The JWAM Framework: Inspired By Research, Reality-Tested By Commercial Utilization

Holger Breitling, Carola Lilienthal, Martin Lippert, Heinz Züllighoven

University of Hamburg
Computer Science Department, SE group
&
APCON Workplace Solutions
Vogt-Kölln-Straße 30
22527 Hamburg, Germany

{breitling, lilienthal, lippert, zuellighoven}@jwam.org

ABSTRACT

Developing a framework is not an easy task and can be done on several levels and for a variety of application domains. There are various techniques within the framework community designed to support the process of framework development on an architectural and an organizational level. But how can we combine these techniques to really succeed in professional software development?

We have concentrated on building frameworks for large-scale software development using the Tools & Materials approach, gaining experience in this area over the past ten years. In this paper, we focus on the Java-based application framework JWAM, which combines research results on framework design with industrial project work. It reifies our experience, knowledge and best practices gleaned over the past ten years.

We outline below the architectural and organizational guidelines that have helped us to build a successful application framework for developing commercial software fast and flexibly. Techniques like framework layering and the partitioning of the framework into a kernel and framework components are combined with eXtreme programming, extreme testing and an extremely useful team development environment.

1 BACKGROUND: FROM ACADEMIC PLAYGROUND TO PROFESSIONAL APPLICATION FRAMEWORK

History

The foundations for the JWAM framework (see [9]) were laid years ago by students at the University of Hamburg's software engineering department in their diploma theses. They decided to build support for constructing Java applications based on the Tools & Materials approach[1]. They combined the experience gained from several research frameworks already built in C++, Smalltalk and Eiffel with the research results from industrial cooperation projects. It soon became clear that the rapidly evolving framework was a great asset to the department's research and teaching activities, which began to focus on it: the staff and students involved decided to concentrate on simultaneously improving both the quality of the framework and its development process. It was obvious that a framework could only be successful if real-world applications were built on top of it and if the framework development were professionally and commercially supported.

As a consequence, a spin-off company, APCON Workplace Solutions, was founded to ensure the framework's commercial utilization.

Today, the JWAM framework consists of more than 800 classes, a small kernel and several additional components. It is used within a number of professional software development projects by the company itself as well as by licensees. APCON Workplace Solutions has initiated the development of commercially relevant components like host-system adaptation or ERP-system integration.

Combining Academic Research with Industrial Software Development

The key idea of our development approach is to combine the inspiration and freedom of academic research (which allows us to try things out of scientific curiosity) with the experience from commercial software development, the latter helping us to identify common design requirements as a basis for framework abstractions.

These two resources for framework development are represented by the two circles in Figure 1. We call this combination "practice-oriented research and research-oriented practice".

We chose the following organizational form for our work: In addition to full-time software developers, APCON WPS employs several part-time developers who also work as research and teaching assistants at the University. The students employed by APCON WPS work part-time for the company while pursuing their studies. The framework development process is steered by a so-called *Architecture Group* with a maximum of eight members. These members are professional software developers, part-time researchers or selected students.

[1] The Tools & Materials approach for object-oriented software development was created and developed by the software engineering group around professor Heinz Züllighoven (see [4]).

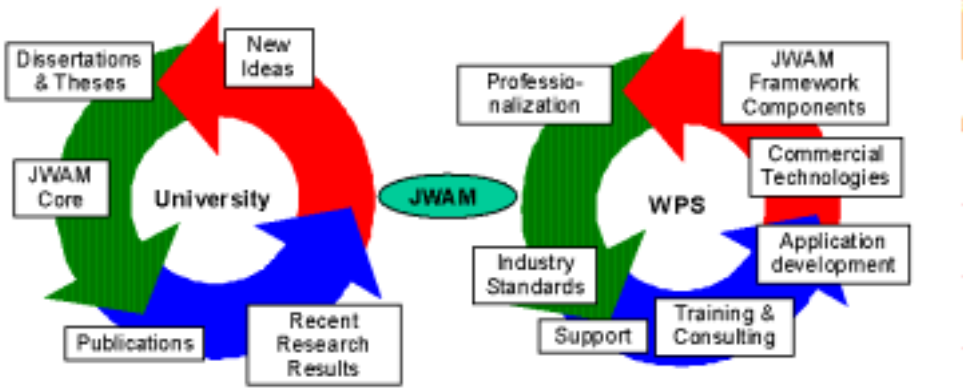

Figure 1: Practice-oriented research and research-oriented practice

Today framework development chiefly means refactoring (see [6]) already existing code. But new ideas from students theses and recent project work are continuously being incorporated.
Using this organizational model, we can maintain a close relationship between the academic and the business side.

2 ARCHITECTURAL PRINCIPLES FOR APPLICATION FRAMEWORKS

First of all, we need criteria for framework use. These will help us identify characteristics of a framework's architecture and guide us through the framework development process. Drawing on our experience in building object-oriented frameworks and the lessons learned from other framework projects like Taligent (see [5]) and IBM's San Francisco, we have reached the following conclusions:

- A framework must be suitable for use by non-framework developers. Building frameworks will only be worth the effort if there is sufficient reuse of the framework in application projects. This usually involves "ordinary" non-framework developers using a framework.
- The chief quality of a framework with a view to its reuse is understandability. No application developer can or will use a framework that he or she does not understand.
- A framework must produce as little overhead as possible. The application built on top of a framework should only need to use precisely those parts that are useful for the application and no others. In addition, a framework should scale up with increasing complexity of the respective application. This means that simple applications should be built from a few framework elements, while an extensive system should be based on more elaborate framework support.
- A framework should clearly separate generic, non-project-specific parts and domain-specific functionality. This allows the separation of concerns when developing framework components. Domain-specific parts should be addable to the generic parts of a framework that provide the technical and architectural foundation.

The most important demand in application framework design is flexibility to changing environments and requirements. We have identified three major areas of change for frameworks and applications:

- Changes related to the application domain and the business of an organization
- Changes to the technical base (here we must distinguish between the underlying technological concepts and their implementation)
- General support for workplace environments with prefabricated components

The requirements for framework use and the three areas of potential change have resulted in a flexible framework architecture based on the following concepts:

- Clear separation of generic and domain-specific parts of the framework, realized by a layered multi-tier architecture
- Structural similarity between the concepts and relations in the application domain and the domain-specific layers of a framework architecture
- A "u"-shaped layering of the generic framework.
- A scalable framework consisting of a kernel and add-on extensions
- A component architecture for easy exchange of technical implementations of generic concepts

The Layering Concept

First, we separate the domain-specific parts of a framework from the generic parts using a layered architecture (see Figure 2). The underlying concept is described in [1] based on a major cooperation project.
The different layers are divided into a domain-specific nucleus and the outer generic framework layers, which are organized in a "U" shape.
The domain-specific nucleus can be structured into a hierarchy, layers representing the different workplace types supported by the framework ("application layer"), the different product families or services offered by a company ("business-section layer") and the key abstractions and concepts underlying the entire business ("subject layer"). The domain-specific nucleus represents the application-domain dimension of a framework.
The two other dimensions (technology and general workplace support) are realized in the generic layers of the "U" shape. In the JWAM framework, we have built four layers.
The "lowest" layer is a "pseudo" layer containing ubiquitous language extensions. As these extensions are jointly used by the others layers of the framework, it is more like an underlying foil. The language extensions consist of a class library providing a contract model according to Bertrand Meyer (see [10]); a small test framework serving as a facade around JUnit (by Erich

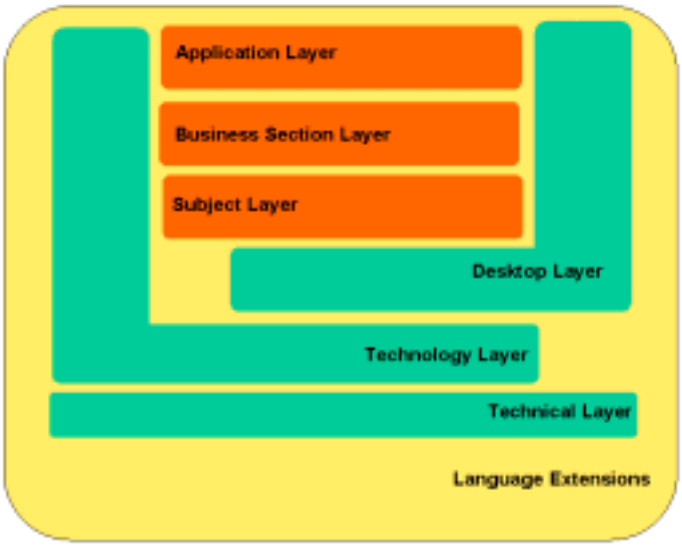

Figure 2: The special layering of frameworks

Gamma and Kent Beck, see [8]); and base classes for domain-value types (see [2]).

- The first "solid" layer is a technical layer providing the encapsulations of system-specific aspects. This layer is very thin because of the rich Java API and its "Write Once, Run Everywhere" philosophy.
- The technology layer provides base technologies such as a messaging service, a persistency service and a class library to separate presentation from interaction.
- The top generic layer is the desktop layer providing general workplace support. It contains sub-frameworks and concepts realizing equipment the user is "in touch" with. This comprises the tool and material construction as well as a desktop for visualizing a workplace or special containers like folders.

A layer inside the framework has no interface of its own. Instead, the elements of the layer (classes or components) have their interfaces. The relations between the elements of different layers usually are defined by design patterns (see [7]).

Components

A crucial requirement for framework design is scalability. We have met this demand in the JWAM framework by partitioning it into a framework kernel and optional framework extensions. This takes account of the fact that in most software projects only a certain part of the framework is used while other parts are not needed.
In partitioning a framework, we have to make sure that a minimal kernel is useful as standalone support and technically independent of the optional parts. This means that complete applications can be built using just the framework kernel and that the framework kernel can be compiled on its own. In addition, a minimal kernel is very easy to understand and supports a good learning curve for novice users. The current JWAM kernel consists of about 100 classes (including interfaces).
The framework extensions can be added as needed. To ensure this, we have designed a component architecture. The components are built on top of the kernel and can be divided into two kinds:

- Equipment components. These typically enhance the workplace environment with ready-to-use items such as a desktop, a registry (as an application- and cooperation-oriented persistency service) or a form-service component.
- Integration components: These realize specific parts of the framework or an equipment component based on a specific technical platform. The registry equipment component, for example, can use a specific integration component to access a relational database system. A second integration component will provide flat-file access instead.

These components can be used independently and as required. This provides an extremely scalable framework.
The component architecture allows us to develop, for example, additional commercial components on a highly professional basis at the company, while the "standard" components can usually be developed at the university with the limited resources available there.
However, the JWAM framework still lacks an infrastructure for software components in the sense of composition units that can be plugged in at runtime, although tools, materials (and automata) would be a natural first choice for such components.

3 PRINCIPLES OF THE FRAMEWORK DEVELOPMENT PROCESS

A further level of complexity is added to the software development process if a framework is involved. Not only does the application development process have to be organized, the framework itself must also be further developed. Both processes must be combined and synchronized to ensure the efficiency of the two strands of development. We are aware that we need to minimize the additional complexity of using a framework within the development process as a whole. This is reflected in our principles for organizing the framework development process.

Versioning a Framework

If a framework is to provide optimal support, it must evolve continuously. However, as it turns into a moving

target, it no longer provides a sound basis for application development. On the other hand, a "frozen" framework will quickly deteriorate. By way of a compromise between continuous changes and a permanently fixed framework, we support the JWAM framework in three versions.

- The current version is the production version of the framework. All ongoing application projects should be based on this version.
- Besides this version, the framework team must support the previous version of the framework for a fixed period. This gives the applications built on top of the previous version the chance to plan the migration process to the current version.
- In order to provide application developers with a preview of the upcoming changes, a preview of the next framework version is announced. This version allows the application developer to plan future application systems with respect to the then current framework base.

These three versions all last for the same period of time. Then the preview version becomes the current version and the current version becomes the previous one. A new preview version is announced and the old previous version is no longer supported. This process means that application projects can be synchronized with framework development.

As the migration of an application from one framework version to the next has to be done largely by hand, one of our current research projects focuses on tools to support this migration process. Initial prototypes are available.

The Architecture Group

A look at the development process of the framework itself highlights certain characteristics of our approach. Since JWAM development is driven by both commercial projects' needs and current research results, we must reconcile these two forces. Our steering organizational unit is the Architecture Group. Meetings are held once a week at which Story Cards are discussed and rated for priority and complexity. The idea of using Story Cards to document requests for new features or changes to the framework is borrowed from eXtreme programming. The Story Cards are written by members of project teams as well as by the framework architects. To ensure framework quality, reviews of all important parts (especially the recent ones) are conducted regularly. These lead to concrete refactoring plans.

Constructive Quality Assurance

The quality of a framework is the crucial factor in framework development. As a framework will be reused by a large number of application projects, errors and inappropriate functionality multiply. In addition, a framework provides the general architecture and design for applications. A lack of quality here, then, is detrimental to the application system's architecture as well.

To ensure top quality of the framework, we use a number of construction techniques. They allow us to build high quality into the system and help us avoid the old mistake of separating construction from quality management.

The techniques employed (many of them part of eXtreme programming, see [3]) are:

- Pair Programming (see [3]). This part of the eXtreme programming paradigm is used exclusively for the framework kernel. Other parts of the framework may be realized by a single developer, but kernel components are invariably integrated into the framework by a pair of programmers.
- Design by Contract (see [10]). The well-known design paradigm by Bertrand Meyer ensures the quality of features by means of pre- and postconditions. The complete framework uses this paradigm, supported by a simple Contract class in the language extension layer.
- Unit Testing using JUnit (see [8]). Currently almost every class within the JWAM framework kernel has its own test class. We have extended JUnit by components that allow us to test the GUI and eventing behavior of tools. In all, the JWAM framework contains about 200 test classes.
- Aggressive Refactoring (see [6]). These refactorings are done in so-called JWAM Sprints, during which we concentrate all our development power on the framework for three consecutive days.
- Team development of software is still an issue within the software engineering community. We tried Kent Beck's idea of eXtreme programming guidelines (see [3]), i. e., we used a single machine to integrate changes in order to prevent clashes and inconsistencies. But this approach was "eXtremely" slow and unattractive for the developers. Standard version-control systems proved to be not really suited to our development processes. We therefore built our own tool, the IntegrationServer. The IntegrationServer is a server-side version control system providing full support for eXtreme

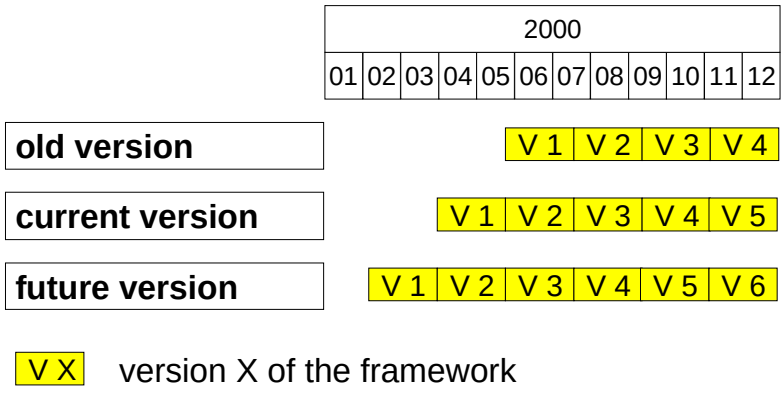

Figure 3: Framework versioning with three supported versions

programming test classes. It integrates sources only if they are consistent throughout the whole system. Consistency is checked by running all test cases automatically and providing feedback on the results. Using this tool, we were able to speed up our framework development during the Sprints to up to 10 or more integrations per day, obtaining a complete and consistent framework version every time. The IntegrationServer and the concepts behind this piece of software form part of Robert Tunkel's degree thesis. The interested readers is referred to this work (available only in German).

Tool Support For Application-Development

Ease of use is crucial for the frameworks. We have therefore considered how to support application developers in using a framework for their specific task. For most application developers, it is attractive to obtain a higher level of support than that offered by pure framework classes and interfaces within a standard programming environment. Another of our current research projects focuses on new framework reuse metaphors and how these might facilitate the use of a framework. The idea is to go one step beyond black-box or white-box to tool-box frameworks. This means enhancing a framework by a set of specific tools supporting the application developer at the design level rather than at the object-oriented-language level. Initial prototypes of these tools are available and will be presented at an OOPSLA demonstration session.

4 ACKNOWLEDGEMENTS

The authors wish to thank the members of the JWAM architecture group for their work, which formed the basis of this position paper. We also thank Robert Tunkel for his work on the IntegrationServer tool, which has been a great help in developing the framework. Last but not least, we would like to thank all users of the framework for their comments and suggestions.

REFERENCES

1. D. Bäumer, G. Gryczan, R. Knoll, C. Lilienthal, D. Riehle, H. Züllighoven: Framework Development for Large Systems, in *Communications of the ACM*, October 1997, Vol. 40, No. 10, pp. 52-59.
2. D. Bäumer, D. Riehle, W. Siberski, C. Lilienthal, D. Megert, K.-H. Sylla, H. Züllighoven: *Values in Object-Systems*, Ubilab Technical Report, Zurich, Switzerland, UBS AG, 1998.
3. K. Beck: *eXtreme Programming Explained – Embrace Change*, Addison-Wesley, Reading, Massachusetts, 1999.
4. U. Bürkle, G. Gryczan, H. Züllighoven: Object-Oriented System Development in a Banking Project: Methodology, Experiences, and Conclusions, in *Human-Computer Interactions, Special Issue: Empirical Studies of Object-Oriented Design*, Vol. 10, No. 2 & 3, 1995, Lawrence Erlbaum Associates Publishers Hillsdale, New Jersey, England, 1995, pp. 293-336.
5. S. Cotter, M. Potel: *Inside Taligent Technology*, Reading, Massachusetts, Addison-Wesley, 1995.
6. M. Fowler: *Refactoring – Improving the Design of Existing Code*, Addison-Wesley, 1999.
7. E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, Massachusetts, 1994.
8. The *JUnit test framework*: http://www.junit.org.
9. The *JWAM framework*: http://www.jwam.org.
10. B. Meyer: *Object-oriented Software Construction*, Second Edition, Prentice-Hall, New York, London, 1997.