume of a media stream using DTMF or speech input
- New VoiceXML properties added
  - playbackspeed
  - playbackvolume
- Volume and Speed property values are consistent with SSML <prosody> ‘volume’ and ‘rate’ attributes

Speed and Volume Control (cont.)

- Relies on VoiceXML 3.0 <reposition> element to restart media playback with adjusted speed or volume at a specific offset

  E.g. Time of last barge-in (application.lastresult$.marktime)

- Makes use of VoiceXML 3.0 ‘expr’ attribute on <property>

Speed / Volume Control Example

```
<form id="media_ctrl">
   <var name="speed_adj“ expr=“0%”/>
   <var name="vol_adj“ expr=“0dB”/>
   <link event="faster"> faster </link>
   <link event="louder"> louder </link>
   <catch event="faster">
            <assign name="speed_adj" expr="'+50%'"/>
            <!-- Implicitly requeue all previously queued prompts -->
            <reposition timeexpr="application.lastresult$.marktime + 'ms'"/>
   </catch>
   <catch event="louder">
            <assign name="vol_adj" expr="'+3dB'"/>
            <!-- Implicitly requeue all previously queued prompts -->
            <reposition timeexpr="application.lastresult$.marktime + 'ms'"/>
   </catch>
   <field name="cmd">
            <property name="playbackvolume" expr="vol_adj"/>
            <property name="playbackspeed" expr="speed_adj"/>
            <grammar src=“gram.grxml" type="application/grammar+xml"/>
            <prompt>
            At any time during this prompt you may say, say LOUDER to raise the volume by
            3dB or FASTER to increase the prompt speed by 50%.
            </prompt>
   </field>
</form>
```

Record Enhancements

- Provide the ability to specify destination of recording and upload recorded audio to an app server while recording
  - New attributes on <record>
    - dest/destexpr – HTTP location to which recording should be written
    - method – HTTP method to be used to submit recording to specified location
  - Must specify HTTP destination to guarantee security of VoiceXML Platform

Record Enhancements (continued)

- method may be ‘post’ or ‘put’
  - ‘put’ will result in recorded data being submitted to destination while record is in progress
    - New File Created: 201 Created
    - Existing File Modified: 204 No Content
    - Failure:
      - 403 Forbidden
      - 404 Not Found
  - ‘post’ will result in recorded data being submitted after recording is complete without replying on VoiceXML <submit> element

Record Enhancements (continued)

- New shadow variables
  - $name.total_size – total size of recording after this recording is complete
    - May differ from ‘size’ shadow variable if destination previously contained data
  - $name.dest_status_code – HTTP error code returned, if any
  - $name.dest_status_description – HTTP error description, if any

<record> Example

```
<form>
   <record name=“recording“ method=“put”
               dest=“http://yourServer/recordingsGoHere”>
       <filled>
         <if cond=“recording$.dest_status_code != ‘201’ &&
                    recording$.dest_status_code != ‘204’“>
               <audio src=“recordingfailed.wav”/>
               <log expr=“recording$.dest_status_description”/>
         <else/>
               <audio src=“recordingcomplete.wav”/>
               <log expr=“recording$.dest_status_description”/>
         </if>
       </filled>
  </record>
</form>
```

Agenda

What is EMMA?

- EMMA: Extensible Multi-Modal Annotation language
  - Standard markup for describing the semantic interpretation of user input
  - http://www.w3.org/TR/emma/
  - At Last Call Working Draft; moving toward Candidate Recommendation
  - Technical contributions from Avaya, AT&T, IBM, Nuance, Volantis, and others

- History:
  - Successor to NLSML from VBWG
    - First draft in November 2000
    - Unpublished draft from May 2001 used by SALT and many MRCP v1 implementations.
  - Moved to MMI and changed editors
    - First draft in August 2003
    - 5 published drafts later, implementation reports are proceeding
  - Publication as a full Recommendation after seven years?

Degenerate Cases

The ‘No Input’ case

```
<?xml version="1.0" encoding="UTF-8"?>
<emma:emma version="1.0" xmlns:emma:="http://www.w3.org/2003/04/emma">
    <emma:interpretation id="int1" emma:no-input="true“ emma:function="dialog"/>
</emma:emma>
```

One possible ‘No Match’ case

```
<?xml version="1.0" encoding="UTF-8"?>
<emma:emma version="1.0" xmlns:emma:="http://www.w3.org/2003/04/emma">
    <emma:interpretation id="int1" emma:uninterpreted="true" emma:mode="dtmf“
                emma:tokens="3" emma:function="dialog"/>
</emma:emma>
```

Trivial Interpreted Input

Simple Value from DTMF Grammar

```
<emma:emma version="1.0“ xmlns:emma:="http://www.w3.org/2003/04/emma“
    xmlns="http://www.example.com/example">


    <emma:grammar id="grammar1“ href="http://www.example.com/simple_dtmf.grxml"/>
    <emma:interpretation id="int1" emma:mode="dtmf" emma:tokens="2“
              emma:grammar-ref="grammar1" emma:function="dialog">
       <emma:literal>no</emma:literal>
    </emma:interpretation>


</emma:emma>
```

More Typical EMMA Document

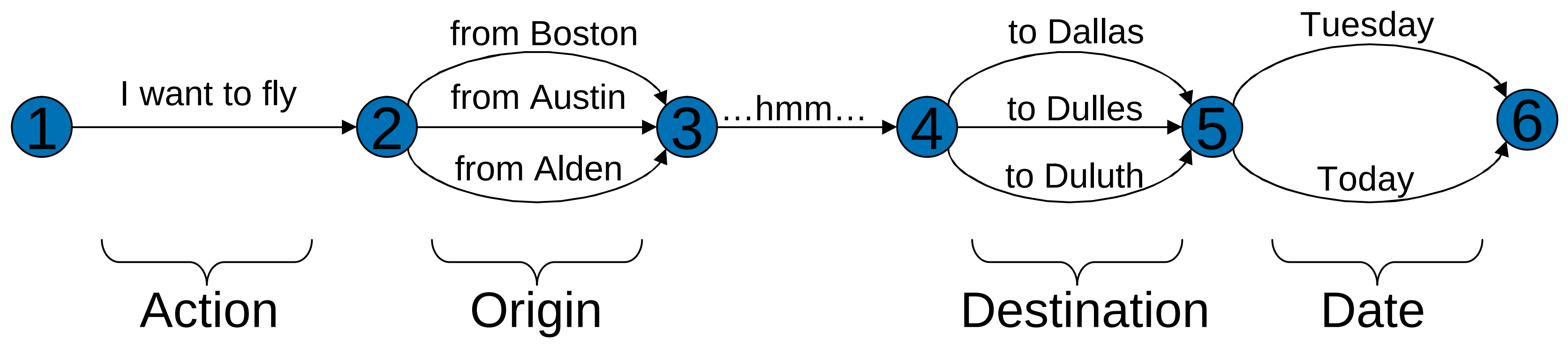
Structured Result from Speech Grammar

```
<?xml version="1.0"?>
<emma:emma version="1.0" xmlns:emma:="http://www.w3.org/2003/04/emma"
    xmlns="http://www.example.com/example">

    <emma:grammar id="grammar1" href="http://www.example.com/simple_speech.grxml"/> <emma:one-of
    id="nbest" emma:grammar-ref="grammar1" emma:mode="speech"
              emma:signal="http://www.example.com/recoding/1234.ulaw"
              emma:media-type="audio/x-ulaw; rate:8000" emma:duration="375"
              emma:function="dialog">
         <emma:interpretation id="int1" emma:tokens="today" emma:confidence="0.821">
              <day>1</day>
              <month>9</month>
              <year>2006</year>
         </emma:interpretation>
         <emma:interpretation id="int2" emma:tokens="Tuesday" emma:confidence="0.282">
              <day>5</day>
              <month>9</month>
              <year>2006</year>
         </emma:interpretation>
    </emma:one-of>


</emma:emma>
```

Basic Results in Review

- EMMA supports several basic annotations
  - ‘no-input’ and ‘uninterpreted’ typically correspond to noinput and nomatch in VoiceXML
  - ‘tokens’ contains the raw input
- Various categories of results exist
  - Simple values contained within <literal>
  - XML results permitted within an <interpretation>
  - N-best lists build from several interpretations within a <one-of> element
  - Composite results using <sequence> or <group>

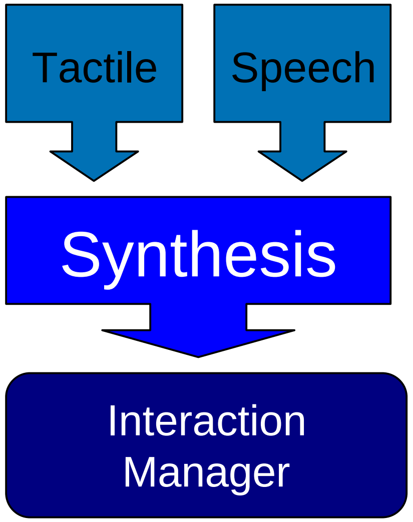
Lattice Results



```
<emma:lattice initial="1" final="6">
   <emma:arc from="1" to="2" emma:confidence="1.0"><action>airline</action></emma:arc>
   <emma:arc from="2" to="3" emma:confidence="0.7"><origin>BOS</origin></emma:arc>
   <emma:arc from="2" to="3" emma:confidence="0.5"><origin>AUS</origin></emma:arc>
   <emma:arc from="2" to="3" emma:confidence="0.1"><origin>IL05</origin></emma:arc>
   <emma:arc from="3" to="4"/>
   <emma:arc from="4" to="5" emma:confidence="0.6"><dest>DFW</dest></emma:arc>
   <emma:arc from="4" to="5" emma:confidence="0.2"><dest>IUD</dest></emma:arc>
   <emma:arc from="4" to="5" emma:confidence="0.1"><dest>DLH</dest></emma:arc>
   <emma:arc from="5" to="6" emma:confidence="0.4">
               <date><month>8</month><day>15</day><year>2006</year></date></emma:arc>
   <emma:arc from="5" to="6" emma:confidence="0.3">
               <date><month>8</month><day>10</day><year>2006</year></date></emma:arc>
</emma:lattice>
```

Multimodal Example

"How do I drive to here from here?"



```
<emma:derivation>
   <emma:group>
      <emma:interpretation emma:mode="speech">
         <mode>auto</mode>
         <origin emma:start="350" emma:end="400"/>
         <destination emma:start="500" emma:end="550"/>
      </emma:interpretation>
      <emma:sequence>
         <emma:interpretation emma:mode="tactile" emma:start="360">
             <x>120</x> <y>253</y>
         </emma:interpretation>
         <emma:interpretation emma:mode="tactile" emma:start="540">
             <x>262</x> <y>189</y>
         </emma:interpretation>
      </emma:sequence>
   </emma:group>
</emma:derivation>
<emma:interpretation>
   <from><addr>48 Winter St.</addr><city>Boston</city></from>
   <to><addr>234 Berkeley St.</addr><city>Boston</city></to>
</emma:interpretation>
```

Tools for Managing Complexity

- Rich inputs may produce complex results
- Unexpected inputs may produce invalid results that are difficult to detect and debug

- EMMA supports schema validation of results
  - Interpretations may contain elements from one or more namespaces
  - Grammar authors can define what constitutes legal results: e.g. format, value restrictions, defaults
  - EMMA consumers can test results against the schema and report errors

- EMMA derivation chains can help with debugging

EMMA in VoiceXML 3.0

EMMA provides a rich set of annotations for describing complex results.

Assertion: Applications should be able to access rich information from EMMA results when available.

The exact mechanism in V3 depends on the datamodel and the preferred XML access method used (e.g. XPath, E4X, DOM, …)

Agenda

- Overview
- DFP architecture
- Overview of Flow/SCXML
- Overview of Data Model
- VoiceXML 3 process
- Modularization
- New media features
- Recording and media control
- EMMA