SPARTACAS – Automating Component Adaptation for Reuse

Brandon Morel

Master’s Thesis Defense
University of Kansas
September 12, 2003

Committee:
Dr. Perry Alexander (Chair)
Dr. Susan Gauch
Dr. Costas Tsatsoulis
Introduction

- Reuse is a sound/practical design technique
- Software engineering slow to embrace reuse
- Benefits
  - Reduce errors
  - Increase productivity of engineers
  - Increase reliability/quality of software
- Costs
  - Effort to create/maintain library of components
  - Effort to search for components
  - Effort to adapt partial matches
Problem and Solution

• Problems
  – How to adapt software?
  – Can adaptation be automated?
  – Will the framework be effective?
• Solution: SPARTACAS
• Outline
  – Specification-level representation
  – Adaptation framework
Outline

– Adaptation architectures
– Adaptation method
– Evaluation results
– Future work and limitations
– Related work
– Concluding remarks
Formal Specifications

• Prior success at the specification-level
• Specification formally describe the functionality without implementation details
• DRIO specification models
  – Domain – typed input parameters
  – Range – typed output parameters
  – Input condition – pre-conditions defining legal inputs
  – Output condition – post-conditions defining valid outputs for legal inputs
  – \( I(d) \Rightarrow O(d, r) \)
Background-Retrieval Methods

- **Feature-based Retrieval**
  - Component/problem assigned domain-specific features
  - Matching is based on a similarity threshold
  - Necessary condition

- **Signature-based Retrieval**
  - Syntactic matching of input and output ports
  - Involves currying, type coercion

- **Specification-based Retrieval**
  - Prove logical relationship between components
  - Match lattice used to determine degree of satisfaction
Background-Component Architectures

- Architecture is a collection of interconnected components
- Architecture theory
  - Parameterized specification
  - Specifies the configuration of sub-components in the composition of a system
  - Specifies the relationship between functionality of the system, sub-components
- To solve a problem, instantiate the theory with the problem as the system, components (other architectures) as the sub-components
SPARTACAS Framework

Problem specification → Retrieval

Sub-problems → Adaptation

Exact/partial matching components → Verification

Potential solutions → Solutions

Component Library
Retrieval Framework

- Problem specification
- Similar components
- Similar components
- Exact/partial matching components

Feature-based Retrieval Engine
- Classify spec. with features
- Retrieval = feature matching
- Necessary condition

Signature-based Retrieval Engine
- Port mapping
- May require signature re-ordering, etc
- Necessary condition

Specification-based Retrieval Engine
- Proof of logical relationship
- Determines degree of match
- Time intensive
Specification Match Lattice

Stronger

Weak Plug-in

(I_P \Rightarrow I_C) \land (I_C \land O_C \Rightarrow O_P)

Satisfies

(I_P \Rightarrow I_C) \land (I_P \land O_C \Rightarrow O_P)

Plug-in Pre

I_P \Rightarrow I_C

Plug-in

(I_P \Rightarrow I_C) \land (O_C \Rightarrow O_P)

Weak Plug-in Post

I_C \land O_C \Rightarrow O_P

Plug-in Post

O_C \Rightarrow O_P

“Specification Matching of Software Components”, Zaremski and Wing, 1995
Port Connection Methods

• Bijective Port Connection
  – One-to-one and onto mapping
  – Component must have equal number of ports
  – Factorial number of port combinations
Less Restrictive PCMs

• One-to-one Port Connection
  – Component can have fewer ports than the problem
  – Binomial number of port combinations

• Onto Port Connection
  – Component can have more ports than the problem
  – Exponential number of port combinations
Adaptation Framework

- Architecture Theories
  - Exact/partial matching components
  - Problem specification
- Adaptation Evaluation
  - Missing functionality
  - Architecture (sub)contract
- Architecture Bin
  - Netlist
- Sub-Problem Synthesis
  - Sub-problem
  - Potential solution
  - Architectural Solution Generation
Verification Framework

Domain Features

Classification

Feature-base Verification

Specification-based Verification

Potential solutions

Feature set

Similar solutions

Solutions
Adaptation Architectures

Problem

comp

comp

comp

comp
Sequential Architecture Theory
BEGIN
  // Problem and components
  Problem(D, R, I, O)
  Component_A(D_A, R_A, I_A, O_A)
  Component_B(D_B, R_B, I_B, O_B)

  // Port constraints
  drConstraint1: D ⊆ D_A
  drConstraint2: R_A ⊆ D_B
  drConstraint3: R_B ⊆ R

  // Behavioral constraints
  behConstraint1: ∀d:D | I(d) ⇒ I_A(d)
  behConstraint2: ∀d:D, x:D_B | I(d) ∧ O_A(d, x) ⇒ I_B(x)
  behConstraint3: ∀d:D, y:R_A, r:R | I(d) ∧ O_A(d, y) ∧ O_B(y, r) ⇒ O(d, r)

END Sequential Architecture Theory
Alternative Architecture Theory

BEGIN

// Problem and components
Problem(D,R,I,O)
Component_A(D_A,R_A,I_A,O_A)
Component_B(D_B,R_B,I_B,O_B)

// Port constraints
drConstraint1: D ⊆ D_A
drConstraint2: D ⊆ D_B
drConstraint3: R_A ⊆ R
drConstraint4: R_B ⊆ R

// Behavioral constraints
behConstraint1: ∀d:D |
    (I(d) ⇒ I_A(d)) ∨ (I(d) ⇒ I_B(d))
behConstraint2: ∀d:D, r:R |
    (I_A(d) ∧ O_A(d,r) ⇒ O(d,r)) ∨ (I_B(d) ∧ O_B(d,r) ⇒ O(d,r))

END Alternative Architecture Theory
Parallel Architecture Theory

BEGIN

// Problem and components
Problem (D, R, I, O)
Component_A (D_A, R_A, I_A, O_A)
Component_B (D_B, R_B, I_B, O_B)

// Port constraints
drConstraint1: D \subseteq D_A \cup D_B
drConstraint2: R_A \parallel R_B \subseteq R

// Behavioral constraints
behConstraint1: \forall d_1 \cup d_2 : D |
  I(d_1 \cup d_2) \Rightarrow I_A(d_1) \land I_B(d_2)
behConstraint2: \forall d_1 \cup d_2 : D, r_1 \parallel r_2 : R |
  I(d_1 \cup d_2) \land O_A(d_1, r_1) \land O_B(d_2, r_2) \Rightarrow O(d_1 \cup d_2, r_1 \parallel r_2)

END Parallel Architecture Theory
Post-match Sequential Adaptation

- Component\textsubscript{B} produces the required results for some set of inputs.
- Tactic: find Component\textsubscript{A} that modifies all inputs to allow Component\textsubscript{B} to execute for all legal inputs.
Example #1

Problem P1

if \( x < 0 \) then \( z' = (-1 \times x) + 1 \) \nelse \( z' = (x + 1) \) end if

\[ a \geq 0 \quad a' \geq 0 \quad b' = (a+1) \]

\[ x : \text{real} \quad a : \text{real} \quad a' : \text{real} \quad b : \text{real} \quad z : \text{real} \]
Example #1

Problem P1

if (x < 0) then (z' = ((-1 * x) + 1))
else (z' = (x + 1)) end if

a >= 0
true
b' = (a + 1)
a' = 0
x: real
true
generate
a: real
z: real

x: real
a: real
generate

a: real
pInc
b: real
Synthesis Method

Sequential Architecture Theory
BEGIN

// Problem and components
Problem(D,R,I,O)
Component_A(D_A,R_A,I_A,O_A)
Component_B(D_B,R_B,I_B,O_B)

// Port constraints
drConstraint1: D ⊆ D_A
drConstraint2: R_A ⊆ D_B
drConstraint3: R_A ⊆ R

// Behavioral constraints
behConstraint1: ∀d:D | I(d) ⇒ I_A(d)
behConstraint2: ∀d:D, x:D_B |
I(d) ∧ O_A(d,x) ⇒ I_B(x)
behConstraint3: ∀d:D, y:R_A, r:R |
I(d) ∧ O_A(d,y) ∧ O_B(y,r) ⇒ O(d,r)

END Sequential Architecture Theory
Post-match Sequential Synthesis

- $D_A = D$

- $D_B = R_A \cup \{ d \in D \mid \neg \exists x \in D_A \mid \rho(x) \rightarrow d \}$

- $I_A = \forall d : D \mid I(d)$

- $O_A = \forall d : D, x : D_B, y : \{ r \in R \mid \exists x \in R_B \mid \rho(x) \rightarrow r \}, r : R \mid I_B(x) \land (\neg O_B(x, y) \lor O(d, r))$
Example #1

if (x<0) then (z' = (-1*x + 1))
else (z' = (x + 1)) end if

true

(o_0' >= 0) and
((not (q_0 = o_0' + 1))) or (if
(i_0 < 0) then
(q_0' = (-1*i_0 + 1)) else
(q_0' = (i_0 + 1))
end if))

true

Problem P1

forall q_0 :: real

Component_A

i_0 :: real

o_0 :: real

a :: real

b :: real

pInc

z :: real

a >= 0

b' = (a + 1)
if \( x < 0 \) then \( z' = ((-1 \times x) + 1) \)
else \( z' = (x + 1) \) end if
Pre-match Sequential Adaptation

- Component\textsubscript{A} accepts the legal inputs, but does not produce valid outputs
- Tactic: find Component\textsubscript{B} that modifies all outputs such that they are valid outputs
Pre-match Sequential Synthesis

- $D_B = R_A \cup \{ d \in D \mid \neg \exists x \in D_A \mid \rho(x) \rightarrow d \}$
- $R_B = R$
- $I_B = \forall d : \{ x \in D \mid \exists y \in D_A \mid \rho(y) \rightarrow x \}, z : R_A \mid I(d) \wedge O_A(d, z)$
- $O_B = \forall d : D, r : R \mid O(d, r)$
Post-match Alternative Adaptation

- Component\textsubscript{A} computes valid outputs for some set of inputs
- Tactic: find Component\textsubscript{B} that computes valid outputs for the rest of the inputs
Post-match Alternative Synthesis

- $D_B = D$
- $R_B = R$
- $I_B = \forall d : D \mid I(d) \land \neg I_A(d)$
- $O_B = \forall d : D, r : R \mid O(d, r)$
Example #1

Problem P1

if \( x < 0 \) then \( z' = ((-1 \times x) + 1) \) else \( z' = (x + 1) \) end if

\( a \geq 0 \)

\( b' = (a + 1) \)

if \( y' = i_0 \) else \( x' = i_0 \) end if

true and not \( (i_0 \geq 0) \)

\( i_0 \) is real

Component B

Problem P1
Example #1

if (x<0) then (z' =((-1)*x)+1)) else (z'=(x+1)) end if

if (i_0<0) then (o_0'=i_0) else (o_1'=i_0) end if
Parallel Adaptation

• Bottom-up behavioral adaptation
  – Find one component, build dynamic adaptation architecture

• Top-down behavioral adaptation
  – Decompose problem into architecture, find components

• Parallel adaptation use slicing for the top-down approach
Slicing Pseudo-Algorithm

1. Pick a range variable as the criterion
2. Select all post-conditions that affect/affected by the criterion
3. Select all pre-conditions that control the execution of the post-conditions
4. Add all range/domain variables constrained
Example #2

Problem P2

true

y' = f(a) and z' = g(a)

true

y' = f(a)

a : real

y : real

P21

true

z' = g(a)

a : real

z : real

P22

a : real

z : real
Find Example Preliminaries

- Classic adaptation example
- Goal is to find a record in a list of records given a unique key
- Library contains no constructors, only observers (e.g. firstRecord, sort, treeSearch)
- Bijective port connection fails to find solution
- Benefit of less restrictive port connection becomes apparent
Evaluation Metrics/Variables

• Evaluation metrics
  – Precision
  – Recall
  – Time-to-solution (TTS)

• Execution variables
  – Search depth (number of components)
  – Port connection methods
Precision and Recall Metrics

• Precision
  – Relates the purity of the retrieval set
  – # solutions retrieved/# results retrieved

• Recall
  – Relates the coverage of the solutions
  – # solutions retrieved/# solutions that exist

• Infinite solutions may exist
  – Example: $f_N$ applied to $f_{N-1}^{-1}$
  – Either never stop searching or always stop with 0% recall
Recall Definitions

- **Recall**\(_1\)
  - \# groups retrieved/\# groups that exist
  - Group is defined as the containment of some combination (without replacement) of components such that a solution exists
  - Reduces influence of multiple/redundant configurations

- **Recall**\(_2\)
  - \# groups retrieved/\# groups that exist
  - Group is defined as the containment of the smallest combination (without replacement) of components such that a solution exists
  - Reduces influence of architecture expansion

- **Recall**\(_3\)
  - \# solutions retrieved/\# solutions that exist
  - A solution has N components or less
### Recall Illustration

#### Equation

<table>
<thead>
<tr>
<th>Equation</th>
<th>Solution Groups</th>
<th>No. Soln.</th>
</tr>
</thead>
</table>
| Recall₁ | Group \{a\}: #1  
          | Group \{b, c\}: #2  
          | Group \{a, b, c\}: #3  | 3 |
| Recall₂ | Group \{a\}: #1, #3  
          | Group \{b, c\}: #2, #3  | 2 |
| Recall₃ | N = 2: #1, #2  | 2 |
Evaluation Library and Queries

• Four libraries
  – 46 mathematical components
  – 106 list manipulation components
  – 33 record manipulation components
  – 42 DSP components

• 103 queries, solved by:
  – Single component architectures
  – 1:N component architectures
  – N>1 component architectures
  – Infinite number of solutions
  – Multiple sub-architectures
  – Components from multiple libraries
Recall vs. Search Depth (1-1)

Recall vs Search Depth

- Well distributed components/arc (1-6)
- Most groups found early, diminishing returns
- Fade due to failed proof strategies on complex sub-problems
Recall(2) vs. Search Depth

Small gains, useful in math library (e.g. power functions)
TTS vs. Search Depth

TTS vs Search Depth

- Bijective
- One-to-one
- Onto
Other Results

• Precision
  – Between 98-100%
  – No tradeoff with recall
    • Formal methods for adaptation/retrieval
    • Theorem-prover precision

• Time
  – 92% spent on retrieval
  – Most of that spent on “dead-ends”
  – Hardware engineers will wait, will software engineers?
Future Work & Limitations

- Assumes shared-variable communication
  - Include communication protocols as search criteria
  - Include connector specifications in the library
- Only synthesizes three architectures
- Limited by theorem-prover, search depth
- Reduce TTS (retrieval limitation)
- Ranking of partial solutions
Related Work

• Specification-based Retrieval
  – Zaremski and Wing – developed match lattice, retrieval engine for Larch/ML specifications
  – Penix/Patil – developed REBOUND/SOCCKER retrieval engine, used feature-based classification
  – Fischer – designed NORA/HAMMR retrieval engine, used a layered architecture, included model checker
Related Work

• Component Adaptation
  – Penix – Suggested using architectures for behavioral adaptation
  – Purtilo and Atlee – created NIMBLE, automated module interface adaptation
  – Jeng and Cheng – identified necessary modifications to reuse general components to specific problems
Related Work

- Synthesis, Slicing, Architecting for Reuse
  - Chen and Cheng – developed ARBIE, an architecture-based reuse framework
  - Zhao – applied slicing to ADL for reuse-of-the-large
  - Bhansali – created a hybrid approach to reuse of geometrics, uses code-level reuse, architectures, and semi-synthesis of code fragments
Conclusions

• Presented framework for specification-based component retrieval and adaptation
• Behavioral adaptation was automated using architectures
• Sequential, alternative, and parallel adaptation implemented to adapt partial matches
• Provided sound definitions to synthesize sub-problems to satisfy component adaptation
• ~94% recall, ~100% precision (tradeoffs for TTS)
• Questions