Frameworks—Trends and Perspectives

Wolfgang Pree
C. Doppler Laboratory for Software Engineering
University of Linz
Altenbergerstr. 69
A-4040 Linz, Austria
Voice: (+43) 70-2468-9432
Fax: (+43) 70-2468-9430
E-mail: pree@swe.uni-linz.ac.at
http://www.swe.uni-linz.ac.at/wolf

A recent discussion on the Internet raised an interesting question: "What was the first object-oriented framework?" Some claimed that class Simulation (Dahl et al., 1970), designed and implemented by the Norwegian Simula67 team, represents the first framework.

Suppose we consider extensibility as an important characteristic of frameworks, the discussion continued. Then we can go even farther back and view operating systems as the first frameworks since they are extended by applications. As many operating system implementations lack any object-oriented flavor, SketchPad was proposed as a candidate deserving the honor of being the first object-oriented framework. This GUI editor system was developed by Doug Engelbart at MIT in the early 60s.

Whichever candidate receives your vote as the harbinger framework, the concepts underlying this technology have been around for at least 30 years. Nowadays many in the software development community hail frameworks as the one and only means to overcome the enormous difficulties encountered in the production of software. In this context we pose the question of what we can expect from framework technology in the next decade or so?

Before we gaze into the future in our crystal ball, let us briefly remain in the past and present in order to better understand the future. Thus far no single panancea has been found to cope with the known deficiencies of software development. Just remember some buzzwords, for example, computer aided software engineering (CASE), prototyping, automated programming and object-orientation. As Pree and Pomberger (1995) pointed out in a virtual roundtable on the future of software, "Those who remain in their lethargy and wait for a silver bullet to overcome the software crisis will be disappointed. No single concept, method or tool will result in a breakthrough." So the first message is that we should not view framework technology as the next panancea.

An issue concerning the present state-of-the-art in framework technology is the question of why it took decades to move frameworks into the mainstream, or at least into becoming a hot topic.

One reason for this might be that in general many companies are stuck with legacy software and often are caught in the trap of believing that they cannot depart from the current trail. This still hinders more widespread dissemination of framework technology. Currently mostly early adopters of the technology are experimenting with frameworks in areas other than GUIs.

Another reason for the slow advance of frameworks is that framework development means an investment, as the development costs are significantly higher than producing one specific application. While these costs pay off in the long run, the application of framework technology does not imply short-term profits. Today numerous projects first apply object-oriented technology in a naive way and fail to meet expectations. For example, just using an object-oriented language definitely does not suffice to exploit object technology, for example, in order to significantly increase software reusability. Nevertheless, many involved in software development still harbor such a naive view.

As a consequence, it is not difficult to predict from the current situtation that many future projects based on object technology will try to move towards the framework camp. But there are other major hurdles to overcome. The current state-of-the-art in framework technology is not much different from that of 30 years ago. And those who have worked with frameworks, no matter whether just adapting them or developing such systems, know that this technology is far from maturity. The following are some of the most striking problems:

- Most available frameworks can be extended or modified only if additional components are developed in the same language (in most cases even by using the same compiler version or development system) as the overall framework.
- It remains unclear how frameworks designed by different teams, probably in different languages, can interoperate.
- The fragile base class problem might overthrow fundamental framework design decisions. Changes in base classes of a framework can fracture numerous classes inheriting from them.
- Except for rudimentary tools such as cookbooks, no tools directly support framework specialization and development.

Component standards, for example, OpenDoc/(D)SOM and OLE/COM, try to attack the first problem. In addition to the fact that these standards have an unnecessary inherent complexity and fail to offer proper garbage collection, which is simply a must for extensible systems, they provide no solution to the

The fragile base class problem can be alleviated by designing frameworks in a way that allows most adaptations by composition instead of inheritance. We discuss this aspect in more detail below.

The above leads us to conclude that framework technolgy still has a long evolutionary way to go before approaching perfection. So be prepared for mainly confusing accompaniment. Lewis et al. (1996) point out one recent example: “Patterns ... is one of the most recent fads to hit the framework camp. ... Expect more buzzwords to appear on the horizon.”

Despite these problems with the state-of-the-art in framework technology, currently available frameworks already represent a significant leap forward over conventional ways of constructing software.

Well-designed frameworks predefine most of the overall architecture, that is, the composition and interaction of its components. (Well-designed means that a framework offers the domain-specific flexibility for adaptations.) So applications built on top of a framework reuse not only source code but also architecture design—which we consider as one of the most important characteristics of frameworks.

Many existing frameworks give an impressive example of the degree of reusability that can be achieved if these systems are well-designed. For example, GUI frameworks with excellent design deliver a reduction in source code size (that is, the source code that has to be written by the programmer who adapts the framework) of *80% percent or more* compared to software written with the support of a conventional graphic toolbox.

The more advanced framework designs available today already point out possible directions in which future frameworks will go. These *pioneering frameworks* rely mainly on *object composition*. In such frameworks most of the adaptations are done by just plugging together objects instead of modifying behavior by means of inheritance. To those who have already experienced the ease and power of such adaptations, it is obvious that future frameworks will rely mainly on composition. We cannot deny the inevitability of this transition. Several implications result from this trend:

- Componentware will indeed mean distributing software components that can be plugged into software systems. The underlying technology will be frameworks whose behavior is modified and/or extended by composition.

  Of course, the world-wide network infrastructure will strongly boost such a component market.

  Note that in the long term this trend remains quite independent of the answer to the question of which of the current or future defacto standards for integrating (distributed) components will dominate.

- Tools will become available that allow end users to configure software systems by handling such framework components.
- Currently software components are quite monolithic. In many cases components represent full-fledged applications. Expect a much finer level of granularity of software components.

Another perspective of frameworks relates to the discussion above and takes into consideration future development stages of the WorldWideWeb (WWW). Steinberg (1996) envisions three further major stages of the Web beyond the current situation where almost all documents are simply displayed:

- First, clients and servers will become smarter. So end users will easily be able to access information systems. The fun-to-use package tracking system of FedEx gives a taste of what will happen at this stage.
- Second, not just data but programs will be exchanged between servers and clients. Steinberg (1996) sketches an example: "A stockbroker's Web site might send out an applet [little application] that acts as front end for displaying a ticker tape at the top of your screen."
- In the third stage, applets will turn into intelligent agents that are sent out like servants and gather information.

In all these stages framework technology will play a key role. In order to make clients and servers smarter, a huge amount of software has to be written. Only appropriate frameworks that significantly reduce the amount of code that has to be written will allow software developers to cope with these demands. The first frameworks for this purpose are NeXT's WebObjects for the server side and Apple's Cyberdog for the client side.

Applets are examples of software components that configure themselves automatically on the end user's client. Again, frameworks will be the core technology underlying such applets. As Sun's Java language, Microsoft's Internet Studio (originally code-named Blackbird) and General Magic's Telescript are specifically designed for building applets, various frameworks will be based on them in the future.

Intelligent agents are quite similar to applets, but more active. Today most of the agents are built from scratch in research labs. Thus when agents become the vogue on the Web, we can expect appropriate agent frameworks.

Overall, we do not view frameworks as yet another silver bullet that will again be replaced by alternatives soon after it became a hot topic. The hype associated with frameworks will quickly calm down, allowing a more realistic view of this technology. Frameworks will remain the long-term players towards reaching the goal of developing software with a building-block approach. Though the state of the art still needs profound refinement, many enabling technology in all areas of software development. So let us continue to make the transition from *manu*factured[1] artifacts towards software produced from adaptable framework components.

References

Dahl O-J, Myhrhaug B, Nygaard K (1970) *Common Base Language*, Norwegian Computing Center, Publ. S-22 (a revision of S-2 of 1968)

Lewis T, Rosenstein L, Pree W, Weinand A, Gamma E, Calder P, Andert G, Vlissides J, Schmucker K (1996) *Object-Oriented Application Frameworks*, Manning Publications/Prentice Hall

Pree W, Pomberger G (1995) *The Past as Prologue*, in Where is Software Headed—A Virtual Roundtable (Ed. Lewis T), IEEE Computer, 28(8)

Steinberg S (1996) *Get Ready for Web Objects*, Wired, 4.02