Object-orientation for Behavior Modeling and Composition

2017 Korea Conference on Software Engineering

Hiun Kim

hiun@divtag.sejong.edu

B.S. Student / Computer Science

Sejong University, Seoul, Korea

Modern Software is Complex

- Examples
  - Web Applications (high-level APIs; service-oriented architectures;)
  - Mobile Applications (business logics, analytics)
  - User Interfaces (rich/advanced UI; single page applications)
  - Robotics (high-level functionalities)
- Factors
  - Modern software contains **‘many’ ‘high-level’ operation**
  - The operations are varies, share some traits and differ some traits
  - **Variability management** is key issue on modern software engineering
  - High modularity is essential to maintain sustainable software evolution
  - Modularity property includes **reusability, flexibility and comprehension**

Issues on Modularity of Modern Software

- Feature : **prominent or distinctive** user-visible aspect, **quality, or characteristic** of a software system or systems (Kang et al.)

- Each ‘features’ of modern software
  - has some variabilities to provides distinctive functionality
  - shares some commonalities to meet quality which the ***domain constrains***

- Examples
  - robots : every move has backups when crash/collision
  - web service : every user operation should be authenticated and logged
  - Many other software product lines problem tries to handle

Related Works

- Researches on localisation/modularisation of commonalities
- Metaprogramming / Metaobject protocol support for OOP
- Aspect-oriented Approaches (Kiczales at el. `97)
  - Asymmetric Approach
    - AspectJ : base program augmented with aspects (Kiczales at el. `01)
    - Delta-oriented Programming : core module and set of delta module to apply changes like adding/modifying and removes (Schaefer at el. `10)
  - Symmetric Approach
    - Hyper/J : Multi-dimensional seperation of concern and its flexible concern composition tools (Ossher at el. `00)

The Essence of Object-orientation

- Made for Simulation - SIMULA67 (Dahl at el. `67)
- Human to **modeling the real world**
- A perspective/framework of thinking, programming paradigm
  - Inheritance/compose/refine to make the desired thing from abstract thing
  - abstract thing - superclass / specific thing - subclass (and sub-subclass)
  - The Thing, an object is consist of data and behavior

- OOP discovers new ways of analysing requirement and design software

- The success of OOP is inevitable by its idea, **modeling the real world**, since most of our software is working for real world

- Other issues on OOP, traceability, performance and collaboration

Bringing The Idea of Object-orientation into Behavior

Inheritance/compose/refine to make **specific ~~thing~~** from **abstract ~~thing~~**

Inheritance/compose/refine to make **specific behavior** from **abstract behavior**

- The independence of behavior from object by supports its own hierarchical relationship and its own system.

- We can achieve
  - **Reusability** by localising commonalities to abstract behavior and variabilities to specific behavior.
  - **Flexibility** by composing/refine variability to specific by inheritance with well-established OO conventions and techniques
  - **Comprehension** by hiding the detail of behavior and enforcing proper level of abstraction in given programming context

Self-composable Programming

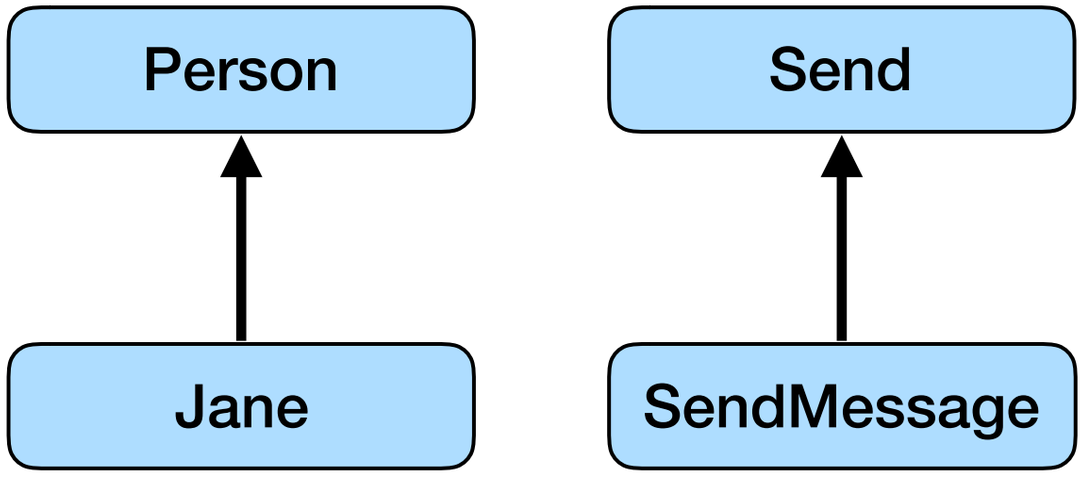
- Introduces mental model of hierarchical relationship of behavior

Event	Object-oriented	Behavior-oriented
Jane picks an apple	Jane => Pick	Pick => Jane
Jane sends an message	Jane => Send	Send => Jane

Abstract Object / Behavior
(localisng commonalities)

Create Instance through Inheritance

Specific Object / Behavior
(refining variabilities)



Self-composability

- For behavior construction and refinement
- To support behavior
  - modular by construction / flexible refinement

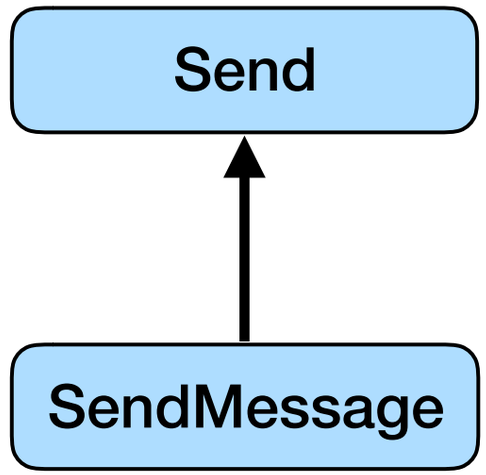
- **Self-addition** : composing behavior sequentially

- **Self-update** : refine specified portion of behavior

- **Self-deletion** : delete specified portion of behavior

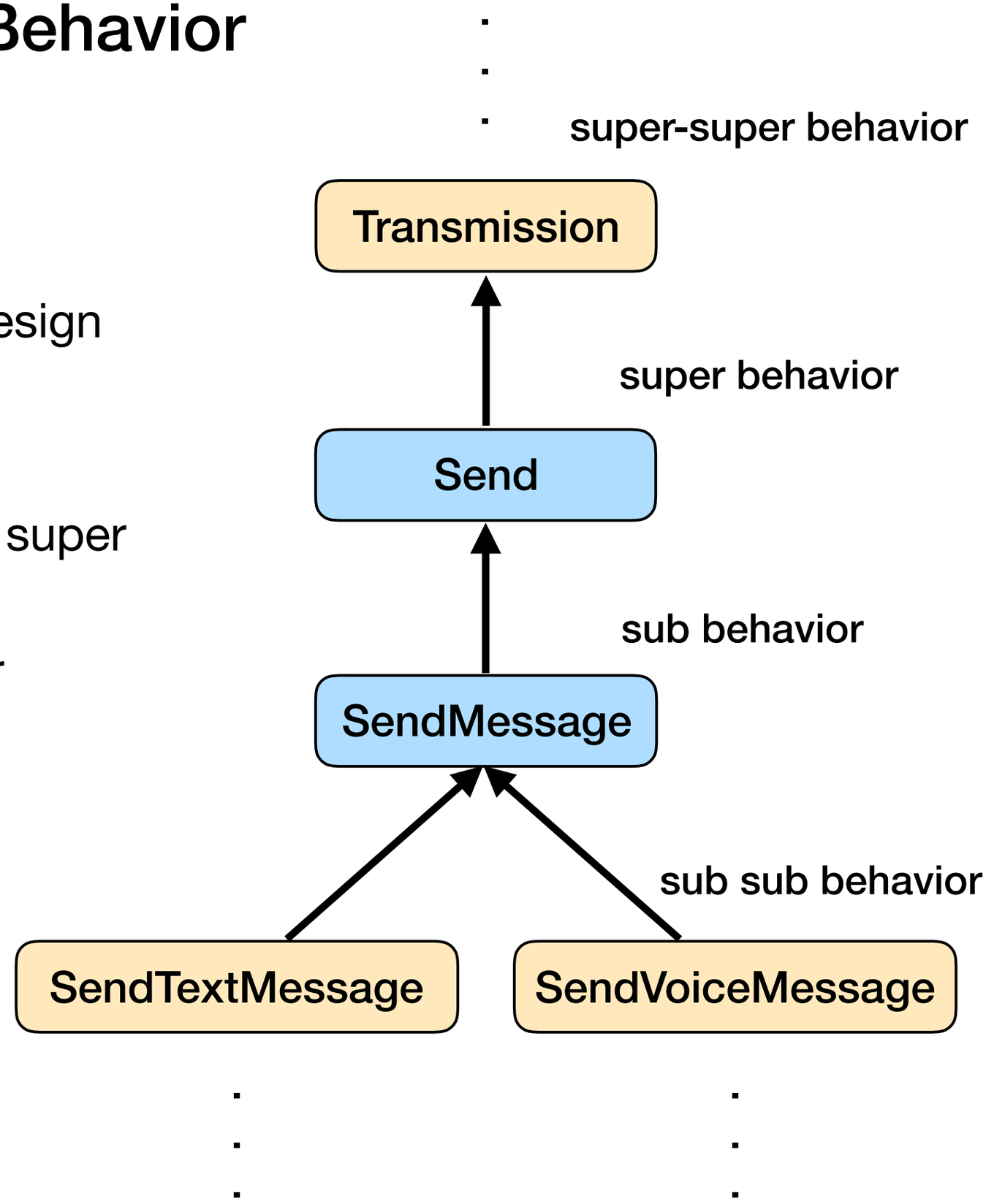
- **Self-manipulation** : free-mode of manipulating portion of behavior

Multi-level Inheritance of Behavior



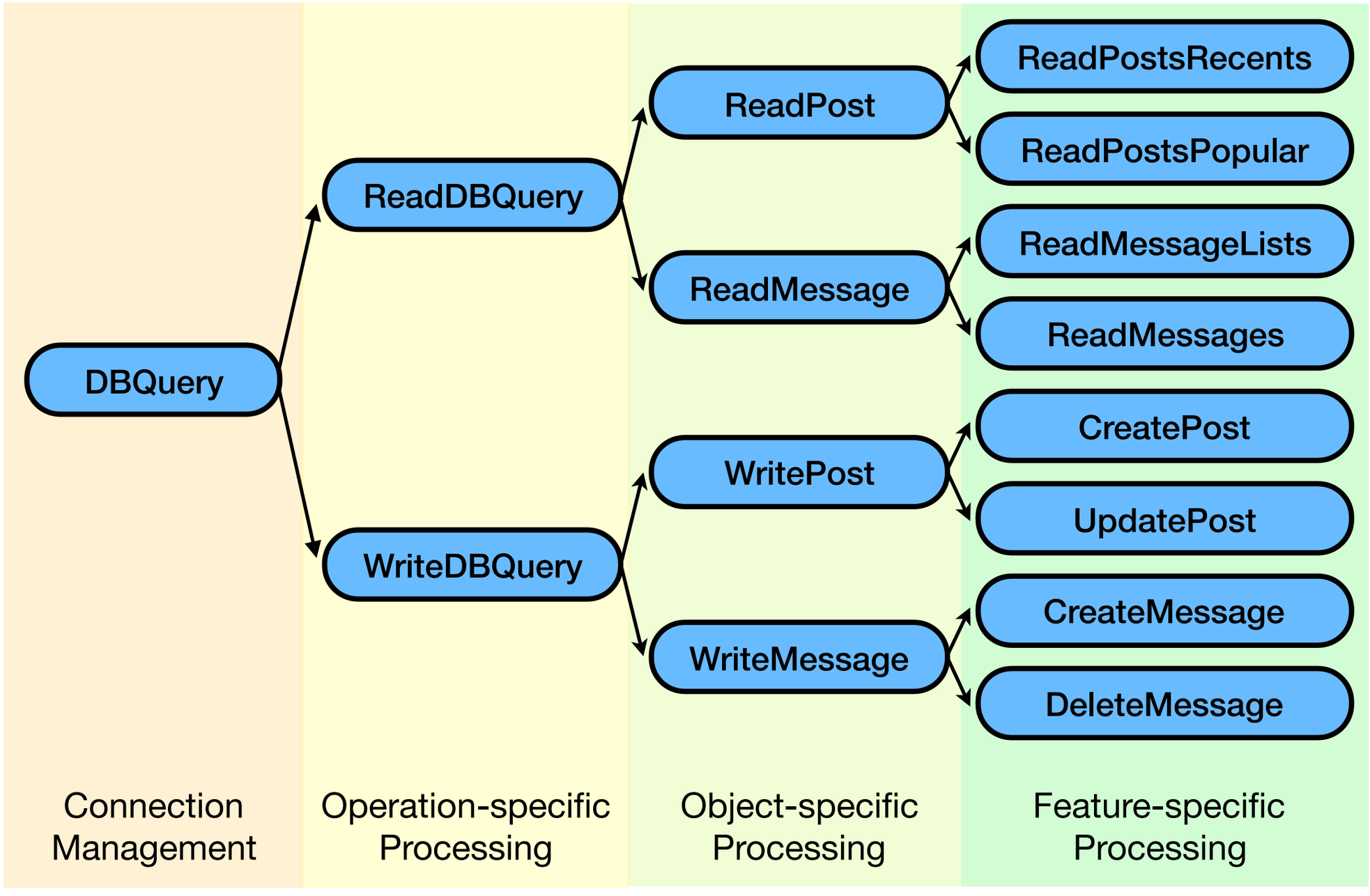
Multi-level Inheritance of Behavior

- Abstract-Specific Relationship
  - Hierarchy on behavior
  - Just like class hierarchy in OO design
- Applying variability
  - inheritance of sub behavior from super behavior
  - apply refinement to sub behavior



Self-composable Domain Analysis

localise commonalities to super behavior (In case of web application)



<Domain of Cross-cutting Concerns per Each Behavioral Level>

Code-level Overview of Self-composable Programming

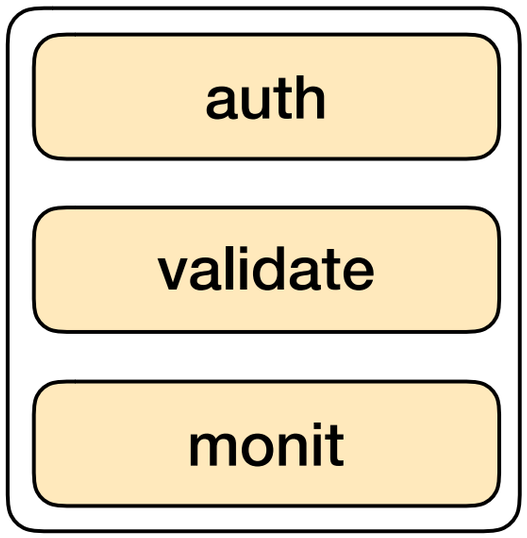
- Self-js : A JavaScript Implementation of Self-composable Programming
- Available at https://github.com/hiun/self-js (will release in stable)
- Method list for Self-composability

Method Name	Description
Behavior#add	Self-addition; Append given sub behavior
Behavior#sub#before	Self-update; Insert given sub behavior before specified sub behavior
Behavior#sub#after	Self-update; Insert given sub behavior after specified sub behavior
Behavior#sub#update	Self-update; Replace sub behavior by given sub behavior
Behavior#sub#delete	Self-delete; Delete specified sub behavior
Behavior#sub#map	Self-delete; Manipulate specified sub behavior in the context of given function

Behavior Construction

- Database-backed, web API that supports both **creation of post and messages with application-wide and object-specific** constrain
- 1st step : behavior construction with application-wide constraint
  - authentication check / data validation / monitoring
- ***Internals.*** Create new behavior array and push each sub behavior

DBQuery behavior



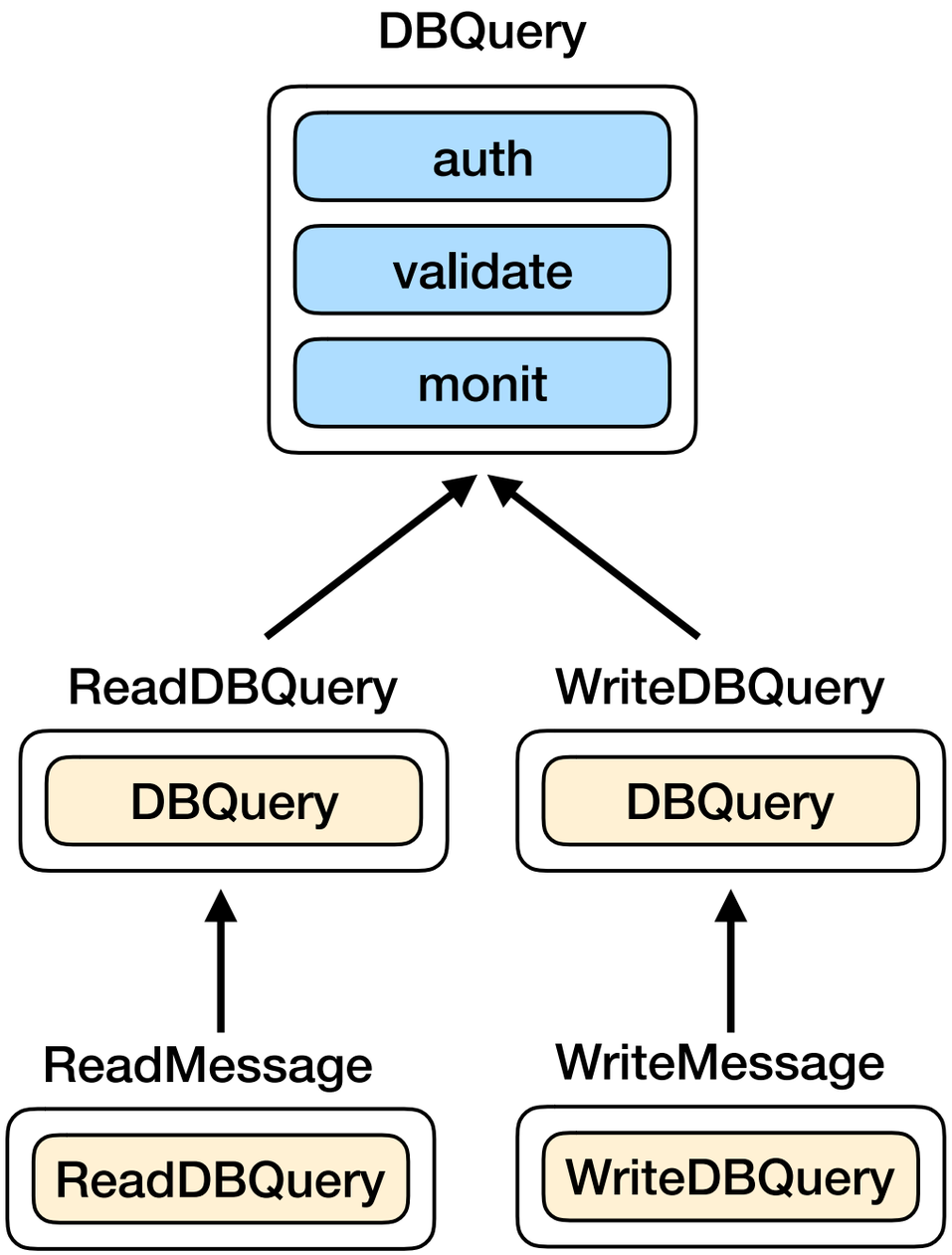

```
var Behavior = require('self');

var DBQuery = new Behavior();

DBQuery.add(auth);
DBQuery.add(validate);
DBQuery.add(monit);
```

Behavior Inheritance

- 2nd step : Inherit constructed super behavior and refine to sub-sub behavior
- ***Internals.*** create new behavior instance, inherit sub behavior and method list

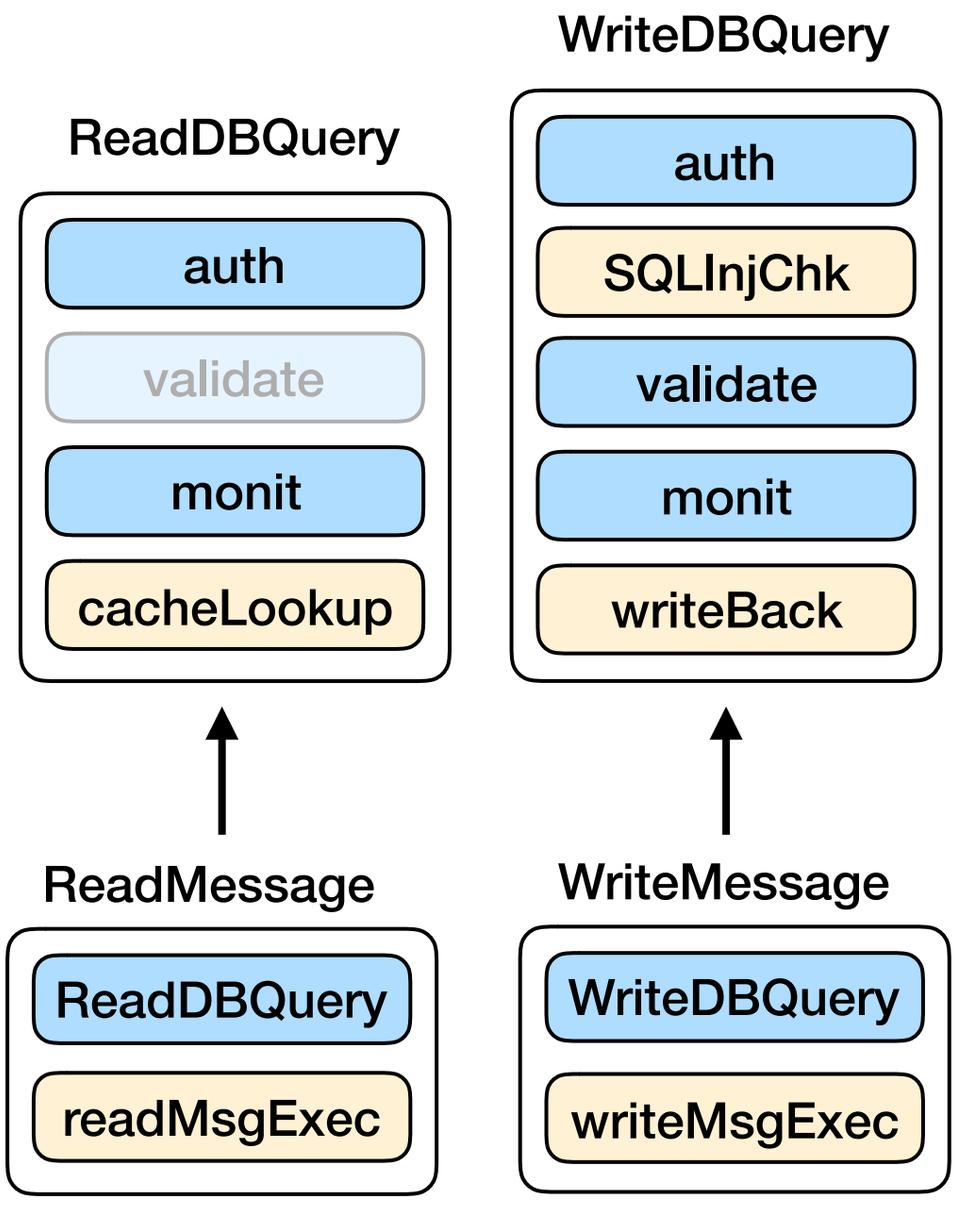



```
var ReadDBQuery = new DBQuery();
var WriteDBQuery = new DBQuery();

var ReadMessage = new ReadDBQuery();
var WriteMessage = new WriteDBQuery();
```

Behavior Refinement

- 3rd step : Refine inherited sub-behavior to create the desired module
- ***Internals.*** Sub behavior placed element in array, manipulating array to refine

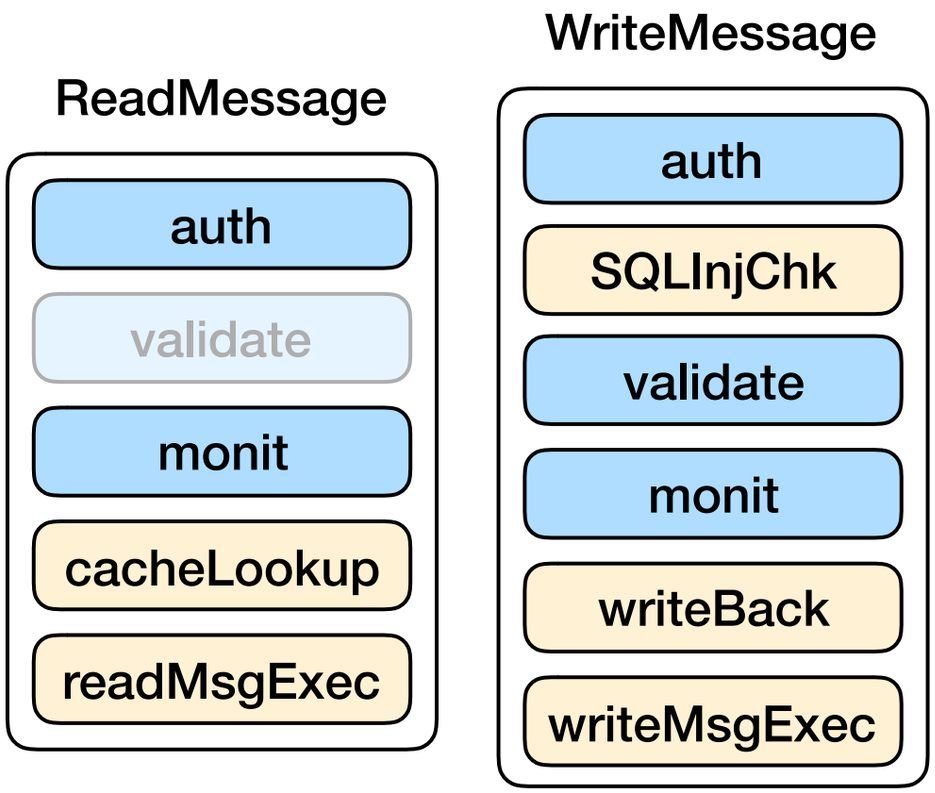



```
//operation-specific refinement
ReadDBQuery.add(cacheLookup);
ReadDBQuery.delete(validate);
WriteDBQuery.add(writeBack);
WriteDBQuery.validate.before(SQLInjChk);

//object-specific refinement
ReadMessage.add(readMsgExec);
WriteMessage.add(writeMsgExec);
```

Behavior Execution

- 4th step : Executed refined behavior with initial arguments
- ***Internals.*** Sequentially invoke sub behavior with initial argument and returning value of succeeding sub behavior




```
CreateMessage.exec([Arguments], Handler);
```

Preliminary Empirical Evaluation Result

- Performs evaluation based on high-level implementation of web service compare to Aspect-oriented Programming(AOP) based implementation
- Implementing web service for database operation around User and Post object with for level of inheritance.
- The efficiency of SLOC come from explicit manipulation for only changed.

Feature Name (Method)
User.getName
User.getProfile
User.getPosts
User.getOnline
Post.getRecentSummary
Post.getRecentsWithoutImage
Post.getPopularSummary
Post.getPopularWithoutImage

Measurements	AOP	Self
Number of Implemented Feature	8	
SLOC for Integration (a)	26	14
SLOC of cross-cutting concerns (b)	18	6
Avg. SLOC per single cross-cutting concern (b/8)	**2.25**	**0.75**

Preliminary Predictive Evaluation Result

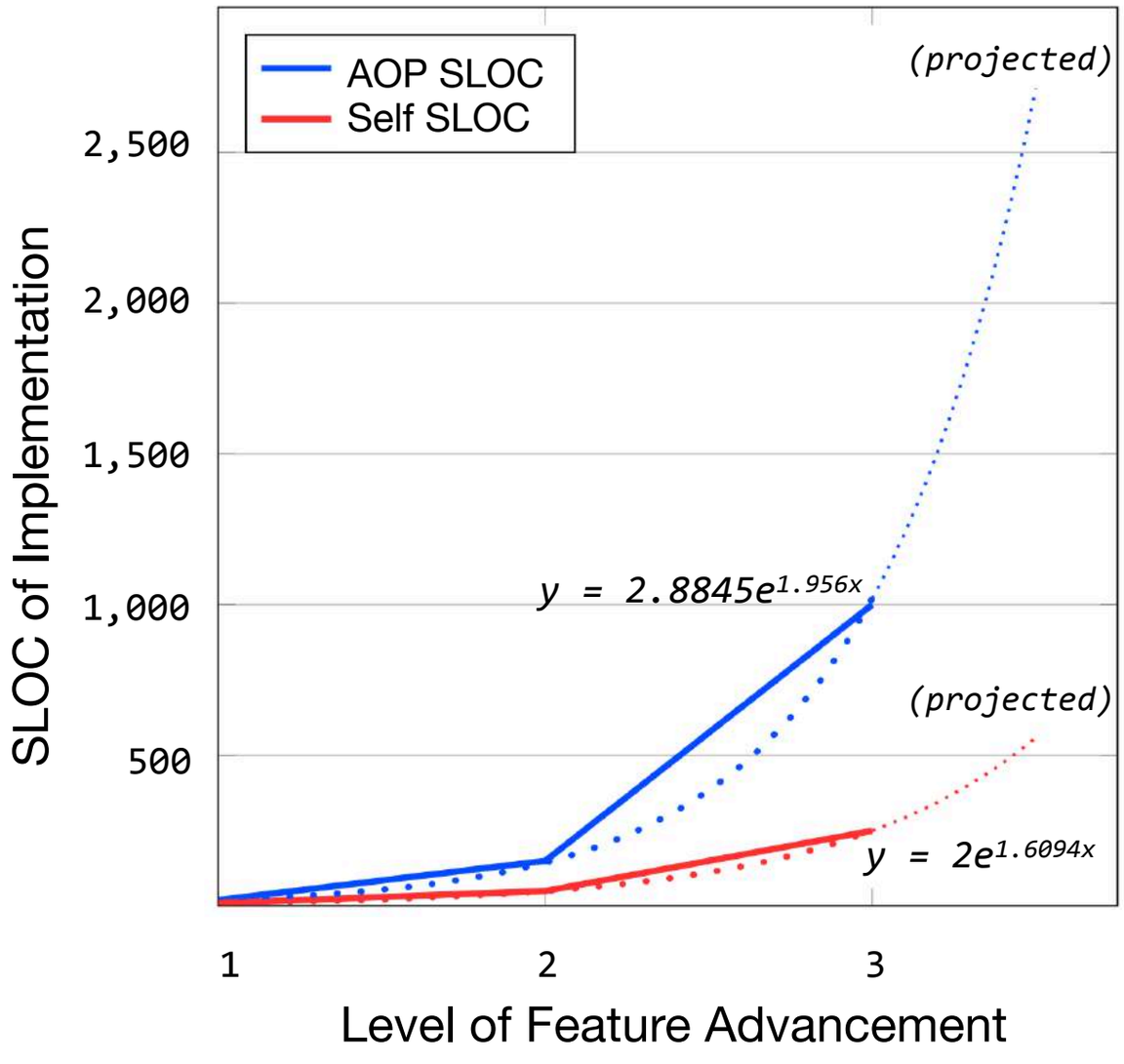
- Using regression analysis to predict SLOC growth per advancement of feature
- As a result, we could confirm efficiency of Self-composable Programming with manipulating only part that has updated

<Self-js Implementation>

Lev. of Inheritance	Num. of super	Num. of sub	SLOC for Refinement	Total SLOC
1st	1	2	**5**	10
2nd	2	5	**5**	50
3rd	5	10	**5**	250
4th	*(projected)*			1,250
5th	*(projected)*			6,249

<AspectJ-like Implementation>

1st	1	2	**10**	20
2nd	2	5	**15**	150
3rd	5	10	**20**	1,000
4th	*(projected)*			7,211
5th	*(projected)*			50,988



Limitations and Future Research Directions

- Scattering and tangling of ~~cross-cutting concerns~~ refinements
  - Same problems of object-oriented design has
  - Robust architectural pattern for representing system behavior directly
    - e.g. DCI architecture for object-oriented collaboration (Reenskaug at el. `09)

- Explicit refinment is like metaprogramming may consider unsafe
  - High-level, implicit refinement by using traits/mixin
  - Domain-specific optimisation by custom module structure/method name

- More empirical studies for proofing efficiency and enhancing theory

- Dedicated language for modeling real-world behavior

Self-composable Programming

- Technique for code-level modeling real-world behavior
  - Modularisation by abstract-specific behavior hierarchy
  - Flexible reusing through OO-fashioned composition and inheritance
  - Opens possibility of advanced refinement by well-established OO theory

- Benefits for highly complex modern software
  - Support code reuse through managed localisation
  - Flexible software composition
  - Improve productivity by raising level of abstraction

- I am looking for Ph.D. position to continue research on programming languages and software engineering please letting me know if you are interested!