Leveraging Software Design to Guide the Development of Sense/Compute/Control Applications

Damien Cassou
Design is Crucial

“...The most important ingredient in ensuring a software system’s long-term success is its design.”

ICSE’11 c.f.p

- A good design improves
- collaboration
- productivity
- maintenance
A good design requires a design framework which guides the architect with

- a language
- a paradigm
A good implementation requires a programming framework which guides the developer with

- abstractions
- services
An implementation must conform to its design
Requirements

1. A design framework to guide the architect
2. A programming framework to guide the developer
3. A guaranteed conformance of the implementation relatively to the design
<table>
<thead>
<tr>
<th>Related Works</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design</strong></td>
</tr>
<tr>
<td>Architect</td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
</tr>
<tr>
<td>Developer</td>
</tr>
<tr>
<td><strong>Conformance</strong></td>
</tr>
<tr>
<td>Conformance</td>
</tr>
</tbody>
</table>
## Related Works

<table>
<thead>
<tr>
<th>Design</th>
<th>Architect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation</td>
<td>Developer</td>
</tr>
<tr>
<td>Conformance</td>
<td>Conformance</td>
</tr>
</tbody>
</table>

**Java**
# Related Works

<table>
<thead>
<tr>
<th>Design</th>
<th>Architect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation</td>
<td>Developer</td>
</tr>
<tr>
<td>Conformance</td>
<td>Conformance</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPL</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>++</td>
</tr>
</tbody>
</table>

**Spring**
# Related Works

<table>
<thead>
<tr>
<th>Related Works</th>
<th>GPL</th>
<th>Library</th>
<th>ADL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design</strong></td>
<td></td>
<td></td>
<td>++</td>
</tr>
<tr>
<td>Architect</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td></td>
<td>++</td>
<td></td>
</tr>
<tr>
<td>Developer</td>
<td>+</td>
<td>++</td>
<td></td>
</tr>
<tr>
<td><strong>Conformance</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conformance</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

C2
## Related Works

<table>
<thead>
<tr>
<th></th>
<th>GPL</th>
<th>Library</th>
<th>ADL</th>
<th>ADL++</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design</strong></td>
<td></td>
<td></td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Architect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td></td>
<td></td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>Developer</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td><strong>Conformance</strong></td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Conformance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ArchJava**
Thesis

A paradigm-oriented framework for both design and implementation which maintains conformance all along the software life-cycle
The Paradigm
Sense/Compute/Control (SCC)

“applications that interact with an environment”
The SCC Paradigm

Environment

- sense
- compute
- control
The SCC Paradigm

sense

compute
direction

GPS, flight plan

control aileron, engine

control

compute
The SCC Paradigm

motion detection

sense

compute

intrusion?

control

trigger alarms

Environment
The SCC Paradigm

Covers various domains

• pervasive computing
• tier-system monitoring
• avionics
• robotics
• ...

Environment
Contributions

1. A paradigm-specific design framework

2. A programming framework dedicated to a design

3. An evaluation of the approach
Contributions

1. A paradigm-specific design framework

2. A programming framework dedicated to a design

3. An evaluation of the approach
Design Language
Design Language

• sources sense the environment
Design Language

- sources sense the environment
- actions impact the environment
Design Language

a concept for handling the interaction with the environment

- sources sense the environment
- actions impact the environment
Design Language

```java
entity Keypad {
  source keycode as Integer;
  action UpdateSt;
}

action UpdateSt {
  updateStatus(message as String);
}
```

- sources sense the environment
- actions impact the environment

a concept for handling the interaction with the environment
Design Language

**entity** Keypad {
  **source** keycode as Integer;
  **action** UpdateSt;
}

**action** UpdateSt {
  updateStatus(message as String);
}

1 entity description for potentially many implementations

a concept for handling the interaction with the environment
Design Language

entity Keypad {
    source keycode as Integer;
    action UpdateSt;
}

action UpdateSt {
    updateStatus(message as String);
}

1 entity description for potentially many instances

a concept for handling the interaction with the environment
Design Language

entity Keypad {
    source keycode as Integer;
    action UpdateSt;
    attribute room as Integer;
}

action UpdateSt {
    updateStatus(message as String);
}

1 entity description for potentially many instances

attributes to characterize instances (color, location, reliability, etc.)
Design Language

raw data

a concept to refine raw data

sources

Entities

actions

Environment
Design Language

context operators

raw data

refined information

a concept to refine raw data

Environment

context BuildingSecured as Boolean {
  source keycode from Keypad;
}

24
Design Language

a concept to take decisions based on application-level data

context
operators

raw data

refined information

sources

Entities

actions

Environment
Design Language

A concept to take decisions based on application-level data.

- **Design Language**
  - **Environment**
    - **Entities**
      - **Sources**
        - **Actions**
          - **Context** BuildingSecured
          - **Controller** BuildingManager
            - **Orders**
              - **Actions**
                - **UpdateSt on Keypad**

- **Control Operators**
  - **Operators**
    - **Orders**
      - **Refined Information**
        - **Raw Data**
Design Language

The language features concepts dedicated to the SCC paradigm.
Information creation

context operators

refined information

control operators

Information use
Design Language

context operators

raw data

refined information

control operators

orders

sources

Entities

actions

Environment

operators

actions
Design Language

actions

control
operators

context
operators

sources
Design Language

- actions
- control operators
- context operators
- sources
Case Study: Anti-Intrusion

- actions
- control operators
- context operators
- sources
Case Study: Anti-Intrusion

Intrusion

- Actions
- Control Operators
- Context Operators
- Sources
Case Study: Anti-Intrusion

Diagram:

- **Alarm OnOff**
- **Intrusion Manager**
- **Intrusion**

Actions:
- Intrusion Manager

Control Operators:
- Intrusion

Context Operators:
- Alarm OnOff

Sources:
Case Study: Anti-Intrusion

Diagram:
- **Alarm**: On/Off
- **Intrusion Manager**
- **Intrusion**
- **Building Secured**
- **Presence**

- **Actions**: Alarm On/Off
- **Control Operators**: Intrusion Manager
- **Context Operators**: Intrusion
- **Sources**: Building Secured, Presence

Description:
- An alarm system is described, with actions including turning the alarm on or off.
- Intrusion management is central, with control operators managing intrusion.
- Context operators provide information on the building's security status.
- Building security and presence are indicated as sources of context.
Case Study: Anti-Intrusion

- Actions: Alarm (On/Off)
- Control Operators: Intrusion Manager
- Context Operators: Intrusion
- Sources: Building Secured, Keypad (keycode), MotionSensor (motion)

Diagram:
- From Intrusion Manager to Intrusion
- From Intrusion to Building Secured, Keypad, MotionSensor
- From Keypad to Intrusion Manager
- From MotionSensor to Intrusion Manager

32
Case Study: Anti-Intrusion

actions

control operators

context operators

sources
Case Study: Anti-Intrusion

- **Sources**
  - Keypad
  - MotionSensor
  - Camera

- **Context**
  - Building
    - Secured
  - Presence
  - Scene
    - Image

- **Operators**
  - Security Manager
  - Intrusion Manager

- **Actions**
  - Keypad: UpdateSt
  - Alarm: OnOff
  - Mailer: Send

This diagram illustrates the components and interactions involved in an anti-intrusion system.
Case Study: Anti-Intrusion

- **Keypad**: UpdateSt
- **Alarm**: OnOff
- **Mailer**: Send
- **Security Manager**
- **Intrusion Manager**
  - **Intrusion**
  - **Building Secured**
    - **keycode**
    - **MotionSensor**
    - **Camera**
      - **image**
  - **Presence**
  - **Scene Image**
  - **Mailer** Send
  - **Keypad** UpdateSt

**Actions**

**Control Operators**

**Context Operators**

**Sources**
Case Study: Anti-Intrusion

Intrusion

Building Secured

Presence
Case Study: Anti-Intrusion

Building Secured: Intrusion → Presence

what the architect has in mind
Case Study: Anti-Intrusion

what the architect has in mind
Case Study: Anti-Intrusion

what the architect has in mind
Case Study: Anti-Intrusion

Possible interpretations

Building Secured → Presence → Intrusion

Building Secured → Intrusion

Presence → Intrusion

Building Secured → Presence
Case Study: Anti-Intrusion

Possible interpretations

Building Secured → Intrusion → Presence

Building Secured → Intrusion → Presence

Intrusion → Building Secured → Presence

Intrusion → Building Secured → Presence
Case Study: Anti-Intrusion

Possible interpretations
Case Study: Anti-Intrusion

Possible interpretations
Case Study: Anti-Intrusion

Possible interpretations
Case Study: Anti-Intrusion

possible interpretations
Case Study: Anti-Intrusion

Possible interpretations
Case Study: Anti-Intrusion

possible interpretations
Case Study: Anti-Intrusion

The architect should express what he has in mind.

Possible interpretations: Intrusion → Presence → Building Secured.
Case Study: Anti-Intrusion

The architect should express what he has in mind, stating the possible interpretations of interaction contracts.
Interaction Contracts

context operator
Interaction Contracts

① Activation condition
Interaction Contracts

① Activation condition

context operator

event
① Activation condition
② Data requirement
Interaction Contracts

1. Activation condition
2. Data requirement
3. Emission
Interaction Contracts

① Activation condition

② Data requirement

③ Emission

```java
context Intrusion as Boolean {
    context Presence;
    context BuildingSecured;
    interaction {
        when provided Presence
        get BuildingSecured
        maybe publish
    }
}
```
Interaction Contracts

① Activation condition

② Data requirement

③ Emission

```java
class Intrusion as Boolean {
    context Presence;
    context BuildingSecured;
    interaction {
        when provided Presence
        get BuildingSecured
        maybe publish
    }
}
```
Interaction Contracts

1. Activation condition
2. Data requirement
3. Emission

```context Intrusion as Boolean {
  context Presence;
  context BuildingSecured;
  interaction {
    1 when provided Presence
    2 get BuildingSecured
    maybe publish
  }
} ```
Interaction Contracts

1. Activation condition
2. Data requirement
3. Emission

```
context Intrusion as Boolean {
  context Presence;
  context BuildingSecured;
  interaction {
    ① when provided Presence
    ② get BuildingSecured
    ③ maybe publish
  }
}
```
Interaction Contracts

1. Activation condition
2. Data requirement
3. Emission

```plaintext
context Intrusion as Boolean {
  context Presence;
  context BuildingSecured;
  interaction {
    1 when provided Presence
    2 get BuildingSecured
    3 maybe publish
  }
}
```

related to automata approaches
Summary

A design framework consisting of a design language, DiaSpec, which guides the architect by offering

- concepts dedicated to the SCC paradigm
- a separation between environment handling and logic
- a separation between information creation and use
- a dedicated description of interactions
Contributions

1. A paradigm-specific design framework

2. A programming framework dedicated to a design

3. An evaluation of the approach
A Programming Framework
Generated from the Design

DiaSpec

DiaSpec design → Code generator → Programming framework

Architect

GPL

Developer
A Programming Framework
Generated from the Design

- separates 2 different roles with 2 different languages
- leverages GPL tools, libraries and expertise
- ensures conformance automatically
A Programming Framework

how to make a programming framework \textit{conform} to a particular design?
A Programming Framework

The code generator maps

- each description to an abstract class
- each interaction contract to an abstract method
A Programming Framework

The code generator maps

• each description to an abstract class
• each interaction contract to an abstract method

By leveraging the GPL type checker, the framework

• guides the implementation of what is required
• forbids anything not specified in the design
A Programming Framework

The code generator maps

- each description to an abstract class
- each interaction contract to an abstract method

By leveraging the GPL type checker, the framework

- guides the implementation of what is required
- forbids anything not specified in the design

different than what is proposed by ADLs or MDE
context Presence as Boolean {
    source motion from MotionSensor;
    interaction {
        when provided motion from MotionSensor
        get motion from MotionSensor
        always publish
    }
}

abstract class AbstractPresence {
    abstract boolean onMotionFromMotionSensor(
        boolean motion, Select select);
}
abstract class AbstractPresence {
    abstract boolean onMotionFromMotionSensor(
        boolean motion, Select select);
}
abstract class AbstractPresence {
  abstract boolean onMotionFromMotionSensor(
    boolean motion, Select select);
}
abstract class AbstractPresence {
    abstract boolean onMotionFromMotionSensor(
        boolean motion, Select select);
}

context Presence as Boolean {
    source motion from MotionSensor;
    interaction {
        ① when provided motion from MotionSensor
        ② get motion from MotionSensor
            always publish
    }
}

Generation according to the interaction contract
abstract class AbstractPresence {
    abstract boolean onMotionFromMotionSensor(
        boolean motion, Select select);
}
Implementing the Behavior

context Presence as Boolean {
    source motion from MotionSensor;
    interaction {
        when provided motion from MotionSensor
            get motion from MotionSensor
            always publish
    }
}

abstract class AbstractPresence {
    abstract boolean onMotionFromMotionSensor(
        boolean motion, Select select);
}

class Presence extends AbstractPresence {
    boolean onMotionFromMotionSensor(
        boolean motion, Select select) {
        return motion;
    }
}

Architect

Developer
Implementing the Behavior

when motion is detected ➞ there is presence

when motion is not detected? ➞ ?

The developer needs to ask all motion sensors

class Presence extends AbstractPresence {
    boolean onMotionFromMotionSensor(
        boolean motion, Select select) {
        return motion;
    }
}

Developer
Entity Selection

Required when an entity is the interaction’s target

Guide the developer with an embedded and type-safe DSL
Entity Selection

Select all motion sensors:
```
select.motionSensors().all()
```
Select all motion sensors in rooms 1 to 3:

```java
select.motionSensors().whereRoom(between(1,3))
```
Commanding Entities

entity Alarm {
    action OnOff {
        action OnOff;
        on();
        off();
    }
}

Triggering all alarms:
select.alarms().all().on();
It is not possible to send unsupported orders:

```java
select.alarms().all().start();
```
It is not possible to discover all kinds of entities:

```java
context Presence as Boolean {
    source motion from MotionSensor;
    interaction {
        when provided motion from MotionSensor
        get motion from MotionSensor
        always publish
    }
}
```

It is not possible to discover all kinds of entities:

```java
select.alarms().all();
```
Implementing the Behavior

context Presence as Boolean {
  source motion from MotionSensor;
  interaction {
    when provided motion from MotionSensor
    get motion from MotionSensor
    always publish
  }
}

abstract class AbstractPresence {
  abstract boolean onMotionFromMotionSensor(...);
}

class Presence extends AbstractPresence {
  boolean onMotionFromMotionSensor(  
    boolean motion, Select select) {
    if (motion) {
      return true;
    }
    MotionSensors sensors = select.motionSensors().all();
    for (MotionSensor sensor : sensors) {
      if (sensor.getMotion()) {
        return true;
      }
    }
    return false;
  }
}

---

Developed by Developer
Summary

The developer is guided with

- a support dedicated to the application
- an embedded DSL for entity selection

Conformance is ensured by

- generating a programming framework
- leveraging a GPL type checker
Contributions

1. A paradigm-specific design framework

2. A programming framework dedicated to a design

3. An evaluation of the approach
Evaluation of the Approach

- Expressiveness
- Usability
- Productivity
Evaluation: Expressiveness

Numerous domains

- home-automation
- avionics
- graphical user interfaces
- health-care
- telecommunications
- tier-system monitoring
- etc.
Evaluation: Usability

Context

• 80 students during 3 years
• sparse and oral-only documentation

Results

• 64 students completed the assignment
• Identification of the interaction contracts
Evaluation: Productivity

We measured the amount of code generated automatically
Evaluation: Productivity

- Framework: 82%
- Specification: 8%
- Implementation: 10%
Evaluation: Productivity

- Framework: 82%
- Specification: 8%
- Implementation: 10%

→ 76% actually executed
Evaluation: Productivity

Complexity of the developer’s code

We used the Sonar platform
to measure code quality
through numerous metrics
Evaluation: Productivity

Complexity of the developer’s code

“number of linearly independent paths in a source code”

McCabe cyclomatic complexity
Evaluation: Productivity

Complexity of the developer’s code

“number of linearly independent paths in a source code”

McCabe cyclomatic complexity
Evaluation: Productivity

Complexity of the developer’s code

"number of linearly independent paths in a source code"
McCabe cyclomatic complexity

"quite well structured"

1 3 7 10 ∞
Evaluation: Productivity

Complexity of the developer’s code

“number of linearly independent paths in a source code”
McCabe cyclomatic complexity

“quite well structured”

“very poor quality”

1 3 7 10 ∞
Evaluation: Productivity

Complexity of the developer’s code

“number of linearly independent paths in a source code”

McCabe cyclomatic complexity

“quite well structured”

“very poor quality”

1 3 7 10

∞

on average

on average
Summary

• The approach covers various domains
• The frameworks are easy to use
• Few code is required and this code is of good quality

Pursuing this evaluation with software engineers
Results

Scientific contributions

• A design language dedicated to SCC (ICSE’11)
• The generation of a dedicated programming framework (GPCE’09)
• The evaluation of this approach (*submitted*)

Technical contributions

• A compiler for the design language (9 KLoC)
• A code generator targeting Java (4 KLoC)

Dissemination

• Demonstrations (PerCom’10), posters (SPLASH’10), visits (Bern, Potsdam)
• Public release ([http://diasuite.inria.fr](http://diasuite.inria.fr))
A Research Vehicle

This design language and code generator are part of a research project which involves

- 4 industrial partnerships
- 2 other research groups
- > 20 real applications
- 24/7 running platform
- 28,000 lines of code
A Research Vehicle

7 PhD students leveraging DiaSpec and the generator

- QoS (FASE’11)
- error-handling (OOPSLA’10)
- virtual testing (Mobiquitous’09 and ’10)
- SIP (ICC’10, ICIN’09, IPTComm’08)
- end-user programming (DSLWC’09)
- security (ICPS’09)
Perspectives

- Can we support other stages of the software life-cycle?

- Can we transpose the approach to another paradigm?

- Can we help creating such approaches?
Facilitating Evolution

- eases developer’s work by
  - showing mismatches
  - leveraging development tools
- ensures conformance all along the software life-cycle