Adding Syntax and Static Analysis to Libraries via Extensible Compilers and Language Extensions

Eric Van Wyk, Derek Bodin, Paul Huntington

Department of Computer Science and Engineering
UM Software Engineering Center
University of Minnesota

www.cs.umn.edu/~evw
www.cs.umn.edu/melt
www.umsec.umn.edu
evw@cs.umn.edu
Outline

- Libraries – the good and the bad
- Extensible Languages / Compilers
- Attribute Grammars, Forwarding
- Embedding SQL into Java
- Project status
Libraries

- Very widely used for programmer-defined (domain specific) abstractions
- Compositional in nature
  - Developed by independent parties
  - Desired collection chosen and imported by programmer
- However, lacking in some respects
  - Cannot define new syntactic forms
  - Cannot easily define static analysis
  - Cannot easily define optimizations (% C++ templates)
Extensible Languages / Compilers

- An **extensible language** allows new language features to be added as first-class language constructs.
- These features may be **new constructs** (syntax), **analyses**, or **optimizing transformations**.
- We package language features from **domain-specific** languages so that they can be **imported** (without implementation level knowledge of the extension) into the programmer’s **general purpose host language**.
- **Aim:** composable language extensions
- For example …
Example: SQL, Java, JDBC library

- In the JDBC library, SQL commands are sent as strings to the database server via a connection.
- Using the JDBC, one writes code like the following:
  ```java
  int value = 100;
  Statement stmt = conn.createStatement();
  stmt.execute("select CUST_NAME from " +
                "CUSTOMERS where QUANTITY > " + value);
  ```
- SQL error checking and optimization is done by the server at run-time, not by the compiler at compile-time.
- A better solution: extend Java with SQL language constructs.
Example: SQL embedded in Java

- The above computation might be implemented as:
  ```java
  Int value = 100 ;
  using conn query ( SELECT cust_name FROM customers WHERE quantity > value )
  ```
- Here `on ... execute` is a new statement which takes a database connection and an SQL/Java construct.
- An extended compiler can statically type check the SQL queries.
Other Examples of Language Extensions

- **Condition Tables** found in synchronous languages like RSML-e or SCR - useful for understanding complex boolean expressions.

- Adding code transformations to prevent data degeneracies in geometric primitive tests from the domain of computational geometry.

- Fast unbounded-precision integers of LN.

- Java 1.5 additions – foreach loops, autoboxing, unboxing.

- Concrete syntax for lists, hash maps, etc.
Programming with Language Extensions

Programmer

writes

Host Spec

imports extensions

Language Extensions

writes

SQL

writes

tables

writes

CompGeom

writes

Silver AG Tools

imports extensions

Custom Lang. Compiler

writes

Reduced Code

writes

Program

Language Feature Designers

writes

Silver AG Tools

writes

Custom Lang. Compiler

writes

Reduced Code

writes
Demo of use of language extensions ...
Attribute Grammars for Extensible Languages and Language Extensions

- An extensible host language is a complete language defined as an attribute grammar (AG).
- A language extension extends a host language by defining
  - new productions – to define new constructs
  - new attributes and their definitions – to define a new semantic analysis or translation.

It is defined as an attribute grammar fragment.

- The union of the host language AG and language extension AG fragment defines the extended language.
- The Silver attribute grammar system is used for composing and implementing the AGs.
Attribute Grammars – Attributed ASTs

- AST nodes annotated with attribute values.
- e.g. `for ( et elem : coll ) { body }`

Attributes: `pp` – pretty print (synthesized),
`env` – environment (inherited)
Attribute Grammars (AG)

- An AG is a context free grammar with definitions for attributes that decorate AST nodes:

- For example:

```plaintext
foreach: f::Stmt ::= et::Type elem::Id
c::Expr  b::Stmt

f.errors =
    if c.type.implements(Collection)
    then no-error else mkError ... f.pp ..
b.env = f.env ;
```
Forwarding – an AG extension

- The “forwarded-to” tree is semantically equivalent to the “forwarding” tree.

- It provides attribute definitions not explicitly specified.
Attribute Grammars + forwarding:

- A distinguished tree is specified for each production implementing an extension:
  ```
  foreach: f::Stmt ::= et::Type e::Id c::Expr b::Stmt
  f.pp = "for each " ...
  for.errors = if ... then no-error ...
  forwards to parse
  " 'et' 'e';
    Iterator 'iter' = 'c'.iterator();
    while ( 'iter'.hasNext() ) {
      'e' = ( 'et' ) 'iter'.next();
      'b'
    }
  }
  where iter = gen_sym();
- The unquote operator is written as '_'
Semantic analysis and Transformation

- An extension can perform semantic analysis.

- The for-each or SQL query can do their own error checking and thus generate error messages regarding code the programmer wrote.

- Semantic analysis can be used to determine what to forward to. e.g. computational geometry

- Forwarding mimics rewriting to allows constructs to transform themselves into (efficient) host language constructs.
Embedding SQL into Java

- `import table` - adds type info to env
- `using...query` - adds this to the env of component SQL query
- Identifiers in SQL examine env to get their type
  - Raise errors if not found
- SQL constructs (`select`, `>=` in `where` clauses)
  - Type check their arguments
  - Generate target Java/JDBC code in higher-order attribute
  - This is forwarded-to by `using...query` construct
Embedding SQL into Java

prod sqlQ e::Expr ::= 'using' c::Id 'query'

   '{' sq::SQL '}'

{  e.pp = "using" ++ ... ++ sq.pp ++ ... ;
  e.typerep = ... Java class ResultSet ... ;
  e.errors = sq.errors ;
  forwards to
       
     '``|c.lexeme|:createStatement().
        executeQuery( |sql.javaExpr| ) ''

}

prod select sq::SQL ::= 'select' f::SqlExpr 'from'
                          table::SqlExpr 'where' w::SqlExpr

{  sq.javaExpr = "``select" + |f.javaExpr| +
     ... ''
Comments

- Not all language enhancements can be implemented as modular language extensions:
  - Adding generics from Java 1.5 to an extensible Java compiler – possible?
  - Adding garbage collection to an extensible C++.
  - Fundamental changes to the core of the language.

- The extensions we write have translations to constructs in the host language.

- May enable incremental adoption.

- Verify composability of extensions … syntactic, semantic …
More information

- evw@cs.umn.edu
- www.cs.umn.edu/~evw
- www.cs.umn.edu/melt

- Thanks to Lijesh Krishnan, Phil Russell, Eric Johnson, August Schwerdfeger

... Thanks for your attention.

- Thanks to National Science Foundation, IBM, and the McKnight Foundation.