Domain Specific Languages for Framework-Based Enterprise Application

Prof. Eli Tilevich

Software Innovations Lab
Dept. of Computer Science
Virginia Tech
Assistant Professor since 2006
- Ph.D. from Georgia Tech, 2005

Direct the Software Innovations Lab
- Distributed Computing / Middleware
  - USENIX Middleware 2009 PC, ICDCS 2010 PC
- Automatic program evolution and maintenance
  - ICSM 2009/2010 PC, OOPSLA 2010 PC
- Using sound to improve Software Engineering
  - ICPC 2009
Software Innovations Lab

- Jeremy Archuleta  Ph.D.*
- Young-Woo Kwon  Ph.D.
- Yang Jiao  Ph.D.*
- John Edstrom  M.S.
- Cody Henthorne  M.S.
- Arif Khokar  M.S.
- Myoungkyu Song  M.S.

* -- co-advised with Prof. Wu Feng
Overview of my research activities

Framework-based enterprise applications

Intermediate Code Enhancement (ICE) language

Pattern-Based Structural Expressions (PBSE) language

Future work directions
Novel Abstractions for Distributed Computing
Novel Abstractions for Distributed Computing

Treating Symptoms (*USENIX Middleware 2009*)

- Flu
  - Cannot eliminate the flu
  - Treat symptoms
  - Improve quality of life

- Network Volatility
  - Cannot avoid network volatility
  - Treat symptoms
  - Improve quality of service (QoS)

Disconnected Operations
- Caching
- Queuing
- Replication
- Hoarding

Dr. OSGi

Prof. Eli Tilevich Software Innovations Lab@VT
Managing program evolution

Examples

Upgrade Rules

Prior

Posterior

Legacy App.

Transformation Engine

Upgraded App.
Novel software engineering paradigms

- Using sound to facilitate program understanding and performance tuning

Sonification
Roadmap

- Overview of my research activities
- **Framework-based enterprise applications**
- Intermediate Code Enhancement (ICE) language
- Pattern-Based Structural Expressions (PBSE) language
- Future work directions
Automatic Programming

Requirements → Programmed × → Application
Automatic Programming

Requirements → “Magic” Generator → Application
Automatic Programming

Requirements → Programmer → Application
Automatic Programming

Requirements → Programmer + Code generators, Libraries, Components, Frameworks → Application
Automatic Programming

- “The Holy Grail” of Software Engineering
  - Take the human programmer out of the loop
    - Code generators, components, libraries, frameworks, etc.
- Declarative prog. models of enterprise frameworks:
  - The programmer
    - writes business logic
    - declares metadata to express extra-functional concerns
      - Persistence, transactions, security, distribution, etc.
  - The framework
    - automatically provides the expressed concerns
Framework-Based Enterprise Applications

Programmer

Framework

High level Description for Extra-func. Concerns

Business Logic
What is Framework?

- **Object-oriented frameworks**
  - Used in enterprise software development
    - Implement extra-functional concerns: persistence, transactions, security, distribution, etc.
  - Streamline software construction
    - Through reusable designs and predefined architectures.

- **Modern enterprise frameworks**
  - Based on Plain Old Java Objects (or POJOs)
    - No special interfaces or framework API methods
    - Bytecode enhancement to introduce framework functionality

- **POJO-based frameworks**
  - Improve separation of concerns, speed up development, and improve portability.
Example: Persistence

Business Logic

```java
public class ManagerEJB {
    private String orderId;
    private String status;

    public String getOrderld(){
        return orderId;
    }

    public String getStatus(){
        return status;
    }
}
```
Example: Persistence

- Business Logic + High level desc.(Java 5 Annotation)

```java
@Entity
@Table(name="Manager")
public class ManagerEJB {
    private String orderId;
    private String status;
    @Id
    @Column(name="orderId", primaryKey=true)
    public String getOrderId(){
        return orderId;
    }
    @Column(name="status", primaryKey=false)
    public String getStatus(){
        return status;
    }
}
```
Compile the source code to bytecode.
Transparent bytecode enhancement

- Add new functionality to the compiled bytecode.
- Enhanced bytecode differs from the original source code.
Transparent bytecode enhancement

Source Code → Compiler → Intermediate Code (Bytecode) → Enhancer (static or at class load time) → Enhanced Bytecode
Problems with Annotations

```java
@Entity
@Table(name="Manager")
public class ManagerEJB {
    private String orderId;
    private String status;

    @Id
    @Column(name="orderId", primaryKey=true)
    public String getOrderId(){
        return orderId;
    }

    @Column(name="status", primaryKey=false)
    public String getStatus(){
        return status;
    }
}
```
Problems with Annotations

- Programmability
  - Error-prone—naming conventions are implicit

- Understandability
  - Express no structural information
  - Examine the entire codebase to extract invariants

- Maintainability
  - Newly-added constructs must be properly annotated

- Reusability
  - Cannot be reused at any level
1. A novel approach to improving the precision and utility of source-level programming tools in the presence of intermediate code enhancements.

2. The Intermediate Code Enhancement (ICE) language, a Domain-Specific Language (DSL) for concisely expressing structural enhancements that modern enterprise frameworks commonly apply to intermediate code.

3. A debugging architecture that enables symbolic debugging of programs whose intermediate code has been transparently enhanced. (OOPSLA ’09 S/W Demo)
Roadmap

- Overview of my research activities
- Framework-based enterprise applications
- Intermediate Code Enhancement (ICE) language
- Pattern-Based Structural Expressions (PBSE) language
- Future work directions
Domain-Specific Languages

- Target specific tasks in a fixed and restricted problem domain
- Provide high-level abstractions and relationships that fit the domain closely
  - Domain tailoring abstractions improves productivity
- Examples: SQL, Make, YACC, POSTSCRIPT, LaTeX, XQuery, etc.
Example: Mortgage Authorization Application

```java
public void displayMaxMortgageEligibility
    (Display display, double projectedIncrease)
{
    ...
    double maxMortgage = calcMaxMortgage(salaryLevel, creditLevel);
    ...
    FrameField mortgageField = display.getMortgageField();
    mortgageField.setVal(maxMortgage);
    ...
}
```

```java
public void displayMaxMortgageEligibility
    (mortgage.Bank$Display, double);
Code:
...
    invokestatic   // jdoGetSalaryLevel;
    aload_0
    invokestatic   // jdoGetCreditLevel;
    ...
    invokevirtual   // calcMaxMortgage
    ...
    invokevirtual   // getMortgageField
```

- What if `getMortgageField` return `null`?
Example: Mortgage Authorization Application

```java
public void displayMaxMortgageEligibility (Display display, double projectedIncrease) {
    double maxMortgage = calcMaxMortgage(salaryLevel, creditLevel);
    FrameField mortgageField = display.getMortgageField();
    mortgageField.setVal(maxMortgage);
}
```

- What if `getMortgageField` return null?

```java
public void displayMaxMortgageEligibility (mortgage.Bank$Display, double);
Code:

```
Example: Mortgage Authorization Application

```java
public void displayMaxMortgageEligibility(Display display, double projectedIncrease)
{
    ...
    double maxMortgage = calcMaxMortgage(salaryLevel, creditLevel);
    ...
    FrameField mortgageField = display.getMortgageField();
    mortgageField.setVal(maxMortgage);
    ...
}

- Difficult to find errors.
- Hinder source-level Prog. tools.
```

Code:

```java
public void displayMaxMortgageEligibility(mortgage.Bank$Display, double);
Code:
...
invokestatic // jdoGetSalaryLevel;
aload_0
invokestatic // jdoGetCreditLevel;
...
invokevirtual // calcMaxMortgage
...
invokevirtual // getMortgageField
```
Managing Bytecode Enhancements

- **What is a structural enhancement?**
  - Large scale program transformations
    - Add, change, remove classes, methods, and fields
  - Limited and well-defined method body changes
    - Replace field accesses with getter/setter
    - Replace method calls with wrappers
Managing Bytecode Enhancements

- **Structural enhancements at the bytecode level**
  - Poorly understood, not properly documented
  - Lack easy-to-understand expression medium
- **Our solution**
  - **Intermediate Code Enhancements (ICE) Language**
    - Domain Specific Language (DSL) for bytecode enhancements
    - Helps express, understand, and maintain enhancements
ICE Constructs

- AddClass
  - [Add|Remove]SuperClass
  - [Add|Remove]SuperInterface
  - [Add|Remove]Method
  - [Add|Remove]Field
- ReplicateMethod
- Field[Get|Set]Replacer
- Iterators
- Patterns
Program JDO Using SUPER_JDO_ICE

Begin

EnhClass = OrgClass
EnhClass.AddInterface ("javax.jdo.spi.PersistenceCapable")

Var Pattern fieldP
Begin

modifiers = ("private" or "protected") and not "static"
End

Var Iterator fieldIter = OrgClass.Fields(fieldP)

EnhClass.AddMethod(FieldSetReplacer("jdoSet", fieldIter))
EnhClass.AddMethod(FieldGetReplacer("jdoGet", fieldIter))

End
Managing Bytecode Enhancements

- ICE scripts can be used to
  - Document bytecode enhancements
  - Improve precision and utility of source-level programming tools

- How can a ICE script be produced?
  - Provided by a framework vendor
  - Reverse-engineered from existing systems
Interpreting ICE

Source Code -> Enhancement Program Constructs

ICE Script

ICE Parser

ICE Interpreter

- Original Source Code
- Enhancement Information
- Bytecode Processor
- Symbol Table
- Bytecode to Src. Map
- Enhanced Bytecode
- Original Source Code
Interpreting ICE

Enhanced Bytecode → Original Source Code

ICE Script

ICE Parser

Src. code Processor

Symbol Table

Bytecode Processor

Bytecode to Src. Map

ICE Interpreter

Original Source Code

Enhancement Information

Enhanced Bytecode

Original Source code
Symbolic Debugger for Enhanced Bytecode

A standard debugger: JDB

- mortgage.Bank.displayMaxMortgageEligibility(), line=31 bci=57
- mortgage.CreditLevel.getCredit(), line=12 bci=0
- mortgage.Bank.displayMaxMortgageEligibility(), line=31 bci=57
- mortgage.CreditLevel.getCredit(), line=12 bci=0
- mortgage.CreditLevel.getcredit(), line=12 bci=25
- org.jpox.state.JDOStateManagerImpl.isLoaded(), line=1,764 bci=5
- mortgage.CreditLevel.jdoGetcredit(), line=-1 bci=25
Roadmap

- Example of bytecode enhancements
- Managing bytecode enhancement
- Enhancements-aware programming tools
- Pattern-Based Structural Expressions (PBSE) language
- Conclusions
A clear exposition of the advantages and shortcomings

- XML and annotations.

Pattern-Based Structural Expressions (PBSE)

- Usability, reuse, and ease of evolution advantages

An automated translation approach

- Automatic annotation of the source with equivalent Java 5 annotations.
Example: Persistence

- Business Logic

```java
public class ManagerEJB {
    private String orderId;
    private String status;

    public String getOrderId() {
        return orderId;
    }

    public String getStatus() {
        return status;
    }
}
```
A PBSE Script Example

01  **Metadata** MyJPA<Package p>
02    Class c in p
03        **Where** (public *EJB)
04            c += @Table
05            @Table.name = (c.name =~ s/EJB$//)
06        Column<c>
07
08  **Metadata** Column<Class c>
09    Field f in c
10        **Where** (private * *)
11            Method m in c
12                **Where** ("get" + (f.name =~ s/^([a-z])/([A-Z])/)) == m.name)
13                m += @Column
14            @Column.name = f.name
15                **Where**(public * get*Id ())
16            @Column.primKey = true
17            m += @Id
18  MyJPA <"package1">
PBSE Constructs

- Metadata \textit{module\_name} \\
\langle[\text{Package}|\text{Class}|\text{Method}|\text{Field}|\text{Parameter}] \text{ val}\rangle \\
- \text{[Class}|\text{Method}|\text{Field}|\text{Parameter}] \text{ iterator in parm} \\
- \text{Where (pattern)} \\
- @\text{Metadata} \\
- @\text{Metadata.property} \\
- @\text{Metadata.property} = \text{ val} \\
- \text{val} \ += @\text{Metadata} \\
- ~s / [^] \text{ old\_val} [$] / \text{ new\_val}
Pattern-Based Structural Expressions

- Programmable
  - Express metadata concisely
  - Use OO and declarative query programming
    - Where clauses and regular expressions
    - Straightforward to learn for an enterprise programmer
Pattern-Based Structural Expressions

- Understandable
  - Encode relationship between program constructs and their metadata
    - Express software framework architecture
    - Capture complex invariants throughout the entire codebase.
Pattern-Based Structural Expressions

- **Maintainable**
  - Express explicit structural information
    - Keep source code consistent with metadata.
    - Eliminate dependencies on specific annotations
      - Switch framework vendors at will
    - Maintain framework conventions during program evolution
Pattern-Based Structural Expressions

- Reusable
  - Leverage naming conventions of framework dependent code
    - Reuse across classes and applications.
    - Modify easily for different naming conventions
PBSE Translator

Java Src.

Java Src.

PBSE Metadata

PBSE Metadata
PBSE Translator

Java Src.

PBSE to Annotations Translator

Annotated Java Src.

Compiler

Bytecode

PBSE Metadata
Conclusions

- **ICE (Intermediate Code Enhancements)**
  - Expresses direct bytecode transformations
  - Improves precision and utility of prog. tools

- **PBSE (Pattern-based Structural Expressions)**
  - Expresses naming patterns between the source code and its metadata
  - Enables systematic reuse across applications
Current Projects

- **Automatic Adaptation of Java Frameworks for X10: IBM**
  - Automatically introduce various non-functional concerns into X10 programs
Current Projects

Java Source → Compiler → ICE + Program Analysis → Enhanced Bytecode → PBSE → nᵗʰ Enhanced Bytecode
Thank you!