ContextBox: A Visual Builder for Context Beans

[Extended Abstract] *

David H. Lorenz Predrag Petkovic†
College of Computer Science
Northeastern University
Boston, MA 02115
{lorenz,predrag}@ccs.neu.edu

ABSTRACT

We present an assembly-design environment that supports the JavaBeans extensible runtime containment and services protocol. The environment provides: a vehicle for demonstrating the Java component model; a third-party client for testing `BeanContext` and `BeanContextChild` components; and a prototype illustrating how a visual builder might unify visual and context nesting during component assembly.

1. INTRODUCTION

The essence of the extensible runtime containment and services protocol [2] is that beans may be placed in, and removed from, their enclosing `BeanContext`. The `BeanContext` becomes a container of objects, which not only introduces a new logical hierarchical structure, but also provides to its inhabitants a service discovery and obtaining protocol.

In order to test in a builder a bean implementing the `java.beans.beancontext.BeanContext` interface, the bean must have a visual representation. Without such a representation, it would be impossible to manipulate it visually during the assembly and design activities. *Assembly* is the act of connecting components into a working application visually. *Design* is the act of fine-tuning the application's look and feel by manipulating the components' visual aspects.

Moreover, assembling the context hierarchy would be difficult unless a corresponding visual containment hierarchy displays it. A `BeanContext` component, however, is not necessarily associated with an AWT component. Meanwhile, the bean can be a `BeanContextProxy` and a `java.awt.-Container`. In that case, there are two hierarchies to maintain, a visual one and a context one, and a possibility for inconsistency between the two.

†Predrag Petkovic is currently at Oracle Corporation (`Predrag.Petkovic@oracle.com`).

*A full version of this paper is available as *ContextBox: A BeanBox environment for design-time assembly of BeanContext components*, Technical Report NU-CCS-99-04, College of Computer Science, Northeastern University, Boston, MA 02115, Nov. 1999, at `www.ccs.neu.edu/home/lorenz/-center/bcdk/contextbox`.

Current builders fail to manipulate `BeanContext` components correctly. Some builders (e.g., `BeanBox`) do not even support a visual hierarchy. Other builders (e.g., IBM VisualAge) support visual nesting, but do not support the containment and services protocol. We present an enhanced `BeanBox` that supports and integrates both.

2. VISUAL DIMENSIONS OF JAVABEANS

JavaBeans are "reusable software components that can be manipulated *visually* in a builder tool" [4]. However, a bean can be visual or non-visual, may or may not be associated with a symbolic image, and at times may be visible or invisible. A *visual* bean has a visual representation during execution. All AWT components are visual beans. A *non-visual* beans is used for its functionality despite not having a visual appearance. Adapters [3] are typically non-visual.

Some components are associated with a *symbolic image*, an icon, others are assigned one by the system, e.g., a label. The icon is specified by the bean author in the `BeanInfo` adjunct class. For a bean without an icon, the system instantiates a `Label` and uses it like an icon. The icon or label is used by the builder during assembly to visually display non-visual beans. The icon or label is also used to display the list of available components (in the `ToolBox` window) when a `jar` is loaded.

A visual bean is associated with a `java.awt.Component` object, which is *visible* during design, regardless of whether or not the bean is associated with a symbolic image. A non-visual bean is represented visually during assembly by its symbolic image, and it is *invisible* during execution. But even visual beans can be at times visible and at times invisible, e.g., by invoking `setVisible(false)` during either design or execution.

In `BeanBox` 1.1 [1], the user can switch back and forth between assembly, design, and execution by toggling two environment options (see Table 1). In the next section, we describe the policy for seamlessly integrating into the `BeanBox` environment support for the runtime containment and services protocol.

Table 1: Mode-toggling

Mode	Enable design	Disable design
Show non-visual	Assembly	Read-only assembly
Hide non-visual	Design	Execution

Table 2: The visual and context combinations

component		non-visual	visual	
			leaf	composite
context	leaf	1. Not an AWT `Component`	2. A `Component`	3. A `Container`
	composite	4. A `BeanContext`	5. A `Component` and a `BeanContext`	6. A `Container` and a `BeanContext`

3. VISUAL/CONTEXT COMPONENTS

Every bean can have a `Component` or `Container`, and/or a `BeanContext`, associated with it. A bean can establish that relationship either through inheritance by extending one of `BeanContext`, `Component` or `Container` classes, or by being a `BeanContextProxy` or a `BeanContextContainerProxy` or a `BeanContextChildComponentProxy` for that object.

In terms of the COMPOSITE design pattern [3], a bean can be both a visual and a context component. A *visual component* is either a *visual leaf* or a *visual composite*. Similarly, a *context component* is either a *context leaf* or a *context composite*. From a visual perspective, however, a context leaf and a component not associated with a context are treated the same. There are therefore 6 combinations to consider [5] (Table 2):

1. **A non-visual context leaf.** The `BeanBox` covers the case of a bean which is neither a `Component` nor a `BeanContext`. During assembly, a special label represents a non-visual bean, which is hidden during design and execution. No beans can be placed inside this bean.

2. **Both a visual and a context leaf.** A bean, which is a `Component` but not a `BeanContext`, should always be represented (during assembly, design, and execution) by the `Component` itself. This behavior would be consistent with the behavior in the `BeanBox`. No beans can be placed inside this bean. It is neither a visual nor a context composite, and therefore its visual representation should be used at all times.

3. **A visual composite context leaf.** A bean, which is a `Container` but not a `BeanContext`, is a common case, typically coded by a user who did not anticipate runtime containment. This kind of visual bean should always represent itself visually. However, if the user places inside this bean other beans that expect services from a runtime environment, then those beans must be added to that container and also to the runtime context of some other bean.

   In the extended `BeanBox` version, every bean that is a `Container` but not a `BeanContext` is automatically associated with a new `BeanContext`. Then, every bean added to that container is also added to its associated `BeanContext`, which propagates the environment services according to the protocol.

4. **A non-visual context composite.** This is a kind of `BeanContext` bean that has no visual representation. It should be represented by a special kind of `Container` (e.g., `TransparentPanel`). During assembly, the user can place inside this bean other beans, and during design/execution the container should become "transparent", i.e., become itself invisible but leave the contained components visible.

5. **A visual leaf context composite.** A bean can be both a `Component` and a `BeanContext`. However, this is an unnatural case that probably ought to be disallowed. It is unnatural because the bean seems to have a contradictory behavior: a leaf (`Component`) in the visual containment hierarchy, and a composite, i.e., a collection (`BeanContext`), in the `BeanContext` containment hierarchy.

   A possible work-around is to represent such beans during assembly by a special kind of `Container` (e.g., `OurPanel`, analogous to `OurLabel`), which will allow the user to visually put in it child beans, and during design/execution by the `Component` associated with the bean. Beans placed inside this bean should be added to its associated container and also to its `BeanContext`.

6. **Both a visual and a context composite.** This is a bean that is a `Container` and a `BeanContext`. It is the simplest case. The bean should always represent itself, and there are no additional problems. Beans placed inside the component should be added to both the `Container` and to the `BeanContext`, either by the bean itself or by the environment.

4. CONCLUSION

`BeanBox` can only test beans against the `BeanContext` for which the `BeanBox` object is a proxy. As a result, one can test one's services beans but one cannot test one's own `BeanContext` beans.

ContextBox is an enhancement of the `BeanBox`, supporting both visual and context nesting. We demonstrate the working of ContextBox through an example: a `ColorBeanContext` panel (extends `Panel` and implements `BeanContextProxy`), which rejects the insertion of `ColorBeans` of a color same as its own, but accepts those of a different color. When the colors are dynamically changed, all beans violating the color restriction are expelled from the `ColorBeanContext` panel, and added to the panel's parent, if possible.

5. REFERENCES

[1] BDK 1.1. JavaSoft, November 1997.

[2] L. Cable. Extensible runtime and services protocol for JavaBeans. Version 1.0, JavaSoft, Dec. 3 1998.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[4] JavaBeans API specification. JavaSoft. Version 1.01, Sun Microsystems, Mountain View, CA, July 24 1997.

[5] D. H. Lorenz and P. Petkovic. Design-time assembly of runtime containment components. In *Proceedings of the* 34^{th} *International Conference on Technology of Object-Oriented Languages and Systems*, pages 195–204, Santa Barbara, CA, July 30-Aug. 4 2000. TOOLS 34 USA Conference, IEEE Computer Society.