A FORMAL APPROACH FOR INCORPORATING ARCHITECTURAL TACTICS INTO THE SOFTWARE ARCHITECTURE

Hamid Bagheri & Kevin Sullivan
University of Virginia Computer Science
How do architects integrate tactics with styles in architecture development?

Outline

- Problem
- Goals
- Background
- Approach
- Evaluation
- Related work
- Conclusions
Important outcomes depend on architectural tactic

Lack a scientific account of choices in this dimension

- Selection of architectural tactics is driven by informal analysis
- Application of architectural tactics is a *manual* activity
- Once “wired in,” architectural changes are very hard to make

Can we develop a scientific account for derivation of software architecture, with respect to both styles and tactics, to enable the automated application of architectural best practices?
Goals

- Develop formal account for incorporation of tactics into the software architecture
- Exploiting that formalism to automate architecture synthesis
- Deliver synthesis approach as a tool capability
Background

\[ t : \text{AppType} \]
\[ m : \text{AppModel} \]
\[ \text{conforms} \]
\[ \text{refines} \]
\[ \text{map}(t,s)(m) \]

\[ s : \text{ArchStyle} \]
\[ \{a_i : \text{ArchModel}\} \]
\[ \text{conforms} \]
\[ \text{in} \]
\[ \text{in} \]
\[ \text{in} \]
\[ \text{out} \]
module SCC

sig **Sensor** extends needHandle{}
sig **Actuator** extends needHandle{}
sig **Controller** extends needHandle{
    sensors : set Sensor,
    actuators : set Actuator,
    ...
}
abstract sig dispatch_protocol{}
one sig periodic, aperiodic, sporadic extends dispatch_protocol{}
...

Application Type (elided)
module II
open OO
abstract sig IIObject extends Object {}
abstract sig EventBus extends Connector {}
abstract sig Publish extends Role {}
abstract sig Subscribe extends Role {}
abstract sig PublishEvent extends Port {}
   one o:IIObject| this in o.ports
   all port: PublishEvent| attachments.port.ran in Publish
abstract sig SubscribeEvent extends Port {}
module LunarLander_SCC
open SCC
one sig FlightControl extends Controller{}
  sensors = Altimeter + FuelLevel + Gyro + EngineControlSwitch
  actuators = DescentEngController + Display + AttitudeControlThruster
  controller_dispatch_state = periodic
  frequency_state = fast
  program = controller_code
}

one sig Altimeter, FuelLevel, Gyro, EngineControlSwitch
  extends Sensor{}

one sig DescentEngController, Display, AttitudeControlThruster
  extends Actuator{} ...
An Architectural Mapping
(Sense-Compute-Control to Implicit-Invocation)

all n:needHandle | one o: IIObject | o.handle = n

all a:Actuator | one port: Port | (port in (a.~handle.ports & Procedure) ) || (port in (a.~handle.ports & SubscribeEvent)
all s:Sensor | one port: Port | (port in (s.~handle.ports & Procedure) ) || (port in (s.~handle.ports & PublishEvent) )

# (Controller.~handle.ports & SubscribeEvent) = # (Sensor.~handle.ports & PublishEvent)
(#SubscribeEvent >0) => # (Controller.~handle.ports & PublishEvent) = 1
(#Procedure >0) => # (Controller.~handle.ports & Call) = 1
all port:Procedure| one conector: procedureCall| port[attachments].ran = conector.roles & Provide

Controller.~handle.call[attachments].ran.connector = //procedureCall
   Actuator.~handle.procedure[attachments].ran.connector +
   Sensor.~handle.procedure[attachments].ran.connector

(Controller.~handle.ports & PublishEvent)[attachments].ran~roles = // EventBus
(Actuator.~handle.ports & SubscribeEvent)[attachments].ran~roles

(Controller.~handle.ports & SubscribeEvent)[attachments].ran~roles =
(Sensor.~handle.ports & PublishEvent)[attachments].ran~roles

...

\[
t : \text{AppType} \\
\text{map}(s,t)(m) \\
m : \text{AppModel} \\
s : \text{ArchStyle} \\
\text{conforms} \\
\text{refines} \\
\text{in} \\
\text{out} \\
\text{conforms}
\]
Satisfying Solution

Constraints of the mapping predicate
Lessons Learned

- Applying architectural maps leads not to a single plausible architecture, but to a set of architectures.

- Style specifications to which application models were being mapped are underspecified.

- Their corresponding architectural maps leave overly large architectural spaces.

- Not any instances in that set necessarily satisfies all required non-functional properties.
Approach
Architectural Tactics

- Architectural decisions that codify and record best practice knowledge for achieving a desired NFP

- Several architectural tactics proposed for various NFPs: Reliability, Performance, Modifiability, ...

- Architectural styles have extensive use in the architectural modeling

Structure of tactics must fit within rules implied by styles
Quality-improving Architectural Transformations

- Application of Architectural Tactics as
  - Quality-improving architectural transformations
  - In the formal process of architecture synthesis

- These transformations should influence NFPs without changing functional behaviors

- After the transformation, architectural specifications are still correct
Our Approach: Parameterized Architectural Tactics

- Architectural tactics not independent
  - Parameterized by choices of application type and architectural style
- Specialize tactics to particular settings induced by a choice of both application type and architectural style
- Define tactic predicates such that they refine the architectural map for the given type/style pair

\[
\text{map}(\text{AppType, ArchStyle}) \overset{\text{extends}}{\leftarrow} \text{tactic}(\text{AppType, ArchStyle})
\]
Applying Parameterized Tactics

- Decision to use a tactic in mapping an application model to an architecture involves:
  - Explicit inclusion of tactic specification with more generic architectural map

- Applying this compound map to the application model yields:
  - A family of correct-by-construction architectures supporting given architectural tactic
Tool Support for Architecture Synthesis
High-Level View
Lunar-Lander Case Study

Implicit Invocation

Lunar Landing control System

Application of type sense-compute-control
module PingEcho_SCC_II
open SCC
open II

pred pingEcho_SCC_II(s:set needHandle){
  one o:IIObject| o.handle = Pinger &&
  no handled:needHandle-Pinger| handled in o.handle &&
  one p: Procedure| p in o.ports &&
  all n:needHandle| n in s =>{
    one c1: procedureCall |
    c1 in n.~handle.procedure[attachments].ran.connector &&
    c1 in o.call[attachments].ran.connector
    
    one c2: procedureCall |
    c2 in n.~handle.call[attachments].ran.connector &&
    c2 in o.procedure[attachments].ran.connector
  }
}
Incorporation of Tactic Specification into the mapping

```plaintext
module LunarLander_SCC_II  // Compound mapping

open LunarLander
open SCC_II
open PingEcho_SCC_II

pred execute{
  map_SCC_II[]  // Architectural map
  PingEcho_SCC_II[FlightControl]  // Parameterized tactic predicate
}
```
Result of Applying compound mapping (in ACME)

```java
import families/PubSubFam.acme;

System LunarLander_II_1 : PubSubFam = new PubSubFam extended with {
    Component FlightControl : ParticipantT = new ParticipantT extended with {...}
    Component AttitudeControlThruster : ParticipantT = new ParticipantT extended with {...}
    Component Display : ParticipantT = new ParticipantT extended with {...}
    Component DescentEngController : ParticipantT = new ParticipantT extended with {...}
    Component EngineControlSwitch : ParticipantT = new ParticipantT extended with {...}
    Component Gyro : ParticipantT = new ParticipantT extended with {...}
    Component FuelLevel : ParticipantT = new ParticipantT extended with {...}
    Component Altimeter : ParticipantT = new ParticipantT extended with {...}
    Component Pinger : ParticipantT = new ParticipantT extended with {...}
    Connector EventBus1 : EventBusT = new EventBusT extended with {...}
    Connector EventBus2 : EventBusT = new EventBusT extended with {...}
    Connector EventBus3 : EventBusT = new EventBusT extended with {...}
    Connector EventBus0 : EventBusT = new EventBusT extended with {...}
    Connector EventBus4 : EventBusT = new EventBusT extended with {...}
    Connector procedureCall10 = {...}
    Connector Connector0 = {...}
    Attachment FlightControl.SubscribeEvent5 to EventBus2.Subscribe4;
    Attachment DescentEngController.SubscribeEvent4 to EventBus4.Subscribe2;
    Attachment Gyro.PublishEvent2 to EventBus3.Publish1;
    Attachment EngineControlSwitch.PublishEvent0 to EventBus2.Publish2;
    Attachment FlightControl.SubscribeEvent2 to EventBus0.Subscribe6;
    Attachment Altimeter.PublishEvent4 to EventBus1.Publish3;
    Attachment Display.SubscribeEvent3 to EventBus4.Subscribe0;
    Attachment AttitudeControlThruster.SubscribeEvent6 to EventBus4.Subscribe1;
    Attachment Pinger.Port0 to procedureCall10.role1;
    Attachment FlightControl.Port0 to procedureCall10.role0;
    Attachment Pinger.Port1 to Connector0.role1;
    Attachment FlightControl.Port1 to Connector0.role0;
}
```
Evaluation
Tested the idea by

- Exhibiting parameterized architectural tactics
  - Specialized for different application types and architectural styles
- Check consistency of automatically generated outputs with
  - Informally and manually produced results in the literature
  - Documentation of tactics

<table>
<thead>
<tr>
<th>STYLE</th>
<th>TYPE</th>
<th>Composition of Function</th>
<th>State-Driven</th>
<th>Sense-Compute-Control</th>
<th>Aspect-Oriented Comp. Func.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipe-and-Filter</td>
<td>KWIC, LL</td>
<td></td>
<td></td>
<td></td>
<td>KWIC</td>
</tr>
<tr>
<td>Object-Oriented</td>
<td>KWIC, LL</td>
<td>KWIC</td>
<td>LL, MIDAS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implicit-Invocation</td>
<td>KWIC, LL</td>
<td></td>
<td>LL, MIDAS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Result of the experimental testing of our approach
Strengths of Work to Date

- Represent architectural tactics in a formal and reusable form
- Tactics are formalized and implemented as executable specifications
- Correct-by-construction synthesis of architectural models
- Validated against arch styles and tactics defined in literature
Weaknesses of Work to Date

- Mainly considers structural refinement, not behavioral
- Supports tactics that refine/extend styles, not alteration of styles
- No attempt yet to scale to practical applications
Related Work

- Formal approaches to model transformation
  - *Evolution styles.* Garlan et al., 2009.
    - Horizontal vs. Vertical

- Formalization of architectural styles
  - Wong et al., 2008.

- Separation of Concerns

- Leveraging architectural styles and tactics to satisfy NFPs
  - Modifiability tactics, Bachmann et al., 2007.
  - Interaction of architectural patterns and tactics, Harrison et al., 2010.
  - Pattern-oriented knowledge model for arch. design, Kumar et al. 2010.
Conclusions
Selected Future Work

- Provide more rigorous representation of application models regarding NFRs
- Study of application types analogous to past work on architectural styles
- Automated design space search
  - There are generally many possible satisfying architectures
  - Combining the approach with architectural analysis tools (e.g. AADL) enables optimizing search for properties of interests
Contributions

- Showed feasibility of representing architectural tactics parameterized by type/style pair in a formal and reusable form, using the notion of architectural maps.

- Showed that formalism enables automated synthesis.

- Created opportunities for future work in this area.
Thank You