Collage: A Declarative Programming Model for Compositional Development and Evolution of Cross-Organizational Applications

Bruce Lucas, IBM T J Watson Research Center (bdlucas@us.ibm.com)
Charles F Wiecha, IBM T J Watson Research Center (wiecha@us.ibm.com)
Collage Motivation and Goals

• Motivated by a mismatch
  – today’s applications are loosely coupled, inter-organizational, inter-networked
  – but programming models are designed for monolithic, freestanding applications

• Collage programming model goals
  – targeted at cross-organizational software
    • programs are built as compositions of web components
    • inherently distributed data, execution, development models
  – highly composable
    • fine-grained “gray-box” aspect-like composition
    • supports loosely coupled cross-organizational development
  – declarative
    • focuses on “what” not “how”
    • therefore more readily composable
  – support evolutionary style of software development
    • rapid prototyping
    • progressive refinement into a deployed, hardened asset
  – radically simplified
    • uniform end-to-end programming model
    • supports fluidity of application design
Outline

• Data Model
  – RDF Distributed Graph Data Structures
  – RDF Classification
  – Collage Resources as Mutable Entities
  – Collage/RDF as a Unifying Data Model
  – Examples – XML, relational

• Execution Model
  – Execution Model Concepts
  – Bind Construct
  – Let and Create Constructs
  – End-to-end Example

• Interaction and Composition
  – Recursive MVC
  – Flexible Decomposition and Styling Example
  – Open Composition and Adaptation Example
  – Device Adaptation Example
DATA MODEL
RDF Distributed Graph Data Structures

- **Resource**: graph node, identified by URI
- **Property**: graph edge label, named by URI
- **Literal**: graph data node, as typed string
- **Triple**: bidirectional graph edge consisting of
  - **Subject**: resource
  - **Predicate**: property
  - **Object**: resource or literal
RDF Classification

- Resources may be classified
- Classes are named by URIs
- Classifications are represented by triples with property rdf:type
- **Multiple classification**: a resource may have zero, one, or more classes
- **Dynamic classification**: a resource’s classification may change
- Classifications may originate from disparate development sources
- Implications of classification are not prescribed by RDF

**RDF Triple Store**

<table>
<thead>
<tr>
<th>subject</th>
<th>predicate</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>p</td>
<td>R1</td>
</tr>
<tr>
<td>R0</td>
<td>q</td>
<td>&quot;foo&quot;</td>
</tr>
<tr>
<td>R0</td>
<td>rdf:type</td>
<td>C</td>
</tr>
<tr>
<td>R0</td>
<td>rdf:type</td>
<td>D</td>
</tr>
<tr>
<td>R1</td>
<td>rdf:type</td>
<td>C</td>
</tr>
</tbody>
</table>
Collage Resources as Mutable Entities

- Collage resources have a composite value
  - recursively composed value, i.e. tree
  - tree of RDF nodes and triples
  - triples forming value distinguished by having property that is subproperty of c:value
- Collage resources have a location
  - identified by URL such as http:
  - value may be read or updated via URL
  - this models mutable entities
Collage/RDF as a Unifying Data Model

<table>
<thead>
<tr>
<th>Collage/RDF</th>
<th>Entity-Relationship</th>
<th>UML</th>
<th>Relational</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>entity class</td>
<td>class</td>
<td>table</td>
<td>---</td>
</tr>
<tr>
<td>resource</td>
<td>entity instance</td>
<td>object</td>
<td>row</td>
<td>element, attribute</td>
</tr>
<tr>
<td>value property</td>
<td>attribute</td>
<td>attribute</td>
<td>column</td>
<td>parent-child relationship</td>
</tr>
<tr>
<td>value tree</td>
<td>composite attribute</td>
<td>---</td>
<td>---</td>
<td>XML (sub)-tree</td>
</tr>
<tr>
<td>non-value property</td>
<td>---</td>
<td>association</td>
<td>PK/FK</td>
<td>---</td>
</tr>
</tbody>
</table>
XML Data Model Example

- Uniform data model: RDF triples uniformly represent
  - relationships *within* XML document (e.g. ssn, name, address)
  - relationships *between* XML documents (e.g. employer)
- Allows uniform navigation across entire data model
- Simplifies program and data model refactoring by eliminating data model boundary between intra- and inter-document
Relational Data Model Example

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>37</td>
<td>F</td>
</tr>
<tr>
<td>Jones</td>
<td>45</td>
<td>M</td>
</tr>
</tbody>
</table>

RDF representation:

```
PERSON
  +---+---+---+
  | row| name| age| gender|
  +---+---+---+
  |     | “Smith”| “37”| “F” |
  +---+---+---+
  | row| name| age| gender|
  +---+---+---+
  |     | “Jones”| “45”| “M” |
```
EXECUTION MODEL
Execution Model Concepts

- **Reactive**: defined in terms of reactions to external events
- **Data-centric**: defined in terms of evolution of state
  - language semantics
  - data-centric abstraction, refinement, encapsulation, interfaces
- **Update-based**:
  - an update is an assignment of a value to a resource
  - update is fundamental semantic unit of action
  - all external events manifest as initiating resource updates...
  - ...that cause a cascade of ensuing updates
- **Distributed**
  - Built on distributed data model
  - Messages as implementation protocol, not programming model
- **Declarative language constructs**:
  - **Bind**: spreadsheet-like connection between resource value updates
  - **Create**: data-driven creation of resources
  - **Let**: data-driven creation of structure
Bind Construct

- Declarative expression of functional relationship between resource values
  - Developer specifies function B to compute output R0 from R1, R2, ...
  - Effectively a one-way conditional constraint on the resource values
  - "Generalized spreadsheet" conceptual model
- May be triggered by an update to an input resource - each input may be
  - active: update to that input triggers execution of bind
  - passive: update to that input does not trigger execution of bind
- Each input may refer to its resource's
  - new value: value at end of execution cycle
    - used for constraint-like computations
  - old value: value at beginning of execution cycle
    - used for non-idempotent operations such as inserting into a set or adding to a value
Let and Create Constructs

- Declarative data-driven creation of structure
  - creation of resources
  - classification of resources
  - creation of triples to connect resources

<let
  anchor="A"
  path="xpath"
  property="p"
  class="C"/>
  for every resource R of class A
  for every resource S reachable by xpath from A
  classify S with class C
  connect R to S with property p

<create
  anchor="A"
  property="p"
  class="C"/>
  for every resource R of class A
  create a resource S
  classify S with class C
  connect R to S with property p
A form (1) represented by WEATHERMAN resource allows querying and updating a relational database (2) of weather information. The <create> construct associates UI elements such as inputs (3) and triggers (4) with the WEATHERMAN class. The <let> construct (5) uses the "city" input field to select a row from the database, recording it using the "selected" property. The <bind> construct (6), triggered by the "set" trigger (4), updates the database with the quantity in the "temperature" input field, after converting Fahrenheit to Celsius. A similar <bind> construct (7) retrieves the temperature from the database, converting Celsius to Fahrenheit. Dashed boxes indicate possible distribution scenario.

demo/weather.xml
INTERACTION AND COMPOSITION
Recursive MVC

- MVC
  - model: resource with a value
  - view: set of associated resources
  - controller: binds connecting model with view

- Recursive
  - view resources may be models to further views
  - turtles all the way down: recursion is grounded in primitive resource classes representing primitive units of interaction

- Abstraction defined by
  - model content
  - model behavior

- Refinement
  - view refines (possibly implements) model abstraction

- Encapsulation
  - model is exposed
  - model encapsulates view

- Data as interface
  - permissible and observed updates to model resource define interface to view

Collage generalizes recursive MVC as a key composition mechanism
Flexible Decomposition and Styling Example

- Define a DATE data structure: every resource of class DATE has associated yr, mo, and da resources as its value.

- Define a DATE3 view that associates three input fields with any data structure that has yr, mo, da resources.

- Style a DATE with a DATE3 view by classifying a DATE resource as DATE3.
- Here every DATE is a DATE3, but DATE3 classification might be applied selectively.
- More flexible than subclassing:
  - DATE3 requires only yr, mo, da fields be present
  - DATE3 classification need not be applied at point of instantiation of resource
Open Composition and Adaptation Example

- **Scenario:** IBM partners with Bookseller to provide IBM employees with supplies
  - requires that IBM be able to modify "stock" Bookseller user interfaces and processes

- **Bookseller defines stock**
  - definition of the order form model (1)
  - order form presentation (2).

- **IBM separately authors code to customize Bookseller form, specifying**
  - the addition of an approver field to the model (3)
  - addition of a corresponding presentation item (4).

- **<with> construct** is comparable to class definition, but more flexible
  - complete definition of a class may be composed from multiple independently specified sources.
  - supports flexible multi-organizational composition of applications.
Device Adaptation Example

- View (1) is search page from desktop book-ordering application
- Views (2) and (3) adapt view (1) to smaller screen of mobile device
- Use recursive MVC: view resources of (1) become model resources of (2) and (3)
- Adaptation accomplished by creating
  - new view elements (4),
  - binds linking the new view to the old (5)
  - binds controlling navigation (6).
BACKUP
Relationship to XForms

• Collage assumes RDF as a uniform underlying data model
  – simplifies programming model
  – eases evolution and refactoring by eliminating boundaries

• Collage leverages and extends concepts familiar from XForms
  – resource-resource bind unifies and generalizes model-view and model-model binds
  – declarative resource instantiation generalizes model-driven view instantiation
  – update-driven execution model regularizes the event model
  – uniform programming model across all application tiers
  – recursively composable
ER/UML Data Model Example

**UML Diagram**

PERSON
- name
- age
- gender

employer

COMPANY
- name
- stock
- color

**Example RDF Instance**

PERSON

employer

COMPANY

name → “Smith”
age → “37”
gender → “F”

name → “IBM”
stock → “IBM”
color → “blue”
Execution model
1. An asynchronous resource update is observed
2. A set of bind constructs to execute is chosen based on currently existing resources and triples
3. The binds are executed in dependency order
4. Repeat/use and let constructs are executed based on updated resources to create new resources and triples