Knowledge Transformation from Task Scenarios to View-based Design Diagrams

Nima Dezhkam
Kamran Sartipi
{
(dezhkam, sartipi}@mcmaster.ca

Department of Computing and Software
McMaster University
CANADA

SEKE'08
July 1, 2008

Outline
- Task Scenarios
- Scenarios in knowledge extraction
- Proposed framework
  - Scenario generation
  - Scenario decomposition
  - Design construction
- Fast-food restaurant case-study
- Conclusions

Outline

Task Scenarios

Different scenario representations:
- Simple text
- Graphical representation
- Relational algebra, etc.

Common applications of scenarios:
- Requirement elicitation and analysis,
- Design representation,
- Testing
- Maintenance

Proposed Framework for Scenario to Design Diagram Transformation

Transforms a set of text-based scenarios into two types of design diagrams, i.e., Data and Function.

Properties:
- Uses a scenario syntax that allows us to define well-structured scenarios.
- Uses a scenario schema to parse the scenarios and populate an object base of actors, actions, and dependencies.
- Uses Guidelines for transforming the elements in the object base into design diagrams.

Proposed Framework for Scenario to Design Diagram Transformation

Stage 1
(Scenario generation)

Stage 2
(Scenario decomposition)

Stage 3
(Design construction)
#### Scenario Syntax

Scenario : \{Actor \times \{Constraints\}^{0..M} \times \{Action \times \{Constraints\}^{0..M} \times \{Working Information \times \{Constraints\}^{0..M}\}}

- Example scenario: "Order taker adds a menu item to an incomplete order."

#### Sample Scenario Template Form

A scenario template forms the knowledge-base of a fast-food restaurant system.

#### Sample Fast-food Scenario

**Scenario #1**: "Order taker reports the price of the order."

- **goal** = ordering order & handling payment
- **actor** = order taker
- **information** = order, price
- **action** = report price
- **data dependency** = associated with an order

**Decomposed scenario:**

- **Action**: report price
- **Data dependency**: associated with an order

#### Example of Scenario Decomposition

**One of the 12 Scenarios**

- **Sample Fast-food scenario**: "Order taker reports the price of the order."

- **Decomposed scenario**:
  - **Action**: report price
  - **Data dependency**: associated with an order

#### Proposed Scenario Schema

- **Objectbase Created from 10 Scenarios**

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Action</th>
<th>Data Dependency</th>
<th>Action Precedence</th>
<th>Data Precedence</th>
<th>Precedence</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>price</td>
<td>order</td>
<td>report price</td>
<td>report price</td>
<td>report price</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>order</td>
<td>price</td>
<td>report price</td>
<td>report price</td>
<td>report price</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>price</td>
<td>order</td>
<td>report price</td>
<td>report price</td>
<td>report price</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>order</td>
<td>price</td>
<td>report price</td>
<td>report price</td>
<td>report price</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>price</td>
<td>order</td>
<td>report price</td>
<td>report price</td>
<td>report price</td>
<td></td>
</tr>
</tbody>
</table>
Design Construction Guidelines: Data View

- **Step 1:** Extract all instances of Actor, Working information, and Data dependency classes from the object base and apply the following rules on them:
  1. Instances of Actor and Working information are candidate entities/attributes.
  2. Instances of Is dependency imply generalization and inheritance relationships, i.e., A is B means A is a sub-entity of B, or B is a super-entity of A.
  3. Candidate entities/attributes that appear on either side of a Is-associated-with, or Is-part-of relationship are considered as entities.
  4. Instances of Is and Belongs-to dependencies are used to identify the attributes of the entities, i.e., A Has B (or B Belongs to A) means B is an attribute of entity A.
  5. Instances of Is-associated-with dependency imply candidate association relationships.
  6. Instances of Is-part-of dependency imply candidate decomposition relationships.

- **Step 2:** Depict every entity by a rectangle, every attribute of an entity as a bubble connected to it and label them by their names. Every relationship between two entities can be represented by a line connecting them. Label every relationship according to the type of dependency it came from, e.g., “Is”, “Ispart”, etc.

Generated E-R Diagram

List of Actions in Order Taking Component and the “Follows” Relation

<table>
<thead>
<tr>
<th>Index</th>
<th>Action</th>
<th>Follows*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Login using ID &amp; password</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Logon the system</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Review orders</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Initiate order</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Call-back unpaid orders</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Edit orders</td>
<td>1.5</td>
</tr>
<tr>
<td>7</td>
<td>Compute price</td>
<td>1.5,6</td>
</tr>
<tr>
<td>8</td>
<td>Report price</td>
<td>1.5,6,7</td>
</tr>
<tr>
<td>9</td>
<td>Deliver order payment</td>
<td>1.5,6,7</td>
</tr>
<tr>
<td>10</td>
<td>Enter cash-in</td>
<td>1.4,5,6,7,8</td>
</tr>
<tr>
<td>11</td>
<td>Return change &amp; receipt</td>
<td>1.4,5,6,7,8,10</td>
</tr>
<tr>
<td>12</td>
<td>Send order to assembly station</td>
<td>1.5,6,7,8,10,11</td>
</tr>
<tr>
<td>13</td>
<td>Send excess cash to cash safe</td>
<td>1.4,5,6,7,8,10,11,12</td>
</tr>
</tbody>
</table>

Design Construction Guidelines: Function view

- **Step 1:** Extract all instances of Action, Action dependency, and Constraint classes from the object base and apply the following rules on them:
  1. Instances of Action class are the functions.
  2. Instances of the Follow and Precede dependencies determine the order of execution of the functions. To simplify the diagram generation, transform all the Precede dependencies to Follow, i.e., for all functions f1 and f2, change “f1 Precede f2” to “f2 Follow f1”.
  3. The participants of a Is-parallel-with dependency must be executed concurrently.
  4. The conditions for a function to follow another are determined by the Constraints associated with the function, actor, and working information in the corresponding scenario that the “following” appears.

- **Step 2:** Generate Follow+ relationship (the transitive-closure of the Follow).

- **Step 3:** Sort the functions in ascending order based on the number of the functions they follow, i.e., based on the number of times they appear on the left hand side of a Follow relationship.

- **Step 4:** Starting from the first of the list, depict the function (name A) with a square and label it by its name. List all the functions that Follow A, then use “AND” and “OR” connectors when necessary to form the graph. Label each arrow with the triggering conditions obtained in rule “4” above. Finally, remove A from the list and repeat Step 4, until the list is empty.

Generated Function Diagram

Conclusion

- **Task scenarios:** are used to generate the ingredients of the design diagrams.
- **Scenario generation:** generating a set of structured text-based scenarios that conform with a regular expression syntax.
- **Scenario decomposition:** mapping generated scenarios onto scenario schema which allows parsing the structured scenarios and generating instances of schema classes.
- **Design construction:** generating design diagrams in Data and Function views using the decomposed scenarios and based on a set of guidelines.
Each time an order is set up, the kitchen should be informed to prepare the order-items.

When the computer system determines that all items of an order are available in
the chute, the order can be assembled.

Each available assembly-station picks the order and displays it on its screen.

The assembly-stations use screen and keypad for interaction with the staff.

The staff assemble the orders, and using keypads inform the system. If the order is paid, the system allows the delivery of the order to the customer; otherwise, the delivery will be postponed to the time that the order is paid.

If the system indicates that an order can be filled, but the chute does not contain a sufficient quantity of some order-item, the staff report the shortage to the system to be prepared.
Inventory Unit

- The inventory unit in the system keeps track of the consumption of all materials used for preparation and packaging of the order-items.
- We refer to these materials as “raw-materials”. This unit has a very close interaction with the preparation unit.
- The system keeps stock, and the inventory of raw materials is updated dynamically.
- The arrival of new materials into storage is entered into the system by the staff, and the consumption of the materials is dictated by the recipes of food-items.
- To preserve stock integrity, the system assumes a minimum threshold for usage of each menu-item in the system. If the number of a certain menu-item drops below this threshold, it is considered unavailable and the inventory unit alerts the order-taking unit to inhibit taking that item.

Management Unit

The management-unit of the restaurant system is responsible for setting up:

- Active stations in order-taking, preparation, and assembly units.
- System tables such as restaurant-menu, recipes, anticipated demands, minimum number of menu-items, and raw-materials in stock.
- List of menu-items to be prepared by each preparation station.
- Cash slips and float.
- Different applicable taxes.
- System time and date.

E-R diagram of the Restaurant System