www.dpi.solutions

Microkernel Architecture based on Pipes and Filters

Anton Semenchenko

www.dpi.solutions

About myself ☺

Co-creator of communities www.COMAQA.BY and www.CoreHard.by, co-founder of company www.DPI.Solutions, «tricky» manager at EPAM Systems.

COMAQA.BY
QA automation community

COREHARD
C++ COMMUNITY

www.dpi.solutions

Agenda

1. Idea
2. History
3. **Microkernel** Architecture
4. **Pipes and Filters**
5. **Microservices** Architecture
6. Summary
7. Examples
8. Quality assurance
9. What's next
10. Books
11. Engineers from 70-s

www.dpi.solutions

Idea

1. Everything new - is totally forgotten old things
2. Study, study and study once again
3. Read books
4. Think about the idea, and do not drown in tons of terminology and positioning slogans

www.dpi.solutions

History ☺

1. **Microkernel** Architecture conception – 1960-s
2. **Pipes and Filters** conception – 1960-s
3. UNIX – 1970 (+ *Dennis MacAlistair Ritchie*)
4. UNIX pipeline – 1973 (*Malcolm Douglas McIlroy*)
5. Component-based software engineering – 1987 (*Niklaus Emil Wirth*)
6. Component-based unification – 1989 (*Bertrand Meyer*)
7. COM – 1993
8. CORBA – 1990-s
9. SOAP – 1990-s \ 2000-s
10. JavaBeans
11. .Net components
12. **Microservices** initial definition – 2005
13. **Microservices** Architecture - 2012
14. PowerShells

www.dpi.solutions

Microkernel

Introduction

1. The Microkernel architectural pattern applies to software systems that must **be able to adapt to changing system requirements**.
2. It separates a **minimal functional core** from **extended functionality** and **customer specific parts**.
3. The microkernel also serves as a socket for plugging in these extensions and coordinating their collaboration.
4. Microkernel systems have mainly been described in relation to the design of operating systems. However, this pattern is **also \ mainly applicable to other domains, e.g. financial applications** and database systems.

www.dpi.solutions

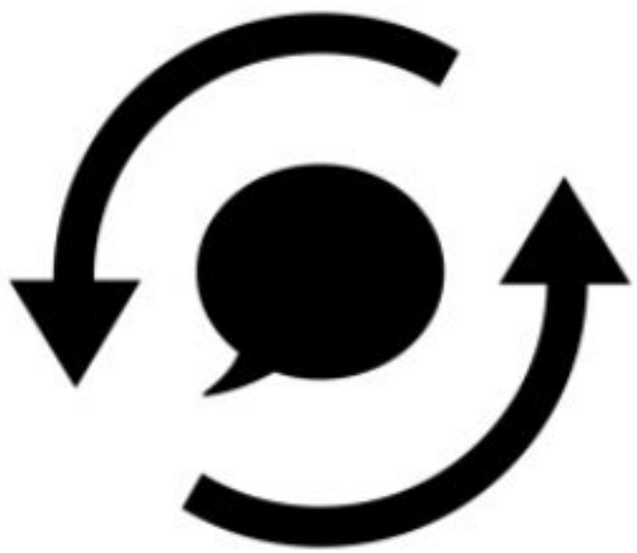
Challange

1. How to design an architecture for the development of **several applications with similar programming interfaces that build on the same core functionality**.

www.dpi.solutions

Context

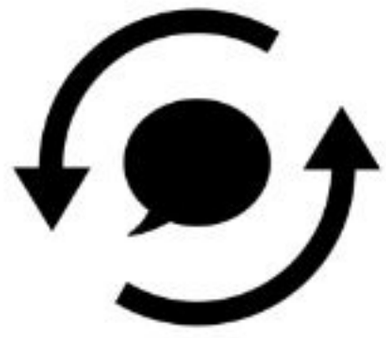
1. **Developing software for an application domain that needs to cope with a broad spectrum of similar standards and technologies is a nontrivial task.**
2. Well-known examples are application platforms such as operating systems and graphical user interfaces.
3. **Such systems often have a long life-span, often ten years or more.**
4. **Over time periods of this length, new technologies emerge and old ones change.**



www.dpi.solutions

Context

5. Some applications exist in multiple versions. – Each version offers a different set of functionality to its users, or differs from other versions in specific aspects, such as its user interface. – **Despite their differences, however, all versions of the application should be based on a common architecture and functional core.**
6. The goal is to **avoid architectural drift** between the versions of the application and to **minimize development and maintenance effort for shared functionality**.
7. It should be easy to provide a particular application version with different user interfaces, and also to run the version on different platforms.



www.dpi.solutions

“Features”

1. The application platform must cope with continuous hardware and software evolution.
2. The application platform **should be portable, extensible and adaptable to allow easy integration of emerging technologies**.
3. The success of such application platforms may also depend on their capability to run applications written for existing standards.
4. To **support a broad range of applications**, there is a need for more than one view of the functionality of the underlying application platform.
5. Ideally, an application platform (such as an operating system or a database) should also be able to emulate other application platforms that **belong to the same application domain.**

www.dpi.solutions

Solution

1. Compose different versions of the application by extending a common but minimal core via a “plug-and-play” infrastructure.
2. The Microkernel pattern defines five kinds of participating components:

- Microkernel
- Internal servers
- External servers
- Adapters
- Clients

Solution

3. The microkernel implements the **fundamental functionality shared** by all application versions and provides the **infrastructure for integrating** version **specific functionality**.
4. Internal servers implement version specific core functionality, and external servers version specific user interfaces or APIs.
5. A specific application version is **configured by connecting the corresponding internal servers with the microkernel**, and providing **appropriate external servers to access its functionality**.
6. Consequently, all versions of the application **share a common functional and infrastructural core**, but provide a tailored function set and look-and-feel.

Solution

7. **Clients**, whether human or other software systems, access the microkernel's functionality solely **via the interfaces or APIs** provided **by the external servers**, which **forward** all requests they receive **to the microkernel**.
8. If the microkernel implements the requested function itself, it executes the function, otherwise it routes the request to the corresponding internal server.
9. Results are returned accordingly so that the external servers can display or deliver them to the client.

www.dpi.solutions

Solution

10. A problem arises if a client needs to access the interfaces of its external server directly.
- *– Every communication with an external server must be hardcoded into the client code.*
- *– Such a tight coupling between clients and servers does not support changeability very well.*

10. We therefore introduce interfaces between clients and their external servers **to protect clients from direct dependencies**.
11. **Adapters represent these interfaces between clients and their external servers**, and **allow clients to access the services of their external server in a portable way**..

www.dpi.solutions

Tech details

1. The internal structure of the microkernel is typically **based on Layers**.

- – The bottommost **layer abstracts from the underlying system** platform, thereby **supporting the portability of all higher levels**.
- – The second layer **implements infrastructure functionality**, such as resource management, on which the microkernel depends.
- – **The layer above hosts the domain functionality** that is shared by all application versions.
- – The topmost layer includes the **mechanisms for configuring internal servers with the microkernel**, as well as for routing requests from external servers to their intended recipient.

www.dpi.solutions

Tech details

2. The **microkernel should be kept as small a s possible** to reduce memory requirements and provide mechanisms that execute quickly.

- – **Additional** and more complex services are therefore **implemented by internal servers that the microkernel activates or loads only when necessary**.

www.dpi.solutions

Internal and External Server

1. Internal servers can be considered extensions of the microkernel and are only accessible by the microkernel component.
2. Internal servers may be implemented as separate processes or as shared libraries:

- – Graphics card drivers are developed as shared libraries because they only act on behalf of clients.
- – In contrast, page fault handlers are separate processes. They always have to remain in main memory and cannot be swapped to external storage.

3. Internal servers follow a similar Layers design as the microkernel, but do not usually provide a routing layer.
4. Each external server is implemented as a separate process that provides its own service interface. The internal architecture of an external server depends on the policies it comprises.

www.dpi.solutions

Benefits

1. **Portability**. A Microkernel system offers a high degree of portability, for two reasons:

- – In most cases you do not need to port external servers or client applications if you port the Microkernel system to a new software or hardware environment.
- – Migrating the microkernel to a new hardware environment only requires modifications to the hardware-dependent parts of the microkernel and does not affect external servers.

2. **Flexibility** and **Extensibility**.

- – If you need to implement an additional view, all you need to do is add a new external server.
- – Extending the system with additional capabilities only requires the addition or extension of internal servers

www.dpi.solutions

Benefits

3. **Separation of policy and mechanism**.

- – The **microkernel** component **provides all the mechanisms necessary** to enable **external servers to implement their policies**.
- – **This strict separation of policies and mechanisms increases the maintainability and changeability of the whole system**
- – It also allows you to add new external servers that implement their own specialized views. If the microkernel component were to implement policies, this would unnecessarily limit the views that could be implemented by external servers.

www.dpi.solutions

Benefits

4. **Scalability**.

- – A distributed Microkernel system is applicable to the development of operating systems or database systems for computer networks, or multiprocessors with local memory.
- – If your Microkernel system works on a network of machines, it is easy to scale the Microkernel system to the new configuration when you add a new machine to the network.

3. **Transparency**.

- – In a distributed system components can be distributed over a network of machines.
- – In such a configuration, the Microkernel architecture allows each component to access other components without needing to know their location. All details of inter-process communication with servers are hidden from clients by the adapters and the microkernel.

www.dpi.solutions

Limitations \ drawbacks

1. **Performance**.

- – A monolithic software system designed to offer a specific view will generally have better performance than a Microkernel system supporting different views.
- – **We therefore have to pay a price for flexibility and extensibility**.
- – If the **communication within the Microkernel system is optimized for performance**, however, this price can be small

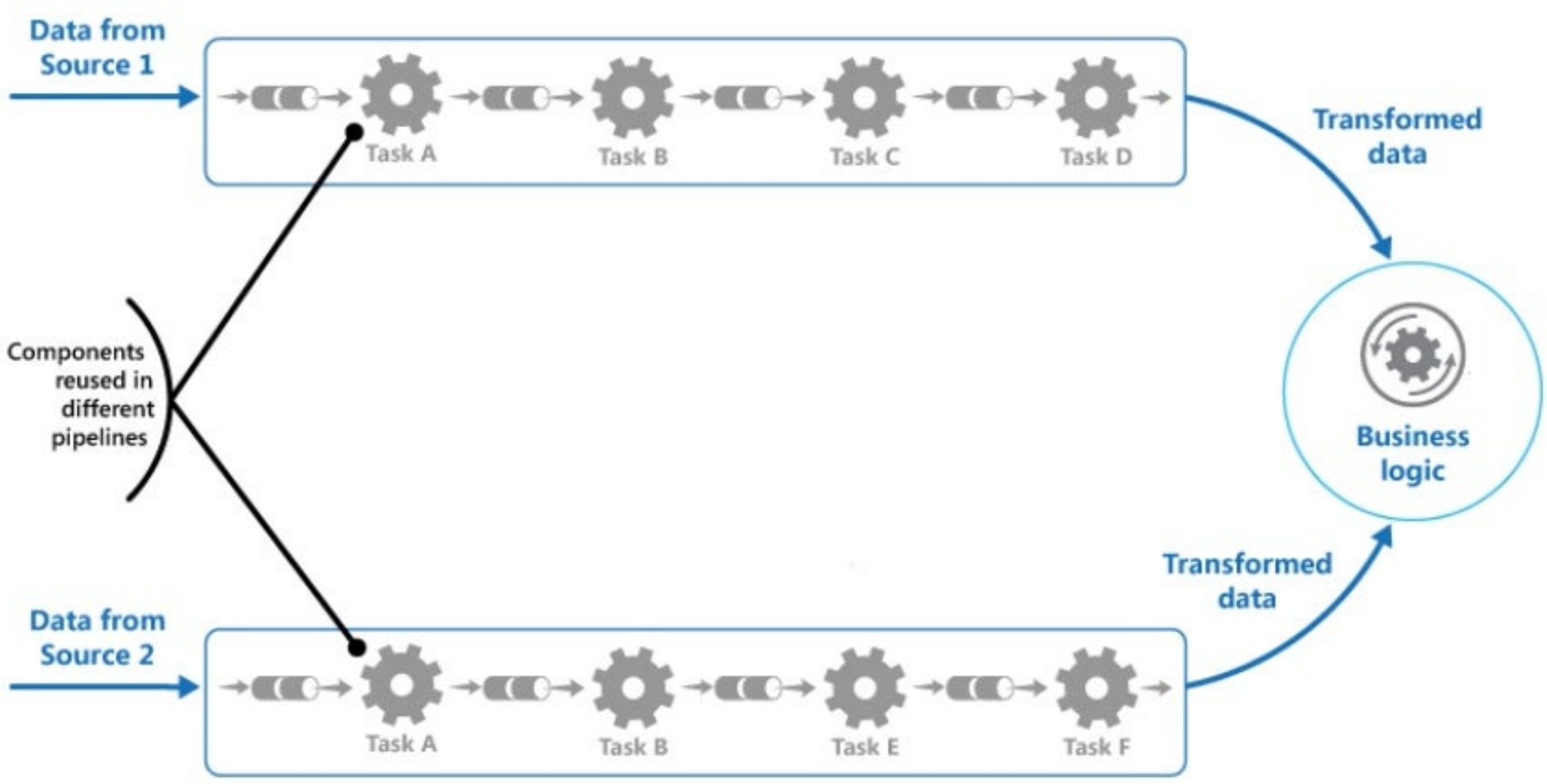
Limitations \ drawbacks

2. **Complexity of design and implementation**.

- – Developing a Microkernel-based system is a non-trivial task.
- – For example, it can sometimes be **very difficult to analyze or predict the basic mechanisms a microkernel component** must provide.
- – In addition, the separation between mechanisms and policies **requires in-depth domain knowledge** and **considerable effort during requirements analysis and design**

www.dpi.solutions

Pipes and Filters



www.dpi.solutions

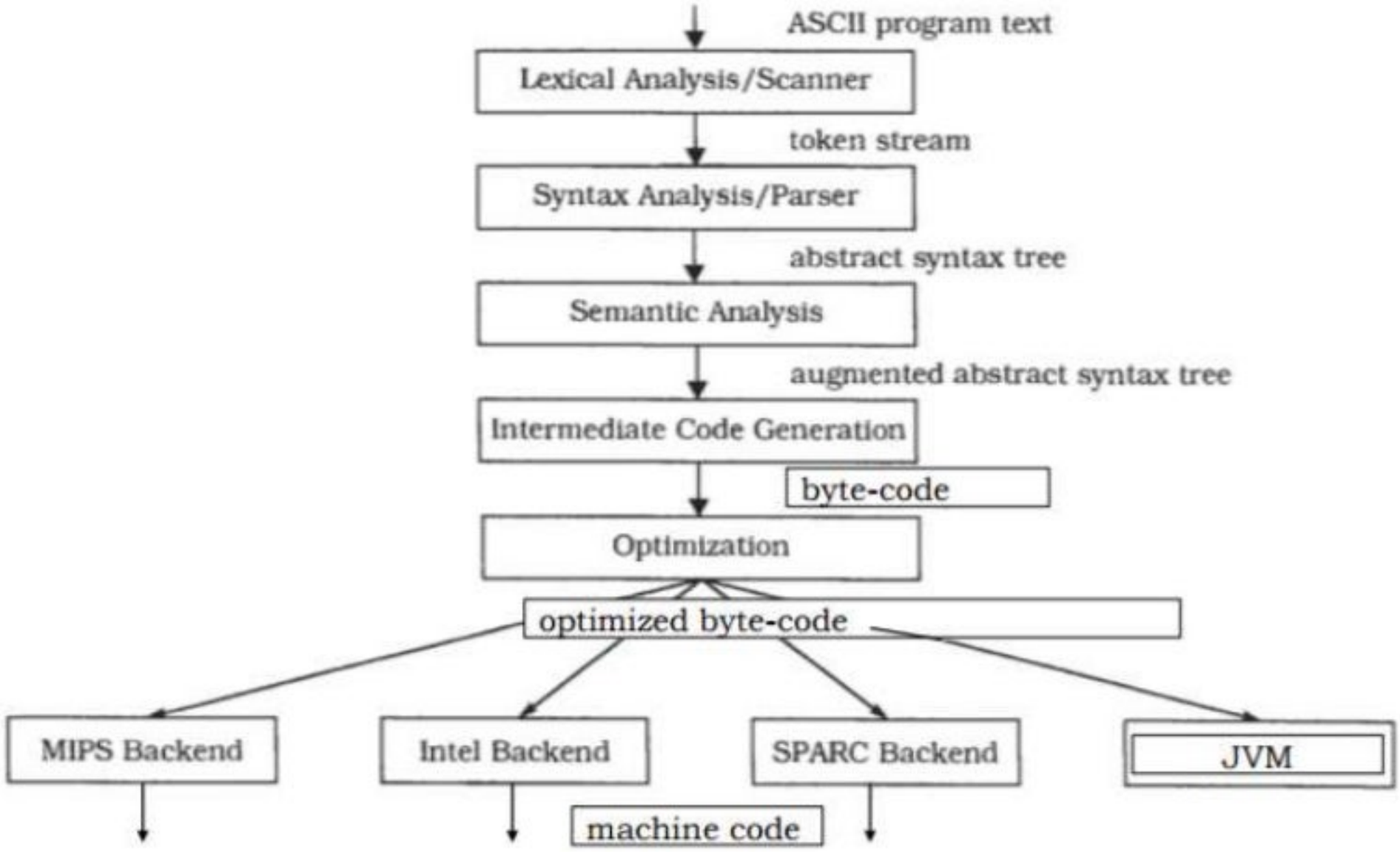
Introduction

1. The Pipes and Filters architectural pattern provides a **structure for systems that process a stream of data**.
2. Each **processing step is encapsulated** in **a filter component**.
3. Data is passed through pipes between adjacent filters.
4. **Recombining filters allows you to build families of related systems**.

Example

1. Suppose we want to develop a new compiler for the Java ☺ programming language.
2. We want the compiler to produce either optimized byte-code to run on a JVM, or optimized machine code to run natively on various platforms
- – Similar to the GNU GCJ compiler.
3. A backend will translate byte-code into the machine instructions of a specific processor for best performance.
4. Conceptually, translation from Java to byte-code consists of the phases lexical analysis, syntax analysis, semantic analysis, byte-code generation, byte-code optimization, and optionally machine-code generation.
5. Each stage has well-defined input and output data.

Example



www.dpi.solutions

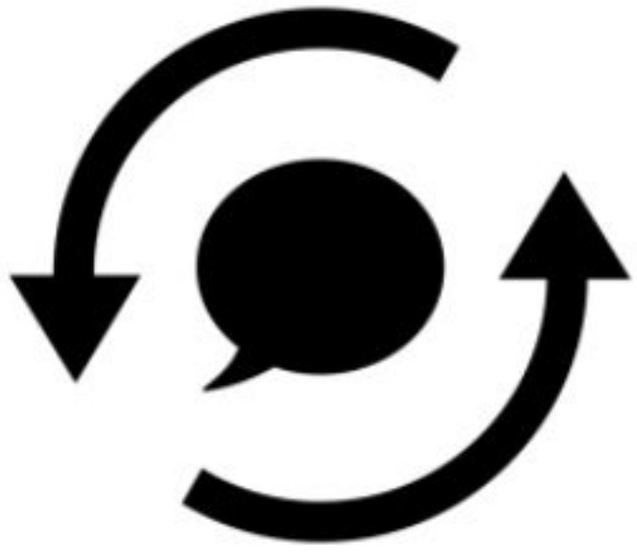
Challange

1. How to provide a design that is suitable for processing data streams.

www.dpi.solutions

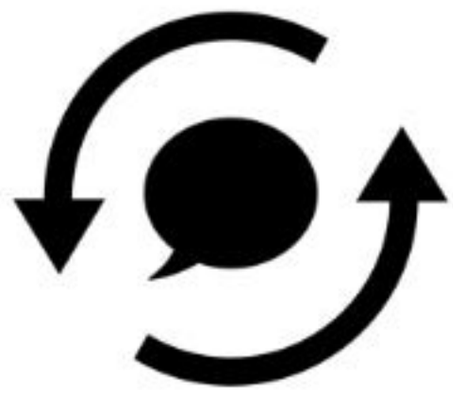
Context

1. Some applications process streams of data.
- – Input data streams are transformed stepwise into output data streams.
2. However, using common and familiar request/ response semantics for structuring such types of application is typically impractical.
3. Instead we must specify an appropriate data flow model for them.



Context

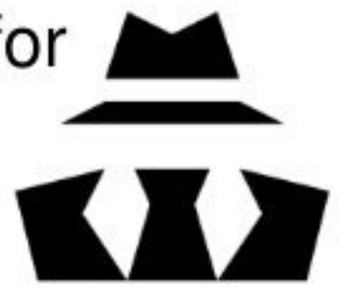
4. Modeling a data-flow-driven application raises some non-trivial developmental and operational challenges.
5. First, the parts of the application should correspond to discrete and distinguishable actions on the data flow.
6. Second, some usage scenarios require explicit access to intermediate yet meaningful results.
7. Third, the chosen data flow model should allow applications to read, process, and write data streams incrementally rather than wholesale and sequentially so that throughput is maximized.
8. Lastly, long-duration processing activities must not become a performance bottleneck.



www.dpi.solutions

“Features”

1. Future system enhancements should be possible by exchanging processing steps or by recombination of steps, even by users.
2. Small processing steps are easier to reuse in different contexts than large components.
3. Non-adjacent processing steps do not share information.
4. Different sources of input data exist, such as a network connection or a hardware sensor providing temperature readings, for example.
5. It should be possible to present or store final results in various ways.
6. Explicit storage of intermediate results for further processing in files clutters directories and is error-prone, if done by users.
7. You may not want to rule out multi-processing the steps, for example running them in parallel or quasi-parallel.



www.dpi.solutions

Solution

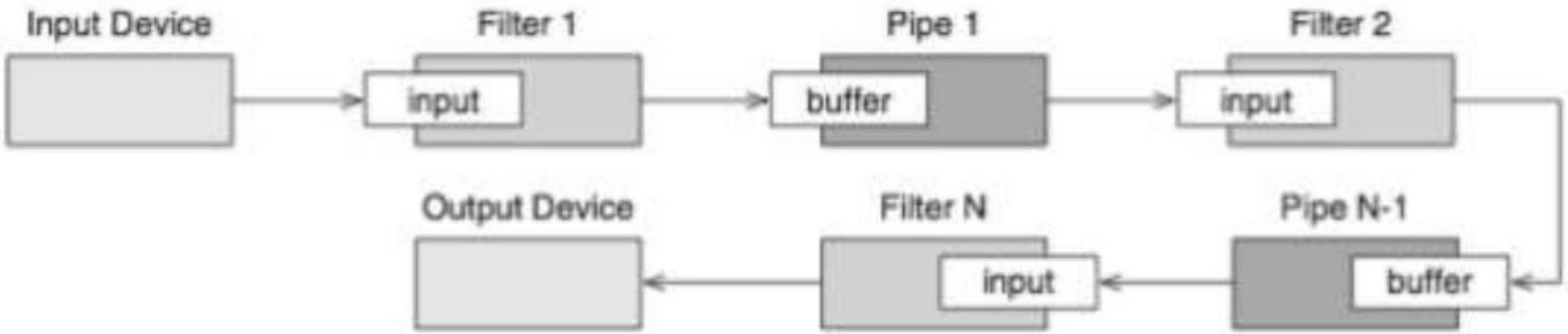
1. The Pipes and Filters architectural pattern divides the task of a system into several sequential processing steps.
2. These steps are connected by the data flow through the system. The output data of a step is the input to the subsequent step.
3. Each processing step is implemented by a filter component.
4. A filter consumes and delivers data incrementally, in contrast to consuming all its input before producing any output, to achieve low latency and enable real parallel processing

Solution

5. The input to the system is provided by a data source such as a text file. The output flows into a data sink such as a file, terminal, animation program and so on.
6. The data source, the filters and the data sink are connected sequentially by pipes that buffer data.
7. Each pipe implements the data flow between adjacent processing steps.
8. The sequence of filters combined by pipes is called a processing pipeline

www.dpi.solutions

Solution



Solution

9. In a single-process arrangement, pipes are typically implemented as queues.
10. In a distributed arrangement, pipes are realized as some form of messaging infrastructure that passes data streams between remote filters.
11. If a filter performs a long-duration activity, consider integrating multiple parallel instances of the filter into the processing chain.

- Such a configuration can further increase system performance and throughput, as some filter instances can start processing new data streams while others are processing previous data streams.

www.dpi.solutions

Filter Tech details

1. Filter components are the processing units of the pipeline.
2. A filter enriches data by computing and adding information, refines data by concentrating or extracting information, and transforms data by delivering the data in some other representation. A concrete filter implementation may combine any of these three basic principles.
3. An active filter starts processing on its own as a separate program or thread. Most commonly, the filter is active in a loop, pulling its input from and pushing its output down the pipeline.
4. A passive filter component is activated by being called either as a function (pull) or as a procedure (push).

www.dpi.solutions

Pipe Tech details

1. Pipes denote the connections between filters, between the data source and the first filter, and between the last filter and the data sink.
2. If two active components are joined, the pipe synchronizes them. This synchronization is done with a first-in-first-out buffer (queue).
3. If activity is controlled by one of the adjacent filters, the pipe can be implemented by a direct call from one filter to the next. Direct calls make filter recombination harder, however.

Options

1. The single-input single-output filter specification of the Pipes and Filters pattern can be varied to allow filters with more than one input and/or more than one output.
2. Processing can then be set up a s a directed graph that can even contain feedback loops.

- – The design of such a system, especially one with feedback loops, requires a solid foundation to explain and understand the complete calculation.
- – A rigorous theoretical analysis and specification using formal methods are appropriate, to prove that the system terminates and produces the desired result.

3. If we restrict ourselves to simple directed acyclic graphs, however, it is still possible to build useful systems.

Benefits

1. A Pipes and Filters architecture **decouples** different data processing steps so that they can evolve independently of one another and support an incremental data processing approach.
2. Pipes and Filters provides for **flexibility** by exchanging or reordering the processing steps.
3. **Recombining** filters allows you to build families of related systems using existing processing components.

www.dpi.solutions

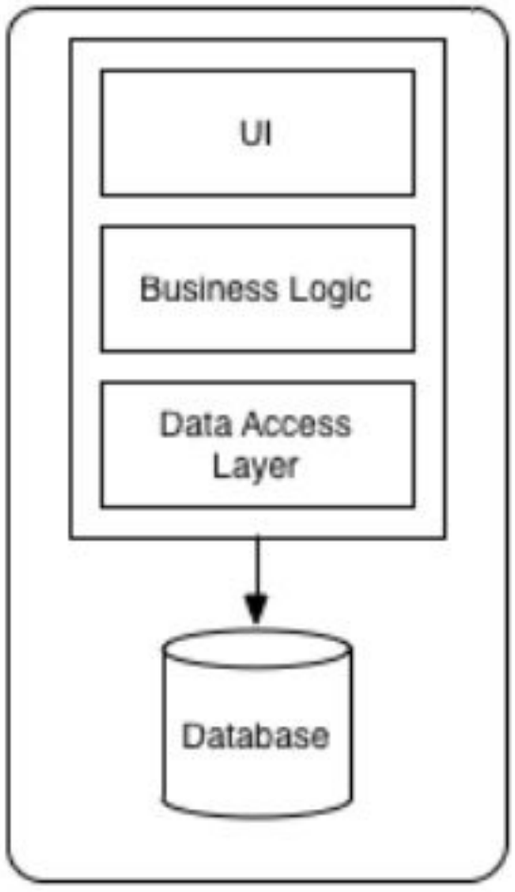
Benefits

4. **Applicability**

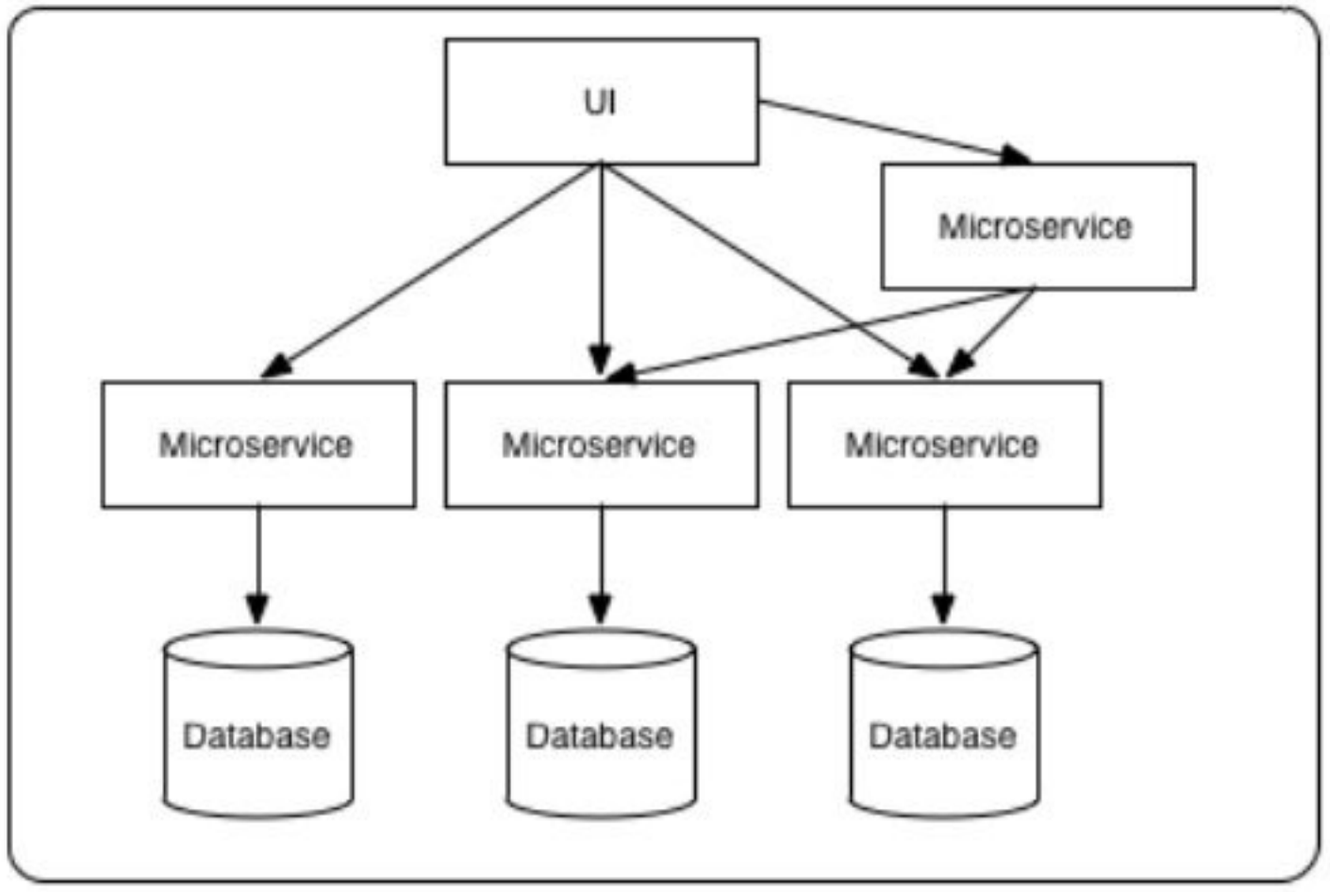
- Whether a separation into processing steps is feasible strongly depends on the application domain and the problem to be solved. For example, an interactive, event-driven system does not split into sequential stages.

www.dpi.solutions

Microservices Architecture



Monolithic Architecture



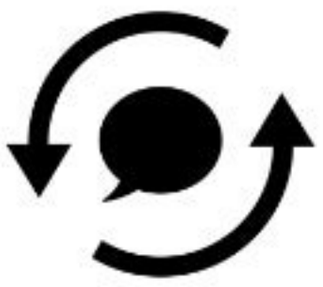
Microservices Architecture

www.dpi.solutions

Context

Just a popular context

1. Server-side enterprise application.
2. A variety of different clients support
   - Desktop browsers
   - Mobile browsers
   - Native mobile applications
3. Expose an API for 3rd parties to consume
4. Integrate with other applications via either web services or a message broker
5. Handles requests (HTTP requests and messages) by executing business logic; accessing a database; exchanging messages with other systems; and returning a HTML/JSON/XML response.
6. There are logical components corresponding to different functional areas of the application.



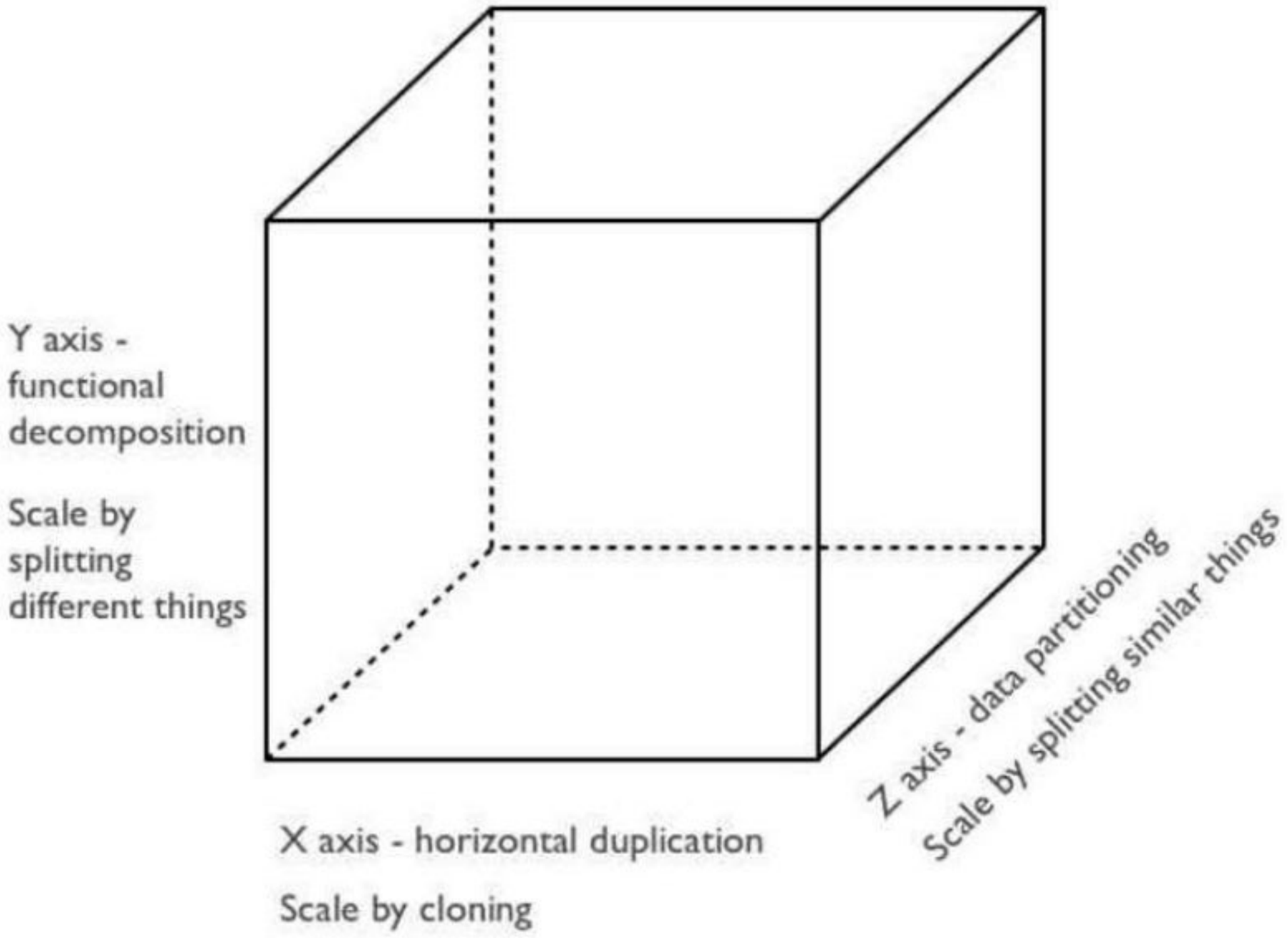
www.dpi.solutions

“Features”

1. There is a team of developers working on the application
2. New team members must quickly become productive
3. The application must be easy to understand and modify
4. You want to practice continuous deployment of the application
5. You must run multiple copies of the application on multiple machines in order to satisfy scalability and availability requirements
6. You want to take advantage of emerging technologies (frameworks, programming languages, etc)

www.dpi.solutions

The Scale Cube



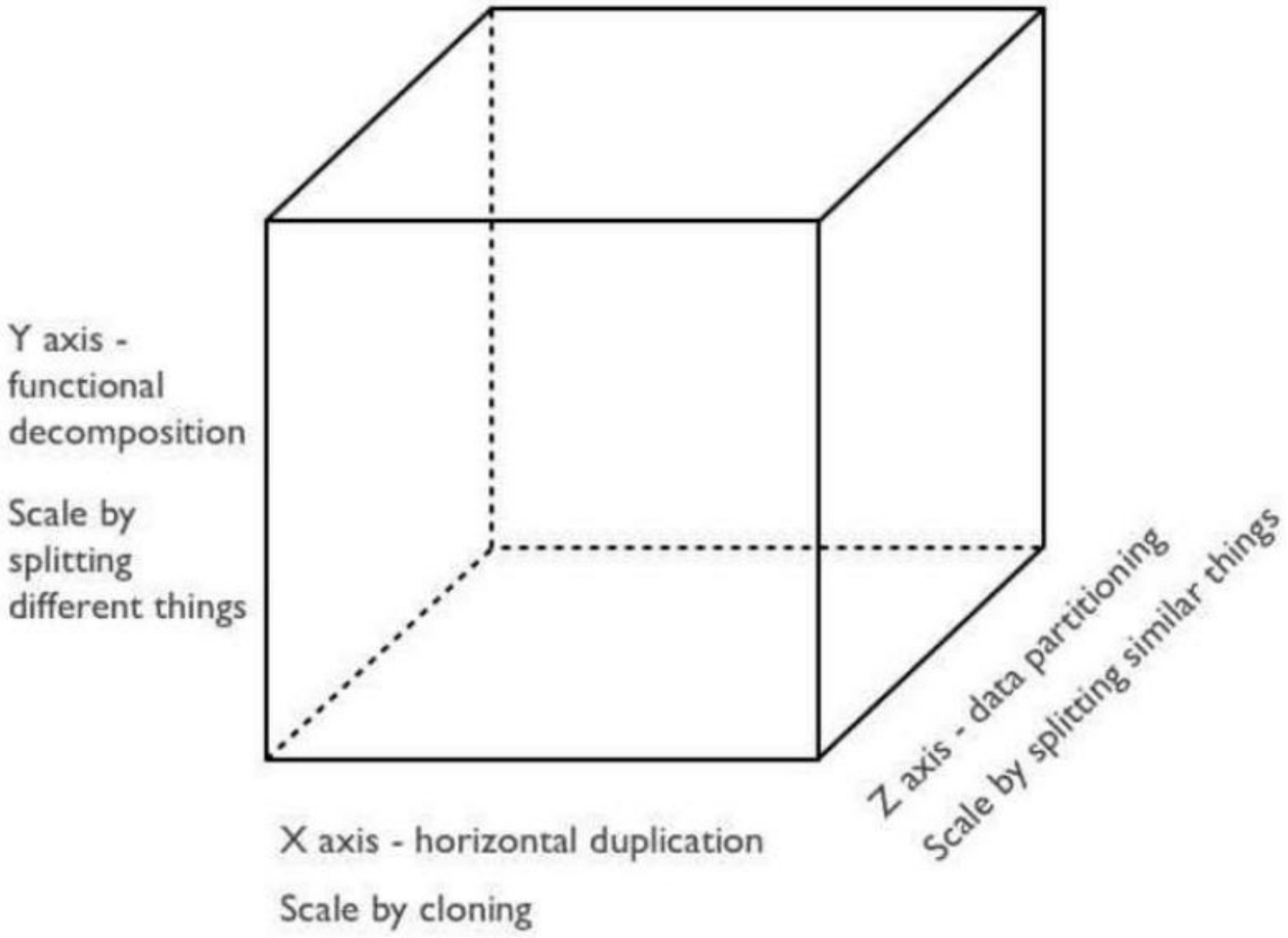
www.dpi.solutions

Solution

1. Define an architecture that structures the application as a set of loosely coupled, collaborating services. **This approach corresponds to the Y-axis of the Scale Cube. Each service implements a set of narrowly, related functions.** For example, an application might consist of services such as the order management service, the customer management service etc.
2. Services communicate using either synchronous protocols such as HTTP/REST or asynchronous protocols such as AMQP. **Services can be developed and deployed independently of one another. Each service has its own database in order to be decoupled from other services. Data consistency between services is maintained using an event-driven architecture**

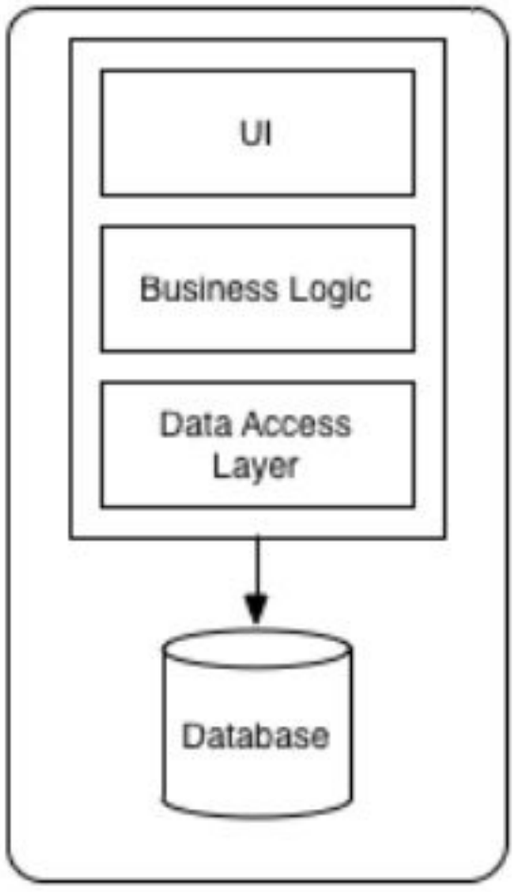
www.dpi.solutions

The Scale Cube

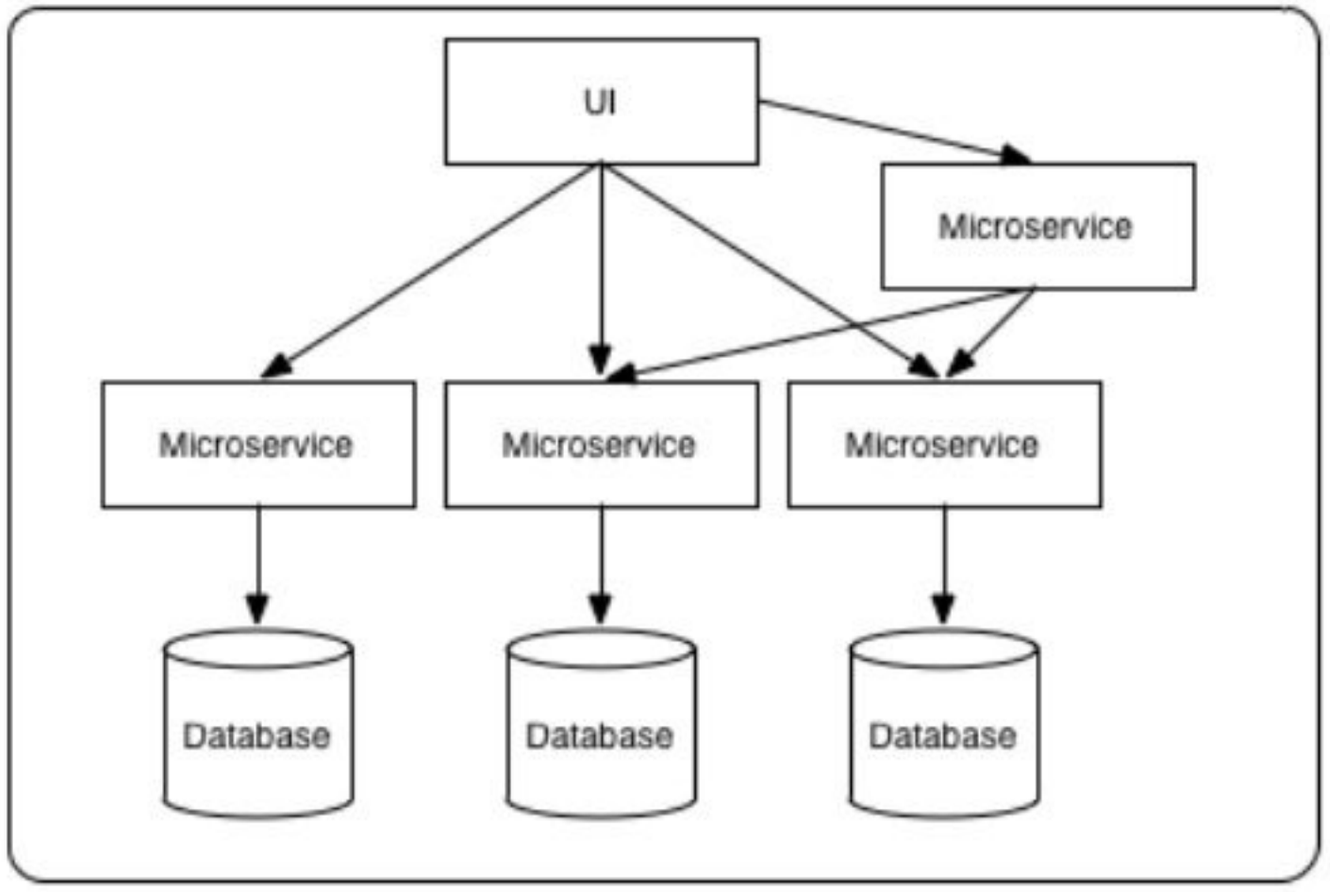


www.dpi.solutions

Microservices Architecture



Monolithic Architecture



Microservices Architecture

www.dpi.solutions

Summary

1. Everything new - is totally forgotten old things
2. Study, study and study once again
3. Read books
4. Think about the idea, and do not drown in tons of terminology and positioning slogans
5. Microkernel, Pipes and Filters, Microservicies – great Architectural options

www.dpi.solutions

Examples

www.dpi.solutions

UNIX philosophy by Malcolm

1. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features".
2. Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.
3. Design and build software, even operating systems, **to be tried early, ideally within weeks**. **Don't hesitate to throw away the clumsy parts and rebuild them**.
4. **Use tools in preference to unskilled help to lighten a programming task**, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them.

www.dpi.solutions

DSL based solutions

1. Health insurance for USA and Canada
2. The world leader of Hardware production

- Trains and wagons “management”
- Calculation, optimization, rule based option selection for electrical networks
- *Note: super- general cores and DSL configurations*

3. Paper’s advertisement for USA

www.dpi.solutions

DSL based - video

1. Analyst Days 2015. Антон Наумович. Роль бизнес-аналитика в разработке собственной Business Rule Engine с нуля как основной платформы проекта
2. Юрий Русович. Особенности системного анализа особо крупных проектов построенных на базе Business Rule Engine «Drools»
3. Александр Василенок. XText – Business Rule Engine в контексте бизнес-анализа
4. Антон Семенченко. BDD or DSL как способ построения коммуникации на проекте
5. Антон Наумович. Разработка своей agile методологии для управления крупными проектами

www.dpi.solutions

Let’s vote ??? ☺

1. Microkernel?
2. Microservices?
3. Pipes and Filters ..?

www.dpi.solutions

Example

EPAM Report Portal

- Transition from Monolithic solution to Microservicies architecture

www.dpi.solutions

Architecture – Methodology

1. Антон Наумович. Разработка своей agile методологии для управления крупными проектами
2. When it's worth moving from Agile to Waterfall

www.dpi.solutions

Quality assurance

www.dpi.solutions

Examples

1. Unix philosophy
2. Health insurance for USA and Canada
3. The world leader of Hardware production

- Trains and wagons “management”
- Calculation, optimization, rule based option selection for electrical networks

3. Paper’s advertisement for USA
4. EPAM Report Portal

www.dpi.solutions

What's next

www.dpi.solutions

Books

www.dpi.solutions

Examples

- «**Pattern-Oriented Software Architecture**» Volume 1-3

Notes: I highly recommend to read it from start to the very end.

- «**Domain Specific Languages**» Martin Fowler

Notes: I highly recommend to read it from start to the very end.

- «**Patterns of Enterprise Application Architecture**» Martin Fowler

Notes: I highly recommend to read it from start to the very end.

- «**The Pragmatic Programmer: From Journeyman to Maste**r» D.Thomas, A. Hunt

Notes: A great book, consisting a whole diversity of atomic advise. IMHO required to be read from the beginning to the end twice and then look through various chapters when preparing to the interview or discussion with the customer

www.dpi.solutions

Engineers from 70-s

www.dpi.solutions

Dennis MacAlistair Ritchie

1. C
2. Unix
3. Dozens of brilliant inventions

www.dpi.solutions

Malcolm Douglas McIlroy

1. Unix pipelines
2. Software componentry
3. Dozens of Unix utilities

www.dpi.solutions

Niklaus Emil Wirth

1. Pascal
2. Component-based software engineering
3. Dozens of brilliant inventions

www.dpi.solutions

Bertrand Meyer

1. Component-based unification
2. Dozens of brilliant inventions
3. Innopolis University lecturer (Russia)

www.dpi.solutions

Thanks! Questions?

Anton Semenchenko

skype: dpi.semenchenko

semenchenko@dpi.solutions