# Framelets—small is beautiful

**Wolfgang Pree**
Software Engineering Group
University of Constance
D-78457 Constance, Germany
pree@acm.org

**Kai Koskimies**
Nokia Research Center
Box 45
FIN-00211 Helsinki, Finland
kai.koskimies@research.nokia.com

**Abstract.** The contribution introduces the notion of framelets. They are essentially small frameworks. Thus framelets can be easily understood and modified. As framelets are not aimed at complex application domains, they can also be assembled without the problems associated with the combination of large frameworks. Our first experiences with framelets demonstrate that these architectural building blocks allow the integration of framework technology into legacy applications. Furthermore, framelets might form a suitable means for structuring software architectures.

## 1 Frameworks and legacy applications

Conventional wisdom implies that legacy applications and framework technology do not match well. Framework experts usually advise the development of a domain-specific architecture for the domain at hand. The legacy application would have to be replaced by the adapted framework. Most companies are not willing to pursue such a radical approach, which renders the investments into legacy applications worthless. Furthermore, the development of a complex framework from scratch represents a formidable risk. The brilliant architects who would be able to design domain-specific frameworks are often not available.

This situation forms the starting point of our considerations, i.e., how framework technology can be harnessed in the realm of legacy applications. The legacy system at hand is a client-/server application of a bank. Our approach is straight-forward: Taking a look at the overall conventional system architecture and then at the underlying source code reveals that some quite small aspects of the system are implemented numerous times again and again in a similar way.

For example, the legacy application relies on remote procedure calls (RPCs) implemented in C. The code associated with a particular RPC implies tedious programming work, in particular handling the parameter value transfer for each RPC. For example, the return parameter types are C-style arrays, which have to be properly processed. The RPCs are too diverse to come up with a simple reusable procedure/function. Instead we found that a small framework is the solution in order to automate the calling of remote procedures. The hot spot of the RPC framelet is the processing of parameters.

Another example of reimplemented code can be found at the client side of the legacy system. Most dialogs provide one or more list boxes (a.k.a. grid controls) together with buttons to add items to the list box, and to modify and remove them. Thus these GUI elements and their interactions have to be implemented again and again. The associated programming task is

another typical example of a piece of programming work that can easily be packaged into a small self-contained framework. The hot spot of the so-called list box framelet is the dialog, which displays an item. Figure 1 shows a sample specialization of the list box framework. The dialog on the left-hand side consists only of the framework's GUI and the button labeled Close. The arrows in Figure 1 indicate the interactions between the framework components.

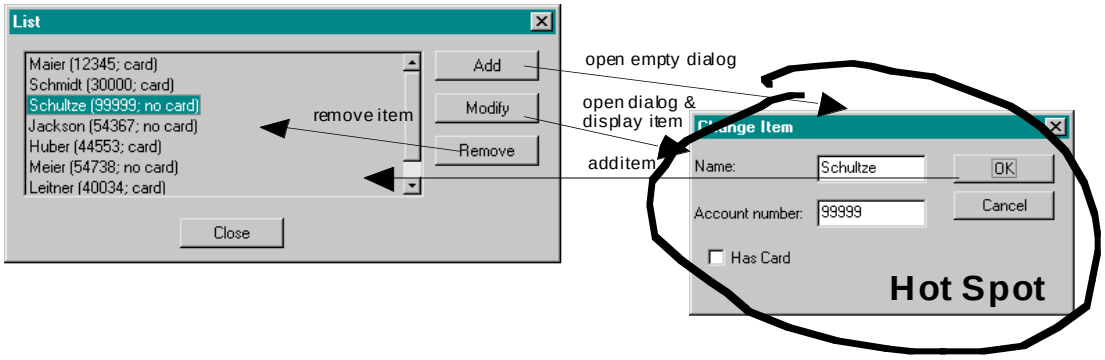


**Figure 1** A sample specialization of the list box framework.

Besides the RPC framework and the list box framework, we found a few other aspects that could be packaged as small frameworks. All these frameworks are implemented in Java and deployed as Java Beans. As the legacy application is steadily extended (new dialogs) and changed, application developers already reuse these assets. Other parts of the legacy application are simplified by replacing source code fragments with calls to these frameworks. This reduces significantly the source code size and thus the maintenance costs in the long term. The next section will introduce the concept of a small framework, *framelet*, and discuss its relationships with other architectural concepts.

## 2 Framelets as architectural units

The conventional idea of an application framework as an application skeleton has well-known drawbacks: a framework easily becomes a large and tightly coupled collection of classes that breaks sound modularization principles and is difficult to combine with other similar frameworks. Complex, implicit specialization interfaces are hard to manage by application programmers. For a discussion of problems in application frameworks, see for example Fayad and Schmidt (1997), Sparks et al. (1996), Lewis (1995), and Casais (1995). Our thesis is that only small software units should be white boxes.

These considerations and our experiences with re-organizing legacy systems led us to the concept of a *framelet*, a small framework. In contrast to a conventional application framework, a framelet

- is small in size (< 10 classes),
- does not assume main control of an application,
- has a clearly defined simple interface.

Otherwise a framelet follows the general principles of frameworks, in particular the Hollywood principle. Essentially, a framelet is thus a component that defines two interfaces: one for calling its services and another to be implemented by the specializer. This notion is particularly suitable for re-organizing legacy code: if one removes a part of a legacy system that represents some logically related services, usually that part will then contain dangling calls to other services provided by the rest of the legacy system. The latter calls correspond to the specialization interface: An implementation of the called routines must be provided in order to make the removed part functional. If the removed part is intended to become a reusable unit, the implementation of these routines can be given in different ways by different reusers. A natural way to implement such a unit is a small framework, that is, a framelet.

The relationships between framelets, layered architectures, components and application frameworks can be explained as follows. Assume that a system has been organized as a layered architecture, and that you wish to make different layers reusable in other systems. First, consider the uppermost layer. If this layer is disconnected from the system, there will be a set of services that are called but not implemented in the layer. On the other hand, the only interface through which its own services can be accessed is the UI. This corresponds therefore to an application framework: the layer is a semi-finished application.

Consider next the lowest layer, or a part of it. When disconnected, such a unit provides an interface for the services it implements, but it needs no specialization. This corresponds to a conventional black-box component.

Finally, consider a middle layer. If the middle layer is disconnected as a separate unit, there will be both an interface defining the services provided by the unit itself, and another interface specifying the methods that have to be implemented by a lower layer to make the unit functional. Such a unit could be implemented as a single framework with large service and specialization interfaces, but this kind of unstructured framework would be very hard to understand and use. The idea of a framelet is to split this framework into smaller parts corresponding to slices of a middle layer, with independent service and specialization interfaces. Hence, framelets can be viewed as a means to bring structure to reusable software.

## References


Casais, E. (1995): An Experiment in Framework Development. Theory and Practice of Object Systems 1, 4, 269-280.

Fayad, M. and Schmidt, D (1997) Object-Oriented Application Frameworks. CACM, Vol. 40, No. 10, October 1997.

Lewis T., Rosenstein L., Pree W., Weinand A., Gamma E., Calder P., Andert G., Vlissides J., Schmucker K. (1995): *Object-Oriented Application Frameworks.* Manning Publications/Prentice Hall.

Sparks S., Benner K., Faris C. (1996): Managing Object-Oriented Framework Reuse. Computer 29,9; September 96.