TÉCNICO LISBOA

Enterprise Integration

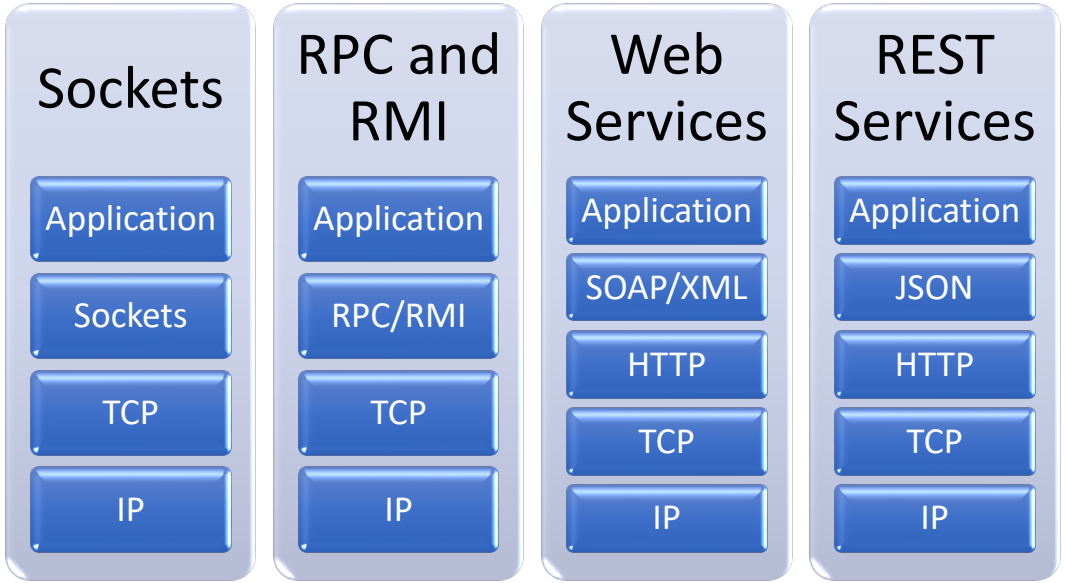
(Micro)Services

Prof. Sérgio Guerreiro
Sergio.guerreiro@tecnico.ulisboa.pt

Department of Computer Science and Engineering
Instituto Superior Técnico / Universidade de Lisboa
INESC-ID
URL: http://www.inesc-id.pt
Rua Alves Redol, 9
1000-029 Lisboa
Portugal

Arriving to Technological Services

- Evolution of protocol stacks for Request-Response integration
- Services can be consumed in a request-reply pattern, usually blocking the client
- Establishes the separation between a client application and the logic layer exposed as a set of services



Service Oriented Architecture (SOA)

The classical approach towards services:

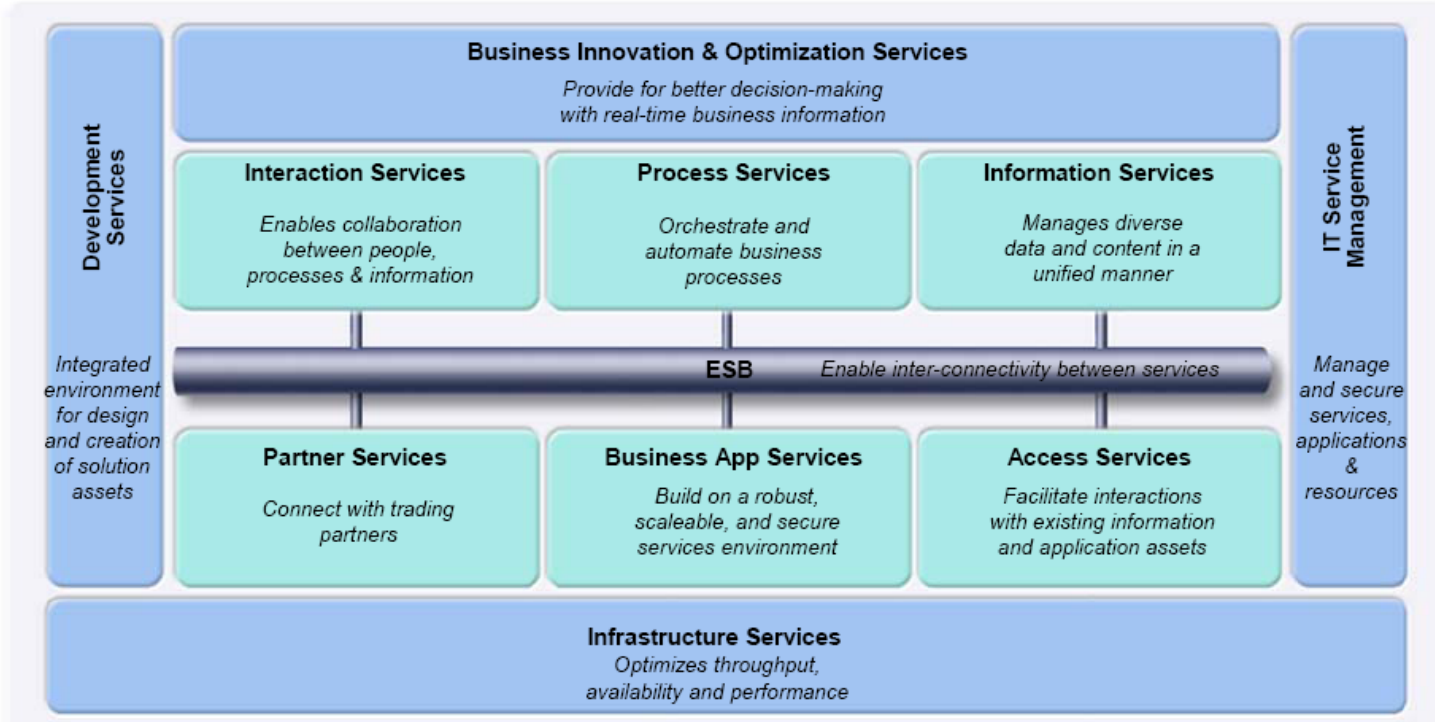
- *What?* is a design approach where **multiple services collaborate** to provide some end set of capabilities. A service here typically means a completely separate operating system process. Communication between these services occurs via **calls across a network** rather than method calls within a process boundary.

- *Why?* SOA emerged as an approach to combat the challenges of the large monolithic applications. It is an approach that aims to promote the reusability of software; two or more end-user applications, for example, could both use the same services.

- *Goal to promote reusability:* It aims to make it **easier to maintain or rewrite software**, as theoretically we can replace one service with another without anyone knowing, as long as the semantics of the service don't change too much.

Newman, S. (2021). Building microservices. " O'Reilly Media, Inc.".

Enterprise Application Integration (EAI)



fonte: **IBM's SOA Foundation,** *An Architectural Introduction and Overview*

SOA specifications

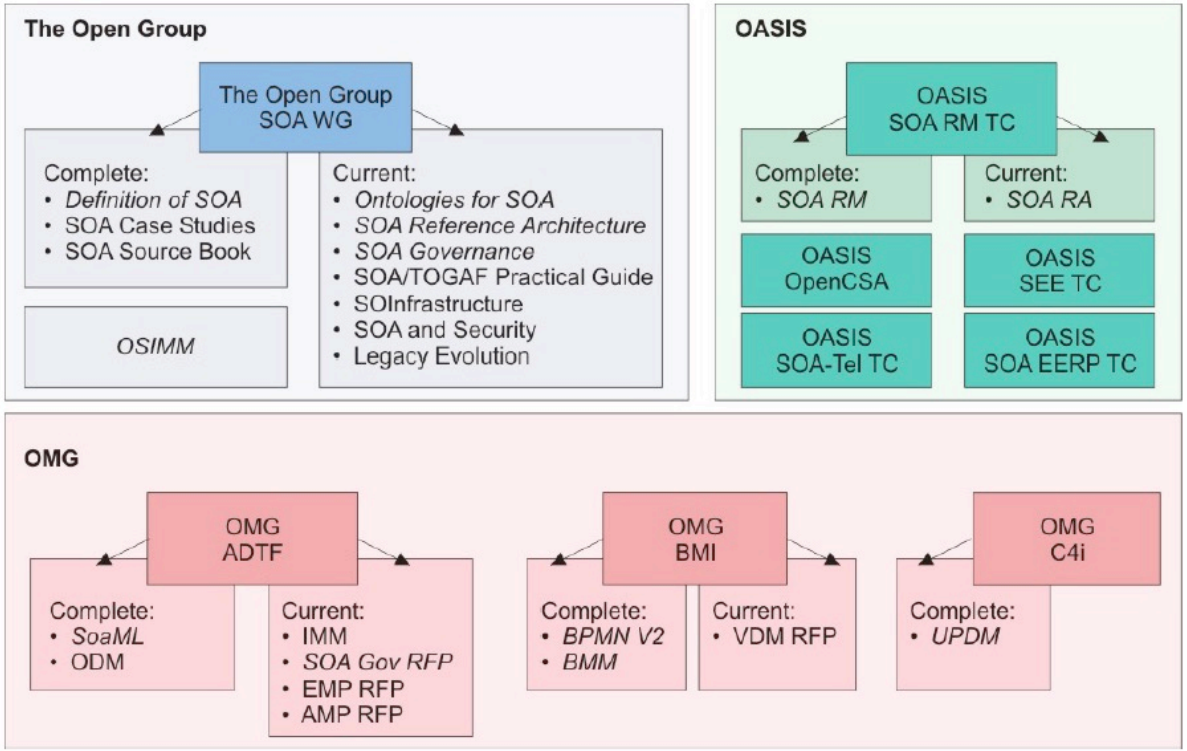


Figure 1: Specifications of SOA Open Standards Working/Work Groups, Technical Committees, and Special Interest Groups

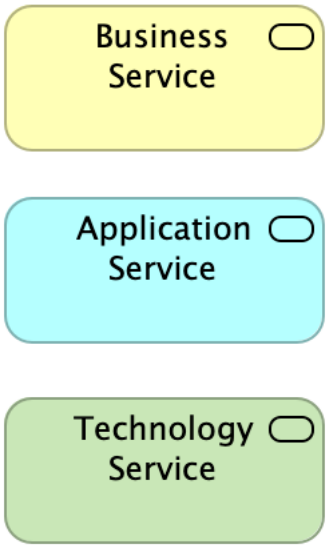
Some SOA pitfalls

- Communication protocols incompatibilities (*e.g.*, SOAP)
- Lack of guidance about service granularity
- Wrong guidance on picking places to split a system
- Many specifications

Newman, S. (2021). Building microservices. " O'Reilly Media, Inc.".

Back to ArchiMate definitions

- A **business service** represents explicitly defined behavior that a business role, business actor, or business collaboration exposes to its environment.
- An **application service** represents an explicitly defined exposed application behavior.
- A **technology service** represents an explicitly defined exposed technology behavior.



https://pubs.opengroup.org/architecture/archimate32-doc/

An application service provides a usage contract composed of:

- Explanation of the functionality provided
- The location where the API can be accessed, e.g., HTTP URL to specify the location
- The input and output parameters for the API, such as parameter names, message format, and data types
- The service-level agreement (SLA) that the API provider adheres to such as response time, throughput, availability
- The technical requirements about the rate limits that control the number of requests that an app or user can make within a given period
- Any legal or business constraints on using the API. This can include commercial licensing terms, branding requirements, fees and payments for use, and so on
- Documentation to aid the understanding of the API

Microservices, *def.1*

- A microservice is a tiny and independent software process that runs on its own deployment schedule and can be updated independently.

Davis, A. (2021). Bootstrapping Microservices with Docker, Kubernetes, and Terraform: A project-based guide. Simon and Schuster.

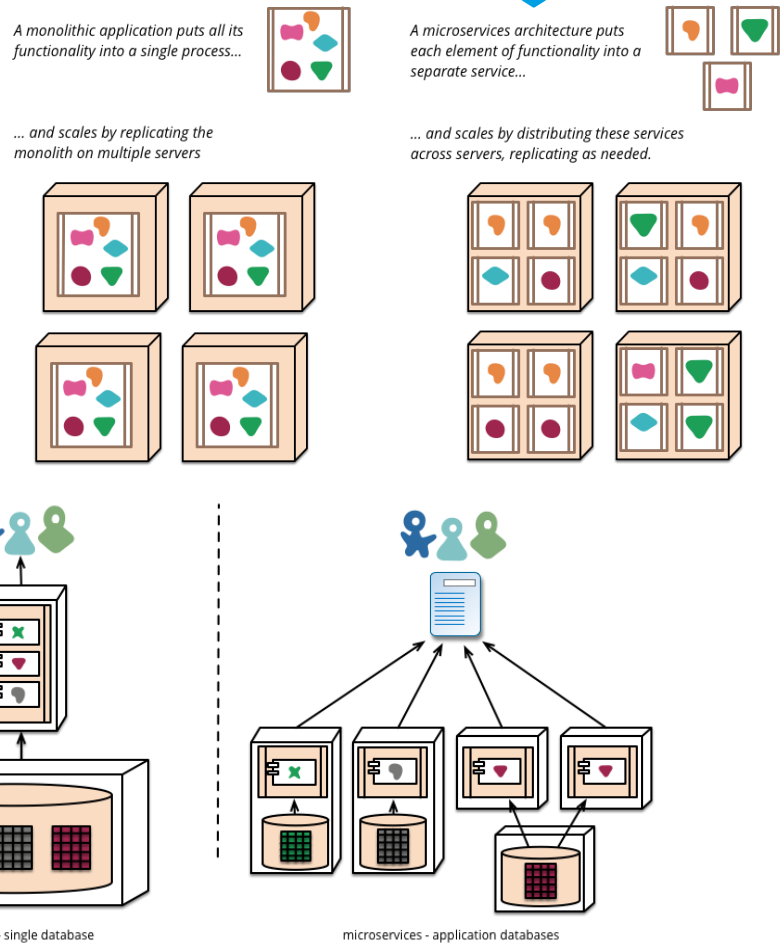
- A microservice is an independently deployable component of bounded scope that supports interoperability through message-based communication. Microservice architecture is a style of engineering highly automated, evolvable software systems made up of capability-aligned microservices.

Nadareishvili, I., Mitra, R., McLarty, M., & Amundsen, M. (2016). Microservice architecture: aligning principles, practices, and culture. " O'Reilly Media, Inc.".

Microservices, *def.2*

An approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms. […] built around business capabilities and independently deployable by fully automated deployment machinery

https://martinfowler.com/articles/microservices.html



Microservices, *def.3*

- Microservices are independently releasable services that are modelled around a business domain. A service encapsulates functionality and makes it accessible to other services via networks—you construct a more complex system from these building blocks. One microservice might represent inventory, another order management, and yet another shipping, but together they might constitute an entire ecommerce system. Microservices are an architecture choice that is focused on giving you many options for solving the problems you might face.

Newman, S. (2021). Building microservices. " O'Reilly Media, Inc.".

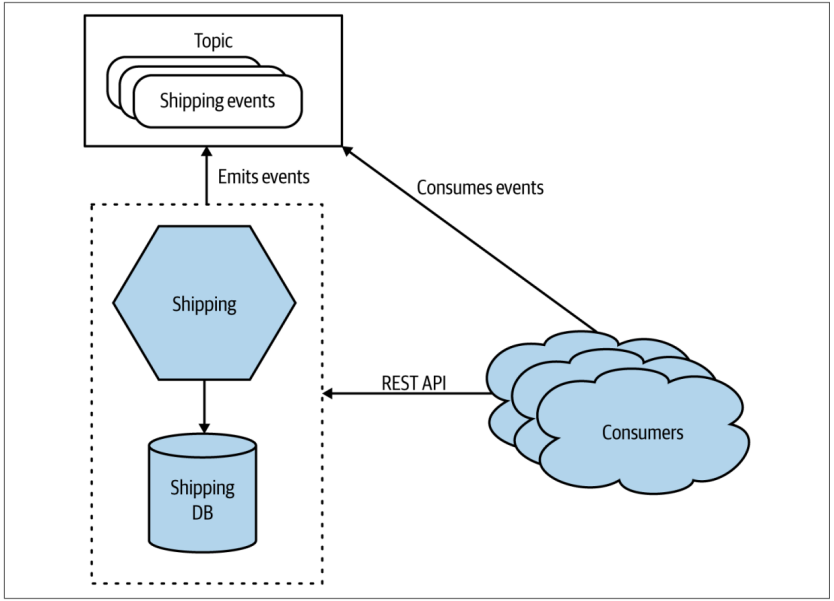


Figure 1-1. A microservice exposing its functionality over a REST API and a topic

Key concepts of microservices to retain

- Run independently
- Independent Deployability
- Distributed data management
  - Owning their own state
  - Bindings to other a bounded context
- Modeled around a Business Domain
  - Size is variable, could small or large. Depends on the business context
- Flexibility
- Alignment with the organization architecture
- Command Query Responsibility Segregation (CQRS)
  - Read/Write your own domain data
  - Read-only representation of other domains data
  - Private data representation (might be different format)

Drawbacks:

- Developers must deal with the additional complexity of creating a distributed system.
- Implementing use cases that span multiple services requires careful coordination between the teams
- Developers must implement the interservice communication mechanism

Newman, S. (2021). Building microservices. " O'Reilly Media, Inc."

Asynchronous **Event-driven** Microservices

Primary benefits:

- **Granularity** –services map neatly to bounded contexts and can be easily rewritten when business requirements change
- **Scalability** – individual services can be scaled up and down as needed
- **Technological flexibility** – services use the most appropriate languages and technologies. This also allows for easy prototyping using pioneering technology.
- **Business requirements flexibility** – ownership of granular microservices is easy to reorganize. There are fewer crossteam dependencies compard to large services, and the organization can react more quickly to changes in business requirements thaht would otherwise be impeded by barriers to data access
- **Loosely coupling** – event-driven microservices are coupled on domain data and not on specific implementation API. Data schemas can be used to greatly improve how data changes are managed.
- **Continuous delivery support** – it's easy to ship a small, modular microservice, and roll it back if needed
- **High testability** – microservices tend to have fewer dependencies than large monoliths, making it easier to mock out the required testing endpoints and ensure proper code coverage

Bellemare, A. (2020). *Building Event-Driven Microservices*. " O'Reilly Media, Inc.".

Types of Coupling

- **Implementation** coupling

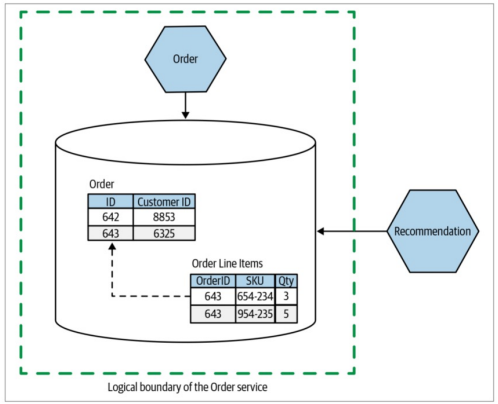


Figure 1-9. The Recommendation service directly accesses the data stored in the Order service

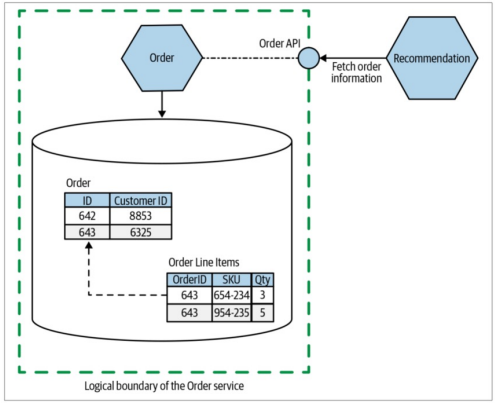


Figure 1-10. The Recommendation service now accesses order information via an API, hiding internal implementation detail

- **Temporal** coupling

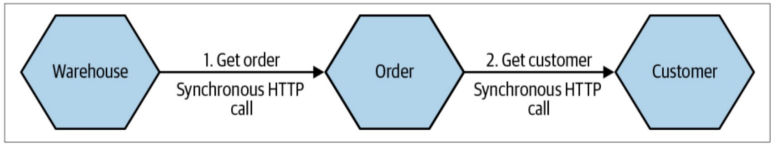


Figure 1-12. Three services making use of synchronous calls to perform an operation can be said to be temporally coupled

- **Deployment** coupling
- **Domain** coupling - the desired coupling for microservices

Newman, S. (2019). Monolith to microservices: evolutionary patterns to transform your monolith. O'Reilly Media.

TÉCNICO LISBOA

Enterprise Integration

The *Reactive systems*

Reactive Manifesto classifies reactive systems as being:

- **Responsive** - focus on providing rapid and consistent response times, establishing reliable upper bounds so they deliver a consistent quality of service.
- **Resilient** - The system stays responsive in the face of failure. This applies not only to highly-available, mission critical systems — any system that is not resilient will be unresponsive after a failure. Resilience is achieved by replication, containment, isolation and delegation. Failures are contained within each component, isolating components from each other and thereby ensuring that parts of the system can fail and recover without compromising the system as a whole
- **Elastic** - react to changes in the input rate by increasing or decreasing the resources allocated to service these inputs. This implies designs that have no contention points or central bottlenecks, resulting in the ability to shard or replicate components and distribute inputs among them
- **Message driven** - rely on asynchronous message-passing to establish a boundary between components that ensures loose coupling, isolation, location transparency, and provides the means to delegate errors as messages. Employing explicit message-passing enables load management, elasticity, and flow control by shaping and monitoring the message queues in the system and applying back-pressure when necessary

https://www.reactivemanifesto.org/pdf/the-reactive-manifesto-2.0.pdf

Reactive 101

- **Responsive** – able to handle requests in a timely fashion
- **Resilient** – able to manage failures gracefully
- **Elastic** – able to scale up and down according to the load and resources
- **Message driven** – using asynchronous message-based communication among the components forming the system

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

Reactive 101

- **Commands** are actions that a user wishes to perform. The action has not yet happened. It may happen in the future, or not; it may complete successfully or fail. In general, commands are sent to a specific receipient, and a result is sent back to the client
- **Events** are actions that have successfully completed. Represents a fact, something that happenned: a keystroke, a failure, an order...
- Events are **immutable**, you cannot change the past. To **refute** a previously sent fact, you need to **fire another event** invalidating the fact
- A **message** is a self-contained data structure describing the event and any relevant details about the event, such as who emitted it, at what time it was emittes, and potentially its unique ID

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

Overview of a reactive system

Commands and Messages are the basis of most of interactions. This pattern handles real-world asynchronicity, and binds together services without relying on strong coupling.

At the edge of the system, this approach uses commands most of the time, often relying on HTTP.

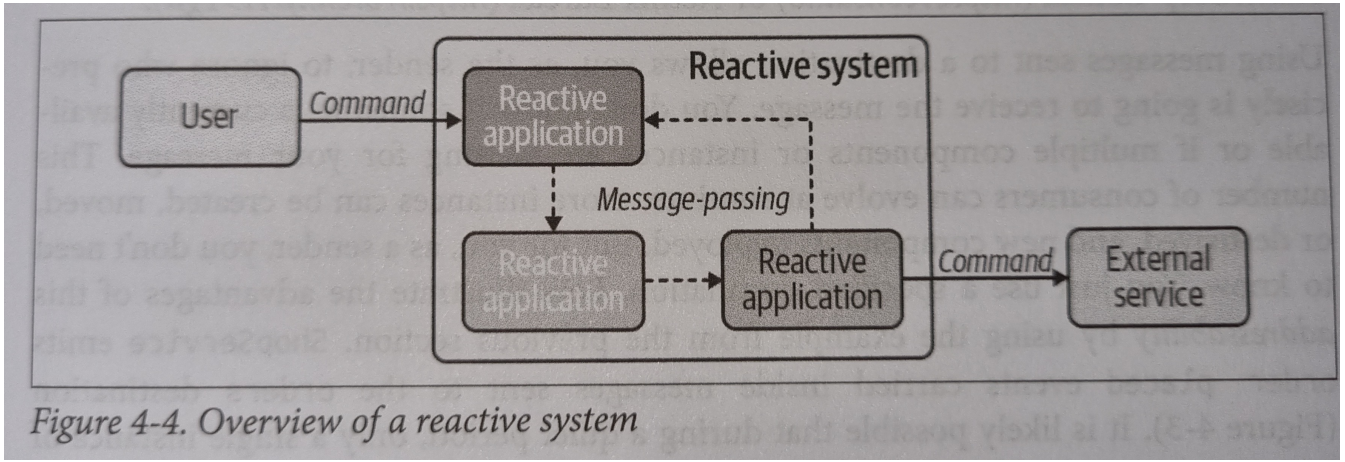


Figure 4-4. Overview of a reactive system

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

Overview of a reactive system: Elasticity

Scale-from-zero ability, starting `OrderService` instances to share the systems' load

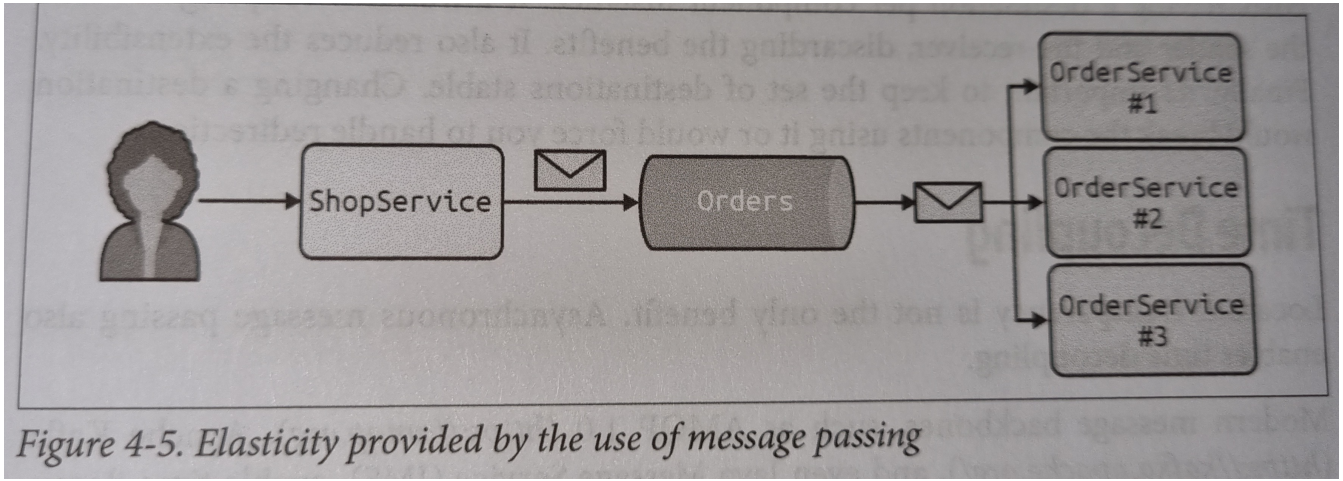


Figure 4-5. Elasticity provided by the use of message passing

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

Overview of a reactive system: Resilience

Message passing enables replication and avoids service disruption

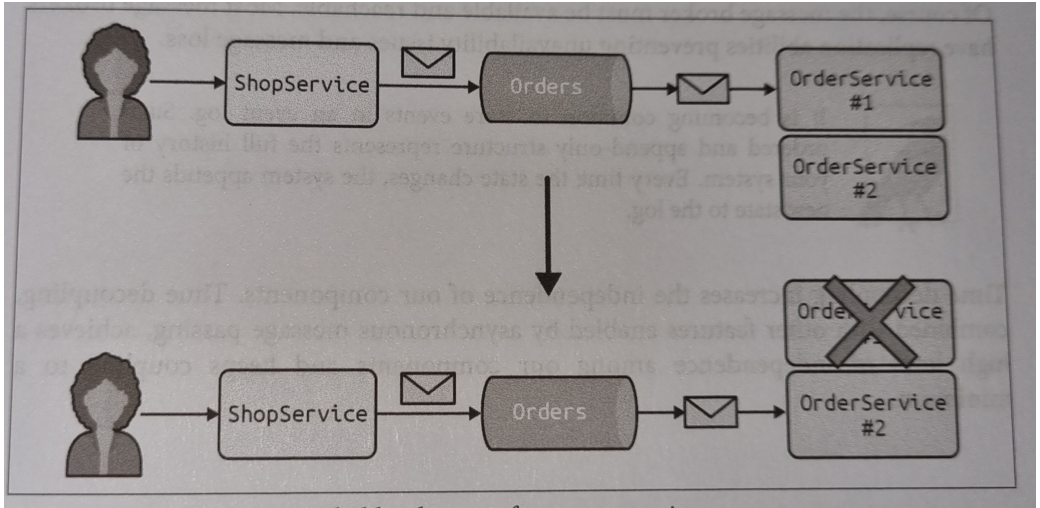


Figure 4-6. Resilience provided by the use of message passing

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

Overview of a reactive system: Time decoupling

- Asynchronous message passing also enables time decoupling.
- Events are not lost if there are no consumers. The events are stored and delivered later.
- Time decoupling increases the independence of components, and keeps coupling to a minimum

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

The role of **Nonblocking** Input/Output

- **Blocking network I/O** – synchronous communication where a client and the server connect before interaction starts. Communication is blocked until the operation completes.

- **Multithread blocking network I/O** – Execute concurrent requests having multiple threads. Resources expended waiting for the clients requests. Concurrency limited by the number of threads available.

- **Nonblocking network I/O** – the system enqueues I/O operations and returns immediately, so the caller is not blocked. When the response comes back, the system stores the result in a structure. When the caller needs the result, it interrogates the system to see whether the operation completed.
  - *Continuation-passing style (CPS)* – style of programming in which control is passed explicitly in the form of a continuation (*usually a callback*)

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

Nonblocking I/O

- Give the possibility to handle multiple concurrent requests or messages with a single thread
- The **reactor pattern** allows associating I/O events with event handlers
- Invokes the event handlers when the expected event is received
- Avoiding the creation of a thread for each message, request and connection

it's a thread iterating over the set of channels

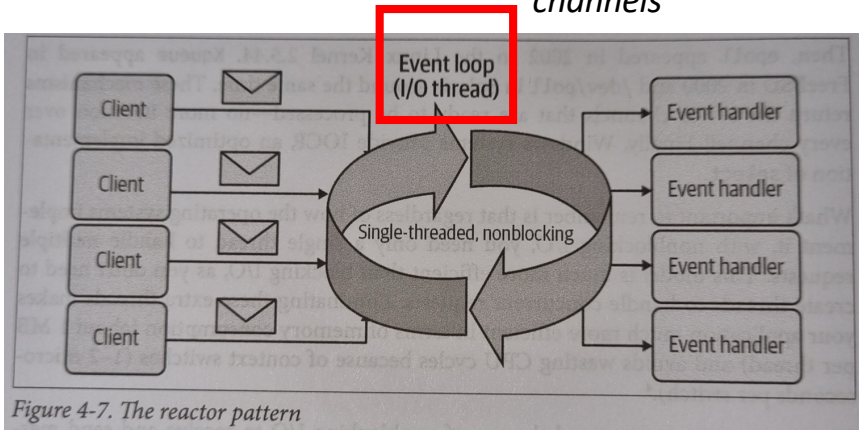


Figure 4-7. The reactor pattern

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

Nonblocking & Blocking I/O

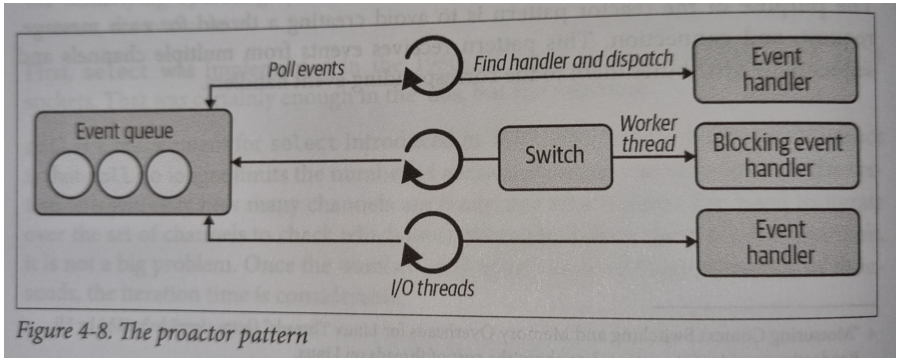


Figure 4-8. The proactor pattern

- The **Proactor pattern** can be seen as an asynchronous version of the reactor
- Useful when long-running event handlers invoke a continuation when they complete
- **Such mechanisms allow mixing nonblocking and blocking I/O**

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

The common architecture

- Bottom layer, handles client connections, outbound requests, and response writing.
- Middle layer, provides easier and high-level APIs such as HTTP requests, responses, Kafka messages
- Top layer, the code developed by you, that is just a collection of event handlers. Uses the features provided by the reactive framework to interact with other services or middleware.
  - **However, your code cannot block the event loop thread, otherwise the architeture will be blocked!**
  - **Design & develop non-blocking code, always!**

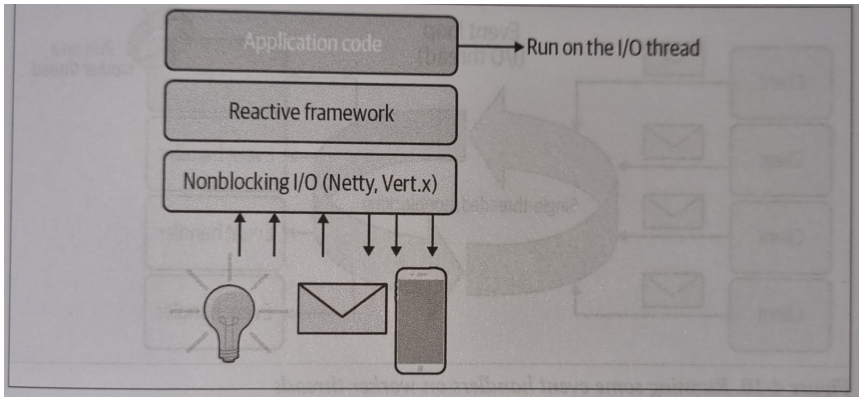


Figure 4-9. The common architecture of reactive frameworks

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

The concepts of the Reactive Landscape

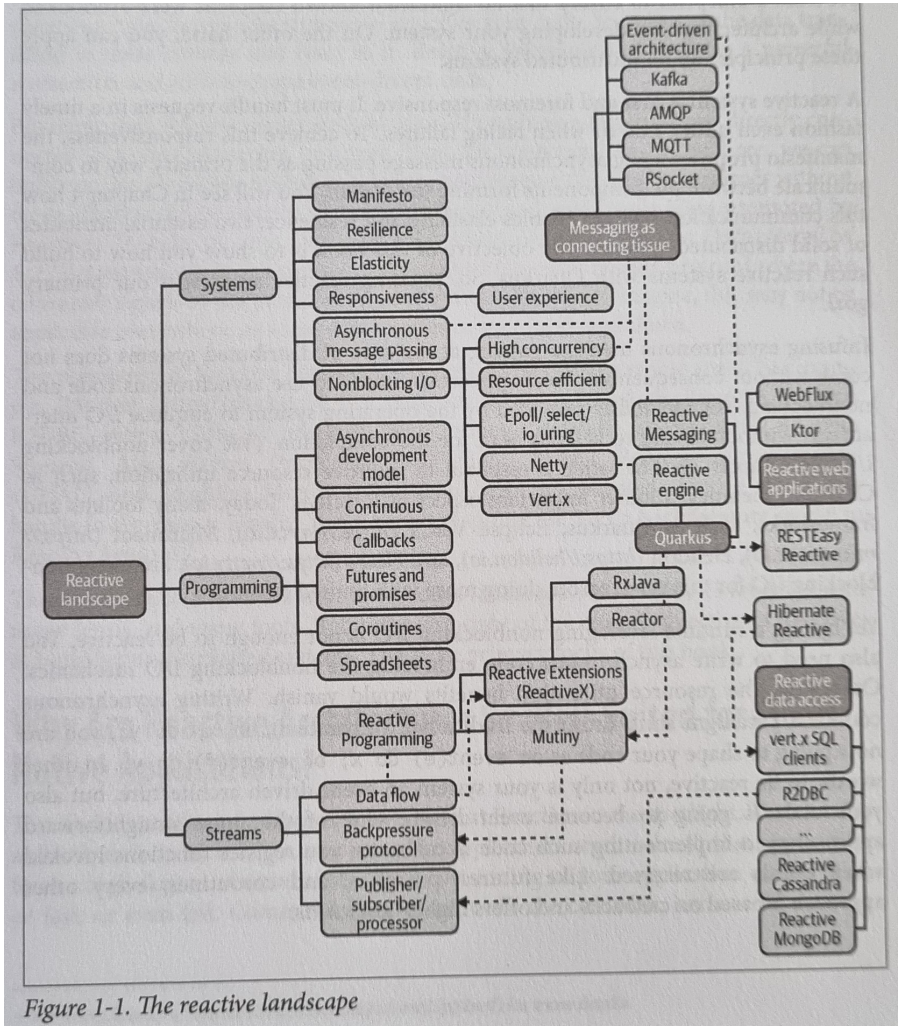


Figure 1-1. The reactive landscape

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

TÉCNICO LISBOA

Enterprise Integration

Quarkus Framework

Mother Tongues

Tracing the roots of computer languages through the ages

Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers-electronic lexicographers, if you will-aim to save, or at least document the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua frangas. Among the most endangered are Ada, APL, B (the predecessor of C), Lsp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Musuem in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at *HTTP://www.informatik.uni-freiburg.de/Java/misc/lang_list.html.* - Michael Mendeno

Key

1954 Year Introduced

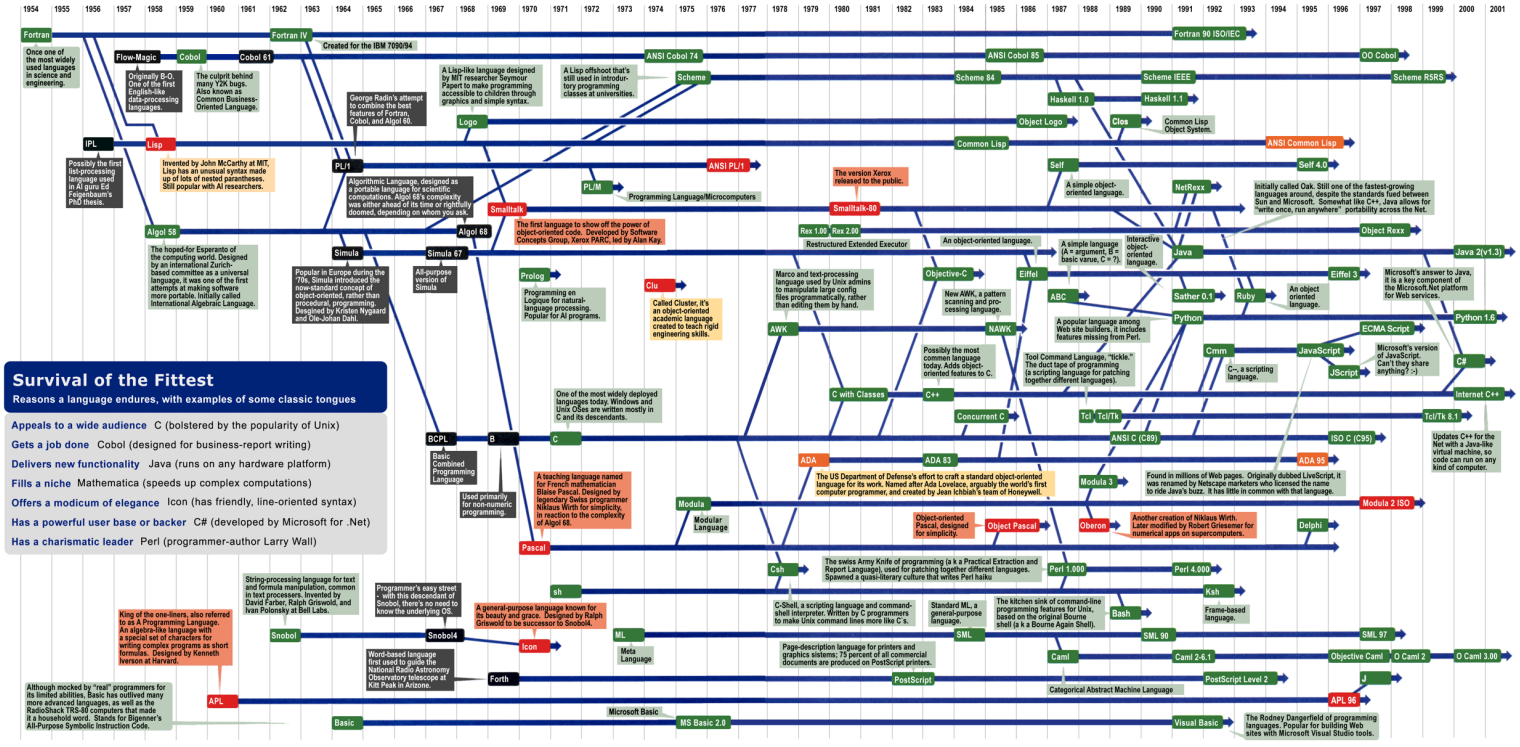
Active: thousands of users

Protected: taught at universities; compilers available

Endangered: usage dropping off

Extinct: no known active users or up-to-date compilers

Lineage continues



Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

https://www.digibarn.com/collections/posters/tongues/

TÉCNICO LISBOA

Programming Languages timeline

Released

Month	JDK 23	JDK 22	JDK 21	JDK 20	JDK 19	JDK 18	JDK 17	JDK 16	JDK 15	JDK 14	JDK 13	JDK 12	JDK 11	JDK 10	JDK 9	JDK 8	JDK 7	JDK 6	JDK 5.0	J2SDK 1.4	J2SDK 1.3	J2SDK 1.2	JDK 1.1	JDK 1	Month
Jan 2023					19.0.2		17.0.6						11.0.18			8u361									Jan 2023
Dec 2022																									Dec 2022
Nov 2022																									Nov 2022
Oct 2022					19.0.1		17.0.5						11.0.17			8u351									Oct 2022
Sep 2022					**19**																				Sep 2022
Aug 2022						18.0.2.1	17.0.4.1						11.0.16.1												Aug 2022
Jul 2022						18.0.2	17.0.4						11.0.16			8u341	7u351							Jul 2022
Jun 2022																									Jun 2022
May 2022						18.0.1.1	17.0.3.1						11.0.15.1			8u333	7u343							May 2022
Apr 2022						18.0.1	17.0.3						11.0.15			8u331	7u341							Apr 2022
Mar 2022						**18**																			Mar 2022
Feb 2022																									Feb 2022
Jan 2022							17.0.2						11.0.14			8u321	7u331								Jan 2022
Dec 2021																									Dec 2021
Nov 2021																									Nov 2021
Oct 2021							17.0.1						11.0.13			8u311	7u321								Oct 2021
Sep 2021							**17**																		Sep 2021
Aug 2021																									Aug 2021
Jul 2021								16.0.2					11.0.12			8u301	7u311								Jul 2021
Jun 2021																									Jun 2021
May 2021																									May 2021
Apr 2021								16.0.1					11.0.11			8u291	7u301								Apr 2021
Mar 2021								**16**																	Mar 2021
Feb 2021																									Feb 2021
Jan 2021									15.0.2				11.0.10			8u281	7u291								Jan 2021
Dec 2020																									Dec 2020
Nov 2020																									Nov 2020
Oct 2020									15.0.1				11.0.9			8u271	7u281								Oct 2020
Sep 2020									**15**																Sep 2020
Aug 2020																									Aug 2020
Jul 2020										14.0.2			11.0.8			8u261 8u260	7u271								Jul 2020
Jun 2020																									Jun 2020
May 2020																									May 2020
Apr 2020										14.0.1			11.0.7			8u251	7u261								Apr 2020
Mar 2020										**14**															Mar 2020
Feb 2020																									Feb 2020
Jan 2020											13.0.2		11.0.6			8u241	7u251								Jan 2020
Dec 2019																									Dec 2019
Nov 2019																									Nov 2019
Oct 2019											13.0.1		11.0.5			8u231	7u241								Oct 2019
Sep 2019											**13**														Sep 2019
Aug 2019																									Aug 2019
Jul 2019												12.0.2	11.0.4			8u221	7u231								Jul 2019
Jun 2019																									Jun 2019
May 2019																									May 2019
Apr 2019												12.0.1	11.0.3			8u212 8u211	7u221								Apr 2019
Mar 2019												**12**													Mar 2019
					...																		...		
Feb 1996																									Feb 1996
Jan 1996																								**1.0**	Jan 1996
Dec 1995																									Dec 1995
Nov 1995																									Nov 1995
Oct 1995																									Oct 1995
Sep 1995																									Sep 1995
Aug 1995																									Aug 1995
Jul 1995																									Jul 1995
Jun 1995																									Jun 1995
May 1995																								1.0_alpha	May 1995

TÉCNICO LISBOA

JAVA history

https://www.java.com/releases/fullmatrix/

JAVA roadmap

TÉCNICO LISBOA

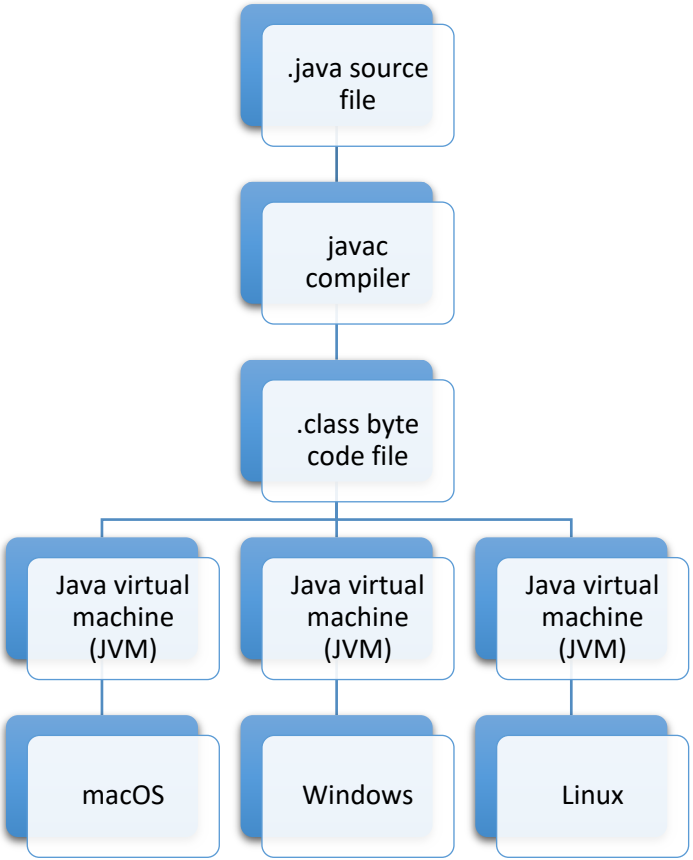
Planned[1]

Month	JDK 23	JDK 22	JDK 21	JDK 20	JDK 19	JDK 18	JDK 17	JDK 16	JDK 15	JDK 14	JDK 13	JDK 12	JDK 11	JDK 10	JDK 9	JDK 8	JDK 7	JDK 6	JDK 5.0	J2SDK 1.4	J2SDK 1.3	J2SDK 1.2	JDK 1.1	JDK 1	Month
Jan 2025	23.0.2		21.0.6				17.0.14						11.0.26			8u441									Jan 2025
Dec 2024																									Dec 2024
Nov 2024																									Nov 2024
Oct 2024	23.0.1		21.0.5				17.0.13						11.0.25			8u431									Oct 2024
Sep 2024	**23**																								Sep 2024
Aug 2024																									Aug 2024
Jul 2024		22.0.2	21.0.4				17.0.12						11.0.24			8u421									Jul 2024
Jun 2024																									Jun 2024
May 2024																									May 2024
Apr 2024		22.0.1	21.0.3				17.0.11						11.0.23			8u411									Apr 2024
Mar 2024		**22**																							Mar 2024
Feb 2024																									Feb 2024
Jan 2024			21.0.2				17.0.10						11.0.22			8u401									Jan 2024
Dec 2023																									Dec 2023
Nov 2023																									Nov 2023
Oct 2023			21.0.1				17.0.9						11.0.21			8u391									Oct 2023
Sep 2023			**21**																						Sep 2023
Aug 2023																									Aug 2023
Jul 2023				20.0.2			17.0.8						11.0.20			8u381									Jul 2023
Jun 2023																									Jun 2023
May 2023																									May 2023
Apr 2023				20.0.1			17.0.7						11.0.19			8u371									Apr 2023
Mar 2023				**20**																					Mar 2023

https://www.java.com/releases/fullmatrix/

JAVA features

- Object oriented
- Simple
- Robust
- Multithreaded
- Distributed
- Dynamic
- Platform independent
- Architecture-neutral
- Portable
- Interpreted
- Garbage collector



JAVA Enterprise Edition (EE)

- Java EE, Java2EE, J2EE, Jakarta EE are evolving names of a set of enterprise specifications that extends JAVA EE

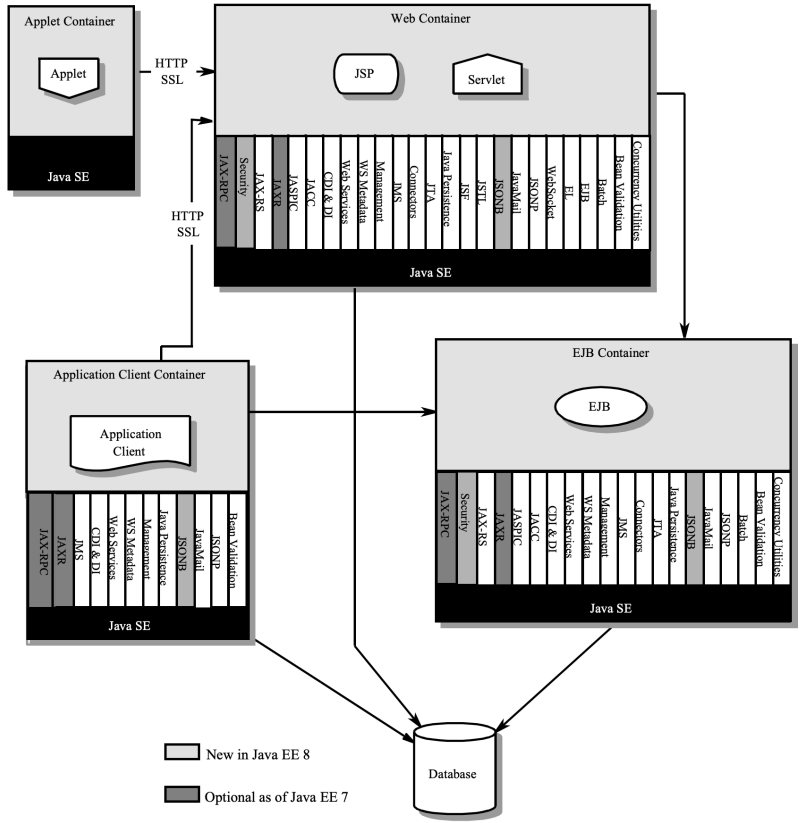
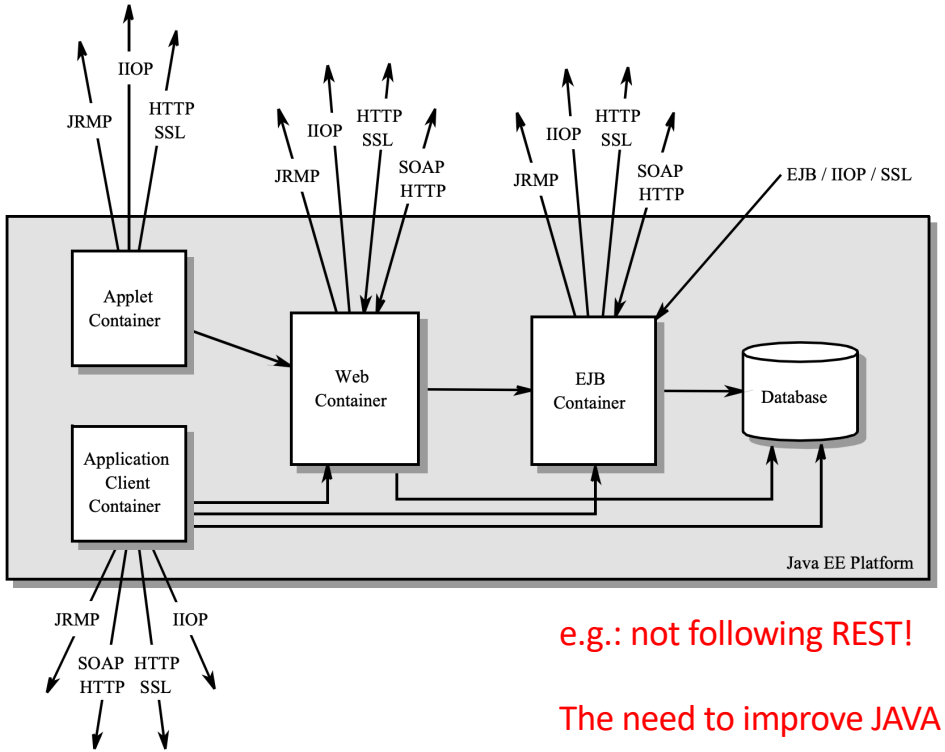


Figure EE.2-1 Java EE Architecture Diagram

JAVA EE Interoperability



e.g.: not following REST!

The need to improve JAVA EE with augmented capabilities

Java™ Platform, Enterprise Edition (Java EE) Specification, v8

Quarkus framework on JAVA

Two distinct models to design your code…

The Imperative Model

The order of the commands cannot be changed or the result will be different.
Sometimes, is the only possible way!

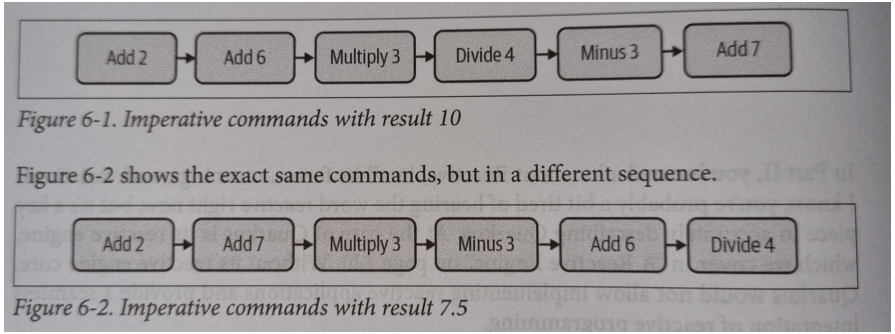


Figure 6-1. Imperative commands with result 10

Figure 6-2 shows the exact same commands, but in a different sequence.

Figure 6-2. Imperative commands with result 7.5

However, in between sending a request to the database and receiving the response, what is the I/O thread doing?

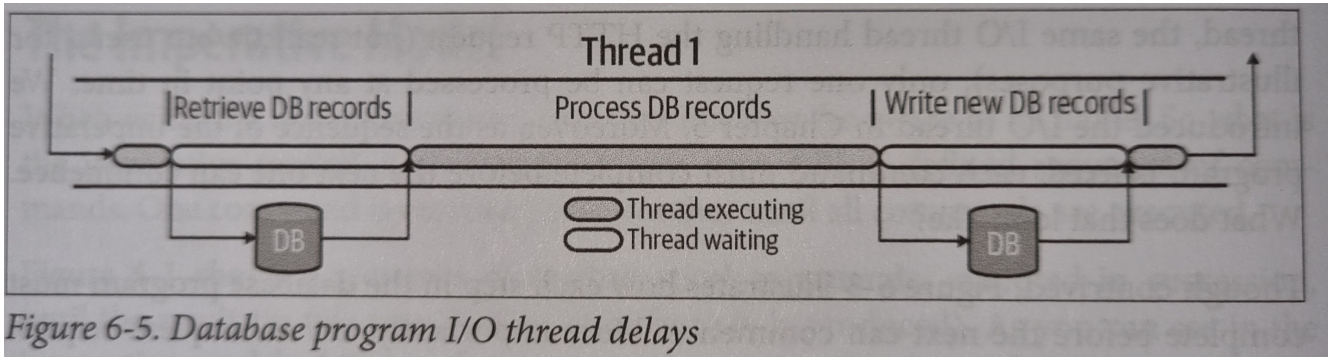


Figure 6-5. Database program I/O thread delays

Many inefficiencies

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

The Reactive Model

- Instead of an I/O thread waiting, it begins processing another incoming request. It continues to do so until it's been notified that a database response is ready for processing.
- How? A continuation (*a callback*) is provided to process the database response

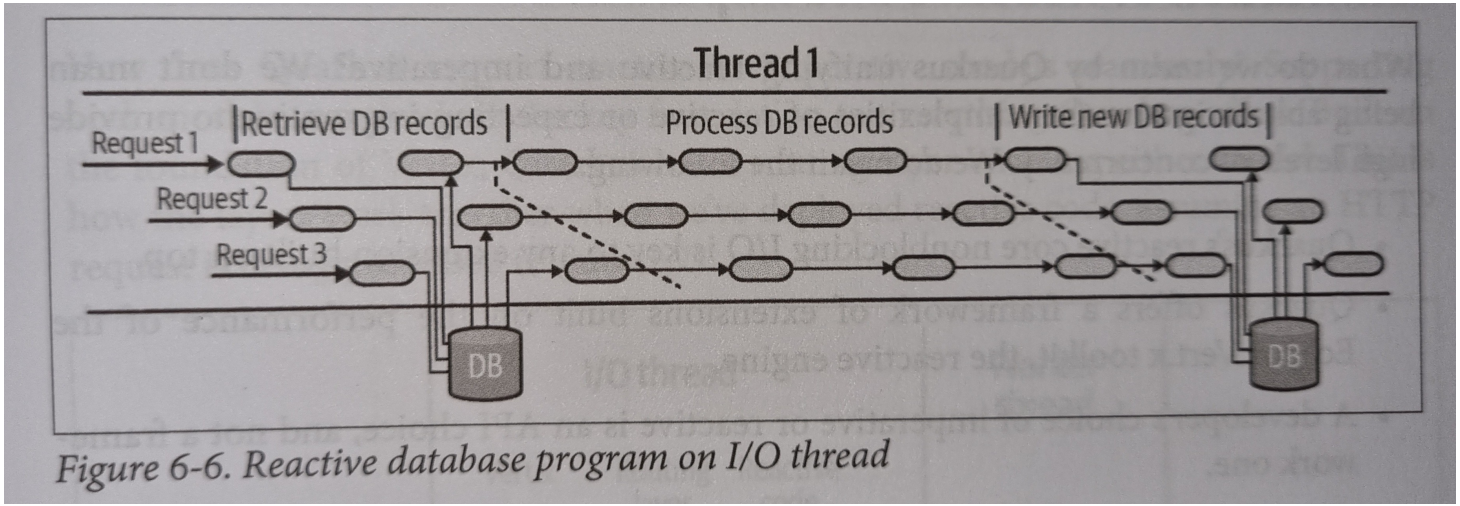


Figure 6-6. Reactive database program on I/O thread

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

The Quarkus supported Models

- A **developer's choice** of imperative or reactive is an API choice, and not a framework one
- Quarkus reactive model is always non-blocking, relying on **Eclipse Vert.x**
- Imperative model requires the execution by a **worker thread** and not I/O thread. Offload from Quarkus is done using **context switch**. **However is cost time and resources**.

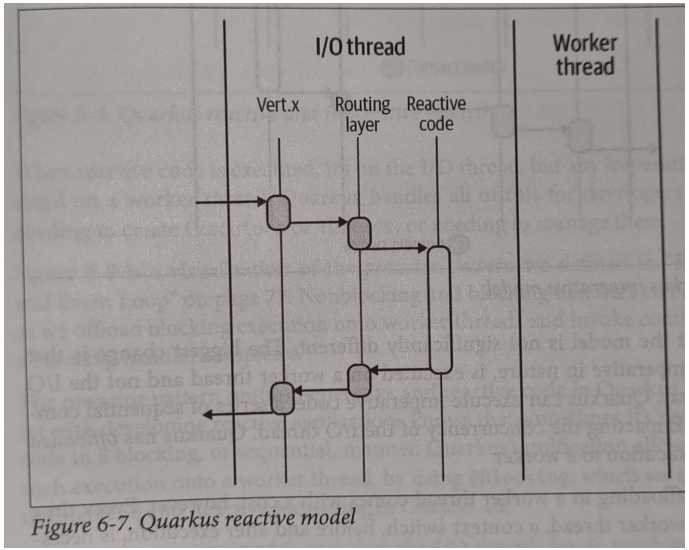


Figure 6-7. Quarkus reactive model

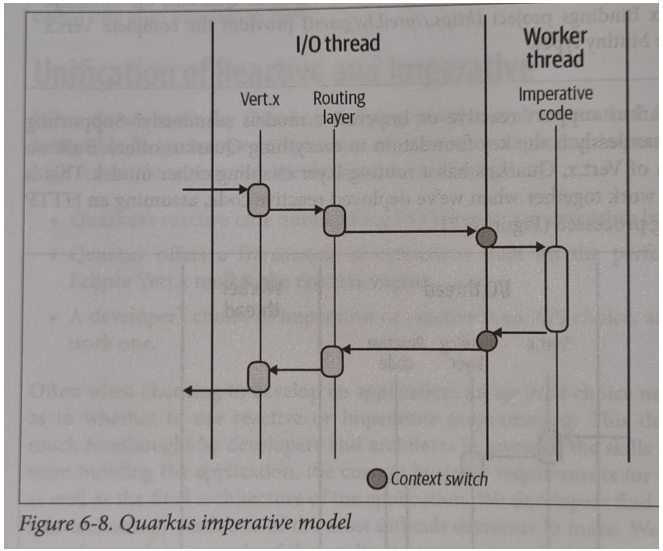


Figure 6-8. Quarkus imperative model

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

Unification of Reactive and Imperative

- Yet, Third option!
  - Non-blocking an blocking handlers can coexist, as long as we offload blocking execution onto worker threads and invoke continuations when a blocking handler completes.
  - Using I/O thread for as much work as possible

- How?
  - Using the `@Blocking` and `@NonBlocking` annotations

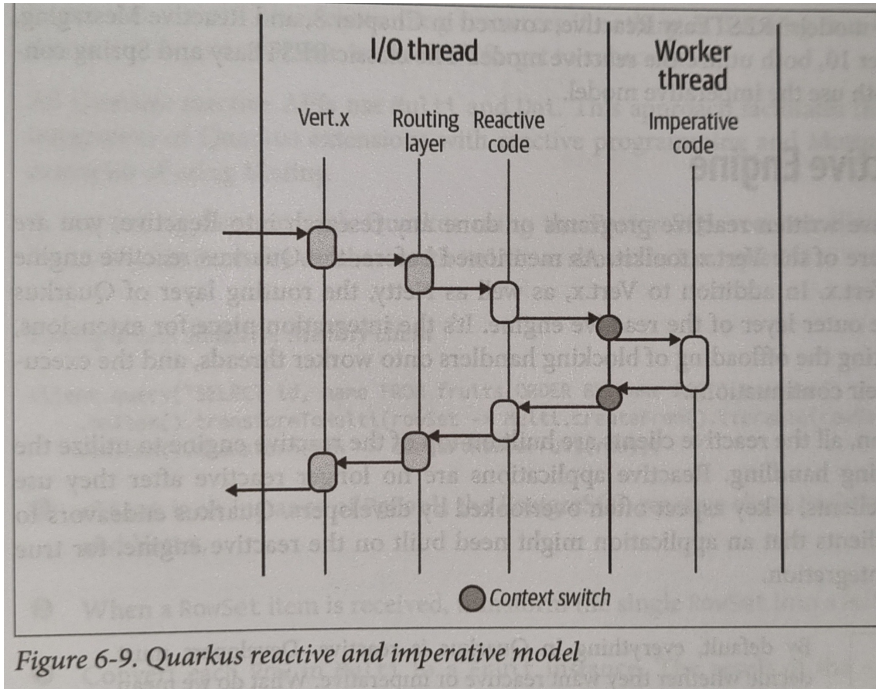


Figure 6-9. Quarkus reactive and imperative model

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

RESTEasy

- With RESTEasy Reactive the annotations `@Blocking` and `@NonBlocking` can be used to indicate on which threads the request should be handled

- **By default, `@NonBlocking` uses an I/O Thread**

- **A method returning Uni or Multi is executed on an I/O thread except if annotated with `@Blocking` that uses a worker Thread**

- **Methods returning any other object is executed on a worker thread, except if the `@NonBlocking` annotation is used**

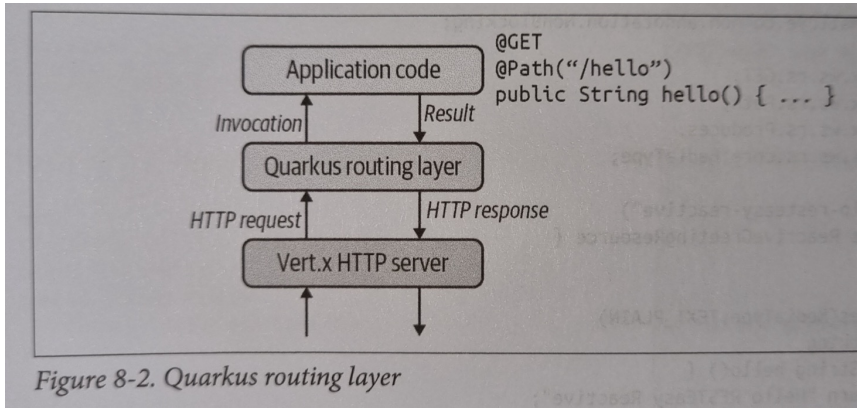


Figure 8-2. Quarkus routing layer

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

RESTEasy

	Annotation required to the method returning Uni or Multi	Annotation required to the method returning any other datatype
I/O Thread	@NonBlocking (it's by default)	@NonBlocking
Worker Thread	@Blocking	Default: @Blocking

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

Mutiny = *Multi and Uni*

- SmallRye Mutiny is the reactive programming library of Quarkus.
- Mutiny is built around three key aspects:
  - Event-driven: listening to events from stream and handling them appropriately
  - Easily navigable API: Navigating the API is driven by an event type and the available options for that event
  - Only two types: `Multi` and `Uni` can handle any desired asynchronous actions

	Events	Use cases	Implement Reactive Streams
`Uni`	Item and failure	Remote invocation, asynchronous computation returning a single result	No
`Multi`	Item, failure, completion	Data streams, potentially unbounded (emitting an infinite number of streams)	Yes

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

Mutiny = *Multi and Uni*

```
@GET
public Multi<Fruit> get() {
        return Fruit.findAll(client);
}

@POST
public Uni<Response> create(Fruit fruit) {
        return fruit.save(client)
          .onItem().transform(id -> URI.create("/fruits/" + id))
          .onItem().transform(uri -> Response.created(uri).build());
}
```

Reactive Object-relational mapping (ORM)

- Three distinct models available:
  - Blocking database client
  - Non-blocking database client
  - Non-blocking database with pipelining – database shared connection *(for compatible databases)*

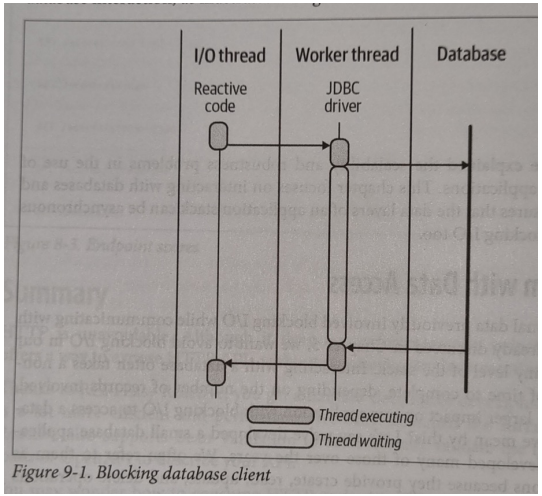


Figure 9-1. Blocking database client

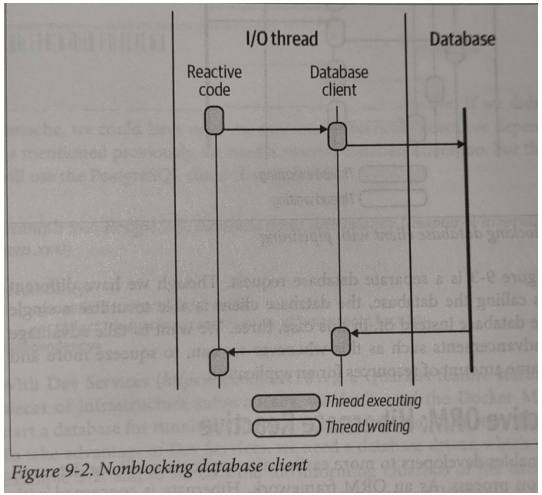


Figure 9-2. Nonblocking database client

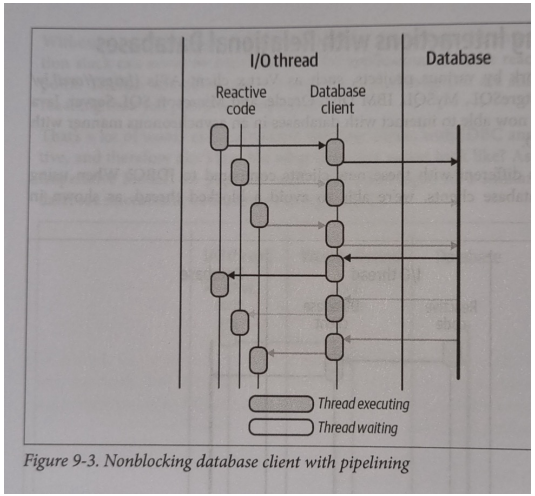


Figure 9-3. Nonblocking database client with pipelining

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

Hibernate Reactive

- Hibernate ORM enables developers to more easily write applications whose data outlives the application process. As an ORM framework, Hibernate is concerned with data persistence as it applies to relational databases. Hibernate provides both imperative and reactive APIs

- These APIs support two facets:
  - non-blocking database clients, and
  - reactive programming as a means of interacting with relational databases.

```
mvn io.quarkus.platform:quarkus-maven-plugin:2.7.5.Final:create \
    -DprojectGroupId=org.acme \
    -DprojectArtifactId=reactive-mysql-transactioncontrol-client-quickstart \
    -Dextensions="resteasy-jackson,quarkus-reactive-mysql-client,resteasy-mutiny,quarkus-smallrye-openapi" \
    -DnoCode

mvn io.quarkus.platform:quarkus-maven-plugin:2.7.5.Final:create \
    -DprojectGroupId=org.acme \
    -DprojectArtifactId=reactive-pg-client-quickstart \
    -Dextensions="resteasy-jackson,reactive-pg-client,resteasy-mutiny,quarkus-smallrye-openapi" \
    -DnoCode
```

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

Reactive Messaging: Quarkus with Kafka

TÉCNICO LISBOA

- Quarkus offers a message-driven development model that is simple but powerful to consume, process, and create messages
- Old JAVA JMS is not fully compatible with Kafka and is a blocking API which do not allow the implementation of reactive principles
- Reactive Messaging can send, consume, and process messages in a protocol-agnostic way
- Messages transit on **`channels`**

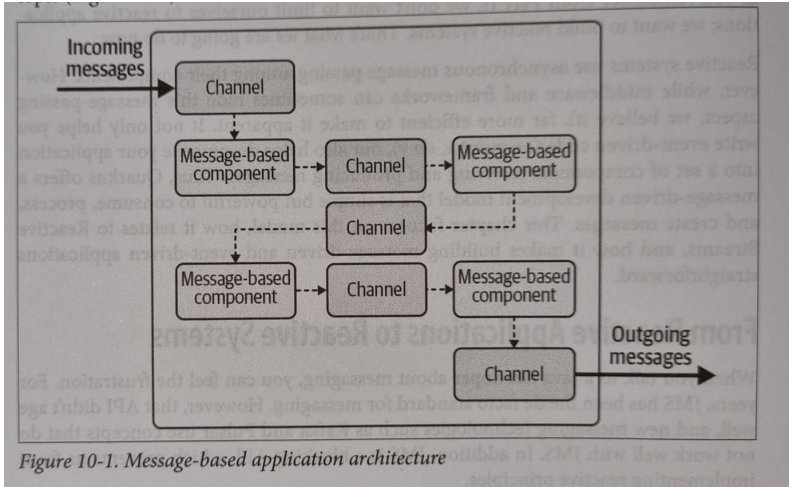


Figure 10-1. Message-based application architecture

```
@Channel("my-channel")

MutinyEmitter<Person> personEmitter;

public Uni<Void> send(Person p) {
   return personEmitter.send(p);
}
```

Producing messages

```
@Incoming("words-in")
@Outgoing("words-out")
public Message<String> todb(Message<String> message)
{
 return message.withPayload(message.getPayload().toUpperCase());
}
```

Processing messages

Escoffier, C., Finnigan, K. (2021). Reactive Systems in Java, O'Reilly Media, Inc., ISBN: 9781492091721

TÉCNICO LISBOA

Q&A

TÉCNICO LISBOA