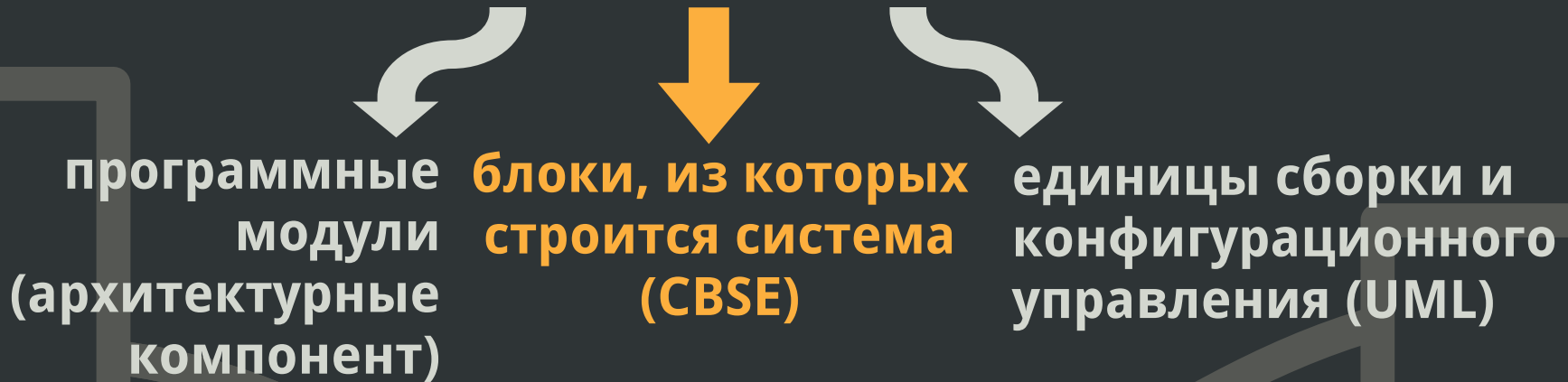


Компонентное программирование и определение типов данных во время исполнения

Амир Шакуров amir-shak@yandex.ru
НИУ ВШЭ, отделение программной инженерии

Терминология

КОМПОНЕНТЫ



Объектные и компонентные технологии

- характеризуются

- гранулярностью

состоят из большого количества сравнительно небольших (по отношению к размеру системы) составных частей

- отличаются

- простотой сочленения экземпляров в готовую систему

- возможностью создания приложений без написания кода

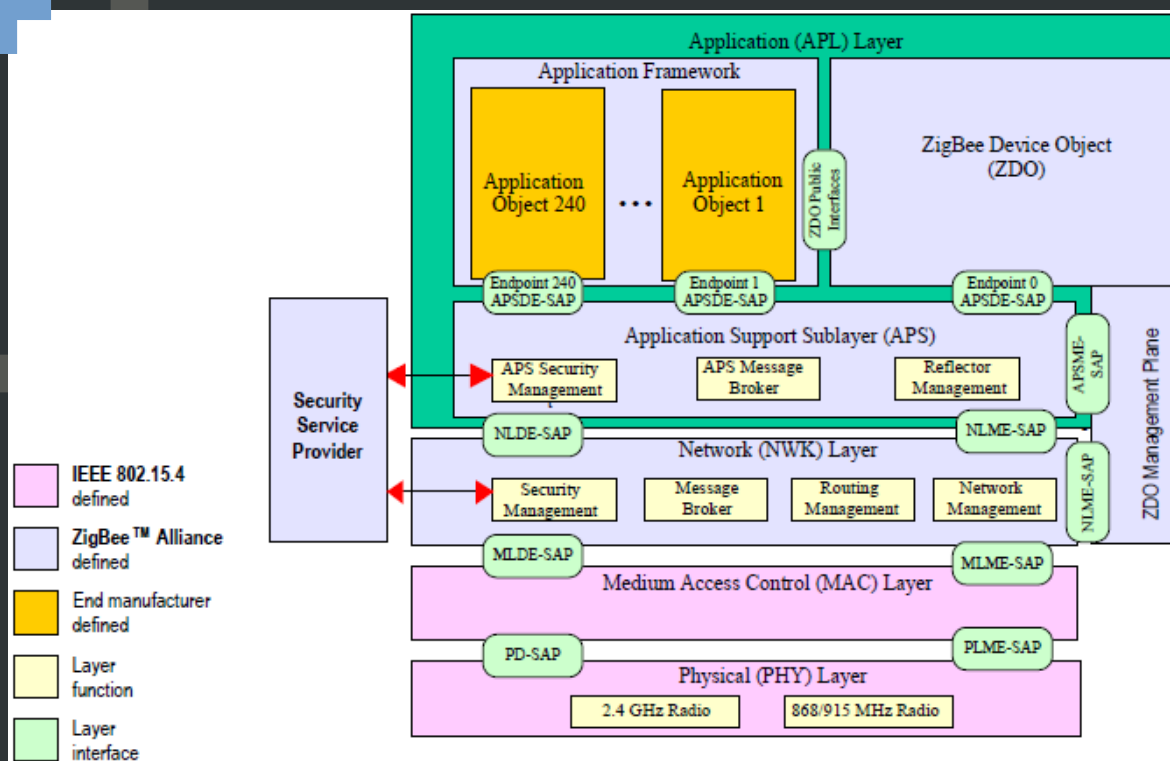
следствие — большая динамичность; возможность инстанцирования без контекста — признак компонента

- большей независимостью «гранул»

- более строгим (ограниченным в количестве используемых концепций) понятием интерфейса

Независимость ведет к гибкости

- частые изменения спецификаций
- отсутствие развитых инструментов разработки
- исключительно удаленный доступ



дорогостоящее ПО

CBSE и динамическая реконфигурация!

структура стека сетевых протоколов ZigBee

Совместить преимущества...

- простота использования
 - гранулярность (separation of concerns)
 - простота сочленения
- универсальность
- эффективность
- строгая формализованность взаимодействия

Компонент JavaBeans



- «повторно используемый элемент программного обеспечения, которым можно управлять с помощью графических инструментов, входящих в состав интегрированных сред разработки»
- java-класс, следующий соглашениям об именовании
- характеризуется:
 - событиями, свойствами и методами,
 - поддержкой интроспекции,
 - поддержка сохраняемости (persistence)

Properties	
action	
background	<input type="checkbox"/> [238,238,238]
font	Dialog 12 Bold
foreground	<input checked="" type="checkbox"/> [51,51,51]
icon	
mnemonic	
text	Custom label
toolTipText	null
Other Properties	
UIClassID	ButtonUI
actionCommand	Custom label
alignmentX	0.0
alignmentY	0.5
autoscrolls	<input type="checkbox"/>
baselineResizeBehavior	CENTER_OFFSET
border	[CompoundBorderUIResou...
borderPainted	<input checked="" type="checkbox"/>
buttonGroup	<none>
componentPopupMenu	<none>
contentAreaFilled	<input checked="" type="checkbox"/>
CURSOR	Default_Cursor

Fractal

- Компонентная модель общего назначения, не привязанная к конкретной технологии
- Компоненты — сущности времени исполнения
- API, который могут реализовывать компоненты, разделен на уровни контроля
 - 1) объект
 - 2) 1 + внешняя интроспекция (определение границы компонента)
 - 3) «уровень конфигурации» (обнаружение и изменение содержимого компонента)
 - 4) инстанцирование компонентов и типизация

<http://fractal.ow2.org/>

Fractal — типизация компонентов

- Тип компонента — совокупность типов интерфейсов, реализуемых экземплярами данного типа
- Тип интерфейса — это:
 - имя,
 - сигнатура (имя интерфейса ЯП),
 - флаг «клиентский/серверный»,
 - флаг гарантированности (contingency) — обязательный/необязательный
 - флаг мощности множества компонентов, реализующих интерфейс (singleton/collection)

VRML и X3D

- формат представления трёхмерной интерактивной векторной графики для использования во всемирной сети
- структура файла:
 - заголовок
 - граф сцены
 - прототипы
 - маршрутизация событий

DAG узлов, имеющих имя и тип

тип (встроенный или пользовательский) =
имя + поля + вх./исх. события + реализация

<http://www.w3.org/MarkUp/VRML/>
<http://www.web3d.org/x3d/>

VRML: пример кода

```
#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 C01 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
      ]
    }
  }
}
```

объявление
(интерфейс)

определение
(реализация)

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

Ptolemy II и MoML

- Основной принцип — разделение структуры и семантики моделируемой системы
- Структура — совокупность акторов
 - Интерфейс актора = порты + параметры
 - Взаимодействие: порт — канал — порт
 - Внешние параметры и порты => иерархическая структуризация
- Семантика — «режиссер»
 - сущность, задающая модель вычислений, управляющая исполнением модели

актор — экземпляр, который:
1) можно запустить на выполнение,
2) может взаимодействовать с другими акторами

<http://ptolemy.eecs.berkeley.edu/>

ComponentJ и ComponentGlue

- компонентный ЯП с фокусом на строгий контроль типов и полноценное динамическое реконфигурирование
- основная идея: сделать зависимости компонента явными, вынеся их в отдельный блок
- объекты + компоненты + конфигураторы
 - имеют порты и методы
 - конфигуратор — сущность, позволяющая модифицировать компоненты: добавлять/удалять компоненты, предоставляемые и требуемые порты

<http://c2.com/cgi/wiki?ComponentGlue>

ComponentJ и ComponentGlue

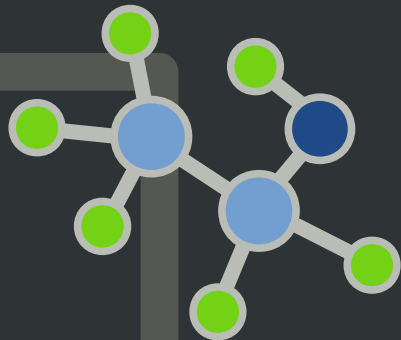
```
component Counter {  
  provides ICounter p;  
  methods m {  
    int s = 0;  
    int tick(int n) {  
      s = s + n;  
      return s;  
    }  
  }  
  plug m into p;  
}
```

```
ZeroCounter = compose (  
  provides ICounter p;  
  uses c = Counter;  
  methods x {  
    int tick (int y) {  
      if (y==0) return c.p.tick(1);  
      else return 0;  
    };  
  }  
  plug x into p ) ;
```

<http://c2.com/cgi/wiki?ComponentGlue>

Принятые руководящие принципы

Структуризация кода и данных

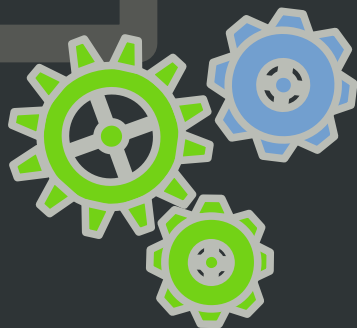


Плоское объединение экземпляров компонентов (в стиле JavaBeans)



Иерархическая группировка экземпляров компонентов (в стиле объектных языков программирования)

Организация потока управления



Методы (в стиле объектных языков программирования)



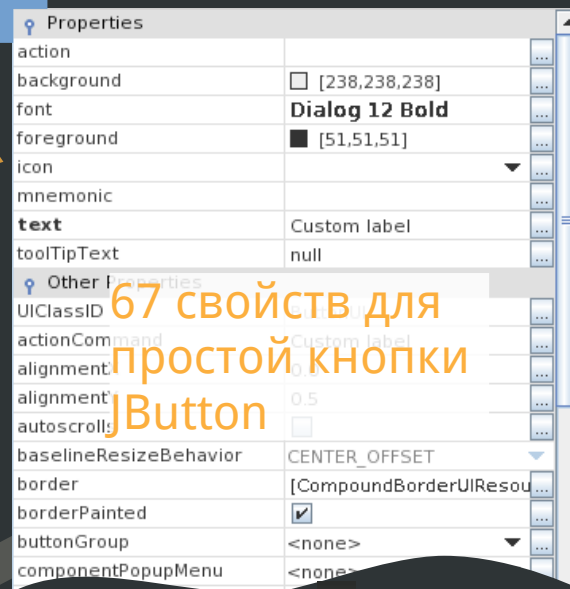
Свойства, доступные для чтения, записи и связывания (в стиле компонентных моделей)

RTTD — зачем? пример с bean'ом JButton



1 установить требуемое значение свойства


2 запустить программу



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

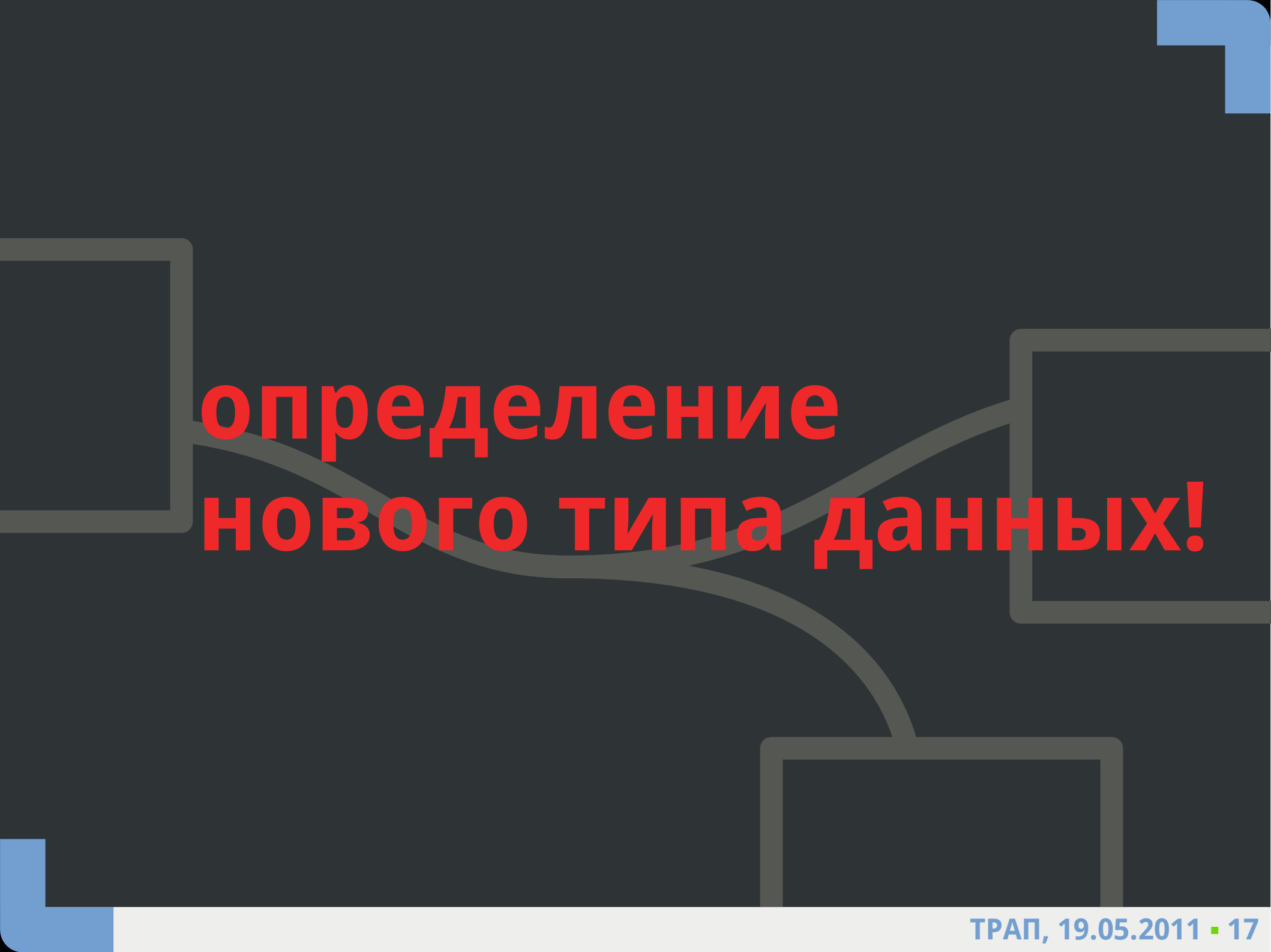
```
final String text =  
"Custom label";
```

Но как сообщить системе, что значение свойства не будет изменяться во время исполнения?



**глубокая подстройка
под контекст**

**изменение и данных,
и метаданных**

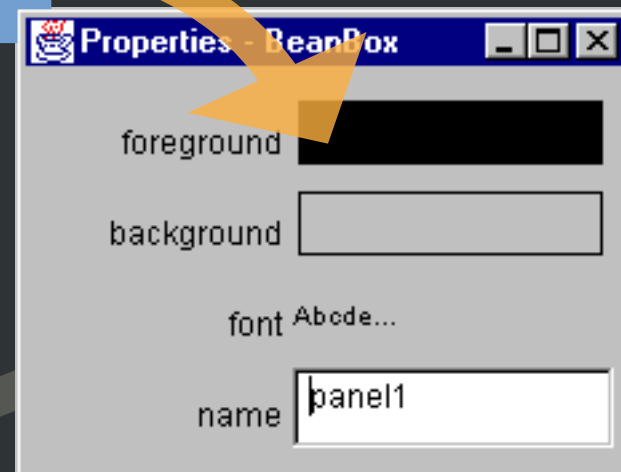
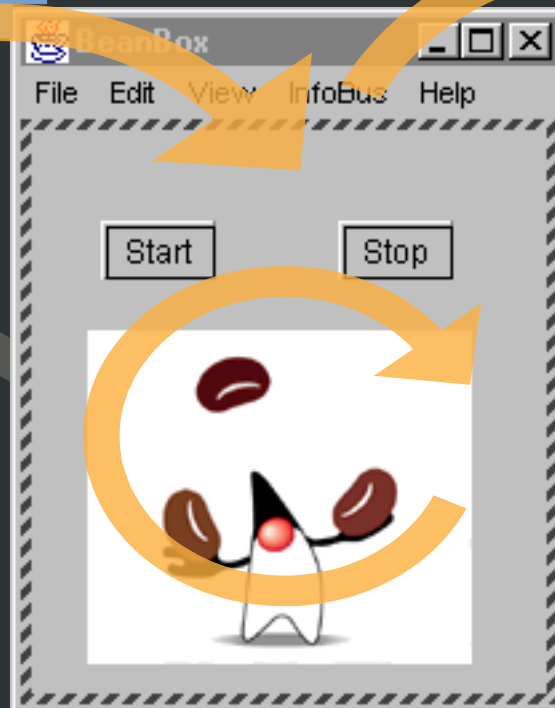
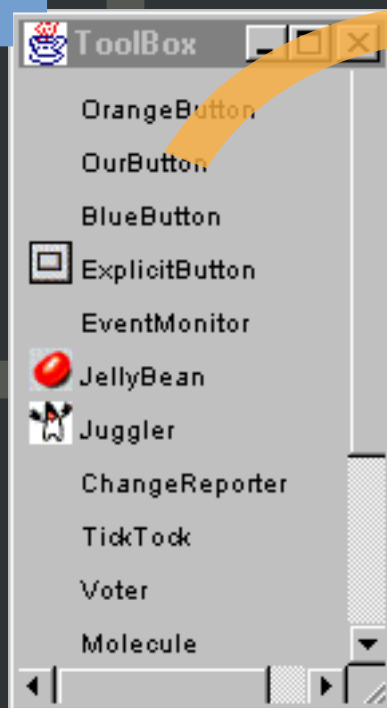


**определение
нового типа данных!**

RTTD — зачем? пример с BDK BeanBox

1 создать экземпляр компонента из predeterminedного набора компонентов

2 настроить экземпляр под контекст использования

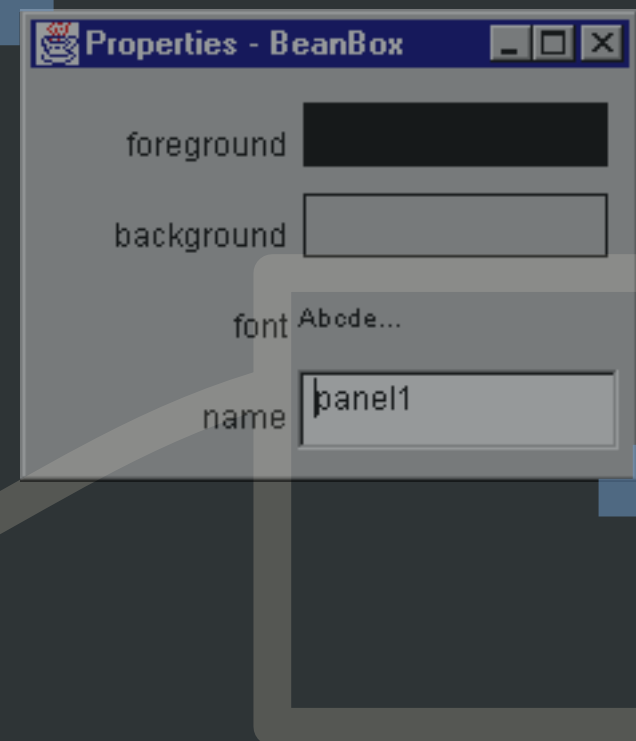
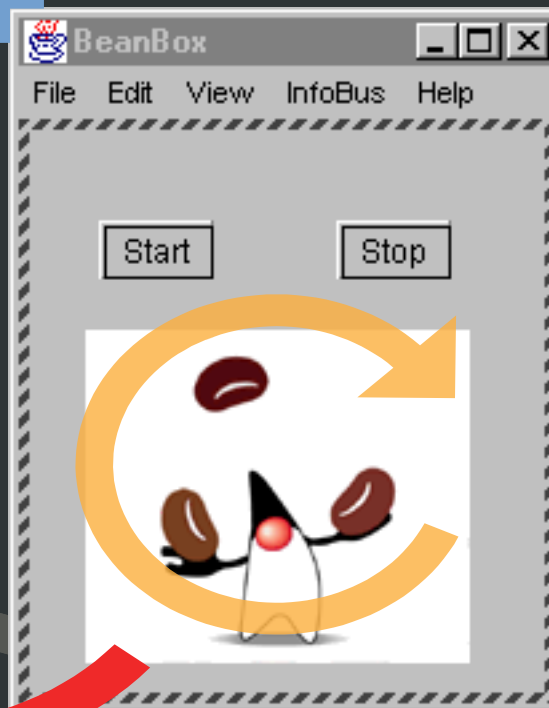
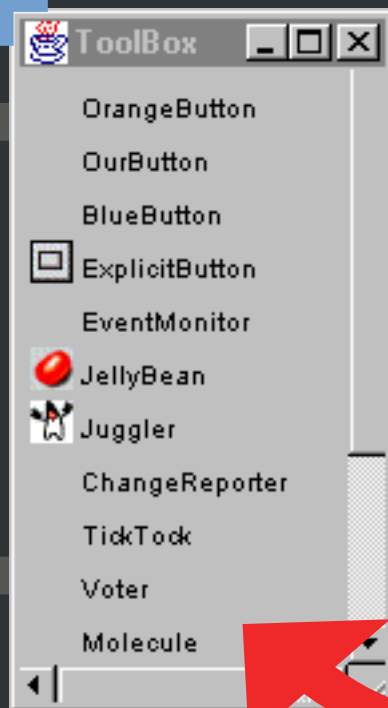


3 повторить для других компонентов

4 организовать компоненты в требуемую структуру

5 запустить структуру на выполнение

RTTD — зачем? пример с BDK BeanBox



Но если хочется добавить
получившуюся структуру в набор компонентов?

**RTTD — это то, в чем
компонентные технологии
уступают объектным ЯП**

RTTD и существующие технологии

- JavaBeans

концептуальная простота

свойства

события

методы

плоская структура экземпляров

неизменяемые метаданные

RTTD и существующие технологии

- **Fractal**

динамическое управление компонентами,
их содержимым и связями

создание НОВЫХ ТИПОВ ИЗ ТИПОВ
интерфейсов

рекурсивное отношение
«КОМПОНЕНТ — ПОДКОМПОНЕНТ»

однонаправленность отношения
«ТИП — КОМПОНЕНТ»

RTTD и существующие технологии

- VRML

иерархическая структура

создание НОВЫХ ТИПОВ

реализация PROTO посредством
макроподстановок

RTTD и существующие технологии

- Ptolemy II

иерархическая структура

создание новых типов возможность превращения модели или актора (с подстроеными параметрами) в класс

ограниченность возможностей

«подстройки» типа

допустимо создание структуры и установка значений параметров, но:

- только числовые и строковые параметры
- **«неглубокая» реконфигурация экземпляров**

RTTD и существующие технологии

- ComponentJ и ComponentGlue

иерархическая структура

динамическая реконфигурация

пример:
добавление порта

совместное использование портов

строгая типизация

необходимость компиляции

необходимость кодирования на императивной части java

Реализация RTTD

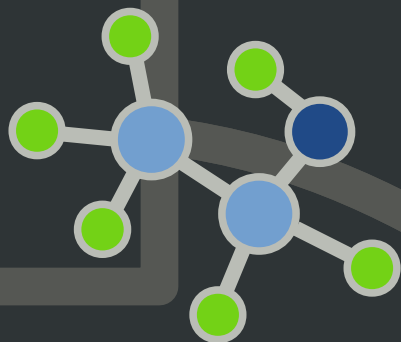
- генерация кода + вызов компилятора
- генерация бинарного или байт-кода
- клонирование
- отражение
- поддержка архитектурой

Принятые руководящие принципы



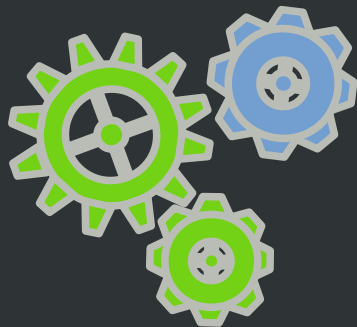
- ✓ Определение типов данных во время исполнения — Run-Time Type Definition (RTTD)

Структуризация кода и данных



- ✗ Плоское объединение экземпляров компонентов (в стиле JavaBeans)
- ✓ Иерархическая группировка экземпляров компонентов (в стиле объектных языков программирования)

Организация потока управления



- ✗ Методы (в стиле объектных языков программирования)
- ✓ Свойства, доступные для чтения, записи и связывания (в стиле компонентных моделей)

Экземпляр компонента



интерфейс —

совокупность свойств

свойство = имя +
текущее значение +
операции:

Λ r — чтение,

V w — запись

U и b — связывание

реализация

различна для

- примитивных,
- скомпилированных
- и скомпонованных компонентов

“А связано с В”



значение
изменено

новое значение
записано в В

Примитивные, скомпилированные и скомпонованные экземпляры

примитивные экземпляры

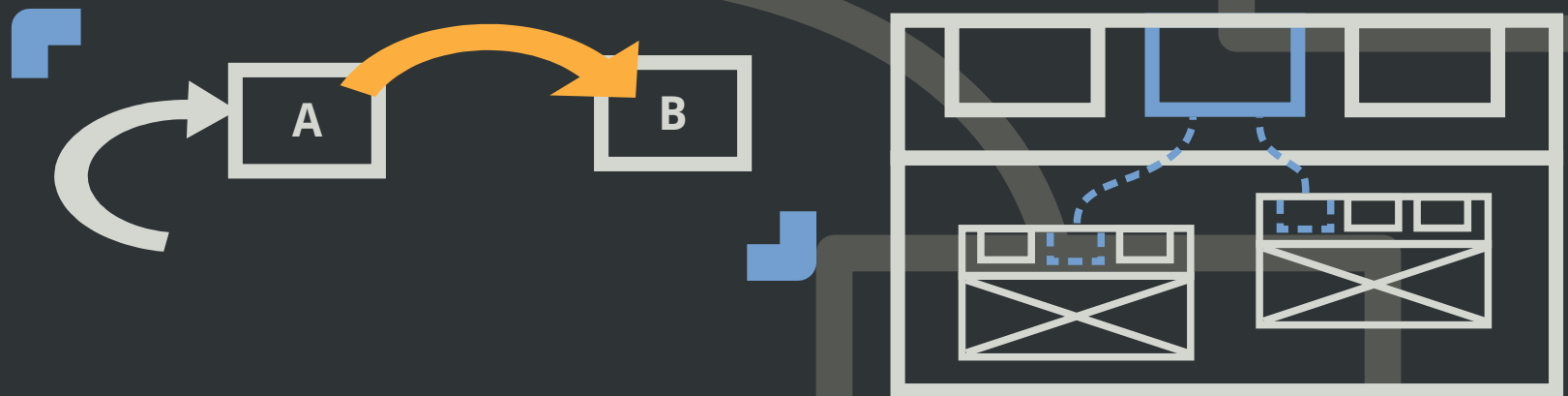
уникальны («объекты-значения») ▪ неделимы ▪ не имеют свойств, значений по умолчанию

скомпилированные экземпляры

реализованы сторонними средствами ▪ имеют свойства, значения по умолчанию ▪ поддерживают сторонние технологии

скомпонованные экземпляры

совокупность экземпляров других компонентов, взаимосвязанных посредством событийных связей и разделяемых свойств



Контейнер среда исполнения и не только

сущест-
вующий
тип

- добавлять, удалять и изменять дескрипторы свойств
 - имена, типы, значения по умолчанию, права доступа
- редактировать структуру реализации, т. е. добавлять и удалять:
 - подэкземпляры,
 - событийные связи,
 - разделяемые свойства



новый
(изменен-
ный) тип

- * экземпляры нового типа **глубоко** подстраиваются под внесенные изменения

Мы стремимся...

...представить

- простую в использовании,
- эффективную,
- гибкую (RTTD без вызова компилятора и т.п.)

компонентную архитектуру

Контейнер среда исполнения и не только

сущест-
вующий
тип

- добавлять, удалять и изменять дескрипторы свойств
 - имена, типы, значения по умолчанию, права доступа
- редактировать структуру реализации, т. е. добавлять и удалять:
 - подэкземпляры,
 - событийные связи,
 - разделяемые свойства

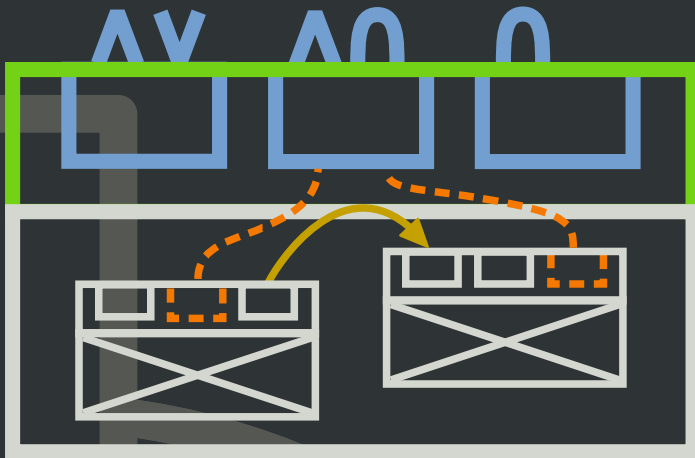


новый
(изменен-
ный) тип

* экземпляры
нового типа
глубоко
подстраиваются
под внесенные
изменения

Под капотом

скомпонованные компоненты и их инстанцирование



метаинфо интерфейса

совокупность дескрипторов свойств

дескриптор свойства = имя +
значение по умолчанию +
права применения операций:

^ r — чтения,
v w — записи
u и b — связывания

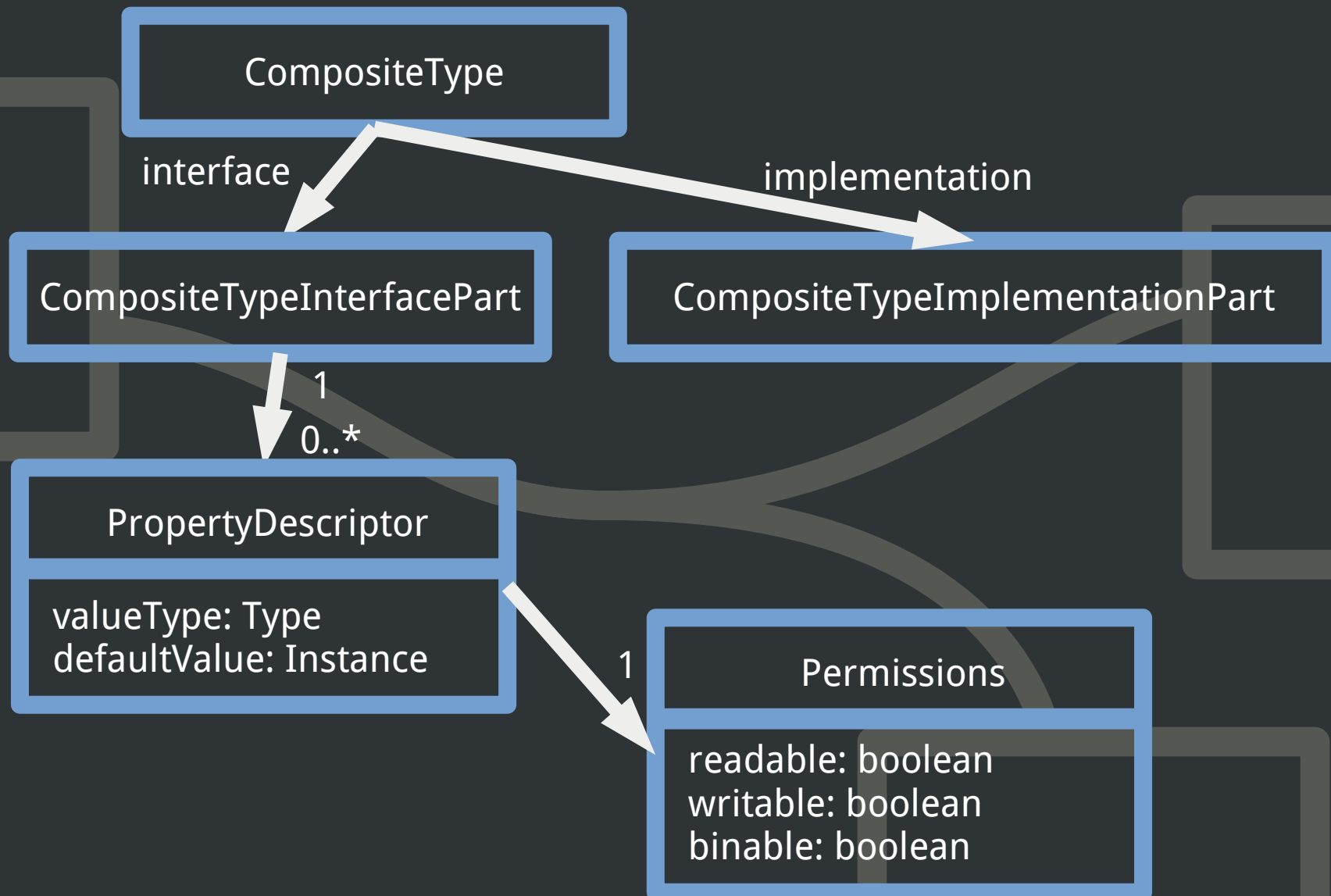
метаинфо реализации

различна для разных компонентов:

- **примитивные**
место для хранения
текущего значения
- **скомпилированные**
инструкции по
получению реализации и
соединению ее с
интерфейсом
- **скопонованные**
 - дескриптор подэкземпляра =
тип + начальное значение
 - «разделения» свойств
 - **событийные связи**

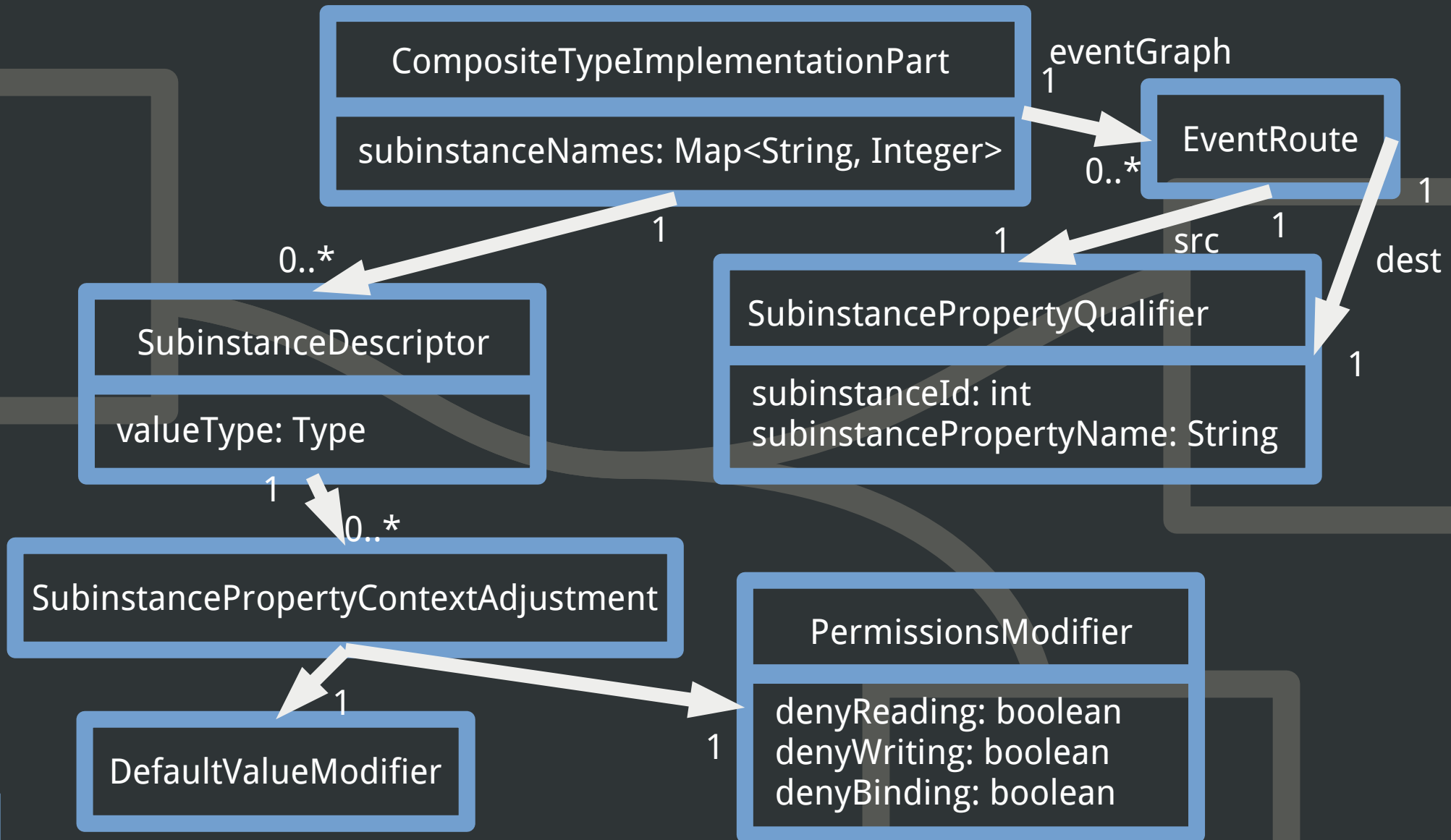
Под капотом

СКОМПОНОВАННЫЕ КОМПОНЕНТЫ И ИХ ИНСТАНЦИИРОВАНИЕ



Под капотом

СКОМПОНОВАННЫЕ КОМПОНЕНТЫ И ИХ ИНСТАНЦИИРОВАНИЕ



Процесс инстанцирования скомпонованного компонента

1

инициализировать ссылки на свойства (поля):

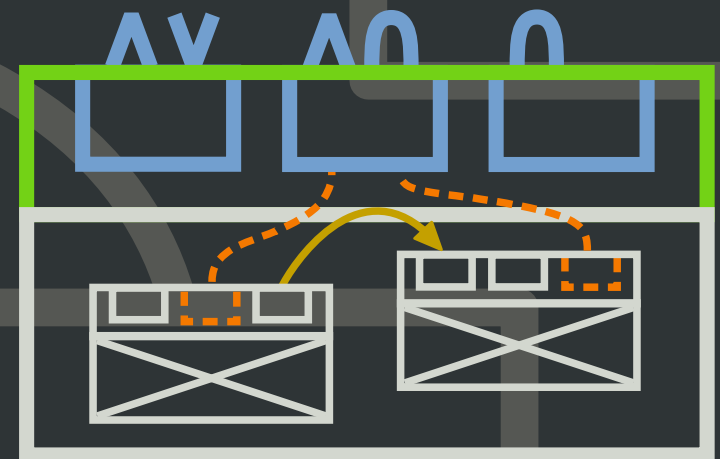
- свойствами надэкземпляра
- вновь созданными экземплярами

2

создать подэкземпляры и передать им ссылки на **разделяемые свойства**

3

установить **событийные связи**



Под капотом

выведение компонента из его прототипа

- Полагаемся на структуру времени выполнения (RS) везде, где это возможно
- Храним только те дополнительные данные, которые не могут быть выведены из RS
- Эмулируем желаемое поведение, когда оно недостижимо без пересоздания всей RS

Планы на будущее

- UI


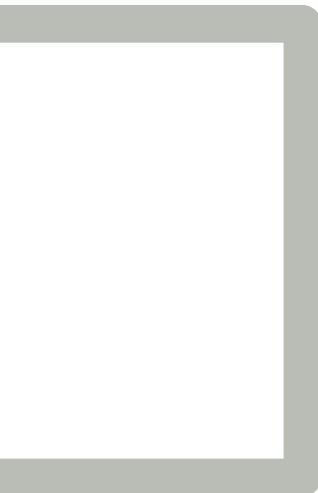

- Скриптовый (в дополнение к XML)
- GUI (с различными выходными форматами)

- Thread safety

- ? Наследование

- **Практическое применение**

- прошивка микроэлектромеханических сенсоров
- 3D-визуализация
- средства разработки GUI
- ...предложения приветствуются!



**Благодарю за
внимание!**



Амир Шакуров amir-shak@yandex.ru
НИУ ВШЭ, отделение программной инженерии

Семинар «Технологии разработки и анализа программ», 19 мая 2011 г.

XML-ВЫВОД

```
<TypeLibrary>
  <Type name="SampleType">
    <Interface>
      <Property name="value" type="Integer" accessR="true"
        accessW="true" accessB="true" defaultValue="5"/>
    </Interface>

    <Implementation>
      <Variable type="String" defaultValue="Hello"/>
      <Variable type="ThirdPartyComponent"/>
      <!-- ... -->
    </Implementation>
  </Type>

  <Type name="Person">
    <Interface>
      <Property name="Surname" type="String" accessR="true"
        accessW="true" accessB="true" defaultValue=""/>
      <Property name="FirstName" type="String" accessR="true"
        accessW="true" accessB="true" defaultValue=""/>
      <Property name="Age" type="Integer" accessR="true"
        accessW="true" accessB="true" defaultValue="1"/>
    </Interface>
    <Implementation>
      <!-- написано на Java -->
    </Implementation>
  </Type>
</TypeLibrary>
```


Script-UI

```
>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
```

гибко... но не совсем

- Объектные языки программирования

разработка
приложений для
специфических задач

динамическая
реконфигурация
системы

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

упрощение разработки
определенного рода ПО