

A TASTE OF STRINGTEMPLATE

Terence Parr
University of San Francisco



Netflix 8/31/09

WHAT WE'LL COVER

- What is ST
- Why we need to separate logic from display mechanisms
- What ST looks like
- ST characteristics
- Localization, altering the look, formatting data (w/o resorting to arbitrary code)
- Goals and lessons learned

WHAT IS ST?

- Evolved from simple “document with holes” into a functional language* that strictly *enforces model-view separation*
- Well suited to code generation & dynamic page generation
- ST is a lightweight library (not a tool or server) with two key classes: **StringTemplate** and **StringTemplateGroup**

*To my great surprise!

DISENTANGLING
BUSINESS LOGIC FROM
DISPLAY

LIFE BEFORE TEMPLATES

Servlet

```
out.println("<html>");
out.println("<body>");
out.println("<h1>Servlet test</h1>");
if ( request.getParameter("user").equals("root") ) {
    out.println("Hello, "+request.getParameter("name")+".");
}
out.println("</body>");
out.println("</html>");
```

JSP

Entangles model and view

```
<html>
<body>
<h1>JSP Test</h1>
<%if ( request.getParameter("user").equals("root") ) {%>
Hello, <%=request.getParameter("name")%>.
<%}%>
</body>
</html>
```

SAMPLE ENTANGLLEMENTS

(DON'T DO THIS)

- `$if(user=="parrrt" && machine=="yoda")$`
- `$price*.90$, $bloodPressure > 130$`
- `$a=db.query("select subject from email")$`
- `$model.pageRef(getURL())$`
- `$ClassLoader.loadClass(somethingEvil)$`
- `$names[ID]$`
- Don't send in output text either: `st.setAttribute("color", "Red");`

SEPARATION MOTIVATION

- *Encapsulation* (separation of concerns, isolates business logic)
- *Clarity* (say what the output looks like, not program+output)
- *Division of labor* (graphics designer works in parallel with developer)
- *Component reuse* (w/o logic, can reuse)
- *Single point-of-change* (e.g., change single template to change every link style)
- *Maintenance* (e.g., don't have to hunt down SQL all over templates)
- *Interchangeable views* (supports site "skins", re-targetable code generators)
- *Security* (E.g., no way for user to make template with infinite loop)

PROPOSITION

- Separating logic from display is widely accepted as a worthy goal
- *Let's enforce it!*
- Competitors only encourage separation, fearing that enforcement will result in fatal weakness
- But, I have formally shown restricted templates generate a sufficiently large language class (E.g., any xml doc with a DTD)
- So, why encourage when we can enforce? (anecdotes: Murphy, IBM keyboard)
- Without enforcement, developers exploit loopholes, encoding logic in templates
- Other engines simply reinvent JSP but with a new language to learn

SEPARATION RULES

- The view cannot modify the model
- Cannot perform computations upon model data values
- Cannot compare model data values
- Cannot make type assumptions
- Data from model cannot contain display, layout information

WHAT ST LOOKS LIKE

CANONICAL OPERATIONS

- Attribute reference:
`$name$`
- Template references (possibly recursive):
`$searchbox()$`
- Apply template to multi-valued attribute:
`$names:bold()$` or
`$users:{u | Hi, $u.name$. Your user ID is $u.id$.}$`
- Conditional include:
`$if(superuser)$You have superpowers.$endif$`

PHILOSOPHICAL DIFFERENCE

Velocity
*JSP with a
different
syntax*

```
<table>  
#foreach( $mud in $mudsOnSpecial )  
  #if ( $customer.hasPurchased($mud) )  
    <tr><td>$flogger.getPromo($mud)</td></tr>  
  #end  
#end  
</table>
```

It's a program+template

Entangled!

ST

```
<table>  
$promotions:{p | <tr><td>$p$</td></tr>}$  
</table>
```

Or

```
<table>  
$promotions()$  
</table>
```

*ST templates only say how
to display, not how to
compute or obtain data.*

FEEDING DATA TO TEMPLATES

- Push don't pull: compute all data then inject
- Pulling data from template can violate separation and can be slower
- The order in which you pull data matters
 - if a_i depends on a_j must fetch a_j before a_i
 - Example: `$model.getNames()` and `$model.getNumberOfNames()`
 - must not request attributes in order violating dependency graph
 - *nothing enforces order*; programmers have to remember, designers won't even know there's an issue

CHARACTERISTICS

- *Dynamically-typed*
 - toString() converts objects to text
 - For, x.y ST calls getY() on x via reflection
- *Pure functional* -- no side-effects, order independent eval
- *Dynamic scoping* -- can refer to attr in enclosing templates
- *Lazy evaluation* -- decouples output order from computations

LAZY EVALUATION

- ST only evaluates the expressions that it has to
- Defers execution until all attributes and embedded templates are available
- We can construct template tree and inject attributes in any order without fear of “premature evaluation”
- Python analogy:

```
>>> e = 'x'+'*'+ '10' # ref x before it's defined
>>> x=3 # must define x before evaluating "x*10"
>>> print eval(e)
30
```

CODE GENERATION

- Template groups are like output grammars
- Set of mutually-referential templates with formal arguments
- Group inheritance and polymorphism

```
group javaTemplates;  
  
method(type, name, args, body) ::= <<  
public <type> <name>( <args:arg(); separator=","> ) {  
    <body>  
}  
>>  
  
assign(lhs, expr) ::= "<lhs> = <expr>;"  
if(expr, stat) ::= "if (<expr>) <stat>"  
call(name, args) ::= "<name>( <args; separator=","> );"  
...
```

EDITING SQL TEMPLATES IN ANTLRWORKS



```
/Users/parrr/tmp/SQL2.stg

Rules
column
columnForSingleValuedField
objectTables
output
tableForMultiValuedField

group SQL2;

javaToSQLTypeMap ::= [
    "int":"INTEGER", // "int" maps to "INTEGER"
    "String":"TEXT",
    "float":"DOUBLE",
    "double":"DOUBLE",
    "Date":"DATETIME",
    default : key // If not found, yield key; don't map
]

/** Common interface to templates for this target */
output(class, fields, arrayFields, nonPrimitiveTypes) ::=
    "<objectTables()>"

/** Inherit class, fields, arrayFields from output template */
objectTables() ::= <<
CREATE TABLE <class.simpleName> (
    ID INTEGER NOT NULL UNIQUE PRIMARY KEY,
    <fields:columnForSingleValuedField(); separator=",\n">
);
<arrayFields:tableForMultiValuedField()>
>>

columnForSingleValuedField(f) ::= "<column(name=f.name, javaType=f.type)>"
```

ALTERING THE “LOOK”

MULTIPLE SITE DESIGNS

- Problem: multiple skins w/o cut/paste
- Observation: much HTML in common, inheritance “feels” right
- ST supports template inheritance from supergroup down to subgroup and template polymorphism
- Design new site as it differs from existing
- Example: jGuru.com premium vs guest

LOCALIZATION

- *Problems:*
 - Text in various languages, site-per-locale issues
 - Display data (e.g., dates, integers) differently
10/01/06: “Oct 1, 2006” vs “Jan 10, 2006”
- How to achieve without:
 - arbitrary actions in view (templates)
 - cutting pasting; don't break single-point-of-change principle!
 - unrealistic expectations of translators, designers
- *Answer: automate it, don't code it!*

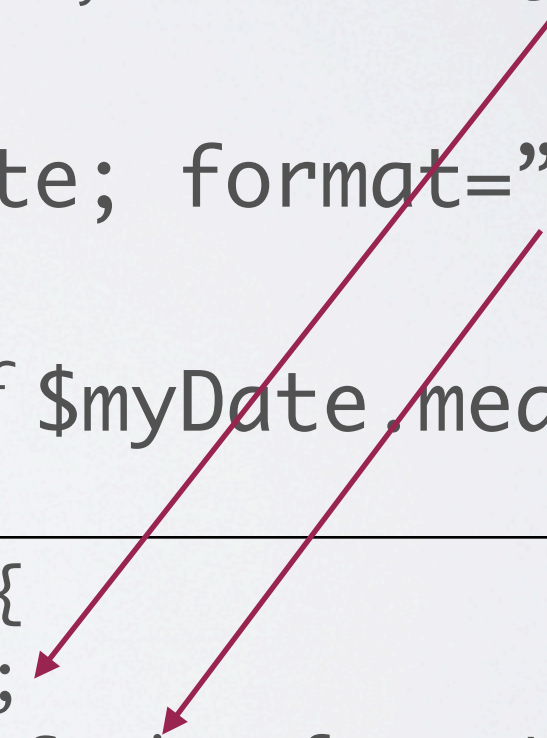
LOCALIZING TEXT STRINGS

- Avoid: `$strings.get("title")` and `<fmt:message key="title"/>`
 - too easy to do `$strings.remove("title")`
 - HTML folks won't grok
 - requires cut-n-paste (evil)
- Idea: treat text as data, pull from database or property files, inject as dictionary, ref as `$strings.title$`
- Translators can work in isolation and in parallel
- Example using ST: <http://www.schoolloop.com>

FORMATTING DATA

- Velocity uses DateTool: `$date.format('medium', $myDate)`
- To avoid arbitrary code, *automate!*
- Choice 1: register renderer for Date objects, ref `$myDate$`
- Choice 2: register renderer, ref `$myDate; format="medium"$`
- Choice 3: auto-wrap Date objects, ref `$myDate.medium$`

```
public interface AttributeRenderer {  
    public String toString(Object o);  
    public String toString(Object o, String formatName);  
}
```



LOOKING BACK

DESIGN GOALS

- Optimized for enforcement of separation not Turing completeness
- Conceptual integrity
 - single-minded fanaticism against entanglement
 - two primary concepts: attributes and templates
 - consistency; no special cases
 - proper language formalisms, semantics, sane syntax
- Simple as possible -- accessible to nonprogrammers, fast, easy to maintain
- Powerful as possible

LESSONS LEARNED

- Language design is hard; implementation is easy in comparison
- Focus on principles and conceptual integrity
- Let the problem dictate the solution
 - Avoid preconceived notions about what solution looks like
 - Evaluate proposed features in light of design goals, use cases
- Don't let implementation details drive design
- Difficult to stick to your principles and still create usable tool

SUMMARY

- ST's distinguishing feature: enforcing model-view separation
- Tiny, moderately simple, but expressive functional language
- Good at web page code generation
- Handles multiple site looks, localization, retargetable generators naturally
- Broad usage around the world; all the cool kids are doing it