Monarch: Model-Based Development of Software Architectures

Hamid Bagheri & Kevin Sullivan
University of Virginia Computer Science
Basic Question

Given a description of an application, how do you get to an architecture for that application?
Key Observations

- For given types of application and styles of architecture, architects have learned how to do this mapping.

- Software engineers have long considered architectural style a separate design decision.

- We lack a rigorous account of the mappings from application descriptions to architectures.
Goal & Contribution of this Work

- Use formal account of mapping as basis for modeling tools that synthesize architectures for applications of given types in given styles
  - Provide description of application (a model)
  - Select an architectural style
  - Push a button and get architectures
Parnas 1972

- Architecture as a separate choice in design

Key Word in Context (KWIC)

- Procedural Decomposition
- Information Hiding Modularization
Wide range of architectural style choices
Toward a theory of architectural maps

KWIC

Instance of a given type of application

Architectural Map from applications of given type to architectures of given style

PF

Instance of a given style of architecture
Application of type sense-compute-control
Architecture in RESTful style
Implicit Invocation Architectures

Why do this?

- Formalize and automate architectural expertise
- Target diverse architectural styles in space, time
- Automate costly & error prone transformations
- Correct by construction architectural models
A Theory of Architectural Maps

\[ t : \text{AppType} \quad \xrightarrow{\text{conforms}} \quad \text{map}_{(t,s)}(m) \quad \xleftarrow{\text{refines}} \quad \{a_i : \text{ArchModel}\} \]

\[ \text{m : AppModel} \quad \xrightarrow{\text{in}} \quad \text{map}_{(t,s)}(m) \quad \xleftarrow{\text{out}} \quad s : \text{ArchStyle} \]
module SCC
open domain
abstract sig dispatch_protocol{}
one sig periodic extends dispatch_protocol{}
... sig Sensor extends needHandle{}
sig Actuator extends needHandle{}
sig Controller extends needHandle{
  sensors : set Sensor,
  actuators : set Actuator,
  ...
}
module II
open OO
abstract sig Publish extends Role {}
abstract sig Subscribe extends Role {}
abstract sig PublishEvent extends Port {}

some s:System | one anEventBus:EventBus | one r1:Publish |
r1 in anEventBus.roles &&
r1->this in s.attachments

abstract sig SubscribeEvent extends Port {}

some s:System | one anEventBus:EventBus | one r1:Subscribe |
r1 in anEventBus.roles &&
r1->this in s.attachments

...
An Architectural Mapping
(Sense-Compute-Control to Implicit-Invocation)

module SCC_II
open SCC
open II
pred handle()
  all n:needHandle | one o: IIObject | o.handle = n
  #Sensor.~handle.ports = #Sensor
  all s:Sensor| one port: Port| (port in (s.~handle.ports & Procedure) )||
  (port in (s.~handle.ports & PublishEvent) )
  (#SubscribeEvent >0) =>{
    # (Controller.~handle.ports & PublishEvent) = 1}
  # (Controller.~handle.ports & SubscribeEvent) =
  # (Sensor.~handle.ports & PublishEvent)
  # (Controller.~handle.ports & Procedure) = 0
  (#Procedure >0) =>{
    # (Controller.~handle.ports & Call) = 1)//Actuator + Sensor
    # EventBus = #PublishEvent
    # Actuator.~handle.ports = # Actuator
    alla:Actuator | one port: Port| (port in (a.~handle.ports & Procedure) ) ||
    (port in (a.~handle.ports & SubscribeEvent) )
    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 // EventBus
    (Controller.~handle.ports & SubscribeEvent)[attachments].ran.~roles = // EventBus set
    (Sensor.~handle.ports & PublishEvent)[attachments].ran.~roles
    #Concern = 0
    #System = 1
  }
pred show{
    handle[]
  }
run show for 12
module MIDAS
open SCC
one sig IntrusionAlarmReceiver extends Sensor{} ...
one sig FireAlarmActuator extends Actuator{} ...
one sig IntrusionAlarmAnalyzer extends Controller{} {
  sensors = IntrusionAlarmReceiver
  actuators = IntrusionAlarmActuator
  controller_dispatch_state = periodic
  frequency_state = fast
  program = IntrusionAlarmAnalyzerSourceCode
} ...
Alloy Analyzer Computes Architectures!
Architectural Mapping
(Sense-Compute-Control, Implicit-Invocation)

For each **Sensor**, **Actuator** and **Controller** there is an *IIObject* that handles it

*Part of the mapping predicate represented in Alloy*
Architectural Mapping
(Sense-Compute-Control, 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) )

Each **Actuator's IIObject** has a port of type: **SubscribeEvent** or **Procedure**
to support invocation -- either implicit invocation or by explicit procedure call

Controller~handle.call[attachments].ran.connector = //procedureCall

Each **Sensor's IIObject** has a port of type: **PublishEvent** or **Procedure**

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

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

...
### Architectural Mapping

**Sense-Compute-Control, Implicit-Invocation**

\[
\text{all } n : \text{needHandle} \mid \text{one } o : \text{IIObject} \mid o.\text{handle} = n
\]

\[
\text{all } a : \text{Actuator} \mid \text{one port } \text{Port} \mid (\text{port in } (a.\neg\text{handle.ports} \& \text{Procedure}) \lor (\text{port in } (a.\neg\text{handle.ports} \& \text{SubscribeEvent}))
\]

\[
\text{all } s : \text{Sensor} \mid \text{one port } \text{Port} \mid (\text{port in } (s.\neg\text{handle.ports} \& \text{Procedure}) \lor (\text{port in } (s.\neg\text{handle.ports} \& \text{PublishEvent}))
\]

\[
\# (\text{Controller.}\neg\text{handle.ports} \& \text{SubscribeEvent}) = \# (\text{Sensor.}\neg\text{handle.ports} \& \text{PublishEvent})
\]

The number of **SubscribeEvent** ports of the **Controller’s IIObject** =

The number of **PublishEvent** ports of **Sensors’ IIObjects**

Each controller’s **SubscribeEvent** port is **connected** to a **Sensor’s PublishEvent** port

---

Part of the mapping predicate represented in Alloy
Architectural Mapping
(Sense-Compute-Control, 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

A Controller’s IIObject has:
one Call port and one PublishEvent port

The procedures of Actuators’ IIObjects could be called explicitly using a Call port or could be invoked when PublishEvent port of the Controller’s IIObject announces an event
Architectural Mapping

Required **structure** for two supported invocation styles in *implicit-invocation* style: **procedure call** and *implicit invocation*

**Procedure Call** method:
For each **Procedure port**, there is a **ProcedureCall connector** connected to it.

The **Call** port of Controller’s **IIObject** is connected to the **Procedure** ports of **Sensors’** and **Actuators’ IIObjects**

Through **ProcedureCall connectors**

```plaintext
call 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
```

**Part of the mapping predicate represented in Alloy**
Architectural Mapping
(Sense-Compute-Control, Implicit-Invocation)

Implicit invocation method call:

The **SubscribeEvent** ports of **Actuators’ IIObjects** are connected to **PublishEvent** port of the **Controllers’ IIObject**

Through an **EventBus** connector

The **SubscribeEvent** ports of **Controller’s IIObject** are connected to **PublishEvent** ports of the **Sensors’ IIIObjects**

```
(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
```

Part of the mapping predicate represented in Alloy
Application type to GME Meta-model
Application of type *sense-compute-control*
Postprocessor translates Alloy to ADL
Evaluation

- **Strengths**
  - Theory explains mapping, supports automation
  - Works with arch styles defined in the literature
  - Case studies are from others' published works
  - Use of Alloy makes approach formally rigorous

- **Weaknesses**
  - Canonical styles aren't really what count in practice
  - We've just cracked open topic of application types
  - No attempt yet to scale to practical applications
Related Work

- Middleware-induced architectural styles
  - Di Nitto and Rosenblum
  - Formal modeling of EJB, Sousa and Garlan
  - ComPAS (Gall et al.), SOA (Baresi et al.)

- Filling the gap between architectures and implementations
  - ArchJava, Aldrich
  - Prism-MW architectural framework. Malek et al.

- Formal approaches to model transformation
  - Architectural evolution patterns, Tamzalit and Mens
  - Evolution styles, Garlan et al.
  - Architectural transformations, Ambriola and Kmiecik
    - Horizontal vs. Vertical

- Formalization of architectural styles
  - Kim and Garlan, Wong et al.

- Separation of Concerns
  - Flexible Packaging, Deline
Selected Future Work

- Realistic architectural styles (under review)
- Compose with maps to *code* (under review)
- Automated design space search
  - Within architectural styles
  - Across architectural styles
- Study of application types analogous to past work on architectural styles
Conclusion

- We have contributed a proof of concept for synthesis of software architectures based on formal theory of architectural maps