A Metric for Frameworks Based on the Deviation of the User’s Understanding from the Manufacturer’s Intention

Ryuichi Kuroda *
kuroriu@fuka.info.waseda.ac.jp

Kouichi Ono **
onono@computer.org

Yoshiaki Fukazawa*
fukazawa@fuka.info.waseda.ac.jp

Abstract: An object-oriented framework is the skeleton of an application program. Application developers who deploy application programs with a framework should reuse whole classes in the framework. Some classes in a framework have virtual functions called “hook methods,” and developers can customize them as function overriding by definition of subclasses. In the process of implementing hook methods, developers must pay attention to the internal structures of classes in the framework and their collaborations. This causes an increase of understanding cost. We believe that framework developers (we call them simply manufacturers) have standard plans for the implementation of hook methods. Frameworks must be designed so as to be suitable for application developers to implement applications with such standard plans. In this paper, we define “indulgence”; a metric for framework design, based on deviations from the standard implementations. This metric measures the likelihood of deviating from the manufacturers’ standard plan. We evaluate the validity of our metric using a real example of a framework.

Keywords: Framework, Metrics, Software Understanding, Design Intention

1 Introduction

A framework, which is a collection of abstract and concrete classes, serves as an object-oriented technique for reusing application designs. In framework-based application development, developers reuse entire classes in frameworks.

Some classes in frameworks have virtual functions called “hook methods,” and application developers can customize frameworks by overriding them. A framework is the skeleton of an application, and involves collaboration between classes. Therefore, developers must understand the internal structures of classes and their collaboration. This increases the cost of framework understanding [1]. To solve this problem, some techniques and tools for application developers have been proposed [2, 3]. However, there are no support techniques or tools for framework manufacturers, and the quality of a framework fully depends on the manufacturers’ skill. Framework manufacturers should provide frameworks that are easy to understand and that ensure reusability and maintainability of applications [4, 5].

* Dept. of Computer & Information Science, Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555, Japan
** Tokyo Research Laboratory, IBM Japan, Ltd., 1623-14 Shimotsuruma, Yamato-shi, Kanagawa-ken, 242-8502, Japan
Thus, if a metric to measure the quality of frameworks is available, it will be useful in the design of the frameworks.

This paper proposes “indulgence,” a metric for domain-specific frameworks. In Section 2, we explain framework manufacturers’ intentions and standard implementations of frameworks, and define the “indulgence” of an application. In Section 3, we devise a method for measuring the indulgence on the basis of the service domain, and in Section 4, we evaluate the validity of our metric by using a real example of a framework.

2 “Indulgence” of Frameworks

2.1 “Manufacturer’s Intention” and “Standard Implementation”

A framework is customized by overriding hook methods in the concrete classes of the framework. Application developers can implement hook methods at will, provided that the hook methods are implemented to behave as part of a framework application. This indicates that application developers must understand the collaborations of classes in the framework.

The framework design process involves identifying common elements among the applications deployed in a target domain. Hook methods are particular elements of these applications. Thus, the framework manufacturer must recognize such uncommon elements that are involved in hook methods. In the design of frameworks, for a particular application domain, a manufacturer must have some intention regarding the plan for implementing hook methods. We call this the “manufacturer’s intention,” and we call the implementation based on it the “standard implementation.”

2.2 Advantages of the Standard Implementation

Application developers who deploy framework applications with standard implementations can obtain the following advantages.

- Easy understanding of implementations
  If some framework applications are developed with the standard implementation, their source code can be compared easily. Even a third person can understand them with ease.
- Improvement of software reusability
  Framework-based application development should not be “claptrap.” An application developed with the standard implementation can correspond to modifications of application requirements. This is related to reusing software resource.
- Improvement of the software reliability
  Assuming that a framework manufacturer has designed the framework properly, the algorithms following its standard implementation would be clearly superior. Thus an application developed with the standard implementation improves software reliability.

2.3 Definition of Indulgence

To implement hook methods with standard implementations, application developers must understand the manufacturer’s intentions. In some cases, application developers can determine a manufacturer’s intentions from the framework source code, API, and specifications. However, in other cases the intentions are unclear and cannot be guessed at by
developers. Applications developed using such frameworks may deviate from the standard implementation, and may not correspond to application users’ requirements. Therefore, it is important to add constraints to a framework design to prevent deviant implementations.

We believe that frameworks should be designed to reject deviant implementations and that the quality of such frameworks will be superior. We therefore define indulgence as a metric of the degree to which a design diverges from the manufacturer’s intention. The expression “a framework has high indulgence” represents a lack of conformance to the manufacturer’s intention in the applications. That is, application developers cannot understand the manufacturer’s intention from the source code and documents of such a framework.

2.4 Premises

We believe the manufacturer’s intention involves the following rules.

- Designation of methods for executing services in the application domain.
- Constraints on the interdependence of method invocations.

A framework provides the methods that execute the domain-specific services. Some methods must be invoked in a hook method. A framework manufacturer designs a framework on the assumption that application developers deploy their applications with standard invocations of these methods. Therefore, these rules represent the standard implementations, and they are part of the manufacturer’s intentions. If a framework has some indulgence with regard to these rules, applications implemented with deviant method invocations may still behave correctly.

3 Measuring Indulgence

Most of the services in frameworks can be represented as state transitions of objects. In our method, therefore, we represent services using statecharts, and use those charts in order to measure the indulgence of the framework.

The purpose of measuring indulgence is to refine a framework through redesign in order to reduce its indulgence. Therefore, this measurement can be regarded as a part of a framework refinement process. Different parts of a framework have different indulgence values. The proposed method measures the indulgence of every object or service, and promotes to redesign those that have the highest indulgence values.

3.1 Statechart for Measuring Indulgence

In our method, nodes of a statechart denote states of the object defined in the framework, and arcs denote method invocations that execute the services most closely related to that object. For example, in a library system domain, a book object has a close relationship with services such as acquiring books, deleting acquired books, lending, and returning, etc. Fig. 1 shows an example statechart for a book object.

First, a framework manufacturer draws a statechart that represents the standard state transitions for a specific object; we refer to this statechart as the SSC (Standard State Chart). Application developers also draw statecharts for their own applications of the same object. If the framework has high indulgence for services related with that specific
object, some statecharts of applications will have paths different from those of the SSC. We define the indulgence based on the statechart as the probability of the appearance of different paths in the statecharts for an application. In this and the next section, we use the word “indulgence” in this specific sense.

To generalize our measurement method, considering a statechart as the set of states and paths connecting those states, we define a notation for statecharts. Tab. 1 shows the components of a statechart and their syntax in our notation.

**Tab. 1: Syntax of statechart notation**

<table>
<thead>
<tr>
<th>component</th>
<th>figure</th>
<th>notation</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source node</td>
<td><img src="source_icon" alt="Source" /></td>
<td>SOURCE</td>
<td>Initial state of an object.</td>
</tr>
<tr>
<td>Sink node</td>
<td><img src="sink_icon" alt="Sink" /></td>
<td>SINK</td>
<td>Final state of an object.</td>
</tr>
<tr>
<td>Service state node</td>
<td><img src="state_icon" alt="State" /></td>
<td>state name</td>
<td>A valid state of an object defined in the target domain of a framework.</td>
</tr>
<tr>
<td>Transit node</td>
<td><img src="transit_icon" alt="Transit" /></td>
<td>A node used for the sake of convenience when an object transits states using two or more method invocations.</td>
<td></td>
</tr>
<tr>
<td>Service execution arc</td>
<td><img src="method_icon" alt="Method" /></td>
<td>method name</td>
<td>A method invocation that executes a service.</td>
</tr>
</tbody>
</table>

A statechart consists of a set of states and a set of paths. In our notation, when a statechart SC is given, S(SC) represents a set of states and P(SC) represents a set of paths in SC. Each complete S(SC) must have two states: SOURCE and SINK.

Let M be a set of methods defined in the framework, a path p which connects a state u to another state v (u, v ∈ S(SC)) by method invocations f₁, f₂, ..., fₙ (∀fᵢ ∈ M) on this order is described as

\[ p = (u, v, (f₁, f₂, ..., fₙ)). \]

And then we define the description of properties of a path, s(p) represents the start state of a path p, t(p) represents the end state of p, and d(p) is the number of method invocations in p.

For instance, the statechart drawn in Fig. 1 is transcribed as:
\[ SC = \{ \{ \text{SOURCE, SINK, OnShelf, OnLoan} \}, \]
\[ \{ \text{(SOURCE, OnShelf, (addBook)), (OnShelf, OnLoan, (lend))}, \]
\[ (\text{OnLoan, OnLoan, (reserve)}), \quad (\text{OnLoan, OnShelf, (return)}), \]
\[ (\text{OnShelf, SINK, (lost)}), \quad (\text{OnShelf, SINK, (deleteBook)}), \]
\[ (\text{OnLoan, SINK, (lost)}) \} \)

3.2 Service Path Table and Implemented Service Table

A statechart has paths for multiple services, therefore it is necessary to identify which services each path represents. Moreover, an application program with a framework does not always use all of the services provided by the framework. We therefore must inquire as to the services adopted by each application program. We now define two types of tables in order to support these measurements.

**Service path table:** To identify the relations between services provided by frameworks and paths drawn in statecharts, the framework manufacturer tabulates them. We call this list the “service path table.” Tab. 2 shows an example of the service path table for the book object appearing in Fig. 1.

<table>
<thead>
<tr>
<th>service</th>
<th>object</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acquiring books</td>
<td>book</td>
<td>((\text{SOURCE, OnShelf, (addBook)}))</td>
</tr>
<tr>
<td>Deleting acquired books</td>
<td>book</td>
<td>((\text{OnShelf, SINK, (deleteBook)}))</td>
</tr>
<tr>
<td>Lending books</td>
<td>book</td>
<td>((\text{OnShelf, OnLoan, (lend)}))</td>
</tr>
<tr>
<td>Returning</td>
<td>book</td>
<td>((\text{OnLoan, OnShelf, (return)}))</td>
</tr>
<tr>
<td>Reservation</td>
<td>book</td>
<td>((\text{OnLoan, OnLoan, (reserve)}))</td>
</tr>
<tr>
<td>Recording loss of books</td>
<td>book</td>
<td>((\text{OnShelf, SINK, (lost)})), ((\text{OnLoan, SINK, (lost)}))</td>
</tr>
</tbody>
</table>

By using the service path table, we can obtain the standard statechart for an application which does not use all services. For example, in the library system domain, the standard statechart for an application without the reservation service is as shown in Fig. 2.

![Standard statechart without reservation service](image)

**Fig. 2:** Standard statechart without reservation service

**Implemented service table:** To survey which services application developers implemented, a manufacturer can send them a questionnaire. An implemented service table is a compilation of questionnaire results. Tab. 3 shows an example. This table consists of the result of five implementations for the application’s requirement.
3.3 Difference Ratio

Our method measures indulgence by means of a difference ratios between statecharts. The difference ratios are calculated at three levels: (1) path-path level, (2) path-statechart level, and (3) statechart-statechart level.

For the first level, \( \text{diff}(p_i, p_j) \) is defined as the difference ratio between two paths \( p_i \) and \( p_j \), given by

\[
\text{diff}(p_i, p_j) = \begin{cases} 
\frac{d(p_i) + d(p_j) - 2 \times m(p_i, p_j)}{d(p_i) + d(p_j)} & \left( s(p_i) = s(p_j) \cap t(p_i) = t(p_j) \right) \\
1 & \text{(otherwise)} 
\end{cases}
\]

where \( m(p_i, p_j) \) is equal to the maximum number of arcs that appear on both \( p_i \) and \( p_j \) in the same order. For example, assume that there are two paths as follows:

\[ p_i = (u, v, (f, g, h, i)) \]
\[ p_j = (u, v, (g, j, h, i, f)) \]

In this case, arcs \( g, h, \) and \( i \) appear in the same order, therefore the \( m(p_i, p_j) \) is 3.

On the second level, \( \text{diff}(p, SC) \) is defined as the difference ratio between a path \( p \) and a statechart \( SC \), given by

\[
\text{diff}(p, SC) = \min(\{ \text{diff}(p, p') | p' \in P(SC) \}).
\]

On the third level, \( \text{diff}(SC_i, SC_j) \) is defined as the difference ratio between two statecharts \( SC_i \) and \( SC_j \), given by

\[
\text{diff}(SC_i, SC_j) = \frac{\sum_{p \in P(SC_i)} \text{diff}(p, SC_j) + \sum_{p \in P(SC_j)} \text{diff}(p, SC_i)}{|P(SC_i)| + |P(SC_j)|},
\]

where \( |P(SC)| \) represents the number of paths in \( P(SC) \).

3.4 Method of Measuring Indulgence Based on the Application Domain

In order to measure indulgence, the following documents are assumed to be given as input.

- \( SSC \) : A standard statechart for an object defined in the framework.
- \( ASC_1, ASC_2, \ldots, ASC_n \) : Statecharts for \( n \) applications \( App_1, App_2, \ldots, App_n \).
- Service path table
- Implemented service table

Then our method measures indulgence according to the following procedures.

Object level indulgence
1. For all $i$ that satisfy $1 \leq i \leq n$, referring to the service path table and the implemented service table for paths of services that are not implemented in $App_i$, remove those paths from $SSC_i$. We refer to these charts as $SSC'_i$.

2. Let $IDG_O$ be the indulgence of the target object. This is,

$$IDG_O = \frac{1}{n} \sum_{i=1}^{n} \text{diff}(SSC'_i, ASC_i). \tag{4}$$

**Service level indulgence**

Let $P'$ be a set of paths to execute a specific service, and $ASC'_1, ASC'_2, \ldots, ASC'_m$ are statecharts of applications implementing that service. Then $IDG_S$: the indulgence value of the service is given as

$$IDG_S = \frac{1}{m \times |P'|} \sum_{i=1}^{m} \sum_{p \in P'} \text{diff}(p, ASC'_i). \tag{5}$$

4 Evaluation

In this section, the relationship between the possibility of deviation from the standard implementation and the indulgence value given by the proposed method is verified. The sample framework used for our experiment is in the library system domain. Services defined in this domain include entering books/users, deleting entered books/users, lending, return, reservation, recording loss of books, and restocking.

This framework was designed to allow application developers to deviate from standard implementations in the lending, reservation, and restocking services. For example, in the lending service, this framework produces the following methods.

**Rental()**: Proper interface for executing a lending service.

**PreRental()**: Method for prior transactions related to the lending service.

**RentalDash()**: Main method for shifting the state of a book object to “on loan.”

Strictly speaking, **Rental()** should be a concrete function, but we designed it to be a virtual function. Accordingly, for example, developers can implement a deviant application such as one that includes in **Rental()** prior transactions that should be implemented in **PreRental()**.

Tab. 4 shows the measurement results of $IDG_S$ for each service and $IDG_O$ for three objects: **Book** as a book object, **User** as a user object and **LibrarySystem** as a library system object. Services that are not included have an indulgence value of 0; that is to say, they follow the standard implementation.

**Tab. 4:** Indulgence values of various objects and services

<table>
<thead>
<tr>
<th>object</th>
<th>$IDG_O$</th>
<th>service</th>
<th>$IDG_S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Book</td>
<td>0.17</td>
<td>Lending</td>
<td>0.29</td>
</tr>
<tr>
<td>User</td>
<td>0.00</td>
<td>Returning</td>
<td>0.29</td>
</tr>
<tr>
<td>LibrarySystem</td>
<td>0.00</td>
<td>Recording Loss</td>
<td>0.29</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reservation</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Restocking</td>
<td>0.64</td>
</tr>
</tbody>
</table>
Of the three services designed to allow developers to deviate, reservation and restock had higher indulgence values than the other services; however, the value for the lending service was same as the other services. The likely reasons for this are as follows.

- Most virtual functions were not overridden but used the default implementation.
- Developers understood the standard implementation of the lending service, because they had already implemented other services whereby deviation was difficult (e.g., the return service).

5 Conclusions

In this paper, we propose that standard implementation affects the maintainability of a framework application, and that frameworks should be designed to encourage developers to follow the standard implementation. Accordingly, a metric called indulgence was proposed to measure the quality of a framework design. The indulgence value measured by the proposed method is related to the potential danger of deviating from the standard implementation. By redesigning their frameworks on the basis of the measured indulgence values, framework manufacturers can refine them to ensure that implementations of applications will be similar to the standard implementation.

However, the present method has the following problems.

- Dependence on sample applications
  As stated in the previous section, in some cases, services designed to allow application developers to deviate have low indulgence values because the proposed method relies on sample applications. Therefore, the method should be extended to cover a wider range of applications and to ensure that services designed to allow application developers to deviate have high indulgence values.

- Effects on other services
  Normally, objects provided in a framework are not related to only one service. Thus, the redesign of a service to decrease its indulgence may change the indulgence of other services. Some tools clarifying for framework manufacturers such effects will be required.

Our future work will include finding a solution to these problems and the development of a new method capable of measuring the indulgence of frameworks in cases not covered by the present method.

Bibliography