DATA ART Enjoy IT

Artyom Astafurov

astaff@dataart.com

DATA ART Enjoy IT

Разработка автоматных объектов в императивных языках с использованием декларативного подхода

Артем Астафуров

<astaff@dataart.com>

Artyom Astafurov

astaff@dataart.com

Разработка автоматных объектов в императивных языках с использованием декларативного подхода

- Об автоматном программировании
- Описание проблемы
- Декларативный vs. Императивный
- Декларативный подход в императивных языках
- Описание подхода
- Реализация на платформе .NET
- Инструментализация vs Context Bound Objects
- Примеры
- Заключение

Artyom Astafurov

astaff@dataart.com

Об автоматном программировании

- Программирование с явным выделением состояний
- **Элементы:**
  - Состояния
  - Условия перехода
  - Входные воздействия
  - Выходные воздействия
- **Широко используется в:**
  - Embedded-системах
  - Системах реального времени
  - В системах с высокими требованиями по надежности
- **Позволяет:**
  - Легко описать сложное поведение при использовании очень простой нотации
  - Верифицировать при помощи различных верификаторов (SPIN, Bagor)
  - Управлять изменениями в сложной логике системы

Artyom Astafurov

astaff@dataart.com

Описание проблемы

- **Существующих средств реализации автоматной логики не достаточно:**
  - Классический SWITCH не подходит в ОО-приложениях
  - Паттерн State из GoF сильно увеличивает связность и децентрализует логику автомата
  - Визуальные методы и генерация кода затрудняют отладку и сборку приложения
- **Необходимо придумать подход который:**
  - Позволяет легко описывать структуру автомата
  - Делает логику переходов легко читаемой
  - Позволяет легко отлаживать автомат, пользуясь стандартными средствами IDE

Artyom Astafurov

astaff@dataart.com

Декларативный vs. Императивный

- **Императивный**
  - Отвечает на вопрос “как сделать?”
  - Предполагает описание логики в явном виде
  - Хорошо подходит для описания логики и действий программы
  - Пример: программы на языке C#, Java, etc
- **Декларативный**
  - Отвечает на вопрос “на что должно быть похоже?”
  - Описывает ЧТО должно быть сделано, не говоря о том КАК именно это должно быть сделано
  - Хорошо подходит для описания свойств компонент, определения взаимодействия и структуры системы
  - Пример: HTML, DSL, даже некоторые T-SQL statements

Artyom Astafurov

astaff@dataart.com

Декларативный подход в императивных языках

- **Необходима метаинформация:**
  - Атрибуты
  - Комментарии с метаязыком внутри
  - Внешние файлы с мета информацией
  - *Пример: Hibernate/NHibernate*
- **Способы интерпретации метаинформации:**
  - Инструментализация сборок
  - Использование перехвата вызовов

Artyom Astafurov

astaff@dataart.com

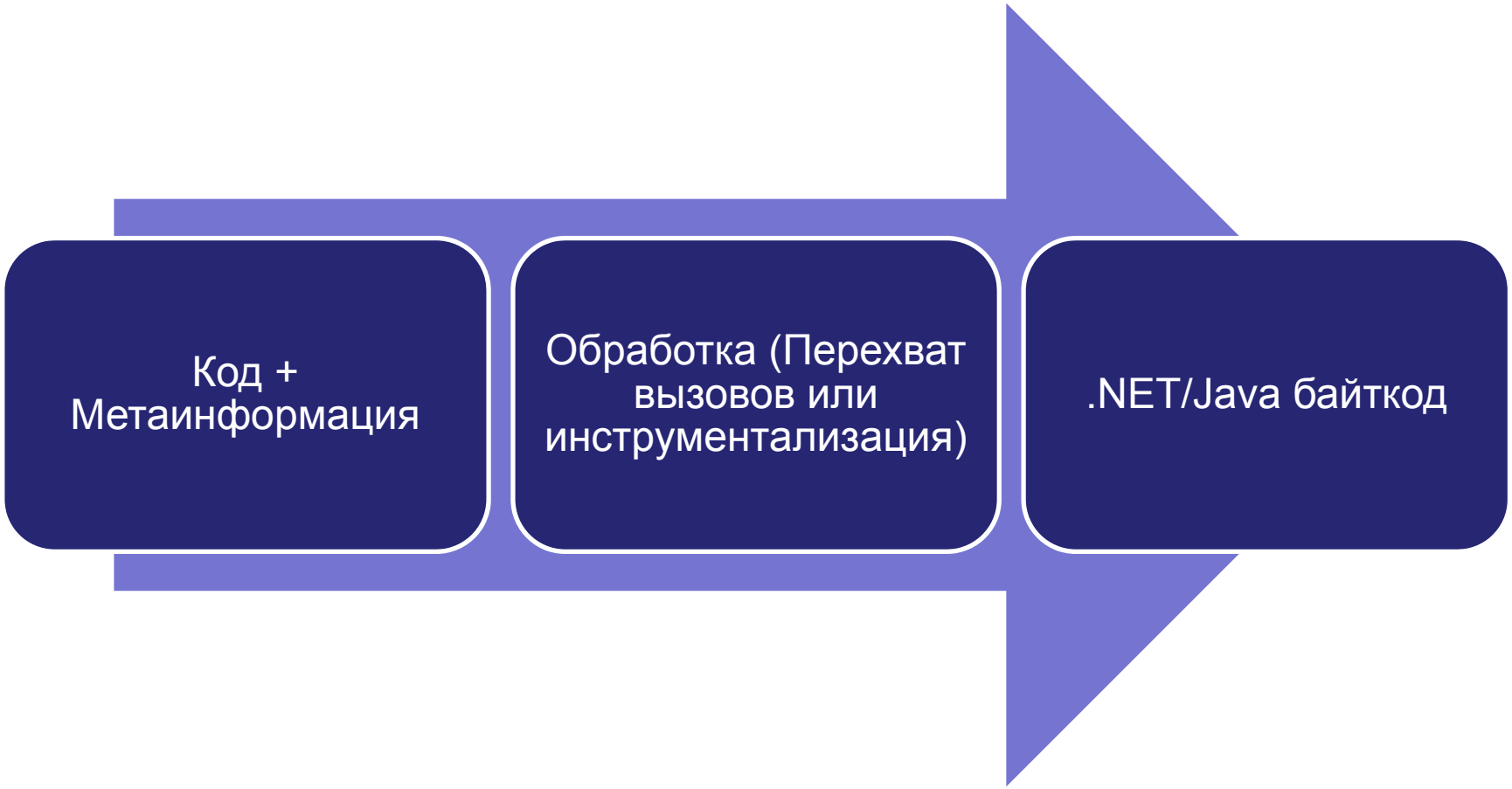
Реализация на платформе .NET

- Атрибуты в качестве мета-языка
- Перхват вызозов (Context Bound Objects в .NET) или инструментализация для перехвата вызовов
- Состояния наследуются от базового класса библиотеки
- Атрибутами определяются вложение состояний и перекрытие состояний при наследовании

Artyom Astafurov

astaff@dataart.com

Как это работает?



DATAART Enjoy IT
Artyom Astafurov
astaff@dataart.com

Инструментализация vs Context Bound Objects

Инструментализация

- Обработка происходит на этапе компиляции
- Изменяет исполняемый код ➔ проблемы с отладкой
- Естественный байт-код ➔ не влияет на производительность

Context Bound Objects

- Обработка происходит на этапе выполнения
- Исполняемый код не изменяется, код СВО доступен для отладки
- Дополнительный механизм времени выполнения ➔ менее эффективен

Artyom Astafurov

astaff@dataart.com

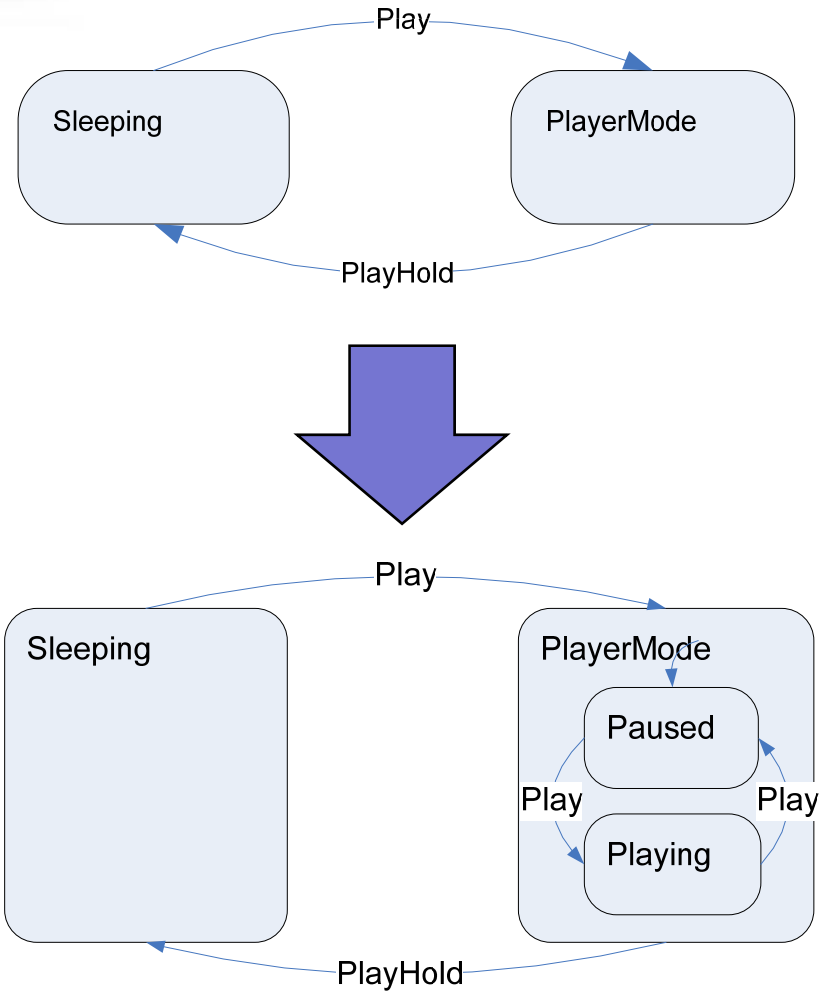
Наследование и вложение

- Вложение макросостояний описывается с помощью метаинформации
- Все вызовы вложенных макросостояний передаются без явного обращения в коде
- Перекрытие состояний при наследовании производится с помощью метаинформации

Artyom Astafurov

astaff@dataart.com

Пример (Apple iPod)

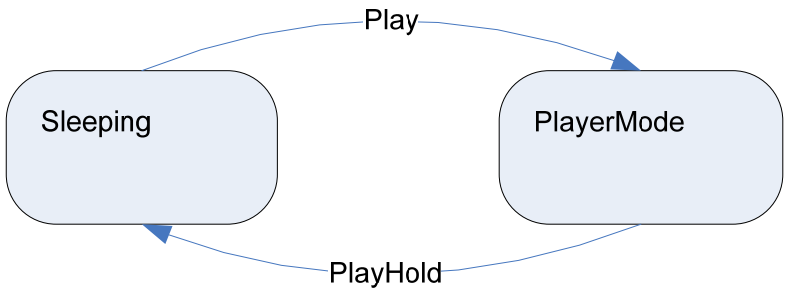


Artyom Astafurov

astaff@dataart.com

Высокоуровневый автомат

```
public interface IAcceptsPlay
{
    void Play();
}
public interface IAcceptsPlayHold
{
    void PlayHold();
}
public interface IIpodPlayer :
 IAcceptsPlay, IAcceptsPlayHold
{
}
```



Artyom Astafurov

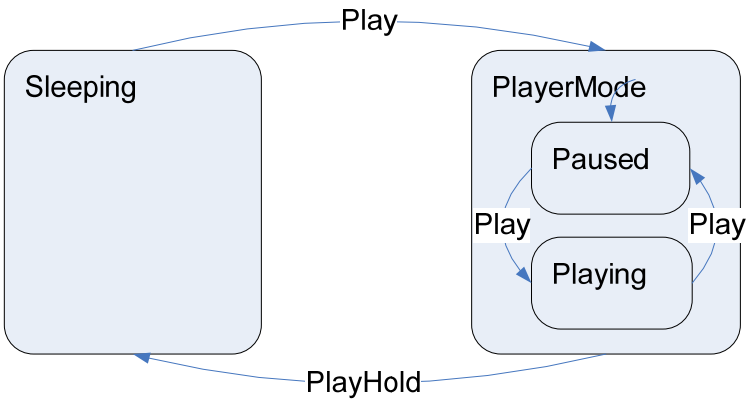
astaff@dataart.com

Пример вложения и наследования

```
public class Paused : State, IAcceptPlay
{
 public void Play()
    {
     Container.SetState(typeof(Playing));
    }
}

public class Playing : State, IAcceptPlay
{
 public void Play()
    {
     Container.SetState(typeof(Paused));
    }
}

[InitialState(typeof(Paused)),
State(typeof(Playing))]
public class PlayerModeWithNestedStates :
PlayerMode
{
 // That's it!
}
```

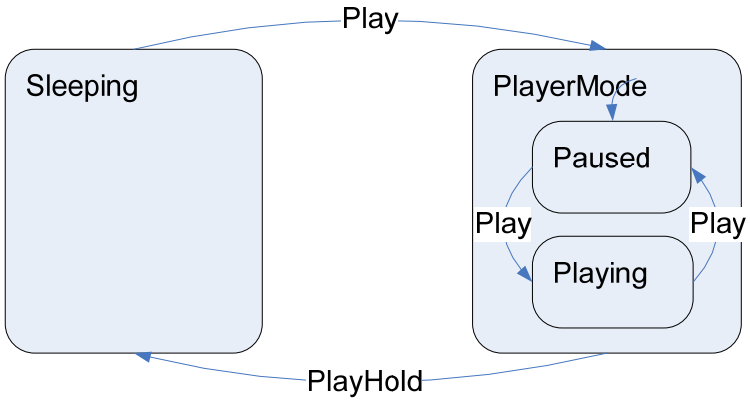


Artyom Astafurov

astaff@dataart.com

Пример вложения и наследования

```
[State(typeof(PlayerMode),
typeof(PlayerModeWithNestedStates))]
public class IpodPlayerExt : IpodPlayer
{
 // That's it!
}
```



Artyom Astafurov

astaff@dataart.com

Заключение

- Проведен анализ существующих подходов к реализации автоматов
- Сформулирован метаописательный подход к построению автоматов в современных ОО-языках
- Разработана библиотека для реализации автоматов в .NET при помощи метаинформации

Artyom Astafurov

astaff@dataart.com

?

astaff@dataart.com

http://is.ifmo.ru