OBJECT METAMORPHISM

Zbyněk Šlajchrt

Type-safe Modeling of Protean Objects

Protean Objects

- *"Able to change into many different forms or able to do many different things"*
- *"Displaying great diversity or variety"*
- Adaptive: chemical reactions, mimicry, adaptive services
- Evolutionary: fetal development, tutorials
- How much is OOP capable of modeling such object?

Airport Scanner Case Study

- AS can recognize objects in baggage
- Two aspects: Material and Shape
- AS outputs a JSON record for each recognized item
-
```
{
  "id": 0,
  "shape": "cylinder",
  "material": "metal",
  "x": 234.87,
  "y": 133.4,
  "z": 12.94,
  "radius": 13.45,
  "height": 0.45,
  "density": 3.8
}
```

Modeling PO In Java

- Poor modeling capabilities
- Using mainly delegation and composition
- "*Has-A/Is-A*" dilemma: "*A wine bottle box or a wine bottle in a box*"
  - Good at "Has-A" relationships
  - Poor at "Is-A" relationship
- Loosing type information, `instanceof` is useless
- Scattered object identity, i.e. *object schizophrenia*

Modeling PO In Scala

- Using traits to compose items
- No delegation, no composition, no schizophrenia
- No loss of type information
  - `item.isInstanceOf[Rectangle with Paper]`
- **Problem: Exponential explosion of classes declarations**
- Proportional to the Cartesian product of all dimensions used to describe the item

Modeling PO In Groovy

- Dynamic traits resolve the explosion issue
- No schizophrenia, types are preserved
- The composition is done step-by-step (imperatively)
- However, this "manual" approach **is prone to:**
  - **Incompleteness**; i.e. forgetting some dimension
  - **Redundancy**; i.e. adding two mutually exclusive parts
  - **Missing** or **ambiguous** dependencies between parts

Solution: Object Metamorphism

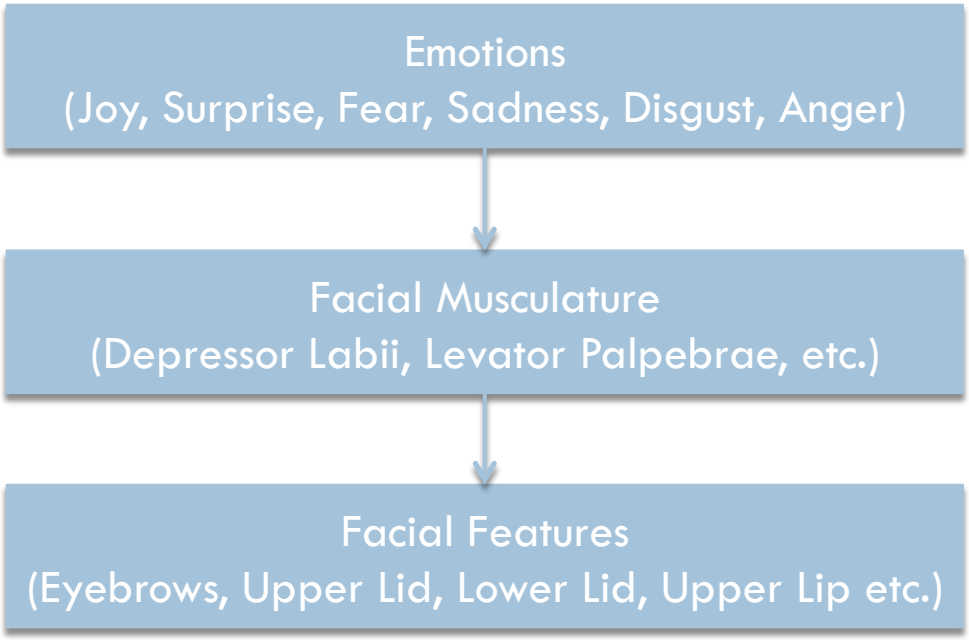
- A capability of an object to assume one or more forms **specified** by the object's **morph model**
- **Morph Model**
  - Describes all possible alternative "shapes" of the object
  - Each alternative consists of a list of traits
  - **Verified at compile-time**
- **Morph**
  - An **instance** of one alternative from the morph model
  - May **mutate** to another alternative to change behavior
- **Morph Strategy:** governs the mutation of the morph
- **Morpheus:** A P-o-C implementation of OM in Scala

Emotions In Face

- Human face as a protean object
- Emotions cause electric stimulation of muscles, which are responsible for facial expressions
- Studied by Ch. Darwin and G. Duchenne

https://en.wikipedia.org/wiki/Facial_expression

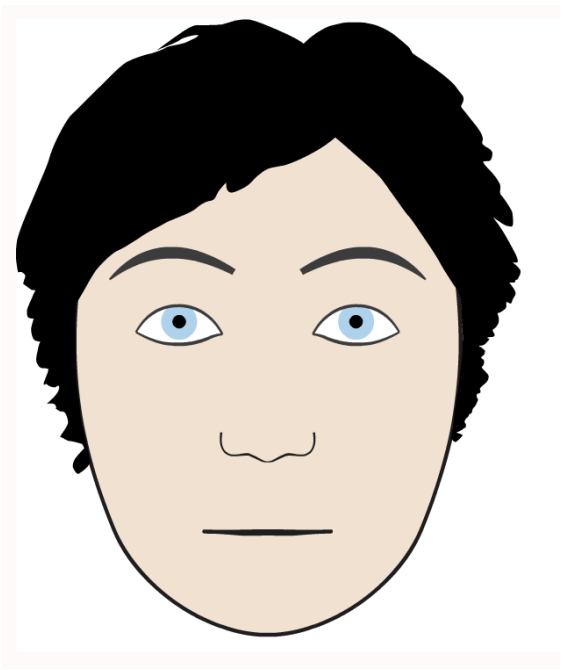
Modeling Emotional Expressions



- Each morph model actually represents one view on the human being
- The emotions influence the facial musculature, which influences the facial features

Emotions Model

- Joy, Surprise, Fear, Sadness, Disgust, Anger
- No more than 2 simultaneous emotions
- 6 simple + 15 combined = 21 alternatives



http://www.grimace-project.net/

Emotions Model

```
type Emotions1D = Joy or Surprise or Fear or Sadness or
   Disgust or Anger
// Emotions2D is Emotions1D “squared”
type Emotions2D = Emotions1D with Emotions1D
// Accompany Emotions2D by additional traits to complete deps
type Emotions = EmotionBase with TensionCalculator with
  Emotions2D

@fragment
trait EmotionBase {
  // Keeps the selection and intensity of one or two emotions
  def setEmoLevel(emoId: Int, intensity: Float) = …
  def getEmoLevel(emoId: Int): Float = …
  def getFirstEmotion(): Option[Int]
  def getSecondEmotion(): Option[Int]

  // Propagate the emotions to the muscles
  def stimulate(): Unit = {}
}
```

Emotion Sample

```
@fragment @wrapper
trait Joy extends Emotion {
  this: MuscleBase with TensionCalculator =>

  private lazy val tcalc = tensionCalc("joy")(_)

  override def influence() = {
    // Propagate the joy intensity through the stack of muscles
    // using applyTension() defined in MuscleBase
    this.applyTension(tcalc, joyLevel)

    // Invoke the other emotion, if any
    super.influence()
  }
}
```

Facial Musculature Model

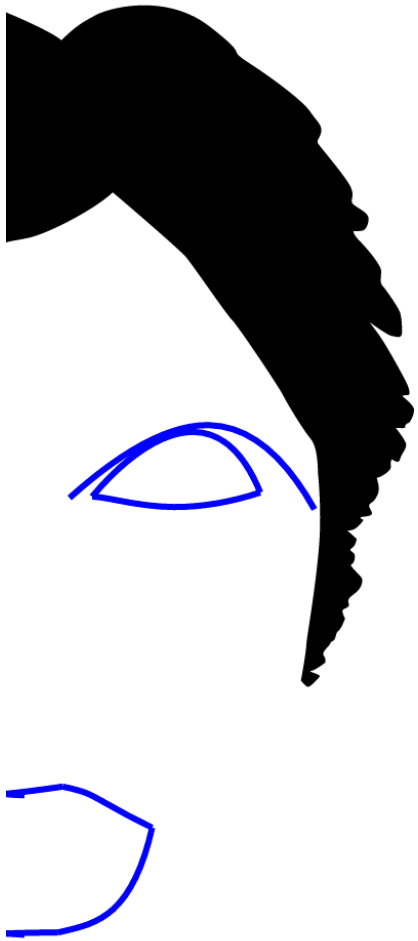
- 26 muscles (M0, M1, M2 …)
- The simplest approximation model assumes that all muscles may be stimulated simultaneously; i.e. there are no mutually exclusive muscles
- The model consists of only one alternative
- Splines used to model the muscle contractions
  - Segments, quadratic and cubic Bezier curves

Facial Musculature Code

```
type Musculature = MuscleBase with M0 with M1 with M2 with …

@fragment
trait MuscleBase {
def applyTension(tensFn: TensionFn, tension: Float) {}
}

@fragment @wrapper
trait M0 extends MuscleBase {
val m0 = new MuscleData(Line("m0", (136f, 144f), (140f,
140f)))

override def applyTension(tensFn: TensionFn, tension: Float){
    m0.updateTension(tensFn("m0")(tension))
    super.applyTension(tensFn, tension)
 }
}
```

Facial Features Model

- 7 most basic features
  - Eyebrow, Upper Lid, Lower Lid, Upper Lip, Upper Lip Joiner, Lower Lip, Lower Lip Joiner
- Any combination is possible
  - 2^7 = 128 alternatives
- Each feature depends on a certain subset of muscles
- Selected features depend on the set of activated muscles
  - There is only one set with all muscles
  - This is why all features are activated



Facial Features Code

```
type /?[T] = T or Unit
type Features = Feature with /?[Eyebrow] with
    /?[UpperLidFragment with UpperLid] with
    /?[LowerLid] with
    /?[LowerLipJoiner] with
    /?[LowerLipFragment with LowerLip] with
    /?[UpperLipFragment with UpperLip] with
    /?[UpperLipJoinerFragment with UpperLipJoiner]

@fragment
trait Feature {
  def render(): List[Spline] = Nil
}
```

Feature Sample

```
@fragment @wrapper
trait LowerLipJoiner extends Feature {
  this: LowerLipFragment with M15 with H_Mli11 =>

  private val spline = CBezier((0,0),(12f,400f),(0f,400f),(-10f,401f)))
  private lazy val muscleBindings = List(
    (3, m15, 1f),
    (2, m15, 1f),
    (1, m15, 1f),
    (3, h_Mli11, 1f),
    (2, h_Mli11, 1f),
    (1, h_Mli11, 1f)
  ).groupBy(_._1)

  override def render(): List[Spline] = {
    val lowerLipSpline = transformLowerLip()
    val lowerLipJoinerSpline = spline.copy(p1 = newLowerLipSpline.p4)
    lowerLipJoinerSpline.transform(muscleBindings) :: super.render()
  }
```

Putting All Together

```
// Parse and validate the model at compile-time
val emotionsModel = parse[Emotions with Musculature with Features]

// Instantiate the morph
val emotionsMorph = singleton(emotionsModel, new EmoStrategy()).~

// Select Joy and Surprise and their intensities
emotionsMorph.setEmoLevel(JoyId, 0.9f)
emotionsMorph.setEmoLevel(SurpriseId, 0.7f)

// Remorph the morph
emotionsMorph.remorph

// Stimulate the muscles
emotionsMorph.stimulate()

// Render the face
print(emotionsMorph.render())
```

Conclusion

- Protean object may be modeled in current OOP languages only with difficulties
- There is a gap between dynamic and static languages; a need for a hybrid approach
- Object Metamorphism addresses this gap
  - Checking the behavioral model at compile-time
  - Controlled dynamism at run-time
- Downsides and future work:
  - Compilation time, 10.000 alts ~ 2 minute
  - To tackle the performance issues
  - Intelligent elimination of alternatives to speed up

Special Thanks To:

- Oliver Spindler and Thomas Fadrus
- Authors of project Grimace
  - http://www.grimace-project.net
- This presentation uses Grimace's pictures and data published in Oliver's thesis
  - Spindler O.: Affective space interfaces, Technische Universitat Wien, 2009
  - http://www.grimace-project.net/assets/affectivespaceinterfaces.pdf

Appendix1: Emotion Model Strategy

```
val emoStrategy1 = mask[Unit or BasicEmotions]({
    case None => None
    case Some(morph) => morph.getFirstEmotion
})

val emoStrategy2 = mask[Unit or BasicEmotions](emoStrategy1, {
    case None => None
    case Some(morph) => morph.getSecondEmotion
})
```

Appendix 2: Muscle Stimulation

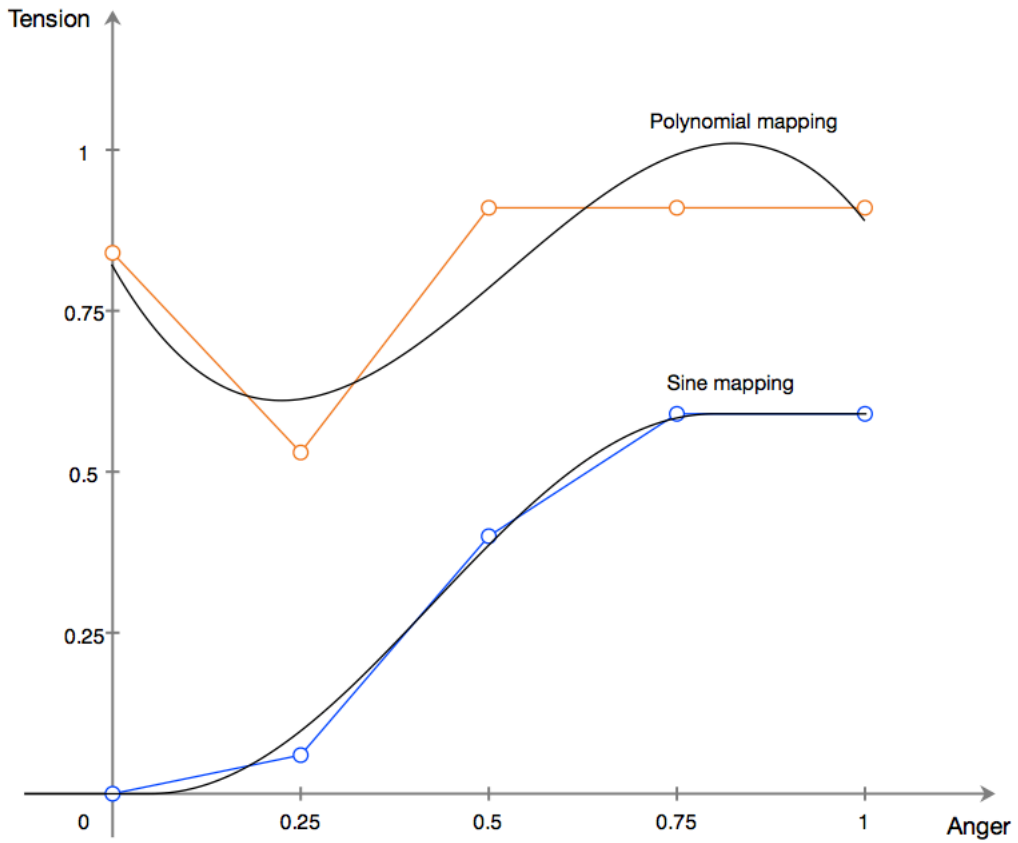


Figure 6.5: Muscle tensions were plotted and interpolated for each emotion.

http://www.grimace-project.net/assets/affectivespaceinterfaces.pdf

Appendix 3: Morpheus

- Project Morpheus: a proof-of-concept of OM
  - https://github.com/zslajchrt/morpheus