Architecture Designs
as
Domain-Specific Languages

David Wile
wile@isi.edu
11/99

PowerPoint Design Editor

- Allows designs to be created by a domain engineer using PowerPoint
  - Semantic interpretations of diagram provided
  - In an architecture-style-specific jargon
- The domain architect designs
  - Style specification (using PPT DE itself)
  - Problem specific analyzers, simulators, semantic actions
Overall Architecture

Registry Details

- Style design registered with its name
- Analyzers
  - COM server for analyzer registered with the DLL/EXE that implements it
  - Domain + analysis registered with analyzer
Analysis results: Analyzer side

- ILatency interface class, in LatencyAnalyzerModel:
  - Public Sub initialize(ByVal P As Proxy)
  - Event Highlight(ByVal P As Proxy)

- Latency class in analyzer has methods containing
  - implements LatencyAnalyzerModel.ILatency
  - Public Sub LatencyAnalyzerModel.Initialize(ByVal P As Proxy) ...
  - Raise Event 'Highlight P' 'used within analyzer

Analysis results: Exchanger side

- Analysis object will be created by Style-specific Exchanger on demand
  - will cause appropriate analyzer (e.g. Latency) to be loaded
  - inside of Exchanger
  - With Events LA As Latency
  - Design Editor sends request for Latency analysis
    - set LA = New Latency
    - LA.initialize p1, p2, ..., pn
  - subroutines responding to events from the analyzer, e.g. Highlight, have prefix of "with events"
  - Public Sub LA_Highlight(ByVal P As Proxy)
  - Turn into domain-independent event by
    - Raise Event HighlightObject(P.idDE) ' in Design Editor Server

- Problem with multiple analyses simultaneously
Analysis invocation: Exchanger side

- Application in I<style>Model
  - Event addLatencyofToSatellite(ByVal sat As Satellite, ByVal lat As Float)
- Application in <style>
  - implements I<style>Model.Application
  - Public Sub <style>Model.initialize()
  - Raise Event addLatencyofToSatellite used inside Exchanger
  - in response to AnalysisActivate method from DE,
    - Dim an As Object
    - Set an = createObject(analyzerSuiteFor(style, analysis) & "." & analysis)
    - an.initialize(application)

Analysis invocation: Analyzer side

- Class <Analyzer>
  - Private WithEvents CA as Application
  - Application object will be passed to analyzer upon creation of Analyzer object
    - Public Sub I<style>Model_initialize(ByVal ap As Application, ByVal Parameters() As String)
      - Set currentApplication = ap
      - End Sub
  - Subroutines respond to events:
    - Public Sub CA_addLatencyofToSatellite(ByVal sat As Satellite, ByVal lat As Float)
An example: Survey Authoring

- Census surveys are administered on laptops using a so-called "survey instrument"
- Myriad questions are conditioned on previous answers
  - e.g. questions depending on whether you live in an apartment, house, trailer, etc.
  - Design of the instrument is very time consuming
- We want to provide an instrument creation tool, based on PowerPoint Design Editor
- Design a domain-specific language – style – to capture the decision process

Instrument Administration Architecture
Instrument Administration

- The Interviewer interviews...
- One or several Interviewees...
- Reading and recording information from and into Interview Forms.
- The particular forms presented are determined by the Instrument Control:
  - Its function is to update the Relational Database with answers from the Interview Forms;
  - Its progress is controlled by data from the Relational Database and by a database of Interview Flow structures (produced by the instrument Author - see next figure);
  - The Instrument Control customizes forms from the Form Template database to exhibit sensitivity to the context of the particular Interviewee.

Authoring Environment
Authoring Environment

- The instrument Author interacts with ISI's PowerPoint Design Editor to produce the Interview Flow Design.
  - This design is the basis for all logical instrument content and flow.
  - A Consistency Analyzer will determine inappropriate flows in the design (e.g., improper loops, control-to-data mismatch, etc.) and report them on the design diagram itself.
  - An Administration Simulator, exhibiting typical behavior, can be used for debugging.
  - The Interview Flow database will be generated by the Flow Generator to be used in the administration of the instrument.
- The instrument Author interacts with COTS database, e.g., Microsoft Access, to produce the Relational Database schema specification for the instrument.
- The Forms Author is responsible for the physical appearance of the information displayed during the instrument administration.
  - The Forms Author designs Templates that are specialized to the context of the interviewee during the instrument administration.
  - He or she will have a COTS Template Design environment tailored to the presentation medium to design the individual forms, e.g., PageMakerPro might be used for an HTML-based presentation.
  - In addition, some analysis support may be provided for the Forms Author.

(Potential) Forms Author Analysis Support

- The Browser Interface Generator will take interface specifications attached to the Interview Flow Design and make them visible to the Forms Author during the Template Design process.
  - These interfaces reside in the Interview Flow database as well, and are used during instrument administration for customization of forms.
- The IDOC Generator will generate Instrument Documentation that augments the Interview Flow Design.
  - This will comprise a set of indices not normally available from the PowerPoint Design Editor.
  - It may need to link to the Form Templates as well to produce some of the indices.
  - Extra information will also be superimposed on the diagram by Author request.
- The Form Analyzer will ensure the compatibility between the Form Templates and the Browser Interfaces used with them.
**Interview Control Flow**

- **Construct (Browser) Form:**
  - Use an SQL-like language to fill in fields of
  - The Browser Interface Language to produce customized questions needing answers
  - A Form Template from the library together with the constructed browser interface are used to compile an HTML or VB form
- The Browser is then invoked
- **Validate Input** is executed on the data returned
  - Through SQL queries on a hypothetical DB - the Relational Database extended with the returned data
  - Use same mechanism as for SKIP data
Interview Control Flow
(continued)

- Update Database with returned data
  - (Schema previously specified in Access)
  - Updates specified in SQL (or perhaps a language more akin to that used in the IDOC for the American Housing Survey)

- Determine skips:
  - set of predicate / next component pairs
  - along with the normal next component exit

Interview Flow Description

- Different component types:
  - Primitive
    - Browser Interface Form as property
    - Validation as property, written in SQL
    - Database update as property, written in SQL
  - Subprocess / abstraction
    - Parameters
    - Subprocess specification on separate slides with the same name
    - No browser interaction associated
    - Skip and next as with others
  - Roster Collect
    - Identification of relevant roster schema
    - Subprocess specification on separate slides with the same name
    - Termination condition as a property, written in SQL
Interview Flow Description

- Different component types (continued):
  - Roster enumerate
    - Existing roster identification
    - Subprocess specification on separate slides with same name
  - Checkbox
    - Primitive with additional consistency condition, written in SQL
    - And linkage to support data collection points

Interview Flow Description

- Connector types
  - Normal response flow arrow
  - Skip response flow arrow
    - with skip condition as a property of the arrow
  - Checkbox support arrows
    - Only explicit cycles allowed
    - May back up to previous questions, marking skip data
  - Roster enumerate / collect
Survey Style

- Survey authoring style specification follows on the next 4 slides
- These are the actual input to the system
- The properties attached to the items were described earlier, but they are actually entered on the diagram when constructing the style
Coherence Constraints

- No flows connected to *labels*
  - Labels defined are referenced
  - Labels referenced are defined
  - Exactly one label per slide

- Only circular flows allowed through:
  - Checkbox support arrows
  - Offpage connections through checkbox support arrows indeed can be part of the flow

- No normal or skip flow arrows into offpage subroutines

- Roster flows only connected by collect, enumerate and identify, in the obvious fashion, viz.
  - Identify in to enumerate
  - Collect out to identify
  - Every enumerate and collect has exactly one identify (subject to potential for correlated enumeration and collection)

- Uli's abstraction mechanism used for subroutines

Browser API

- Used to identify information requested and returned, not its format in the display or presentation style

- Identify Form Fields:
  - Name: identifier to link to template
  - Query string / value pairs: list of strings to present with values to be returned as the value(s) of the field for the Name.
  - Default value
  - Fills (all other words): named strings used to fill in variable questions, e.g. "he" or "she"

- Possible value restrictions:
  - Arbitrary text
  - one of { set of items } or range 1..J
  - predefined (using macro, with set or range), e.g. Month
  - table [NOT DONE]

- Represented as an XML type
  `<Field-Spec NAME=...>` (see following examples)
Template Language

• One option: HTML with parameterized presentation types: text, radio, select, etc.

<template type=checkbox name=sibling>
And how about [fill possessive] brothers?
</template>

• Using Templates with Output API we will automatically
  - (statically) check consistency between API and Template
    • Template can display all the fields
    • Every template field can be requested by the API
  - (dynamically) produce context sensitive HTML

  <field-spec name=sibling
    possessive="her"
    value="alive",true
    default=false
  >
  >&lt;template type=checkbox>
  &lt;input type="checkbox"
  name="sibling"
  value="alive">
  </template>

Template + API => HTML

• E.g. <FIELD-SPEC name=happiness
  values="very", "extremely"
  default="very">
  together with template types (italicized) below are used to map field
  names to input methods
    - checkbox, becomes:
      <INPUT TYPE="checkbox" NAME="happiness"
      VALUE=1 CHECKED=very
      VALUE=2>extremely
    - radio, becomes:
      <INPUT TYPE="radio" NAME="happiness" VALUE=1>very
      <INPUT TYPE="radio" NAME="happiness" VALUE=2>extremely
    - select, becomes:
      <SELECT>
        <OPTION SELECTED>very
        <OPTION>extremely
        </SELECT>

15
Template + API => HTML

- Text fields: `<FIELD-SPEC name=happiness
value = (text, "40,5")
default = "Type Here"
gender = "Her">`
  - text: `<INPUT TYPE="text" NAME="happiness" size=40>`
  - textarea: `<TEXTAREA NAME="happiness" cols=40 rows = 5>`
    Type Here
    `</TEXTAREA>`
- Tables: [ NOT DONE ]

AML: an Architecture MetaLanguage

David Wile
USC / ISI
wile@isi.edu
Architecture Description Languages (ADLs)

- Provide for the introduction of a set of objects: nodes, modules, components, connectors, etc.
- Allow one to impose topological constraints on the way the nodes are connected
- Provide a problem domain-specific language for stating properties of and constraints on the specification
- Provide a tool set for:
  - analysis
  - simulation
  - animation
  - compilation or synthesis

ADL Semantics

- Generally informal
- Because they are pretty much trivial!
- But, there are some significant differences from other specification languages
  - Introduction of constants
  - Restriction to style
  - Dynamic concerns
AML provides

- Declarations for:
  - elements - to be found in the artifact described by the ADL
  - relationships - to be ascertained to hold in the artifact
  - kinds - to restrict construction primitives of an artifact to a proscribed set
- Based entirely on translation of the declarations to underlying logic(s)

Elements and Assumptions

- Elements in the "real world" elements Controller, Main, Helper;
- Identification of elements analogous to asserting their existence at some time
  \[ \text{assume } \text{identified}(\text{Main}) \land \text{identified}(\text{Controller}) \land \text{identified}(\text{Helper}) \]
  - \[ \text{assume all } x, y : \text{identified-with}(x,y) \Rightarrow \exists z : \text{identified-with}(x,z) \land z \neq y \]
  - \[ \text{assume all } x : \exists y : \text{id} x = \text{exists } y : \text{identified-with}(x,y) \]
Closedness

- May want to constrain relationships between elements:
  \textbf{assume} \; \text{has-part(Controller, Main)}
  \wedge \; \text{has-part(Controller, Helper)};

- May want to say these are Controller's only parts - i.e. the set of parts is closed.
  \textbf{assume all} \; x:
  \text{has-part(Controller, x)}
  \Rightarrow x = \text{Helper} \lor x = \text{Main}

Derivative Information

- Want to say that, using the assumptions as axioms (in some logic), properties should be derivable from these axioms:
  \textbf{derive all} \; w,x,y,z:
  \text{identified-with}(w,x)
  \wedge \text{identified-with}(y,z) \wedge z \not<-> x
  \Rightarrow w \not<-> y
AML is used to (state that one must):

- Identify the elements of the model with items in the artifact.
- Ensure that these identified elements satisfy appropriate topological relationships, again in the artifact itself.
- Ensure that certain closure properties hold in the artifact.
- Establish the non-topological properties. This is a purely domain-specific activity, and is actually the major source of leverage of ADLs.