Using Predicate Fields in a Highly Flexible Industrial Control System

Shay Artzi*, Michael D. Ernst
CSAIL, MIT

* Work done while at Rafael, Ltd.
Evaluating Predicate Fields

- Predicate oriented programming is a promising research idea that has never been evaluated in practice
  - Dynamic classification of an object into subclasses:
    - Predicate classes [Chambers et al. 93]
    - Kea language classifiers [Mugridge et al. 91, Hamer et al. 92]
    - Modes [Taivalsaari 93]
  - Predicate Dispatch [Ernst et al. 98, Millstein 04]
- We successfully deployed them in an industrial application
- Conclusion:
  - Increase software flexibility to handle changing and unknown requirements
  - Simplify certain development task
A predicate field is present or not, depending on the values of other fields

First name: Shay
Last name: Artzi
Parking required: Yes
Dates: ...........

<table>
<thead>
<tr>
<th>obj:Reservation</th>
</tr>
</thead>
<tbody>
<tr>
<td>-firstName -- &quot;Shay&quot;</td>
</tr>
<tr>
<td>-lastName -- &quot;Artzi&quot;</td>
</tr>
<tr>
<td>-parkingRequired -- true</td>
</tr>
<tr>
<td>-licensePlate</td>
</tr>
<tr>
<td>-dates</td>
</tr>
</tbody>
</table>
Implementation with Predicate Fields

// Definition
pred arriveWithCar (needsParking==true);
class Reservation {
    ...
    bool needsParking;
    String licensePlateNum when@arriveWithCar;
}

// Use
Reservation r = new Reservation();
r.licensePlateNum = "44GT23"; //RUN-TIME ERROR
r.needsParking = true;
r.licensePlateNum = "44GT23"; //OK
Advantages of Predicate Fields

- Allow an object to change its structure during its life cycle
  - Recover from user errors in user interface
  - Emulate dynamic classification of an object into subclasses
- Expedite user interface development
- Fine-grained customization of objects
Outline

- Introduction
- Case Study: Experiment control system
- Predicate Fields Motivation
- Developer Experience
- Summary
Case Study: Experimental Control System

- System goal: define, control, execute, and examine results of experiments

- Experiment:
  - Ordered instructions on a set of devices
  - Control complex events and vast number of devices
Requirements and Design

- Non functional requirement: adaptability to physical hardware changes (new devices, device locations)
- MML language to create experiments
- Two-level system architecture
  - Knowledge level: legal configuration of operational objects.
  - Operational level: concrete model of the system.
Implementation 1

- Development:
  - Fifteen man years
  - Written in Delphi IDE and the Object Pascal language
  - Component based (COM/DCOM)
  - ~100,000 lines of code
- In daily use
- Won several internal prizes
- Its deficiencies inspired the use of predicates in Implementation 2
Implementation 2

- In development since 2002 in Visual Studio .NET and C#
- Currently in integration phase (adding controlled hardware)
- Five developers
- Implementation 1 functionality was subsumed in less than two years
- Controls more complicated hardware
- Uses predicate fields.
Implementation 2 tiers

- C# library
- Developer: Knowledge Level in Database
- Developer: Operational Level in C#
- User: Implementation in MML
- Predicate Library
- MML Interpreter and Editor
- MML Interpreter Predicate Definitions
- Experiments
Outline

- Introduction
- Case Study: Experiment control system
- Predicate Fields Motivation
- Developer Experience
- Summary
Predicate Fields Motivation in Implementation 2

- Implementation 1 deficiencies were resolved using predicates:
  - Tight coupling of persistent objects with their user interface
  - Many custom made user interface forms
  - Can’t change object types
  - Inflexibility to some hardware changes
Motivation 1
Tight coupling

- **Cause:** MML statements which are persistent objects with UI representation had tight coupling with other components
- **Problem:** Changes to the structure of the MML statement required cross cutting modifications

- **Example:** adding a max_repeat field
- **Solution:** Dynamic objects. Structure and connections defined using predicates. Predicate fields carry the rest of the information
- **Outcome:** Changes to the MML statement data type can be easily done in one place (database)
Motivation 2
Many Custom Made UI Forms

- **Cause:** One UI form per MML statement type, and device type
- **Problem:** UI development and changes were costly
- **Example:** Adding a new measurement device type with a different number of channels
- **Solution:** Adopting .NET editing concept
  - One adjustable properties form
  - Object exposing properties to be edited
  - PropertyGrid uses reflection to query a selected object structure
  - Dynamic objects can be easily wrapped to expose properties
- **Outcome:** Homogeneous look and feel and reduced user interface development effort.
Editing concept example

Setting Properties

Defining an MML instruction
Motivation 3

Can’t Change Object Types

- **Cause:** The user is unable to change an object type in the MML UI
- **Problem:** Losing mutual information of the new and the old object type
- **Example:** Changing an automatic statement to a manual one
- **Solution:** Using predicate fields to dynamically classify into subclasses.
- **Outcome:** Allowing objects to “switch type” while maintaining mutual information
Motivation 4
Inflexibility to Hardware Changes

- **Cause**: New device types with components that exists in the set of known devices required cloning information
- **Problem**: Introducing clones into the system. Maintenance complexity increase
- **Solution**: Using predicate fields to support fine grained combination of existing fields
- **Outcome**: More flexibility to new device types
Outline

- Introduction
- Case Study: Experiment control system
- Predicate Fields Motivation
- Developer Experience
- Summary
Definitions Modifications

- Developers making modification to the MML interpreter definitions:
  - Modify the dynamic types (rarely)
  - Modify predicates, fields and fields’ types (usually).

- Initially found to be difficult due to the library use and integral limitations
Limitations

- **Declarative approach**
  - Far-reaching, system behavior depends on the metadata
  - Developers need to master the knowledge level
  - Type safety cannot be guaranteed

- **Implemented as a library**
  - Incur performance overhead
  - Software is harder to understand, less readable
  - Poor UI (MML interpreter definitions were saved in database)
Developer Experience (after further use)

- Familiarity and ease
- Easily perform seemingly complex task
- Surprising uses (E.g. wizards for the knowledge level editor)
- Change in perspective toward designing the UI
- Dynamic type errors cause distrust
- Active interest from other development teams
Summary

- Used predicate fields in a large industrial application
- Developers find predicate fields useful
- Software flexibility is increased
- UI development costs were greatly decreased
- Lack of static type checking is a problem