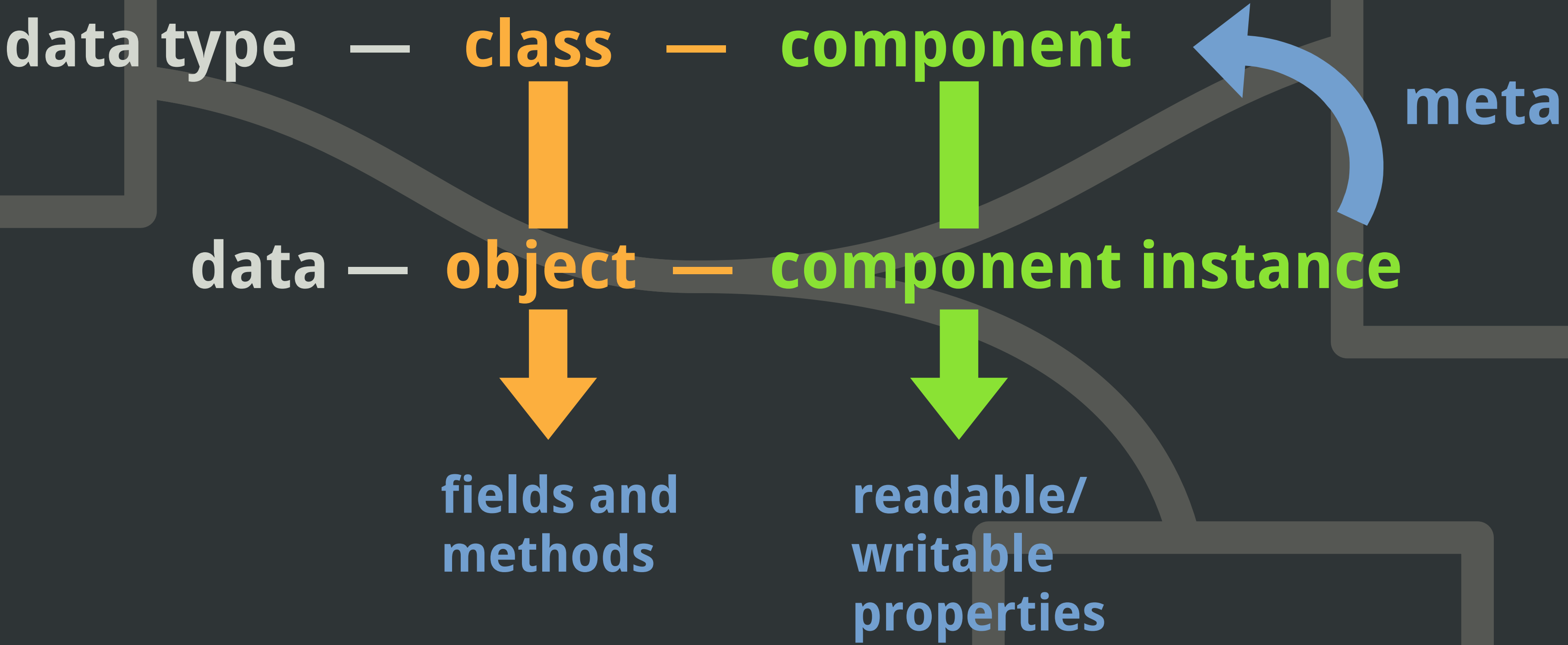
Component Architecture with Run-Time Type Definition

Bringing the power of object-orinented and component-based paradigms together

Amir Shakurov
amir-shak@yandex.ru
Higher School of Economics, Russia

Terminology


data type — class — component
meta
data — object — component instance
fields and methods
readable/ writable properties

flexible… but not enough

- Object-based programming languages

specific software applications development

- ComponentJ
- COM, COM+, DCOM
- VRML & X3D
- .Net components
- OmNet++
- The Fractal component model
- Ptolemy II
- JavaBeans

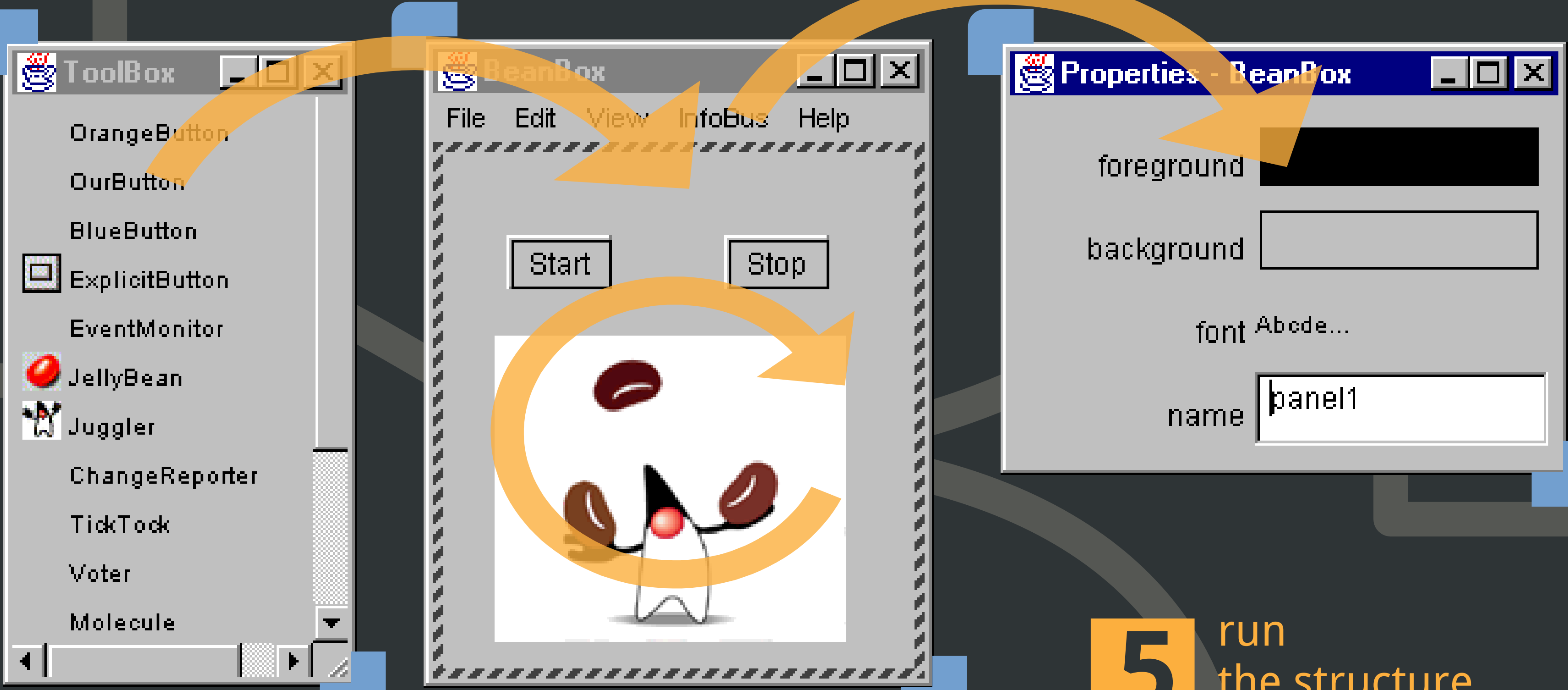
dynamical system reconfiguration

simplifying development of certain kinds of software

Why RTTD? the BDK BeanBox example

1 instantiate component instance from predefined set of components

2 adjust the instance to the context of usage

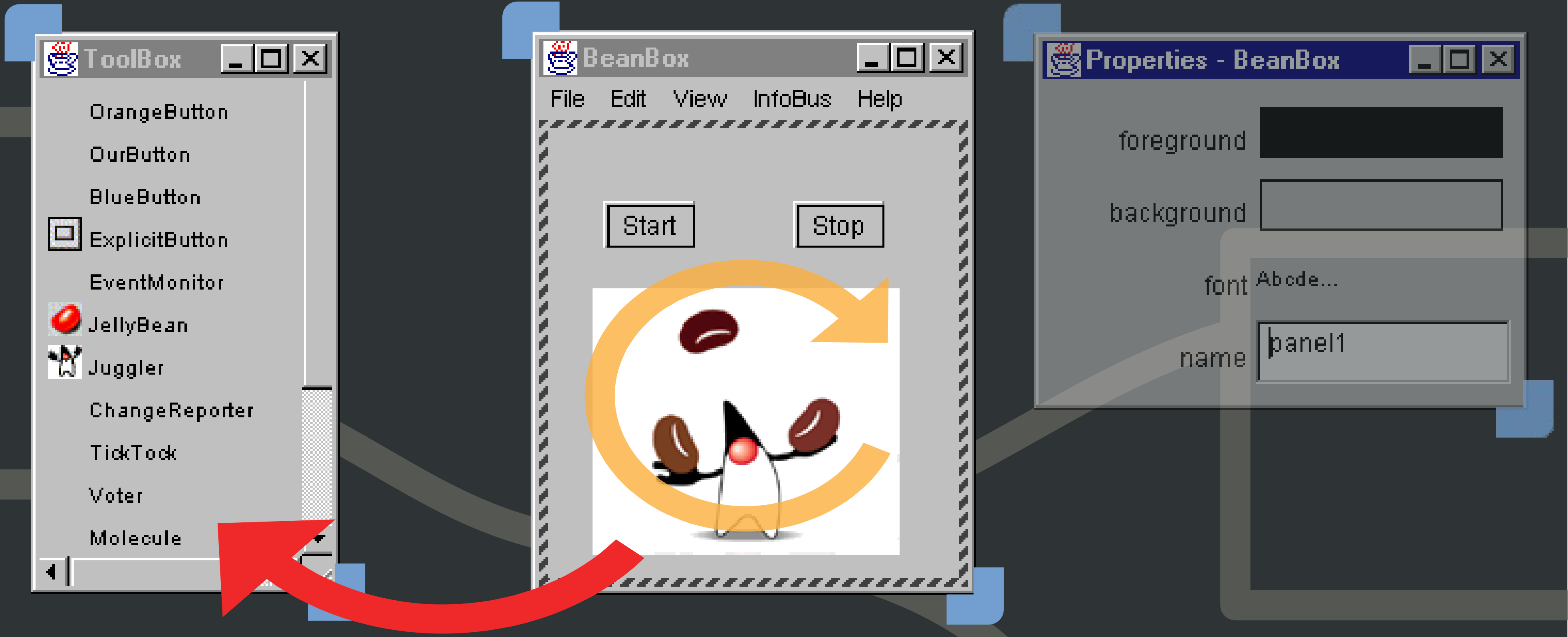


3 repeat for other components

4 arrange components into the desired structure

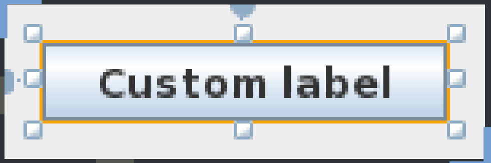
5 run the structure

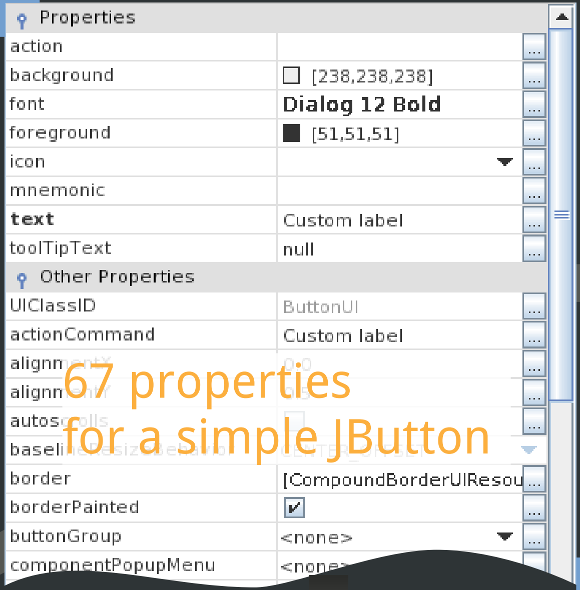
Why RTTD? the BDK BeanBox example



But what if you'd like to add
the resulting structure to the set of components?

Why RTTD? the PushButton bean example





Properties	
action	
background	☐ [238,238,238]
font	**Dialog 12 Bold**
foreground	■ [51,51,51]
icon	
mnemonic	
text	Custom label
toolTipText	null
Other Properties	
UIClassID	ButtonUI
actionCommand	Custom label
alignmentX	0.0
alignmentY	0.5
autoscrolls	☐
baselineResizeBehavior	CENTER_OFFSET
border	[CompoundBorderUIResou
borderPainted	✓
buttonGroup	<none>
componentPopupMenu	<none>

67 properties for a simple JButton

1 set the desired value to the property

2 run the program

```
String text =
"Custom label";
```

```
final String text =
"Custom label";
```

But how should one inform the system that the value of the property will never be changed during run-time ?

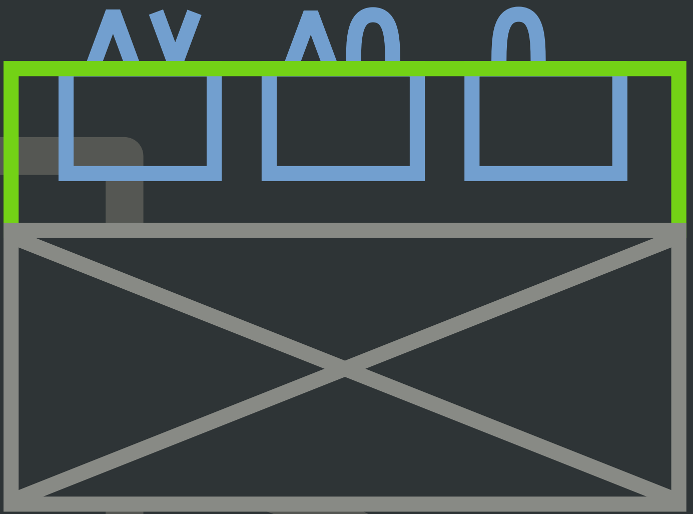
Our goal is...

...to introduce a

- simple to use,
- efficient,
- flexible (RTTD without runtime compilter calls etc)

component architecture

Component instance



interface

is a set of properties

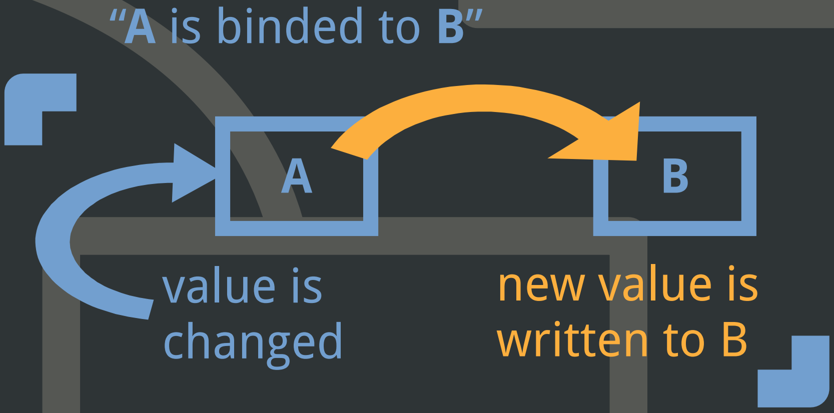
property = name + current value + operations:

∧ reading,
∨ writing
∩ and binding

implementation

is different for

- **primitive,**
- **compiled**
- **and composed components**



Primitive, compiled and composed instances from the implementational point of view

primitive instances

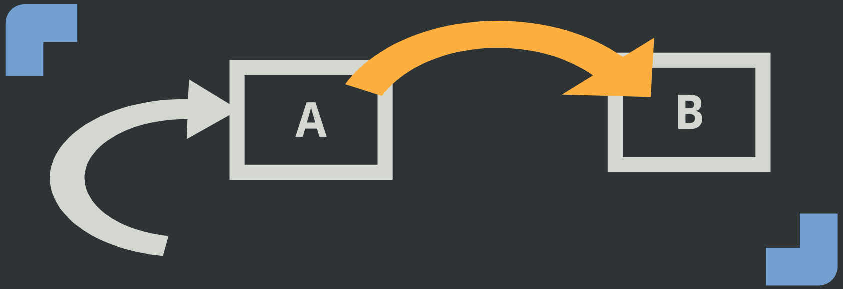
"value objects" ▪ indivisible ▪ have no default value, no properties ▪ unique

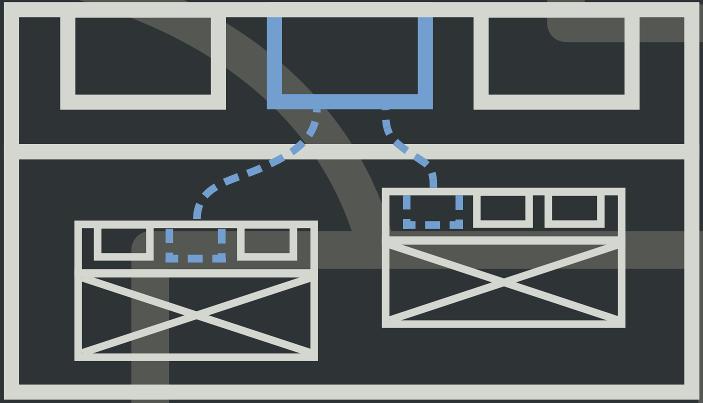
compiled instances

implemented by off-site means ▪ have default value, properties ▪ support 3rd party technologies

composed instances

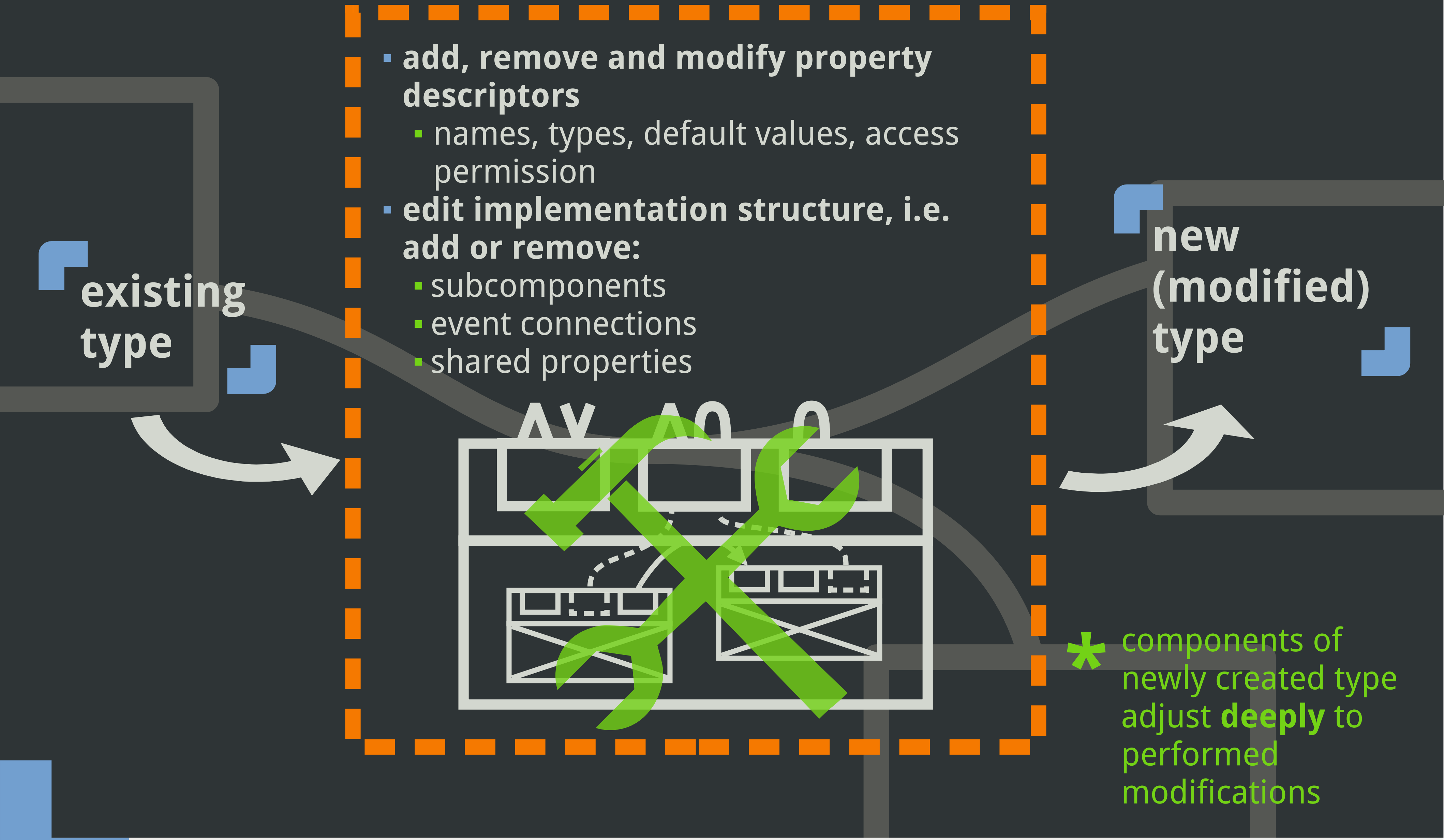
set of other components interconnected by event connections and shared properties





Container runtime environment and more

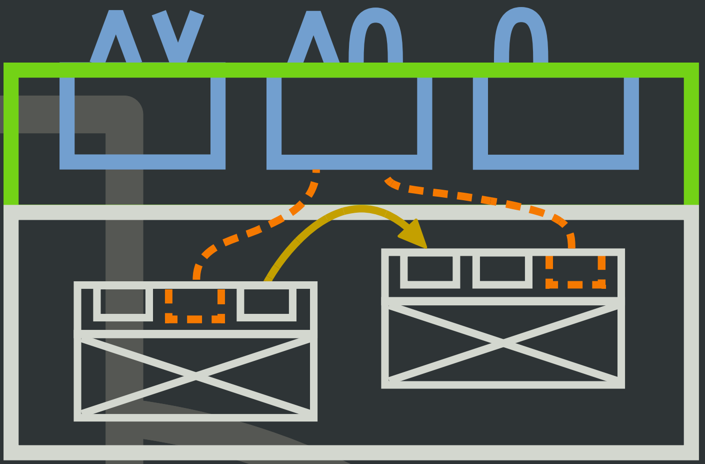
- **add, remove and modify property descriptors**
  - names, types, default values, access permission
- **edit implementation structure, i.e. add or remove:**
  - subcomponents
  - event connections
  - shared properties



* components of newly created type adjust **deeply** to performed modifications

Under the hood

composite components and their instantiation



interface metainfo

is a set of **property descriptors**

property = **name** + **value type** + **default value** + **permissions** to apply

∧ reading,
∨ writing,
∩ and binding operations

implementation metainfo

is different for components:

- **primitive**
  storage to hold current value
- **compiled**
  instructions to obtain the implementation of the component and connect it to the interface
- **composed**
  - subcomponent descriptors = type + initial value
  - property sharings
  - event connections

composed component's instance construction process:

1. initialize property references (fields) to point to:
   - properies of superinstance
   - newly constructed instances
2. create subcomponents and pass them references to shared properties
3. establish event connections

Under the hood
deriving component from its prototype

- Rely on the runtime structure (RS) as far as possible
- Store only those additional data that cannot be derived from the RS
- Emulate desired behaviour when it's not achievable without recreating the whole RS

Future plans

- UI
  - Script-like (in addition to XML)
  - GUI (with multiple output types)
- Thread safety

? Inheritance

- Real-life applications
  - firmware for microelectromechanical sensors
  - 3D visualization
  - development tools for GUI applications
  - ...any ideas are welcome!

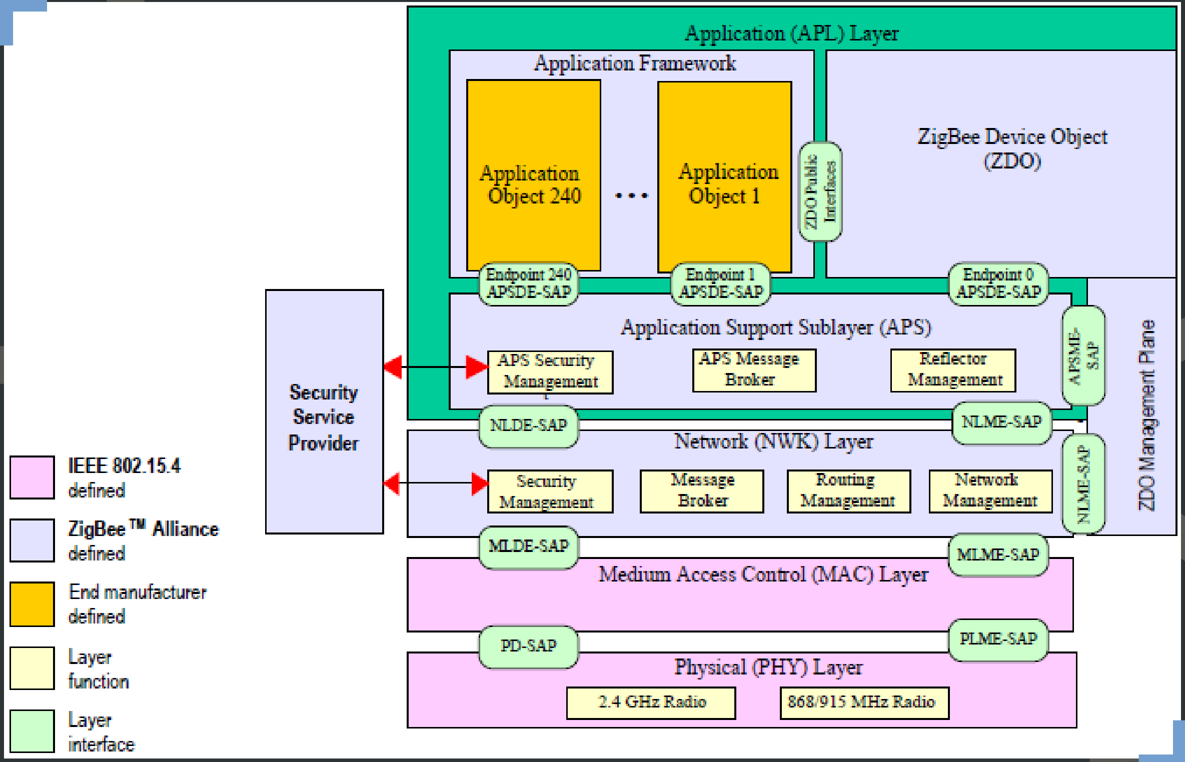
Thank you for your attention!

Amir Shakurov
amir-shak@yandex.ru
Higher School of Economics, Russia

- frequent changes in specs
- lack of development tools
- remote-only access

expensive firmware

CBSE & dynamic reconfiguration!



ZigBee network protocol stack structure

Adopted principles

✓ Run-Time Type Definition (RTTD)

Structuring code & data

✗ Flat conglomeration of components (JavaBeans™ style)

✓ Hierarchical grouping of components (object-based programming languages style)

Organizing control flow

✗ Methods (programming languages style)

✓ Readable, writable, bindable properties (component models style)

context adjustments!

Usage example

```
>list types
PropertyDescriptor, ImageViewerBean,
Str, Int, Bol
>print ImageViewerBean
Type 'ImageViewerBean'.
Property list:
     UIClassID : Str | fileName : Str |
name : Str | text : Str |
toolTipText : Str
Subcomponent list:
>ImageViewerBean iwb = new
>list vars
Iwb
>iwb.fileName =
"/some/path/to/some/file"
>print iwb
iwb : ImageViewerBean = Composite;
properties=( text=;  name=;
fileName=/some/path/to/some/file;
toolTipText=;  UIClassID= );
subcomponents=()
>~ImageViewerBean IwbEditor
>list type editors
IwbEditor
>IwbEditor >> text
>IwbEditor >> name
>IwbEditor >> fileName
>IwbEditor >> UIClassID
>IwbEditor >> toolTipText
>IwbEditor << txt : Str
>IwbEditor << num : Int
>IwbEditor -> NewType
```

```
>list types
NewType, PropertyDescriptor, ImageViewerBean, Str, Int,
Bol
>~ImageViewerBean editor2
>editor2 << age : Int
>editor2 <<< NewType = txt fileName
>editor2 <<< NewType = num age
>editor2 -> NewTypeWithSharedProperties
>print NewTypeWithSharedProperties
Type 'NewTypeWithSharedProperties'.
Property list:
     UIClassID : Str | fileName : Str | name : Str | text
: Str | toolTipText : Str | age : Int
Subcomponent list:
     0. NewType; 1. NewType;
>NewTypeWithSharedProperties abc = new
>print abc
abc : NewTypeWithSharedProperties = Composite;
properties=( text=;  age=0;  name=;  fileName=;
toolTipText=;  UIClassID= ); subcomponents=(0. :NewType =
Composite; properties=( num=0;  txt= );
subcomponents=()1. :NewType = Composite;
properties=( num=0;  txt= ); subcomponents=())
>abc.fileName = "some text"
>abc.age = 42
>print abc
abc : NewTypeWithSharedProperties = Composite;
properties=( text=;  age=42;  name=;  fileName=some text;
 toolTipText=;  UIClassID= ); subcomponents=(0. :NewType
= Composite; properties=( num=0;  txt=some text );
subcomponents=()1. :NewType = Composite;
properties=( num=42;  txt= ); subcomponents=())
>exit
```

Concerning VRML

```
#VRML V2.0 utf8
PROTO P1 [ exposedField SFColor myColor 0 0 0 ]
{
    DEF DL1 DirectionalLight {
        direction .642 -.514 -.569
    }
    DEF VP1 Viewpoint {
        description "Test viewpoint"
        isBound TRUE
    }
    DEF SH1 Shape {
        appearance DEF AP1 Appearance {
            material DEF MT1 Material {
                diffuseColor IS myColor
            }
        }
        geometry DEF IFS1 IndexedFaceSet {
            coord DEF CO1 Coordinate {
                point
                [
                    3.0 -1.0  1.0
                    4.0 -1.0 -1.0
                    3.0  1.0  0.0
                ]
            }
            coordIndex
            [
                0 1 2 -1
            ]
        }
    }
}

DEF MyProtoInstance P1{ myColor 1 0 0}
```