Checking Framework Interactions with Relationships

Ciera Jaspan

Advised by Jonathan Aldrich

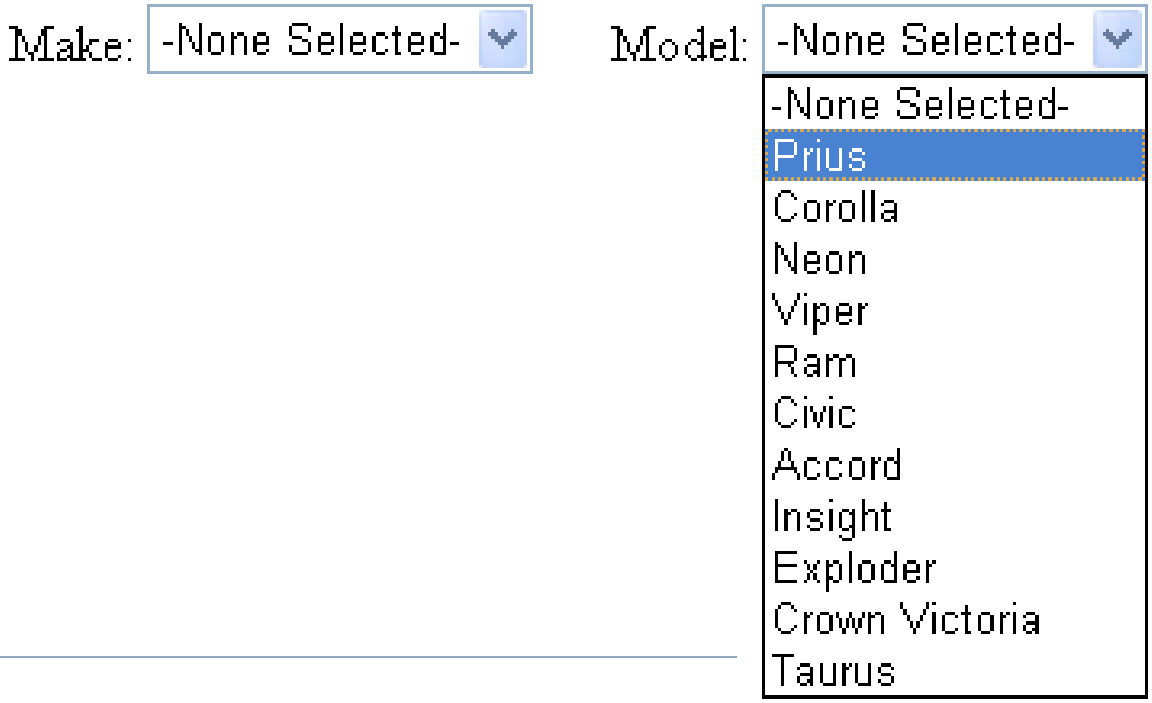
OOPSLA Doctoral Symposium

DropDownList

- Can add drop down lists to a web page
- Can change the selection programmatically
- Only one item is selected at a time

DropDownList

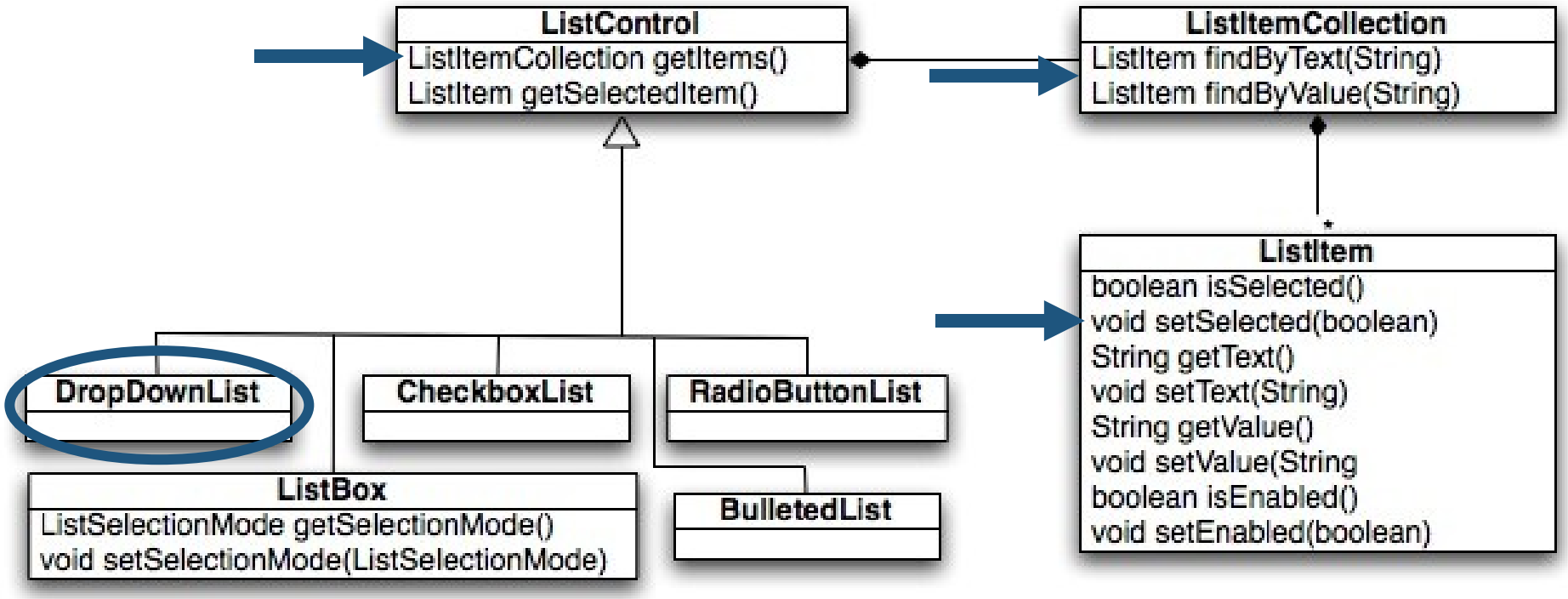
- Can add drop down lists to a web page
- Can change the selection programmatically
- Only one item is selected at a time



DropDownList

- Can add drop down lists to a web page
- Can change the selection programmatically
- Only one item is selected at a time

Class diagram



Let’s change the selection

```
String searchTerm = &;
ListItem newItem;
DropDownList ctrl = getControl(  myList  );




newItem = ctrl.getItems().findByValue(searchTerm);
newItem.setSelected(true);
```

Cannot have multiple items selected in a DropDownList.

Stack Trace:

```
[HttpException (0x80004005): Cannot have multiple items selected in a DropDownList.]
   System.Web.UI.WebControls.DropDownList.VerifyMultiSelect() +133
   System.Web.UI.WebControls.ListControl.RenderContents(HtmlTextWriter writer) +206
   System.Web.UI.WebControls.WebControl.Render(HtmlTextWriter writer) +43
   System.Web.UI.Control.RenderControlInternal(HtmlTextWriter writer, ControlAdapter adapter) +74
   System.Web.UI.Control.RenderControl(HtmlTextWriter writer, ControlAdapter adapter) +291
```

Correct code

```
String searchTerm = &;
ListItem newItem, oldItem;
DropDownList ctrl = getControl(  myList  );

oldItem = ctrl.getSelectedItem();
oldItem.setSelected(false);

newItem = ctrl.getItems().findByValue(searchTerm);
newItem.setSelected(true);
```

- Must `setSelected` to `false` before setting to `true`
- `oldItem` and `newItem` constrained only in `DropDownList`
- `oldItem` and `newItem` members of the same `DropDownList`

Motivating example: LoginView

- Can specify different controls to be shown when a user is logged in
  - Ex: username and password fields v. “Welcome, Username!”

```
<asp:LoginView ID=  LoginScreen   runat=  server  >
   <AnonymousTemplate>
      You can only setup accounts when you are logged in.
   </AnonymousTemplate>
   <LoggedInTemplate>
      <h4>Location</h4>
      <asp:DropDownList ID=  LocationList   runat=  server  />
      <asp:Button ID=  ChangeButton   runat=  server   Text=  Change
   </LoggedInTemplate>
</asp:LoginView>
```

Retrieve controls and set them up

```
LoginView LoginScreen;

private void Page_Load(object sender, EventArgs e) {

   DropDownList list = (DropDownList)
    LoginScreen.FindControl(  LocationList  );

   list.DataSource = &;
   list.DataBind();
}
```

NullReferenceException at list.DataSource = ...;

Correct code

```
LoginView LoginScreen;

private void Page_Load(object sender, EventArgs e) {
   if (this.getRequest().IsAuthenticated()) {
      DropDownList list = (DropDownList)
       LoginScreen.FindControl(  LocationList  );

      list.DataSource = &;
      list.DataBind();
   }
}
```

- DropDownList called LocationList must exist inside of LoggedInTemplate in the associated ASPX file
- Request must be the Request object for the page that LoginScreen is on
- Can only make call when IsAuthenticated is true

Problem characteristics

- Multiple objects and multiple classes
- Non-local constraints
  - Class A may add additional constraints to class B when used together
- Semantic properties
  - Temporal: ordering matters
  - Identity: how objects are related matters
  - Values: runtime values of objects matter
  - Not just syntactic concerns! (but it plays a part)
- Spans multiple kinds of data
  - ASP.NET: ASPX, C#
  - Eclipse: Java, XML
  - Spring: Java, JSP, XML, Annotations, Properties files
  - OpenSpeedShop: C++, XSD, Makefiles

Thesis statement

Frameworks impose constraints on plugins that are about how **muliple objects** may interact. Framework constraints are **non-local** and are dependent on the **identities** of objects, their **values**, and the **order of operations** used by the plugin.

These constraints can be defined by specifications that describe the **relationships** between objects and how these relationships change. A branch sensitive, dataflow analysis can **statically** check a plugin's conformance to these

framework/plugin interactions

- Plugin developers typically don’t make new data; they modify existing objects
  - Fields initialized by dependency injection
  - Few constructor calls; use abstract factories instead
- Plugin is provided with some given state in the callback
- Plugin may request modifications by calling framework operations

knowledge

- Every operation gives a plugin developer additional knowledge
  - Method call changes relationships between parameters, calling object, and return value (and sometimes objects no longer in caller's scope!)

```
oldItem = ctrl.getSelectedItem();
//we know oldItem is a child of ctrl
//we know oldItem is selected
oldItem.setSelected(false);
//we know oldItem not is selected
```

  - Branch test provides knowledge of the objects relationships with each other

```
if (list.contains(val)) {
   //we know val is an item of list
}
```

Relationships as Interaction Model

- Can add and remove relationships and change on branch
- Can track this knowledge in the plugin

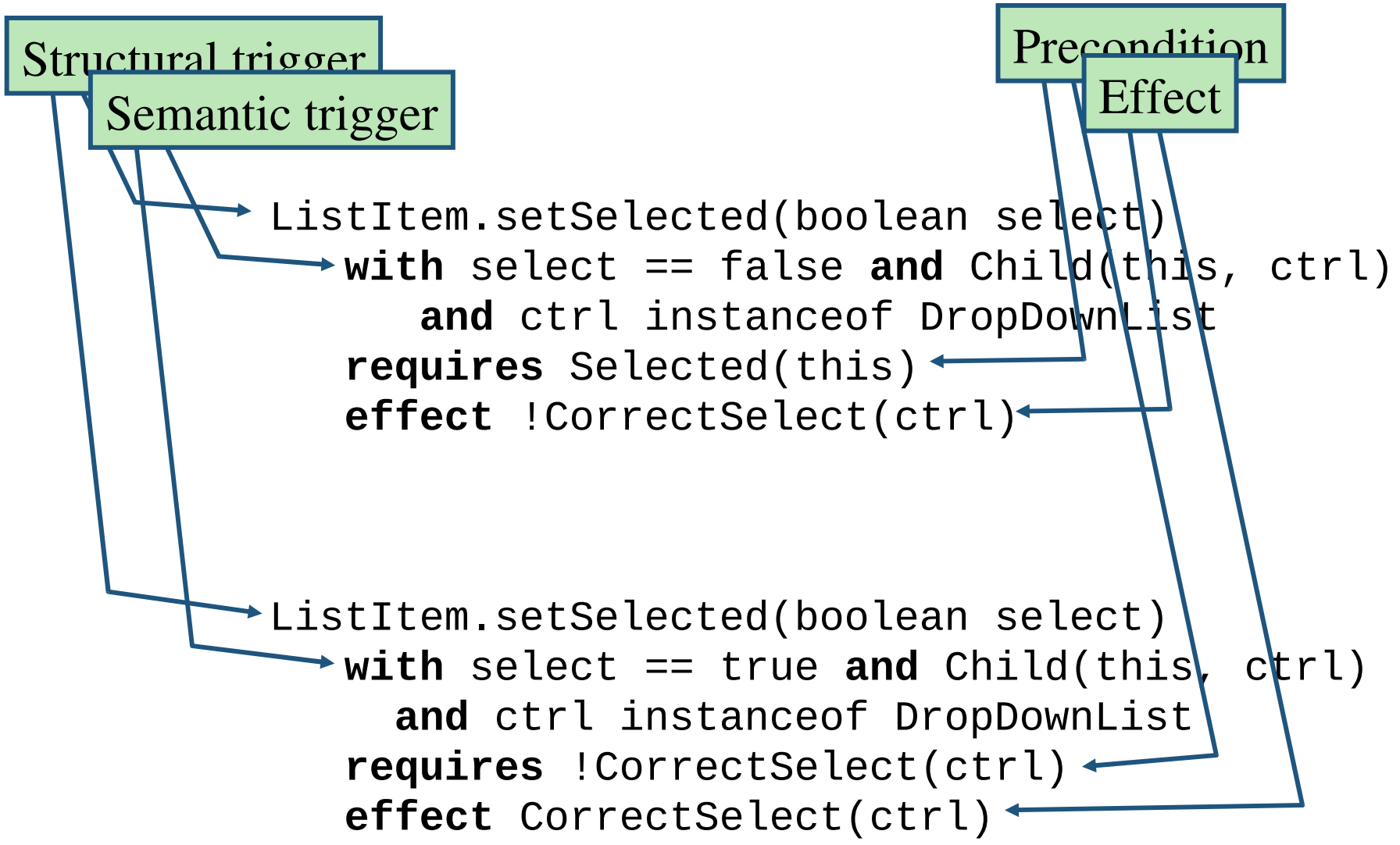
```
public class ListControl {
   [Child(ret, this)]
   [Selected(ret)]
   ListItem getSelectedItem();

   [Items(ret, this)]
   ListItemCollection getItems();

}
```

User-defined meaning

- Captures developer design intent
- `Child` has no pre-defined architectural meaning

```
old.setSelected(false);
{Control(ctrl, this), Child(old, ctrl), !Selected(old)}
```

Relationship based constraints



Checkable by a static analysis

- Requires only annotations at framework interface
- Dataflow analysis runs on plugin code
  - Tracks state of all possible relationships given accessible objects
  - Branch sensitive; uses a constant analysis
- Aliasing is done by any points-to analysis
  - Depends on a simple interface
  - Precision of points-to analysis impacts main analysis

Walkthrough: good example

```
{
String searchTerm = &;
ListItem newItem, oldItem;
ListItemCollection coll;
DropDownList ctrl = getControl(  myList  );

oldItem = ctrl.getSelectedItem();


oldItem.setSelected(false);


coll = ctrl.getItems();


newItem = coll.findByValue(searchTerm);



newItem.setSelected(true);



...
}
```

```
[Child({return, this})]
[Name({return, name})]
Control getControl(String name);
```

Walkthrough: good example

```
{
String searchTerm = &;
ListItem newItem, oldItem;
ListItemCollection coll;
DropDownList ctrl = getControl(  myList  );
{Child(ctrl, this), Name(ctrl,   myList  )}
oldItem = ctrl.getSelectedItem();


oldItem.setSelected(false);


coll = ctrl.getItems();


newItem = coll.findByValue(searchTerm);



newItem.setSelected(true);



...
}
```

```
[Child({return, this})]
[Selected({return})]
Control getSelectedItem();
```

Walkthrough: good example

```
{
String searchTerm = &;
ListItem newItem, oldItem;
ListItemCollection coll;
DropDownList ctrl = getControl(  myList  );
{Child(ctrl, this), Name(ctrl,   myList  )}
oldItem = ctrl.getSelectedItem();
{Child(ctrl, this), Name(ctrl,   myList  ),Child(oldItem, ctrl),
Selected(oldItem)}
oldItem.setSelected(false);


coll = ctrl.getItems();


newItem = coll.findByValue


newItem.setSelected(true);


...
}
```

```
ListItem.setSelected(boolean select)
   with select == false and Child(this, ctrl)
      and ctrl instanceof DropDownList
   requires Selected(this)
   effect !CorrectSelect(ctrl)
```

```
[Selected({this}, TEST, select)]
void setSelected(booelan select);
```

Walkthrough: good example

```
{
String searchTerm = &;
ListItem newItem, oldItem;
ListItemCollection coll;
DropDownList ctrl = getControl(  myList  );
{Child(ctrl, this), Name(ctrl,   myList  )}
oldItem = ctrl.getSelectedItem();
{Child(ctrl, this), Name(ctrl,   myList  ), Child(oldItem, ctrl),
Selected(oldItem)}
oldItem.setSelected(false);
{Child(ctrl, this), Name(ctrl,   myList  )Child(oldItem, ctrl), !
Selected(oldItem), !CorrectlySelected(ctrl)}
coll = ctrl.getItems();


newItem = coll.findByValue(searchTerm);



newItem.setSelected(true);



...
}
```

```
[Items({return, this})]
ListItemCollection getItems();
```

Walkthrough: good example

```
{
String searchTerm = &;
ListItem newItem, oldItem;
ListItemCollection coll;
DropDownList ctrl = getControl(  myList  );
{Child(ctrl, this), Name(ctrl,   myList  )}
oldItem = ctrl.getSelectedItem();
{Child(ctrl, this), Name(ctrl,   myList  ), Child(oldItem, ctrl),
Selected(oldItem)}
oldItem.setSelected(false);
{Child(ctrl, this), Name(ctrl,   myList  ), Child(oldItem, ctrl), !
Selected(oldItem), !CorrectlySelected(ctrl)}
coll = ctrl.getItems();
{Child(ctrl, this), Name(ctrl,   myList  ), Child(oldItem, ctrl), !
Selected(oldItem), !CorrectlySelected(ctrl), Items(coll, ctrl)}
newItem = coll.findByValue(searchTerm);



newItem.setSelected(true);



...
}
```

```
[Items({return, this})]
ListItemCollection getItems();
```

Walkthrough: good example

```
{
String searchTerm = &;
ListItem newItem, oldItem;
ListItemCollection coll;
DropDownList ctrl = getControl(  myList  );
{Child(ctrl, this), Name(ctrl,   myList  )}
oldItem = ctrl.getSelectedItem();
{Child(ctrl, this), Name(ctrl,   myList  ), Child(oldItem, ctrl),
Selected(oldItem)}
oldItem.setSelected(false);
{Child(ctrl, this), Name(ctrl,   myList  ), Child(oldItem, ctrl), !
Selected(oldItem), !CorrectlySelected(ctrl)}
coll = ctrl.getItems();
{Child(ctrl, this), Name(ctrl,   myList  ), Child(oldItem, ctrl), !
Selected(oldItem), !CorrectlySelected(ctrl), Items(coll, ctrl)}
newItem = coll.findByValue(searchTerm);



newItem.setSelected(true);



...
}
```

```
[Item({return, this})]
[Value({return, val})]
ListItem findByValue(String val);
```

```
if: Item({item, coll}) and Items(coll, ctrl)
infer: Child(item, ctrl)
```

Walkthrough: good example

```
{
String searchTerm = &;
ListItem newItem, oldItem;
ListItemCollection coll;
DropDownList ctrl = getCon
{Child(ctrl, this), Name(c
oldItem = ctrl.getSelected
{Child(ctrl, this), Name(ctrl,   myList  ), Child(oldItem, ctrl),
Selected(oldItem)}
oldItem.setSelected(false);
{Child(ctrl, this), Name(ctrl,   my
Selected(oldItem), !CorrectlySelect
coll = ctrl.getItems();
{Child(ctrl, this), Name(ctrl,   myList  ), Child(oldItem, ctrl), !
Selected(oldItem), !CorrectlySelected(ctrl), Items(coll, ctrl)}
newItem = coll.findByValue(searchTerm);
{Child(ctrl, this), Name(ctrl,   myList  ),Child(oldItem, ctrl), !
Selected(oldItem), !CorrectlySelected(ctrl), Items(coll, ctrl), Item(newItem,
coll), Child(newItem, ctrl)}
newItem.setSelected(true);



...
}
```

```
ListItem.setSelected(boolean select)
   with select == true and Child(this, ctrl)
      and ctrl instanceof DropDownList
   requires !CorrectSelect(ctrl)
   effect CorrectSelect(ctrl)
```

```
[Selected({this}, TEST, select)]
void setSelected(booelan select);
```

Walkthrough: good example

```
{
String searchTerm = &;
ListItem newItem, oldItem;
ListItemCollection coll;
DropDownList ctrl = getControl(  myList  );
{Child(ctrl, this), Name(ctrl,   myList  )}
oldItem = ctrl.getSelectedItem();
{Child(ctrl, this), Name(ctrl,   myList  ), Child(oldItem, ctrl),
Selected(oldItem)}
oldItem.setSelected(false);
{Child(ctrl, this), Name(ctrl,   myList  ), Child(oldItem, ctrl), !
Selected(oldItem), !CorrectlySelected(ctrl)}
coll = ctrl.getItems();
{Child(ctrl, this), Name(c
Selected(oldItem), !Correc
newItem = coll.findByValue
{Child(ctrl, this), Name(c
Selected(oldItem), !Correc
coll), Child(newItem, ctrl)}
newItem.setSelected(true);
{Child(ctrl, this), Name(ctrl,   myList  )Child(oldItem, ctrl), !
Selected(oldItem), CorrectlySelected(ctrl), Items(coll, ctrl), Item(newItem,
coll), Child(newItem, ctrl), Selected(newItem)}
...
}
```

```
End-of-method
   with ctrl instanceof DropDownList
   requires CorrectSelect(ctrl)
   effect -
```

Walkthrough: incorrect example

```
{
String searchTerm = &;
ListItem newItem, oldItem;
ListItemCollection coll;
DropDownList ctrl = getControl(  myList  );
{Child(ctrl, this), Name(ctrl,   myList  )}
coll = ctrl.getItems();
{Child(ctrl, this), Name(ctrl,   myList  ), Items(coll, ctrl)}
newItem = coll.findByValue(searchTerm);
{Child(ctrl, this), Name(ctrl,   myList  ),Items(coll, ctrl), Item(newItem,
coll), Child(newItem, ctrl)}
newItem.setSelected(true);
...
}
```

```
ListItem.setSelected(boolean select)
   with select == true and Child(this, ctrl)
      and ctrl instanceof DropDownList
   requires !CorrectSelect(ctrl)
   effect CorrectSelect(ctrl)
```

Soundness vs. Completeness vs. Compromise

- Which variant is more cost effective?
- Sound variant requires sound aliasing analysis
- Complete variant requires complete aliasing analysis
- Compromise variant adds two features to make it more expressive

Closest Related Work

- Typestate-based solutions
  - Easier for abstract class invariants
  - Harder for non-local constraints
  - Treating framework callback as a state transition is unweildy
  - Aims to be statically sound
- Tracematch-based solutions
  - Easier for describing a single known bad path
  - Harder when there are multiple bad paths
  - Requires a state machine for each bad path
  - Relationships raise the level of abstraction
  - Not branch sensitive

Validation

- Formal description and motivating examples
  - Gather examples from internet forums
  - Determine the situations which cause false positives and false negatives for each variant
  - Compare expressiveness and annotation cost
- Large case study
  - Specify constraints of the Spring Web App Framework
  - Find real-world code from internet forums
  - Show that the defects would have been caught by the analysis
  - Compare results to existing checkers
  - Extend to associated frameworks

Status and schedule

- Find real-world examples and define problem (done)
- Prototype compromise variant (done)
- Formalize and prove variants (mostly done)
- Full implementation (3 months)
- Propose thesis (2 months)
- Complete Spring case study (6 months)
- Extend case study to more frameworks (3 months)

Questions for the panel

- Any general advice on the form of a thesis about an abstraction (as opposed to a technique)?
- Should I extend this to other kinds of data?
  - JSP files?
  - File configurations?
- Should I generalize to APIs or leave specific to frameworks?
  - Frameworks: control program flow and architecture
  - Libraries: assume client controls program flow and architecture

This slide intentionally left blank.

Hypotheses 1-3

1. Relationships can be used to express the kinds of constraints that frameworks employ on plugins.
2. These constraints are statically checkable in a way that emphasizes the quality of the results; the false positives signal poorly written code and the false negatives occur from highly unlikely scenarios.
3. When the static analysis discovers a broken constraint, it directs the developer to the root cause of the error rather than the location where

Hypotheses 4-6

1. Relationship specifications can express interactions between multiple languages and files.
2. Relationship specifications lower the cost of creating framework-specific checkers for plugins.
3. Relationship specifications allow developers to compose the specifications of different frameworks and interactions between them that plugins must be aware of.

Less Related Work

- Structural constraints
  - Describe constraints about code structure
  - Can not express semantic constraints
- Design pattern and description based solutions
  - Describe the interactions or task
  - Formalized documentation
  - Usually requires input from plugin developer
  - Useful for beginners, but does not scale well
- Formal methods
  - Describe the constraint with logic
  - Prove that it is not broken
  - Heavyweight solution

Solution goals

- Capture developer design intent
  - Model relationships, including their constraints and effects
- Direct plugin developers to cause of error in plugin
- Low cost
  - No specifications for plugin developer
  - Few and conscice specifications for framework developer
  - Incremental gain on specifications written

Alternate validation ideas...

- Experiment with the Web Apps class