G Java & Component Models & Jini

G.1 Overview

- Software Components
- Component models
- Java & component models: JavaBeans
  - ➤ Architecture
  - ➤ Properties
  - ➤ Events
  - ➤ Introspection
- Jini
  - ➤ a distributed component model
- Programming component software, applicability of software components

G.2 References

Edw99. W. Keith Edwards. *Core Jini.* Prentice Hall, Upper Saddle River, NJ, 1999.

Edw00. W. Keith Edwards. *Core Jini.* Prentice Hall, München, 2000 (deutsche Übersetzung, umfasst bereits Jini 1.1!).

Szy98. Clemens Szyperski. *Component Software — Beyond Object-Oriented Programming.* Addison-Wesley, Harlow, England, 1998.

Grif98. Frank Griffel. *Componentware — Konzepte und Techniken eines Softwareparadigmas.* dpunkt-Verlag, Heidelberg, 1998.

JavaBeans. http://www.sun.com/beans/

Jini. http://www.sun.com/jini/specs/

G.3 Component Models

1 Software Components

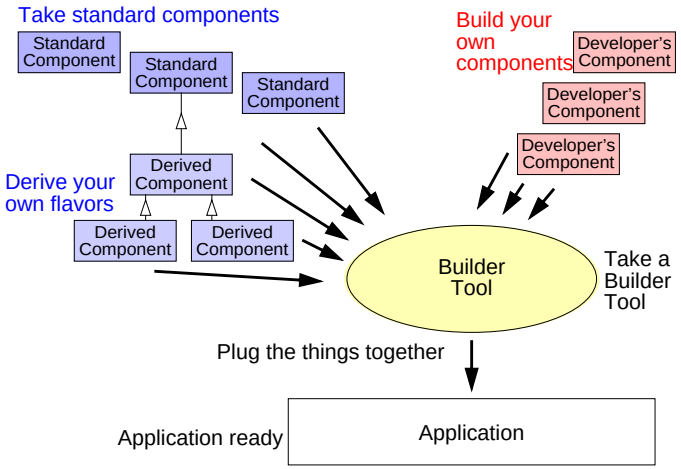
- Definition (Szyperski)
  A reusable piece of software that:
  - ➤ has a well-specified public interface
  - ➤ can be used in unpredictable combinations
  - ➤ is a stand-alone, marketable entity
- Examples
  - ➤ elements of a window system
  - ➤ software modules that represent devices (measuring dev., printer, …)
  - ➤ spelling checker module of text processing system

? just classes in the object oriented sense — or what's new here?

1 Software Components (2)

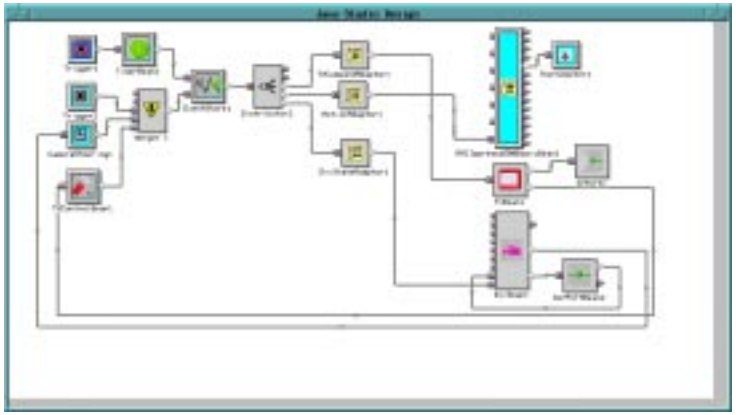
- Standard conventions for the interfaces of software components
  - ↑ **Component model**
  - ➥ Easy to use + easy to reuse
- Software components are self-describing
  - ➤ automatic analysis of interface and properties possible
    - ➤ introspection / reflection mechanism
    - + naming conventions
- Software components can be combined (visually) to complex applications
  - ➤ software kit
  - ➤ visual programming with builder tools

2 Philosophy



3 Example of a Builder Tool

- JavaStudio



4 Example of an Application

- Interface for our coffee machine

5 Component Model

- Guidelines for software components
  - ➤ naming conventions
  - ➤ repositories, introspection mechanism
  - ➤ interface semantics
  - ➤ programming method
- Rival Component Architectures
  - ◆ JavaBeans
  - ◆ CORBA Component Model
  - ◆ ActiveX
    - ➤ not portable, proprietary
  - ◆ OpenDOC
    - ➤ pretty much dead
  - ◆ Proprietary Solutions
    - ➤ GUI builder class libraries

G.4 Java & Software Components

1 Java — Goals

- Solve today's problems with development and distribution of software
  - ◆ various operating systems (Unix, Windows, MacOS, …)
  - ◆ various hardware architectures
- Java: language and environment for *secure*, *high performance*, and highly *robust* applications on *multiple platforms* in *heterogeneous, distributed networks*

2 Java — Key Properties for Components

- Object oriented
- Polymorphism based on class/interface conformance
- Highly dynamic (loading & linking)
- Reflection/introspection mechanisms

3 Java Component Models

- JavaBeans
- Jini

G.5 JavaBeans

1 Definition

- JavaBeans is an API specification for creating reusable software components using Java
  - ➤ defines the Java software components
  - ➤ and how they fit together
- A Bean is any Java class that follows the JavaBeans conventions
- The official definition:

  A Java Bean is a reusable software component that can be visually manipulated in builder tools.

2 Beans — Architecture

- Properties
  - ➤ allow customization of the Bean
- Methods
  Events
  - ➤ the wiring points that allows Beans to be interconnected
- Adapters
  - ➤ if Beans do not fit together
- Introspection
  - ➤ instead of a repository — just look into the beans

3 Example Beans

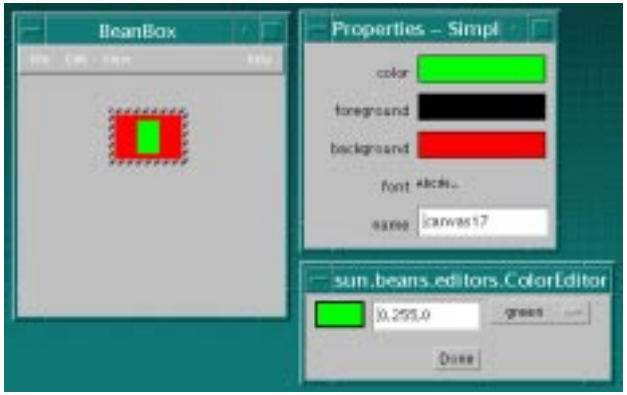
- Visual Beans
  - ➤ custom GUI components
  - ➤ HTML rendering Bean
  - ➤ OpenGL canvas
- Non-visual Beans
  - ➤ database connectivity
  - ➤ timer bean
    - – triggers events at certain time intervals
    - – may encapsulate complex date/time logic
- Software components that represent devices
  - ➤ switches
  - ➤ sensors
  - ➤ actuators
  - ➤ video recorder

4 Properties

- Describe properties of components
- Each property has
  - ➤ a **name** — symbolic description of the property (e. g. Color, Font …)
    `private Color color;` (instance variable of the Bean class)
  - ➤ a **type** — Java class, encapsulating the value(s)
  - ➤ constraints (optional) — e. g. read-only or write-only
- Naming convention for accessor methods (methods of the Bean class)
  - ➤ get method for reading
    ```
    public Color getColor(){ return color; }
    ```
  - ➤ set method for modification
    ```
    public void setColor(Color newColor){
          color = newColor;
          repaint();
    }
    ```
- Examination & modification in property dialogue

4 Properties (2)

- Example: Property Color as property of Bean "SimpleBean"
  - ➤ SimpleBean loaded in BeanBox, clicking on color opens ColorEditor



4 Properties (3)

- Simple properties
  - ➤ represent a single value, access by set/get methods
- Indexed properties
  - ➤ represent an array of values, set/get methods take an index parameter
- Bound properties
  - ➤ notify other objects when the value changes (PropertyChange event)
- Constrained properties
  - ➤ proposed changes may be rejected if invalid

5 Events

- Builds on Source-Listener Pattern
  - ➤ source object notifies listener(s) about state changes
  - EventSource
    - ➤ offers methods for listener(s) to register/unregister themselves

same name

```
public void addTimerListener(TimerListener l)
public void removeTimerListener(TimerListener l)
```

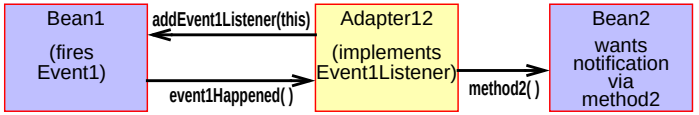
add/remove

TimerListener must implement an EventListener interface

  - EventObject
    - ➤ information about the event
  - EventListener
    - ➤ Class that implements EventListener interface (subtype of **`java.util.EventListener`**)

6 Adaptors

- Adaptation of events of one bean to methods of another bean



  - ➤ Adapter12 implements appropriate Event-Listener interface
  - ➤ Adapter12 registers for Event1
  - ➤ Event1 happens, Bean1 invokes the method, which was defined in the event interface (**`event1Happend()`**), at all registered event listeners
  - ➤ **`event1Happend()`** in Adapter12 invokes **`method2()`** at Bean2
- Adapter may manipulate event data
- Automatic generation of simple adapters possible

5 Events (2)

- PropertyChangeSupport
  - ➤ allows a Bean to send out notifications whenever a property value changes
- VetoableChangeSupport
  - ➤ allows Beans to reject property values that are out of range

7 Introspection

- Allows automatic analysis of beans
- Java1.1 Reflection API
  - ➤ analysis of Java classes at runtime
  - ➤ members: name & type
  - ➤ methods: name, parameters & return type
- JavaBeans naming conventions
  - ➤ get/set methods -> properties
  - ➤ add/remove methods -> events
  - ➤ other methods -> ordinary methods
- Alternative: Information in BeanInfo class
  - ◆ some sort of interface repository
    - ➤ developer may explicitly specify properties and events

8 JavaBeans — Summary

- Beans = pluggable software modules
  - ➤ state = properties
  - ➤ inputs = methods
  - ➤ outputs = events
  - ➤ if connectors do not fit: adaptor

➥ e. g. especially suited for a software representation of devices

- Problems / deficiencies
  - ➤ Assembling simple components to more complex components
    - ➥ hierarchical beans
  - ➤ attaching new software components to running applications
  - ➤ software components in a distributed system

2 Key concepts

- ➤ Services
- ➤ Leasing
- ➤ Events
- ➤ Security
- ➤ Lookup Service (the naming service)
- ➤ Java RMI
- ➤ Transactions

G.6 Jini

1 Overview

- Standard for the communication between "intelligent" devices (Sun 1998)
- Component architecture for distributed systems
  - ➤ components (Services) register with a lookup service
  - ➤ clients may obtain references from the lookup service (RMI stub or *Smart Proxy*)
  - ➤ references time out automatically (*Leasing*)
- Goal: Devices can flexibly log on and off from a network and find thei communication partners automatically
  - ➤ components configure their links autonomously
  - ➤ no configuration by a builder tool

3 Jini Services

- Members of a Jini community (*djinn*) federate to share access to services
- Service = an enitity that can be used by a *person*, a *program* or *another service*
  - ➤ computation
  - ➤ storage
  - ➤ communication channel
  - ➤ software filter
  - ➤ hardware device
  - ➤ another user
- Services are registered with the Lookup Service
  - ➤ clients may obtain reference (stub of of an RMI remote ref. or *smart proxy*)
- Services are composed for performance of a particular task

4 Leasing

- Partial failure in distributed systems causes special problems
  - ➤ references become invalid
  - ➤ resources cannot be freed
- Services in Jini are *leased* based on time
  - ➤ Lease = grant of guaranteed access to a service over a time period
  - ➤ may be renewed if needed longer
  - ➤ renewal can be denied by the service provider
  - ➤ if client does not renew (or cannot in case of failures) lease times out
  - ➤ after timeout resources can be freed

G.7 Real Applications for Component Software?

- Applications that need komplex configuration between implementation and employment
  - ➤ input & query software
    (e. g. SAP systems)
  - ➤ control systems
    flexible test systems
    - ➥ assembly from software components that represent the devices
      (e. g. building control systems)

+ less knowledge needed for configuration

+ application architecture more flexible and open
  - ➥ customer-specific enhancements easier to realize

→ problems with highly dynamic systems
(many new objects are created and deleted at run time)

5 Events

- JavaBeans supports only local events
  - ➤ object firing the event and all receiving objects must be ion the same VM
- Jini supports distributed events
  - ➤ registration by RMI call and passing a stub for the listener
- Jini events are only delivered as long as the lease is valid

6 Security

- Security model based on *principal information* and *access control lists*

G.8 Software Components for Real Applications?

- Class libraries with components for graphical user interfaces
- Devices that "come with" their software representation
- Componets in the office automation domain
  (text processing, database access, calculation, …)

  …

? **What's the problem?**
  - ➤ no standard component model
  - ➤ component models have too strong restrictions
  - ➤ builder tools are not powerful enough + are not adaptable

? **Solution**
  - ➤ open component models
  - ➤ extensible builder tools that provide interoperability