Attempting to Increase Longevity of Applications Based on New SaaS/Cloud Technology

● Toyoaki Furusawa

(Manuscript received November 11, 2009)

In recent years, SaaS/Cloud technology has advanced significantly in development and operation, and IT vendors including Fujitsu must keep up with these developments. In this field, we need to ensure applications have a long life cycle. This paper introduces the concept of a meta-framework, which is a framework of frameworks for resolving the issues involved in extending the life of applications. The basic idea of a meta-framework is to validate the design and development process, and separate the design and framework processes. This will bring us closer to the ideal form of software development in which the end of design signifies the end of testing. By separating the design and the framework, the design can be a permanent asset. In addition, by creating a common base, i.e. the meta-framework, it will be possible to easily migrate a system to the latest framework. We will be able to create templates that are used to automatically generate source codes, and expanding the scope of automatic generation will reduce the cost of migration.

1. Introduction

In the early part of the 21st century, a high-speed Internet has become widespread and the environment surrounding IT is undergoing a significant transformation. As represented by the new concept of Web 2.0, the Web is no more just an environment for browsing information but has become indispensable as infrastructure for the whole of society as well as for the activities of individual corporations, and it has also considerably changed in quality.

Various applications such as groupware, Customer Relationship Management (CRM) and Enterprise Resource Planning (ERP) are now offered as services on the Internet, and these are referred to as SaaS (Software as a Service). Among such applications, salesforce.com, which has achieved success in the field of CRM and boasts over one million user licenses around the world, is especially well-known.

In addition to applications, IT resources themselves such as hardware, OS, middleware and development and verification environments have started to be globally offered as services (Microsoft, Google and Amazon are leading this field). As a result, a substantial reduction in the cost of IT investment in relation to the adoption and operation of IT is being brought about. For example, when designing networks systems engineers are not required to construct or test system load sharing. In other words, the amounts of customers' investment in environment construction, and operation and maintenance, which have so far been a significant source of revenue for system integrators, are about to be greatly reduced.

A similar change is taking place in system development as well. The concept of agile development has long been in existence and the frameworks for realizing it[note 1)] have made rapid progress in the last five years. Among them are Ruby on Rails[1)] based on the principle of Convention over Configuration (COC)[note 2)] and Force.com.[2)] They have features in which defining models allows automatic generation of most source codes and the results can be immediately checked in the verification execution environment. With Force.com, even self-customization screens for end users are automatically generated.

Furthermore, in Cloud computing, system development and an operation environment is offered as a service and customers expect the life of the system, which is an asset on top of the environment, to be extended. Up to now, a vast amount of investment has been required every time the CPU, OS, middleware or framework was updated but it may be remarkably reduced or equalized at a low level, which is what customers expect.

In this way, the environmental change has created a pressing need for a considerable reduction in all of the development, introduction, and operation and maintenance phases and enduring it and expanding business has now become a requirement for IT vendors to survive. In SaaS/Cloud, competition without borders is appearing. To survive in such an age, a highly competitive Cloud platform and development infrastructure that are globally applicable are required.

This paper describes the direction of a next-generation application development style suggested by new technologies such as SaaS/Cloud.

2. Issues in system development

The following issues must be resolved to reform development and operation and have a longer life.

1) Different frameworks

In system integration, so many different frameworks exist that every type of business or every customer company is said to have their own framework. To develop an application, a data model, workflow, screen layouts/items and screen transitions exist on each framework, and they are dependent on the type of business. This is hampering the standardization and reuse of know-how and components. Personnel with development skills are fixed in their positions and it is not easy to reassign them according to demand, resulting in higher costs.

In addition, each framework must independently pursue the progress of the software technologies in existence and the investment for enhancing the framework itself is not efficient. Eventually, keeping up with the technological progress in the world becomes impossible and what is manually made by system engineers comes to take on greater importance to compensate for a shortage of functions and the costs grow even higher. This is likely to make the framework outmoded and weeded out.

2) Too much emphasis on scratch building as system integrators' style of development

System integrators' ways of thinking, processes and frameworks have so far been premised on development from scratch. Architecture that allows gradual functional enhancement has been believed to be impossible in technical terms as well. A modification that is made to add functions may affect the whole system and substantial costs were required. In recent years, frameworks have appeared that output initial sources from design information, which often contribute to improved efficiency, but attempting to add functions to a system that has been already completed unavoidably requires wide-scale modification including automatically generated source codes.

For this reason, to meet customers' requests

note 1) Implementation method in application development: Eclipse + Struts is widely known in Java programming.

note 2) Concept that uses conventions for naming and such like to simplify software structure more than controlling applications by configuration: indispensable for efficient automatic generation of source codes.

for extra functions, costs that exceed the added value of the additional functions are required.

3) No approach to extending life of system

Customers do not readily invest in systems without added value. When a system that is actually running is migrated along with a renewed piece of hardware or OS, the functions basically do not change significantly. However, estimates of system integrators show huge costs because the renewal of the platform alone requires everything to be rebuilt and customer satisfaction cannot be obtained. What has been made and is running requires huge amounts for migration alone.

This has been the fate of open systems because the CPU, OS, middleware and framework are renewed beyond system integrators' reach. To offer the system in a Cloud environment, however, the system infrastructure is hidden from customers, which may make it difficult to account for the costs of renewal.

As for the above issues, some say that the use of standardized frameworks can resolve the issues described in 1) and 2). However, frameworks have limited lives. As software technologies progress, standardized frameworks also undergo transformation. With Struts, for example, differences existed that could be described as generational changes when it shifted from V1 to V2. These changes were made to keep up with new innovative functions such as Grid computing and capture of distributed database, and with future improvements in development performance, operability and maintainability. As a result, frameworks that have failed to keep up in these areas are weeded out and frameworks also undergo structural change. In such cases, the frameworks themselves go through substantial generational changes accompanied by no little incompatibility.

3. Proposal of meta-framework

Frameworks involve the challenges described in the previous section. To take an example from the case of Fujitsu, there are general-purpose frameworks such as EZDeveloper,[3] QuiQpro,[4] and eProad,[5] and it may be possible to integrate them. These architectures are basically based on the MVC model and the compatibility between them starts off being high. It is not impossible to share design information and a source code generation engine and data model between such frameworks.

We have assumed that further pursuing this direction may solve the problems presented in the previous section and listed the requirements for the framework as shown below.

1) Standardization of data and screen models

To start with, data models and screens are not dependent on the language, architecture or framework of the system but are the most basic design information that can be standardized. Describing the design information of individual systems in a form independent of the language or framework and controlling that information allows us to extract pure design information, which is invariable regardless of the form of infrastructure.

2) Multi-framework

It would be ideal to have the capability to migrate from the same design information to the most appropriate framework for a given type of business or form of operation as required. For that purpose, we need to have extendibility that allows source codes to be output based on various data models and screen standards dependent on the type of business. If a template or source code generation engine for outputting source codes from design information can be easily extended for each type of business, source code generation based on customer-specific frameworks as well as many business fields becomes possible, and this is expected to help reduce the costs of migrating to new hardware or OS's in addition to improving productivity and quality in a wide range of fields.

3) Bi-directionality

In many cases, automatically generated

source codes as they are cannot satisfy customer requirements and the output source codes need to be edited. Once edited, the result ceases to synchronize with the original design information and modification of the result requires manual modification of the original information and vice versa. For example, modification of design after source codes have been changed cannot take advantage of the automatic generation of source codes. For this reason, the function to automatically generate source codes can only be used in the initial phase of scratch development (development of an original system) and cannot help to improve the efficiency of developing packages or SaaS that undergo constant functional enhancement or incremental SI development.

In such a system, the development process flows only in one direction. Making this bi-directional can improve the efficiency of modification and allow incremental development that starts small and grows large. For this purpose, we need a system for extracting and managing the added source to reflect it in the source generation template. To address this issue, the direction is indicated by the Generation Gap pattern and Hook Operation pattern[6] in the design patterns but the restrictions that apply when the patterns are used in the actual applications must be verified.

4) Real-time operation verification

With agile frameworks such as Force.com and Ruby on Rails, when a model (information) has been modified, it is immediately reflected in the source code and distributed to servers automatically in a verification environment and the operation can be verified at once. This is an effective way to minimize any additional work when a problem is found, and will likely help to reduce development costs. In the next-generation development environment, the ability to conduct a real-time check on the result of design change will become common.

The systems mentioned in 1) to 4) are not dependent on the framework and provide a very versatile development method. Because it is a foundation common to multiple frameworks, we call this a "meta-framework."

The following explains the concept of the development style assumed in the meta-framework shown in **Figure 1**.

1) Model design

A model library has major data models registered in advance including persons, organizations and commodities. A GUI tool for model design is used to select data models to use as bases, create derivative models and associate between the models.

2) Screen design

A screen design template is used to select screens to use as bases and a GUI tool is used to specify the relationships with the models created in 1).

3) Automatic generation engine

1), 2) and source templates dependent on the language and framework are input for outputting a source code.

4) Editing

The source code is modified as required. The result of the modification is converted into an additional source template, which constitutes part of the input in 3). This achieves bi-directionality.

5) Development and verification

The generated source is compiled and distributed to the development and verification environment and then developed and executed.

Ideally, to add a simple attribute, 2), 3) and 5) following 1) should be automatically executed and the result verified at once, and this has been partly achieved at a framework level with Force.com and Ruby on Rails.

4. Effect of meta-framework and future development

The following summarizes the effects that can be expected when a meta-framework is realized.

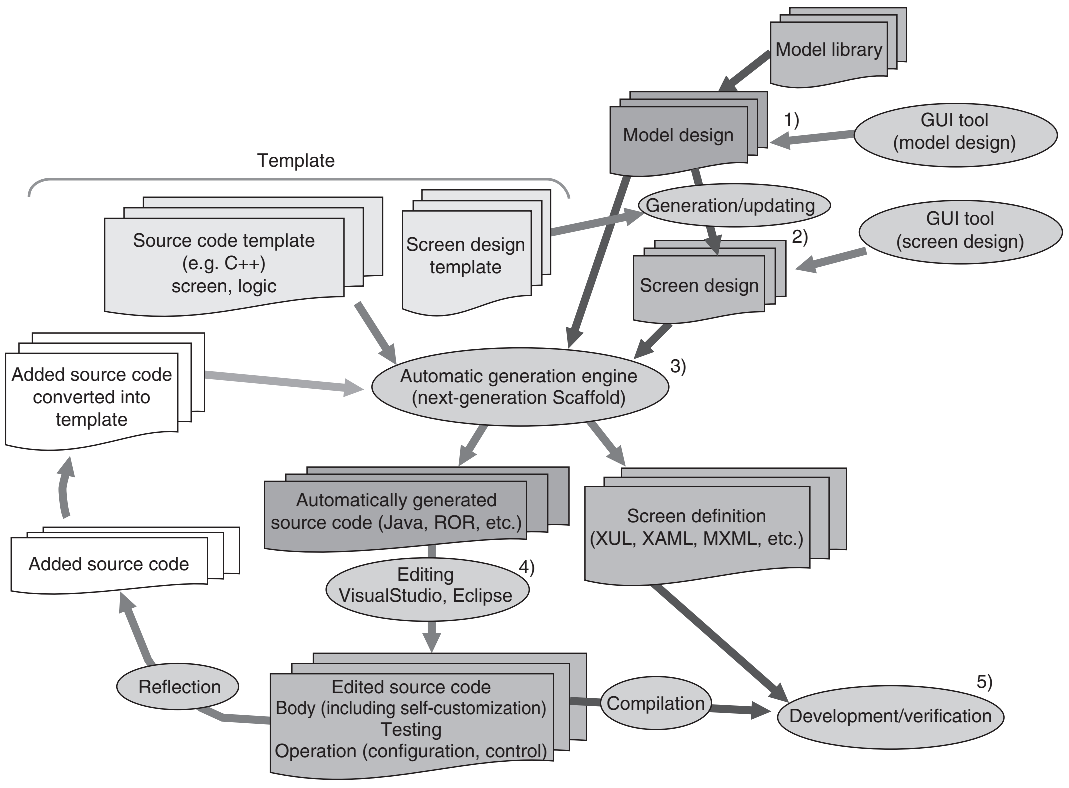

Figure 1
Concept of meta-framework.

4.1 Extended life of developed asset

For migrating to a new environment, design information can be utilized to output a source for a new framework. Porting only the additional source code of the older version minimizes the migration costs. Accordingly, survivability of business know-how, which is a core asset, can be improved.

4.2 Mobilization of resources by standardization of styles

Work is mostly model design and testing. The development procedure is common, which makes it easier to reassign personnel.

4.3 Improved efficiency and response to cumulative business issues by agile development

Starting small and growing large becomes possible. New development costs are said to account for 20% of customers' IT investment budgets. How to propose system improvement at reasonable costs while carrying out operation and maintenance, rather than scratch building, will be one of the main issues of business in the future.

5. Conclusion

Fujitsu and Fujitsu Laboratories have started studying the feasibility of a development and operation environment in the age of Cloud computing and a meta-framework is among the themes of this study. We intend to move forward with the research and study for technical verification that will allow us to give an interim report by the end of fiscal year 2009.

References

1) Ruby on Rails.
http://rubyonrails.org/
2) Force.com.

http://www.salesforce.com/platform/
3) Fujitsu: "Manufacturing Innovation" in Software Development (especially, EZDeveloper). (in Japanese), *FUJITSU JOURNAL*, Vol. 33, No. 5, pp. 2–7 (2007).
http://jp.fujitsu.com/about/journal/publication_number/300/topstory/03.shtml
4) QuiQpro. (in Japanese).
http://quiq.fsol.co.jp/
5) Fujitsu Kansai Systems: SDAS for .NET /eProad. (in Japanese).
http://jp.fujitsu.com/group/fks/services/eproad/
6) H. Yuuki: Generation Gap pattern and Hook Operation pattern. (in Japanese).
http://www.hyuki.com/dp/dpinfo.html

Toyoaki Furusawa
Fujitsu Ltd.
Mr. Furusawa received a Ph.D. in Elementary Particle Theory from Osaka University, Osaka, Japan in 1986. He joined Fujitsu Ltd. in 1988 and has been engaged in developing artificial intelligence products, middleware products based on Java, and ERP packages. Currently, he is in charge of research and development of SaaS/Cloud technology for system engineers.