SOLVING THE FAMILIES TO PERSONS CASE USING EVL + Strace

Model-Driven Software Engineering Research Group

Department of Software Engineering
University of Isfahan
Leila Samimi-Dehkordi
samimi@eng.ui.ac.ir
Bahman Zamani
zamani@eng.ui.ac.ir
Shekoufeh Kolahdouz-Rahimi
sh.rahimi@eng.ui.ac.ir
SOLUTION OVERVIEW

EVL+Strace

Epsilon Validation Language (EVL) + Specific trace metamodel (Strace)

Source Metamodel

Target Metamodel

http://lsamimi.ir/EVLStrace.htm
EVL+Strace

- EVL expresses constraints between heterogeneous models and evaluates them to resolve the occurred violations.

- An EVL constraint contains two main parts including check and fix blocks.

- The Epsilon Object Language (EOL) specifies the checking expressions and fixing statements.

- The defined constraints in EVL+Strace are applied on the elements of three metamodels including source, specific trace, and target.

- A specific trace metamodel consists of strongly typed trace links.

- EVL+Strace can recognize four atomic updates including element deletion, addition, relocation, and value modification.
SPECIFIC TRACE METAMODEL (SOLUTION)
THREE CONSISTENT MODELS
TRANSFORMATION CODE

```java
context Families2Persons!FamilyMemberSourceEnd{
    guard: not self.isRemoved() and not self.refFamilyMember2Persons.endTypeIsRemoved()
    constraint nameIsModified{
        check: not self.nameIsModified()
        message: 'name of '+self+' is modified'
        fix{
            title: 'Propagate the modification'
            do{ self.namePropagates();}
        }
    }
}
```
TRANSFORMATION CODE

context

```
context Families2Persons!FamilyMemberSourceEnd{

guard: not self.isRemoved() and not self.refFamilyMember2Persons.endTypeIsRemoved()

constraint nameIsModified{

check: not self.nameIsModified()

message: 'name of '+self+' is modified'

fix{

title: 'Propagate the modification'

do{ self.namePropagates(); }

}

}
```

- **Conditions for guard**
- **Conditions for check block**
- **Message shows a modification is occurred**
- **Statements for fix block**

**Constraint for recognizing modification**

**EOL operation propagate the recognized modification**
EXECUTION (INTERACTIVE FIXING)
INTERACTIVE VS. AUTOMATIC EVL+Strace

```
constraint Foo {
    check : <some expression>
}

constraint Foo {
    check {
        var condition = <some expression>;
        if (condition == false) {
            // Code that automatically fixes the problem
        }
        return true;
    }
}
```

http://lsamimi.ir/MoDEBiTE.htm
To test the approach, we use auto-fix code for EVL+Strace
EUnit and Workflow tools of the Epsilon framework are used for testing.
It is possible to automatically produce the trace metamodel and some parts of transformation code with the use of our toolkit, MoDEBiTE.
### RESULTS

<table>
<thead>
<tr>
<th>#</th>
<th>direction</th>
<th>Policy</th>
<th>Change Type</th>
<th>Test Case Name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fwd</td>
<td>fixed</td>
<td>-</td>
<td>testInitialSynchronisation</td>
<td>expected pass</td>
</tr>
<tr>
<td>2</td>
<td>fwd</td>
<td>fixed</td>
<td>attribute</td>
<td>testFamilyNameChangeOfEmpty</td>
<td>expected pass</td>
</tr>
<tr>
<td>3</td>
<td>fwd</td>
<td>fixed</td>
<td>add</td>
<td>testCreateFamily</td>
<td>expected pass</td>
</tr>
<tr>
<td>4</td>
<td>fwd</td>
<td>fixed</td>
<td>add</td>
<td>testCreateFamilyMember</td>
<td>expected pass</td>
</tr>
<tr>
<td>5</td>
<td>fwd</td>
<td>fixed</td>
<td>add</td>
<td>testNewFamilyWithMultiMembers</td>
<td>expected pass</td>
</tr>
<tr>
<td>6</td>
<td>fwd</td>
<td>fixed</td>
<td>add</td>
<td>testNewDuplicateFamilyNames</td>
<td>expected pass</td>
</tr>
<tr>
<td>7</td>
<td>fwd</td>
<td>fixed</td>
<td>add</td>
<td>testDuplicateFamilyMemberNames</td>
<td>expected pass</td>
</tr>
<tr>
<td>8</td>
<td>bwd</td>
<td>runtime</td>
<td>add ($e \land p$)</td>
<td>testCreateMalePersonAsSon</td>
<td>expected pass</td>
</tr>
<tr>
<td>9</td>
<td>bwd</td>
<td>runtime</td>
<td>add ($e \land \neg p$)</td>
<td>testCreateMembersInExistingFamilyAsParents</td>
<td>expected pass</td>
</tr>
<tr>
<td>10</td>
<td>bwd</td>
<td>runtime</td>
<td>add ($e \land \neg p$)</td>
<td>testCreateMalePersonAsSon</td>
<td>expected pass</td>
</tr>
<tr>
<td>11</td>
<td>bwd</td>
<td>runtime</td>
<td>add ($e \land \neg p$)</td>
<td>testCreateMembersInExistingFamilyAsParents</td>
<td>expected pass</td>
</tr>
<tr>
<td>12</td>
<td>bwd</td>
<td>runtime</td>
<td>add ($e \land \neg p$)</td>
<td>testCreateDuplicateMembersInExistingFamilyAsChildren</td>
<td>expected pass</td>
</tr>
<tr>
<td>13</td>
<td>bwd</td>
<td>runtime</td>
<td>add ($\neg e \land p$)</td>
<td>testCreateMalePersonAsParent</td>
<td>expected pass</td>
</tr>
<tr>
<td>14</td>
<td>bwd</td>
<td>runtime</td>
<td>add ($\neg e \land p$)</td>
<td>testCreateMembersInNewFamilyAsParents</td>
<td>expected pass</td>
</tr>
<tr>
<td>15</td>
<td>bwd</td>
<td>runtime</td>
<td>add ($\neg e \land p$)</td>
<td>testCreateDuplicateMembersInNewFamilyAsParents</td>
<td>expected pass</td>
</tr>
<tr>
<td>16</td>
<td>bwd</td>
<td>runtime</td>
<td>add ($\neg e \land p$)</td>
<td>testCreateMalePersonAsSon</td>
<td>expected pass</td>
</tr>
<tr>
<td>17</td>
<td>bwd</td>
<td>runtime</td>
<td>add ($\neg e \land p$)</td>
<td>testCreateFamilyMembersInNewFamilyAsChildren</td>
<td>expected pass</td>
</tr>
<tr>
<td>18</td>
<td>bwd</td>
<td>runtime</td>
<td>add ($\neg e \land p$)</td>
<td>testCreateDuplicateFamilyMembersInNewFamilyAsChildren</td>
<td>expected pass</td>
</tr>
<tr>
<td>19</td>
<td>fwd</td>
<td>fixed</td>
<td>add</td>
<td>testIncrementalInserts</td>
<td>expected pass*</td>
</tr>
<tr>
<td>20</td>
<td>fwd</td>
<td>runtime</td>
<td>del</td>
<td>testIncrementalDeletions</td>
<td>expected pass*</td>
</tr>
<tr>
<td>21</td>
<td>fwd</td>
<td>fixed</td>
<td>attribute</td>
<td>testIncrementalRename</td>
<td>expected pass*</td>
</tr>
<tr>
<td>22</td>
<td>fwd</td>
<td>fixed</td>
<td>move</td>
<td>testIncrementalMove</td>
<td>expected pass*</td>
</tr>
<tr>
<td>23</td>
<td>fwd</td>
<td>fixed</td>
<td>add+del</td>
<td>testIncrementalMixed</td>
<td>expected pass*</td>
</tr>
<tr>
<td>24</td>
<td>fwd</td>
<td>fixed</td>
<td>move</td>
<td>testIncrementalMoveRoleChange</td>
<td>expected pass*</td>
</tr>
<tr>
<td>25</td>
<td>fwd</td>
<td>fixed</td>
<td>-</td>
<td>testStability</td>
<td>expected pass</td>
</tr>
<tr>
<td>26</td>
<td>fwd</td>
<td>fixed</td>
<td>-</td>
<td>testHippocraticness</td>
<td>expected pass</td>
</tr>
<tr>
<td>27</td>
<td>bwd</td>
<td>fixed</td>
<td>add</td>
<td>testIncrementalInsertsFixedConfig</td>
<td>expected pass</td>
</tr>
<tr>
<td>28</td>
<td>bwd</td>
<td>runtime</td>
<td>add</td>
<td>testIncrementalInsertsDynamicConfig</td>
<td>expected pass</td>
</tr>
<tr>
<td>29</td>
<td>bwd</td>
<td>runtime</td>
<td>del</td>
<td>testIncrementalDeletions</td>
<td>failure</td>
</tr>
<tr>
<td>30</td>
<td>bwd</td>
<td>runtime</td>
<td>attribute</td>
<td>testIncrementalRenamingDynamic</td>
<td>expected pass</td>
</tr>
<tr>
<td>31</td>
<td>bwd</td>
<td>runtime</td>
<td>del+add</td>
<td>testIncrementalMixedDynamic</td>
<td>failure</td>
</tr>
<tr>
<td>32</td>
<td>bwd</td>
<td>runtime</td>
<td>add</td>
<td>testIncrementalOperational</td>
<td>expected pass</td>
</tr>
<tr>
<td>33</td>
<td>bwd</td>
<td>runtime</td>
<td>-</td>
<td>testStability</td>
<td>expected pass</td>
</tr>
<tr>
<td>34</td>
<td>bwd</td>
<td>runtime</td>
<td>-</td>
<td>testHippocraticness</td>
<td>expected pass</td>
</tr>
</tbody>
</table>
CONCLUSION

- A bidirectional model-to-model transformation solution to the TTC 2017 Families toPersons case study based on a novel approach named EVL+Strace.

- The trace metamodel (correspondence metamodel) is specific to the domains of the Families and Persons case studies.

- The approach defines constraints to check user updates with the use of EVL.

- It is possible to program more than one fixing ways, and interactively ask user to restore the consistency.

- To test the solution, we change the constraints to fix the violations automatically.

- The evaluation presents that from all 34 test cases, automatic EVL+Strace has 32 expected pass and two failures.

http://lsamimi.ir/EVLStrace.htm