

# Лекция 10: Архитектурные стили

Юрий Литвинов  
yurii.litvinov@gmail.com

15.11.2019г

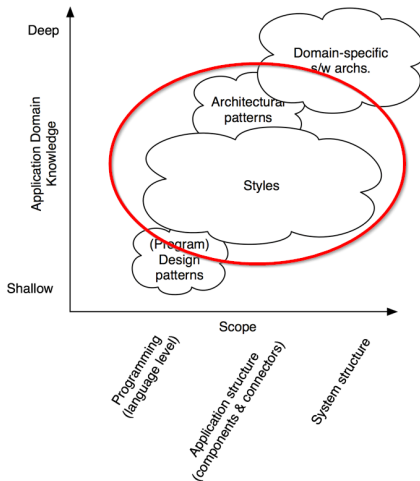
# Архитектурные шаблоны и стили

Архитектурный стиль — набор решений, которые

1. применимы в выбранном контексте разработки,
2. задают ограничения на принимаемые архитектурные решения, специфичные для определённых систем в этом контексте,
3. приводят к желаемым положительным качествам получаемой системы.

Архитектурный шаблон — именованный набор ключевых проектных решений по эффективной организации подсистем, применимых для повторяемых технических задач проектирования в различных контекстах и предметных областях

# Архитектурные шаблоны и стили, классификация

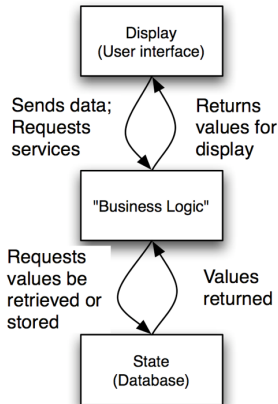


# Пример: трёхзвенная архитектура

## State-Logic-Display

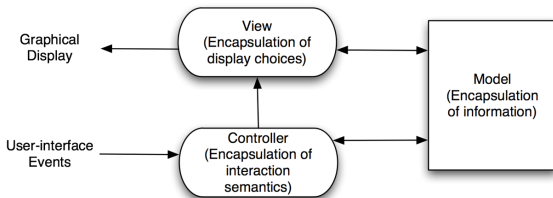
### Примеры применения

- ▶ Бизнес-приложения
- ▶ Многопользовательские игры
- ▶ Веб-приложения



© N. Medvidovic

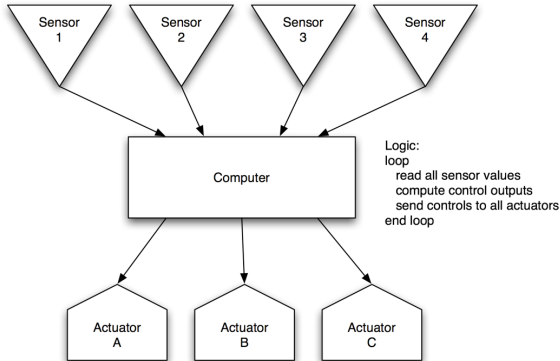
# Пример: Model-View-Controller



© N. Medvidovic

- ▶ Разделяет данные, представление и взаимодействие с пользователем
- ▶ Если в модели что-то меняется, она оповещает представление (представления)
- ▶ Через контроллер проходит всё взаимодействие с пользователем
  - ▶ Естественное место для паттерна “Команда” и Undo/Redo

# Пример: Sense-Compute-Control

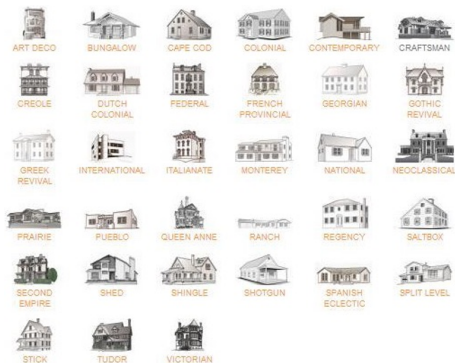


© N. Medvidovic

- ▶ Применяется во встроенных системах и робототехнике

# Архитектурные стили

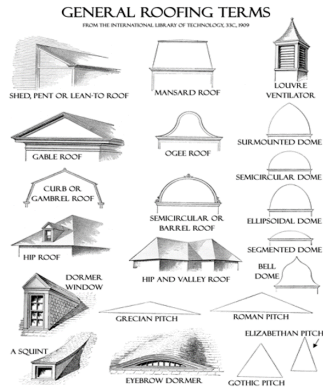
- ▶ Именованная коллекция архитектурных решений
- ▶ Менее узкоспециализированные, чем архитектурные паттерны



© N. Medvidovic

# Архитектурные стили

- ▶ Одна система может включать в себя несколько архитектурных стилей
- ▶ Понятие стиля применимо и к подсистемам



© N. Medvidovic



# Преимущества использования стилей

- ▶ Переиспользование архитектуры
  - ▶ Для новых задач можно применять хорошо известные и изученные решения
- ▶ Переиспользование кода
  - ▶ Часто у стилей бывают неизменяемые части, которые можно один раз реализовать
- ▶ Упрощение общения и понимания системы
- ▶ Упрощение интеграции приложений
- ▶ Специфичные для стиля методы анализа
  - ▶ Возможны благодаря ограничениям на структуру системы
- ▶ Специфичные для стиля методы визуализации

# Основные характеристики стилей

- ▶ Набор используемых элементов архитектуры
  - ▶ Типы компонентов и соединителей, элементы данных
    - ▶ Например, объекты, фильтры, сервера и т.д.
- ▶ Набор правил конфигурирования
  - ▶ “Топологические” ограничения на соединение элементов
    - ▶ Например, компонент может быть соединён с максимум двумя компонентами
- ▶ Семантика, стоящая за элементами

# Игра “Посадка на луну”

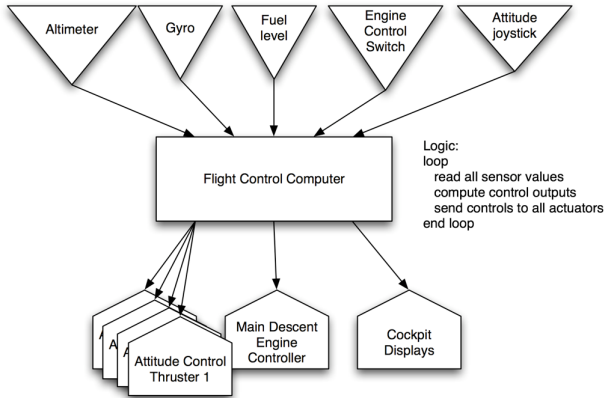
## Lunar Lander

- ▶ Игрок управляет двигателем спускаемого аппарата
- ▶ Топливо ограничено
- ▶ Заданы начальная высота и скорость
- ▶ Победа засчитывается, если скорость при касании грунта меньше заданной
- ▶ Продвинутая версия позволяет управлять горизонтальным движением



© N. Medvidovic

# Sense-Compute-Control-реализация

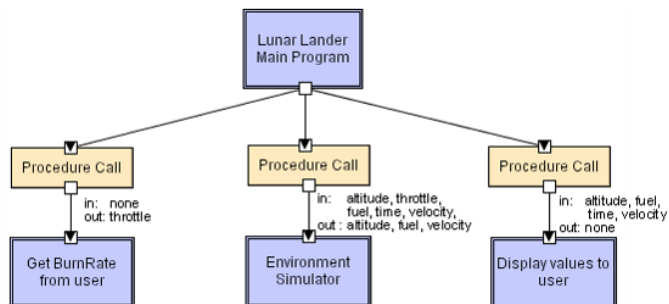


© N. Medvidovic

# Некоторые известные стили

- ▶ “Традиционные”, связанные с языком
  - ▶ Главная программа/подпрограммы
  - ▶ Объектно-ориентированный
- ▶ Уровневый стиль
  - ▶ Виртуальные машины
  - ▶ Клиент-сервер
- ▶ Стили, ориентированные на поток данных
  - ▶ Пакетное исполнение
  - ▶ Каналы и фильтры
- ▶ Peer-to-peer
- ▶ Общая память
  - ▶ Blackboard
  - ▶ Ориентированные на правила
- ▶ Интерпретаторы
  - ▶ Интерпретатор
  - ▶ Мобильный код
- ▶ Неявный вызов
  - ▶ Событийно-ориентированный
  - ▶ Издатель-подписчик
- ▶ “Производные” стили
  - ▶ Распределённые объекты
  - ▶ REST
  - ▶ C2

# Главная программа/подпрограммы

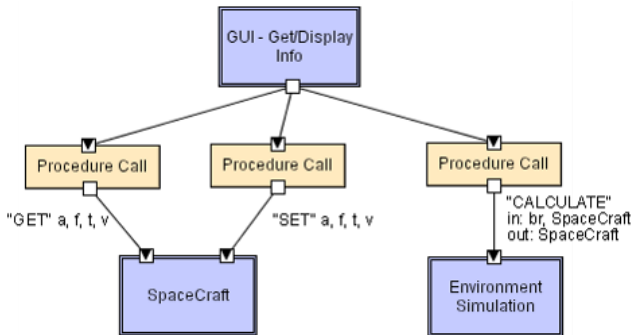


© N. Medvidovic

# Объектно-ориентированный стиль

- ▶ Компоненты — объекты
- ▶ Соединители — сообщения и вызовы методов
- ▶ Инварианты:
  - ▶ Объекты отвечают за своё внутреннее состояние
  - ▶ Реализация скрыта от других объектов
- ▶ Преимущества:
  - ▶ Декомпозиция системы в набор взаимодействующих агентов
  - ▶ Внутреннее представление объектов можно менять независимо
  - ▶ Близко к предметной области
- ▶ Недостатки:
  - ▶ Побочные эффекты при вызове методов
  - ▶ Объекты вынуждены знать обо всех, от кого зависят

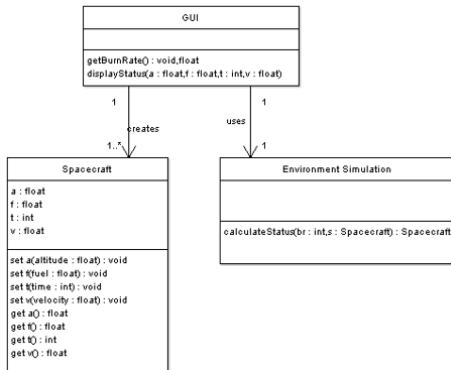
# Объектно-ориентированный стиль, Lunar Lander



© N. Medvidovic



## Или то же на UML

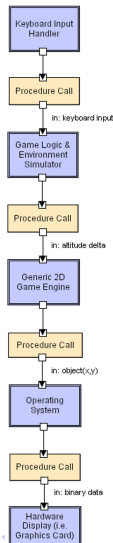


© N. Medvidovic

# Слоистый стиль

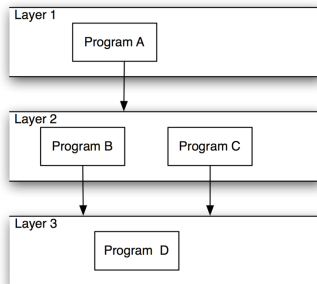
Layered style

- ▶ Иерархическая организация системы
  - ▶ “Многоуровневый клиент-сервер”
  - ▶ Каждый слой предоставляет интерфейс для использования слоями выше
- ▶ Каждый слой работает как:
  - ▶ Сервер — предоставляет функциональность слоям выше
  - ▶ Клиент — использует функциональность слоёв ниже
- ▶ Соединители — протоколы взаимодействия слоёв
- ▶ Пример — операционные системы, сетевые стеки протоколов



# Слоистый стиль, подробности

- ▶ Преимущества:
  - ▶ Повышение уровня абстракции
  - ▶ Лёгкость в расширении
  - ▶ Изменения в каждом уровне затрагивают максимум два соседних
  - ▶ Возможны разные реализации уровня, если они удовлетворяют интерфейсу
- ▶ Недостатки:
  - ▶ Не всегда применим
  - ▶ Проблемы с производительностью

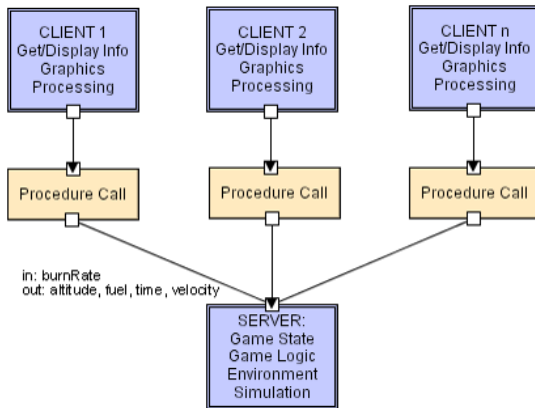


© N. Medvidovic

# “Клиент-сервер”

- ▶ Компоненты — клиенты и серверы
- ▶ Серверы не знают ничего о клиентах, даже их количество
- ▶ Клиенты знают только про сервера и не могут общаться друг с другом
- ▶ Соединители — сетевые протоколы

# “Клиент-сервер”, Lunar Lander

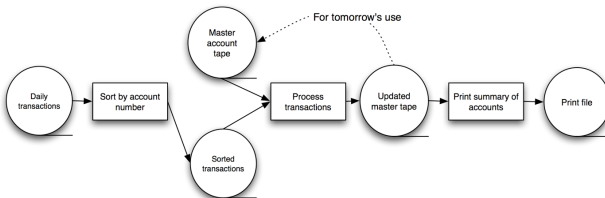


© N. Medvidovic

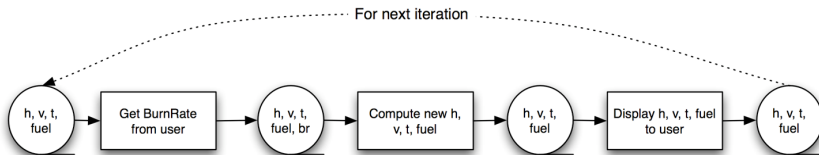
# Пакетная обработка

- ▶ Система строится как набор отдельных программ, выполняющихся последовательно
- ▶ Данные стандартным для ОС способом передаются от программы к программе
  - ▶ Pipes, named pipes, файлы
- ▶ Данные — в явном виде всё, необходимое для работы

Типичен для финансовых систем глубокой древности (“Прадедушка стилей”)



# Пакетная обработка, Lunar Lander



© N. Medvidovic

## Play-by-email?

# Каналы и фильтры

## Pipes and filters

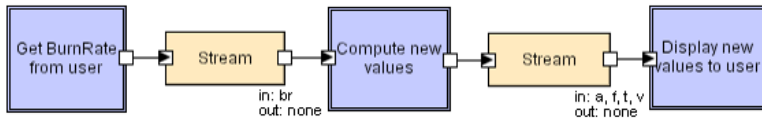
- ▶ Компоненты — это фильтры, преобразующие данные из входных каналов в данные в выходных каналах
- ▶ Соединители — каналы
- ▶ Инварианты:
  - ▶ Фильтры независимы (не имеют разделяемого состояния)
  - ▶ Фильтры не знают о фильтрах до или после них
- ▶ Вариации:
  - ▶ Конвейеры — линейные последовательности фильтров
  - ▶ Ограниченные каналы — где канал это очередь с ограниченным количеством элементов
  - ▶ Типизированные каналы — где каналы отличаются по типу передаваемых данных



# Каналы и фильтры, подробности

- ▶ **Преимущества:**
  - ▶ Поведение системы — это просто последовательное применение поведений компонентов
  - ▶ Легко добавлять, заменять и переиспользовать фильтры
    - ▶ Любые два фильтра можно использовать вместе
  - ▶ Широкие возможности для анализа
    - ▶ Пропускная способность, задержки, deadlock-и
  - ▶ Широкие возможности для параллелизма
- ▶ **Недостатки:**
  - ▶ Последовательное исполнение
  - ▶ Проблемы с интерактивными приложениями
  - ▶ Пропускная способность определяется самым “узким” элементом

# Каналы и фильтры, Lunar Lander

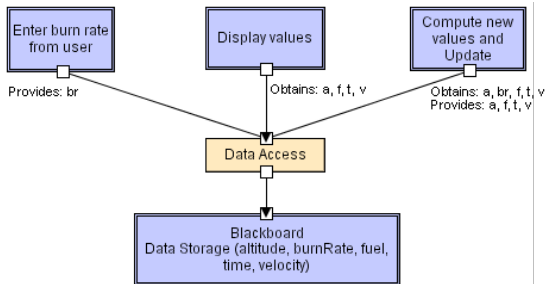


© N. Medvidovic

# Blackboard

- ▶ Два типа компонентов:
  - ▶ Центральная структура данных — та самая “Blackboard”
  - ▶ Компоненты, работающие с blackboard
- ▶ Инварианты:
  - ▶ Управление системой осуществляется только через состояние доски
  - ▶ Компоненты не знают друг о друге и не имеют своего состояния
- ▶ Часто применяется в системах искусственного интеллекта

# Blackboard, Lunar Lander



© N. Medvidovic

# Стили с неявным вызовом

- ▶ Оповещение о событии вместо явного вызова метода
  - ▶ “Слушатели” могут подписаться на событие
  - ▶ Система при наступлении события сама вызывает все зарегистрированные методы слушателей
- ▶ Компоненты имеют два вида интерфейсов — методы и события
- ▶ Два типа соединителей:
  - ▶ Явный вызов метода
  - ▶ Неявный вызов по наступлению события
- ▶ Инварианты:
  - ▶ Те, кто производит события, не знают, кто и как на них отреагирует
  - ▶ Не делается никаких предположений о том, как событие будет обработано и будет ли вообще

# Стили с неявным вызовом, преимущества и недостатки

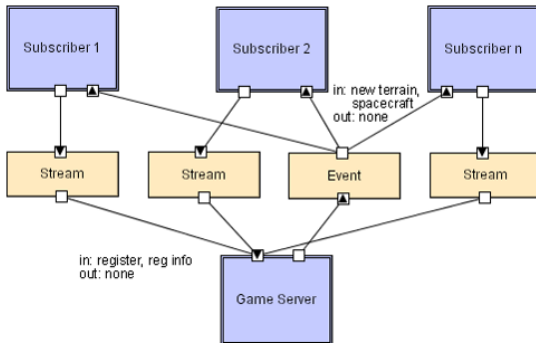
- ▶ Преимущества:
  - ▶ Переиспользование компонентов
    - ▶ Очень низкая связность между компонентами
  - ▶ Лёгкость в конфигурировании системы
    - ▶ Как во время компиляции, так и во время выполнения
- ▶ Недостатки:
  - ▶ Зачастую неинтуитивная структура системы
  - ▶ Компоненты не управляют последовательностью вычислений
  - ▶ Непонятно, кто отреагирует на запрос и в каком порядке придут ответы
  - ▶ Тяжело отлаживаться
  - ▶ Гонки даже в однопоточном приложении

# Издатель-подписчик

## Publish-subscribe

- ▶ Подписчики регистрируются, чтобы получать нужные им сообщения или данные. Издатели публикуют сообщения, синхронно или асинхронно.
- ▶ Компоненты: издатели, подписчики, “маршрутизаторы”
- ▶ Соединители: как правило, сетевые протоколы, часто механизм наподобие паттерна “Наблюдатель”
- ▶ Данные: подписки, нотификации, публикуемая информация
- ▶ Топология: подписчики подключаются к издателям напрямую, либо через посредников
- ▶ Преимущества: очень низкая связность между компонентами, при этом высокая эффективность распределения информации

# Издатель-подписчик, Lunar Lander



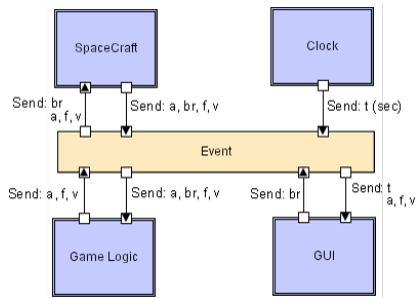
© N. Medvidovic



# Событийно-ориентированный стиль

- ▶ Независимые компоненты посылают и принимают события, передаваемые по шинам
- ▶ Компоненты: независимые генераторы или потребители событий
- ▶ Соединители: шины событий (хотя бы одна)
- ▶ Данные: события и связанные с ними данные, посылаемые по шине
- ▶ Топология: компоненты общаются только с шинами событий, не друг с другом
- ▶ Варианты: push- и pull-режимы работы с шиной
- ▶ Преимущества: лёгкость масштабирования и добавления новой функциональности, эффективно для распределённых приложений

# Событийно-ориентированный Lunar Lander

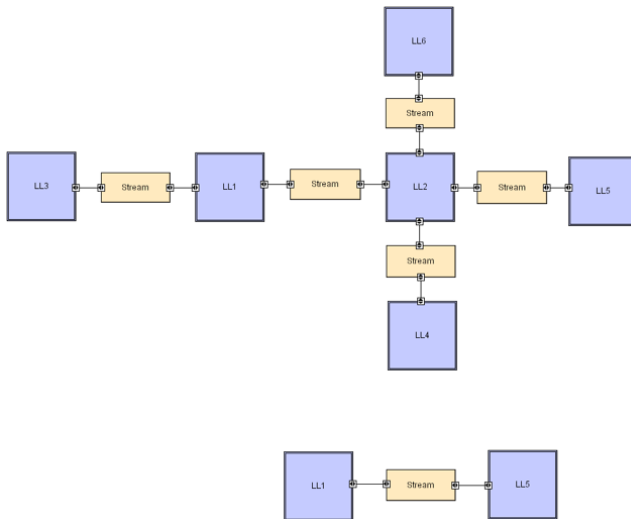


© N. Medvidovic

# Peer-to-peer

- ▶ Состояние и поведение распределены между компонентами, которые могут выступать как клиенты и как серверы
- ▶ Компоненты: имеют своё состояние и свой поток управления
- ▶ Соединители: как правило, сетевые протоколы
- ▶ Элементы данных: сетевые сообщения
- ▶ Топология: сеть (возможно, с избыточными связями между компонентами), может динамически меняться
- ▶ Преимущества:
  - ▶ Хорош для распределённых вычислений
  - ▶ Устойчив к отказам
  - ▶ Если протокол взаимодействия позволяет, легко масштабируется

# Peer-to-peer Lunar Lander



© N. Medvidovic

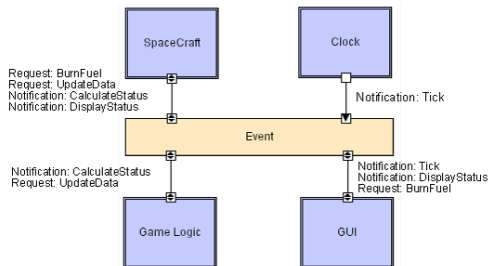
# Гетерогенные стили

- ▶ Более сложные стили, полученные соединением простых стилей
  - ▶ REST
  - ▶ C2
  - ▶ Распределённые объекты
    - ▶ OO + клиент-сервер
    - ▶ CORBA

## C2

- ▶ Стилль с неявным вызовом, где компоненты общаются только через коннекторы, маршрутизирующие сообщения
- ▶ Компоненты: независимые, потенциально параллельные производители или потребители
- ▶ Соединители: маршрутизаторы сообщений, которые могут фильтровать, преобразовывать и рассылать сообщения двух видов: нотификации и запросы
- ▶ Элементы данных: сообщения, содержащие данные
  - ▶ Нотификации анонсируют изменения в состоянии
  - ▶ Запросы — запрашивают выполнение действия
- ▶ Топология: слои компонентов и соединителей с определённым “верхом” и “низом”, где нотификации шлются “вниз”, а запросы — “вверх”

# C2 Lunar Lander



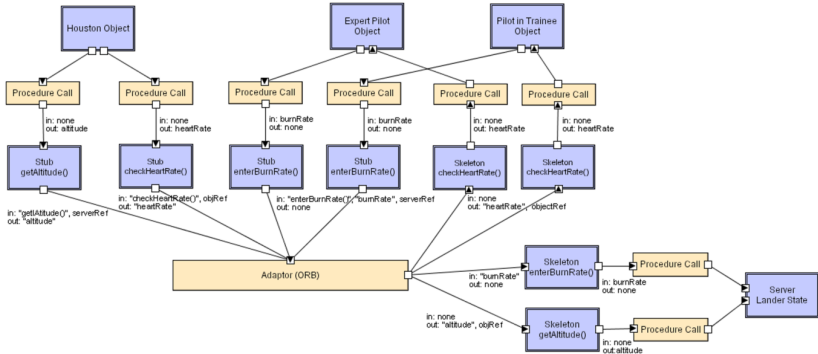
© N. Medvidovic

# CORBA

- ▶ “Объекты” работают на гетерогенных хостах, реализованные на разных языках программирования
- ▶ Объекты предоставляют сервисы через чётко определённые интерфейсы и вызывают методы через RPC-протоколы
- ▶ Топология: граф объектов в самом общем смысле
- ▶ Дополнительные ограничения:
  - ▶ Передаваемые при вызове метода данные должны быть сериализуемы
  - ▶ Вызывающие должны обрабатывать ошибки, связанные с работой сети
- ▶ Преимущества: независимость от платформы, языка и местоположения сервиса



# CORBA Lunar Lander



© N. Medvidovic