HILDA: HIgh level Language for Data driven Application

Fan Yang, Jayavel Shanmugasundaram, Mirek Riedewald, Johannes Gehrke, Alan Demers
Data Driven Application

- E.g. online shopping, online auctions
- Usually follows standard three-tier architecture
  - database
  - application logic
  - web interface
- Representative frameworks
  - J2EE-based application servers
  - Java Servlets/JSPs
  - ASP
  - PHP
  - WebML
  - Strudel
  - Relational transducer
Drawbacks

- **Impedance mismatch**
  - Different data model for different layers
  - Hard to develop and maintain

- **Not declarative**
  - Low level
  - Limit optimization opportunities

- **No unified handling of Queries and Updates**
  - Strudel, AutoWeb are declarative but no updates

- **No structured programming for Web Sites**
  - Websites are connected graphs of pages
  - “jump” between pages is like goto statement
  - No single entry and single exit as in function calls
Drawbacks (Cont.)

- No support for Conflict detection
  - Application level conflicts are hard to detect
- Mixes Application Logic and Presentation
  - Separation between application logic and presentation should be enforced
Outline

- Hilda Overview
  - CMS Case Study
  - Syntax and semantics of AUnit
  - Mini-CMS with Hilda
  - AUnit inheritance
  - PUnit

- Implementation

- Future work

- Summary
CMS

- Assignment Creation
- Viewing student grades
- Student group Management
Hilda Overview

* A single data model – relational model
  - Represent the state of all parts of the application

* Application Unit (AUnit)
  - Single entry and single exit
  - Instances of AUnits form a regular tree structures
  - Models application logic and web site navigation as state transitions

* Presentation Unit (PUnit)
  - HTML-based presentation construct
  - Associated with an AUnit and describe how the content of the AUnit is to be presented
  - AUnit deals with application logic and web site structure while PUnit deal with presentation issues like page layout, font size etc.
**Application Unit**

- Single entry and single exit
  - Optional input and output schema
- Instances of AUnit
  - Activation: creation of an instance
  - Deactivation: destroying of an instance
- Basic AUnit
  - Can interact with users
  - E.g. *ShowRow*: show a row(input) to the user in web browser. *UpdateRow*: show a row(input) to the user and get users’ update(output)
  - *SelectRow*: let user choose one row(output) from a table(input)
- User-defined AUnit
  - Optional local schema and local query
  - Optional persistent schema and persistent query
  - Child AUnits (Basic or User-defined)
  - Logic to activate instances of child AUnits (Called *Activator*)
  - Root AUnit (main function)
User-defined AUnit

- All information are defined using Relational model
  - OutputSchema { Table1(id:integer, value:string) }
- SQL is used to specify update operation
  - E.g Table1 :- SELECT * FROM Table1
  
  Table1 :- SELECT genKey(), ‘foo’
- Activator
  - Activation table and query: For each tuple in activation table, we activate one instance of child AUnit.
  - Input query: Prepare input for child AUnit instances
  - Output handlers: Handle output from the returned child AUnit instance. May cause the instance itself return
Persistent schema defines tables used for course management

Get user name as input

For each course, activate one CourseAdmin instance

Prepare input for each CourseAdmin instances activated

Action will be performed, when one child instance returns
One instance of CreateAssignment always will be activated

Show the list of assignments

Short hand for input and output schema
New assignment information

Condition to check

Construct the output tables and cause the instance to return

Initialize local table

Update assignment information

Reset the date if the constraint is not satisfied

Get a new problem

User's input
State Transition

* Activation phase (Bootstrap for each session)
  - For each incoming session, a new instance of Root AUnit is activated.
  - An nested tree-structure is created recursively which keeps all application states in all three layers. We call it activation tree.
  - The process is idempotent and considered as an atomic action

* State transition consists of return phase and reactivation phase

* Return phase
  - Initiated by returning from instance of Basic AUnit
  - One of the output handlers of the activator which activates the returned instance will be invoked.
  - If the handler is a return handler, the parent will return and deactivates itself.
  - Go on recursively until we invoke an non-return handler or the root
State Transition (Cont.)

- Reactivation phase
  - Proceed just like activation phase except that
    - Instances with identical activation tuple in previous state preserve their local tables
    - Returned instances lose their local state.
  - Intuitively activation tree not by the return phase shouldn’t run local query and lose their local tables
Shared Persistent tables

| Assign | Course | Other tables ...
|--------|--------|-----------------
| Aid   | Cid   | Name | Release | Due |
| 01    | 10    | Assign01 | 03/5    | 03/09 |
| 02    | 10    | Assign02 | 03/10   | 03/14 |
| 04    | 11    | Assign01 | 03/6    | 03/20 |
| Cid   | Name  |
| 10    | CS632 |
| 11    | CS633 |

Session 1. Incoming user is an admin

Session 2. Incoming user is an admin
Shared Persistent tables

**Assign**

<table>
<thead>
<tr>
<th>Aid</th>
<th>Cid</th>
<th>Name</th>
<th>Release</th>
<th>Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>10</td>
<td>Assign01</td>
<td>03/5</td>
<td>03/09</td>
</tr>
<tr>
<td>02</td>
<td>10</td>
<td>Assign02</td>
<td>03/10</td>
<td>03/14</td>
</tr>
<tr>
<td>04</td>
<td>11</td>
<td>Assign01</td>
<td>03/6</td>
<td>03/20</td>
</tr>
<tr>
<td>05</td>
<td>10</td>
<td>Assign03</td>
<td>03/21</td>
<td>04/01</td>
</tr>
</tbody>
</table>

**Course**

<table>
<thead>
<tr>
<th>Cid</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>CS632</td>
</tr>
<tr>
<td>11</td>
<td>CS633</td>
</tr>
</tbody>
</table>

Other tables, student, group etc ...

---

Session 1. Incoming user is an admin

CreateAssignment returns

Session 2. Incoming user is an admin
Shared Persistent tables

### Assign

<table>
<thead>
<tr>
<th>Aid</th>
<th>Cid</th>
<th>Name</th>
<th>Release</th>
<th>Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>10</td>
<td>Assign01</td>
<td>03/5</td>
<td>03/09</td>
</tr>
<tr>
<td>02</td>
<td>10</td>
<td>Assign02</td>
<td>03/10</td>
<td>03/14</td>
</tr>
<tr>
<td>04</td>
<td>11</td>
<td>Assign01</td>
<td>03/6</td>
<td>03/20</td>
</tr>
<tr>
<td>05</td>
<td>10</td>
<td>Assign03</td>
<td>03/21</td>
<td>04/01</td>
</tr>
</tbody>
</table>

### Course

<table>
<thead>
<tr>
<th>Cid</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>CS632</td>
</tr>
<tr>
<td>11</td>
<td>CS633</td>
</tr>
</tbody>
</table>

Other tables, student, group etc ...

---

**Session 1. Incoming user is an admin**

- CMSRoot
- Cid: 10
- Other activators
- CourseAdmin
- Aid: 01
- Name: Assign01
- Aid: 02
- Name: Assign02
- Aid: 05
- Name: Assign03

Activators to change properties of assignment, get new problems
- CreateAssignment
- ID: 50
- ShowRow
- ID: 100
- ShowRow
- ID: 101

Show input on screen

**Session 2. Incoming user is an admin**

- CMSRoot
- Cid: 11
- Other activators
- CourseAdmin
- Aid: 01
- Name: Assign01
- Aid: 02
- Name: Assign02
- Aid: 05
- Name: Assign03

Activators to change properties of assignment, get new problems
- CreateAssignment
- ID: 50
- ShowRow
- ID: 100
- ShowRow
- ID: 101

Show input on screen
Inheritance

- **AUUnit** support inheritance for code reuse
  - Add extra input, output, local and persistent tables
  - Add extra activator
  - Add more condition for activation query
  - Add actions for handler

- **We use AUnit to define both application logic and content and structure of the website**
  - Separate them further using inheritance
  - First consider the general application logic
  - Then refine the structure of the website by inheritance
The course user chosen from course list
Show a course list
Activate only one CourseAdmin instance for the course chosen by the user
PUnit

- HTML based
- Only in charge of how information are shown not what to show.

```html
punit ShowNavCMS for NavCMS {
  <body bgcolor="yellow">
    <hr>
    <punit activator='"ActSelectRow"'
      name='"ShowSelectRow"'>
    <hr>
    <punit activator='"ActCourseAdmin"'
      name='"ShowCourseAdmin"'>
    <hr>
    ...
  </body>
}
```
Outline

★ Hilda Overview
  – CMS Case Study
  – Syntax and semantics of AUnit
  – Mini-CMS with Hilda
  – AUnit inheritance
  – PUnit

★ Implementation

★ Future work

★ Summary
Compiler

```
Hilda Program
     ↓
  Hilda Compiler
     ↓
   Generate
       ↓
  DB Scripts
     ↓ Create
  Backend database
     ←
  Application server
     ←
Web browser client
     ↓ Deploy
  Servlet and Java classes
     ↓ Generate

```
Compiler(Cont.)

- Local and persistent tables are stored in database
- Input and output are passed around as view definition.
- Activation tables are cached in main memory for fast creation of activation tree
- Each Aunit is translated to a java class with toHTML method which recursively call toHTML methods of its children.
Future work

- Client-Server code Partitioning
- Data Caching
- Application concurrency control
Question?