ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Corso di Laurea Triennale in Ingegneria e Scienze Informatiche

Multi-platform distributed systems with aggregate computing in Kotlin: the case of proximity messaging

Tesi di laurea in:
Programmazione ad Oggetti

Relatore
Prof. Pianini Danilo

Candidato
Luca Marchi

Correlatore
Dott.ssa Cortecchia Angela

II Sessione di Laurea
Anno Accademico 2024-2025

Abstract

To my family.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

1.1 Context

In a world where Internet of Things (IoT) devices are becoming increasingly prevalent, the need for efficient and reliable communication systems is paramount (Figure 1.1).

The interactions between neighboring devices play a crucial role in enabling seamless data exchange and coordination, so there is the need to design networks with infrastructures that support scalability, adaptability and reusability [BPV15]. In the past, it was reasonable to use a programming model that focused on the individual computing device, and its relationship with one or more users. However, as systems have grown in scale and complexity with the number of computing devices rising, this method has become inadequate.

Traditional network architectures, rely heavily on centralized infrastructures, making them unsuitable for scenarios such as disaster recovery or interactions with neighboring devices. The computational model of *aggregate computing* provides a promising approach to address these challenges by enabling decentralized and self-organizing systems [VBD+19].

Building on this concept, this thesis explores how aggregate programming can support a proximity and decentralized messaging system in a network.

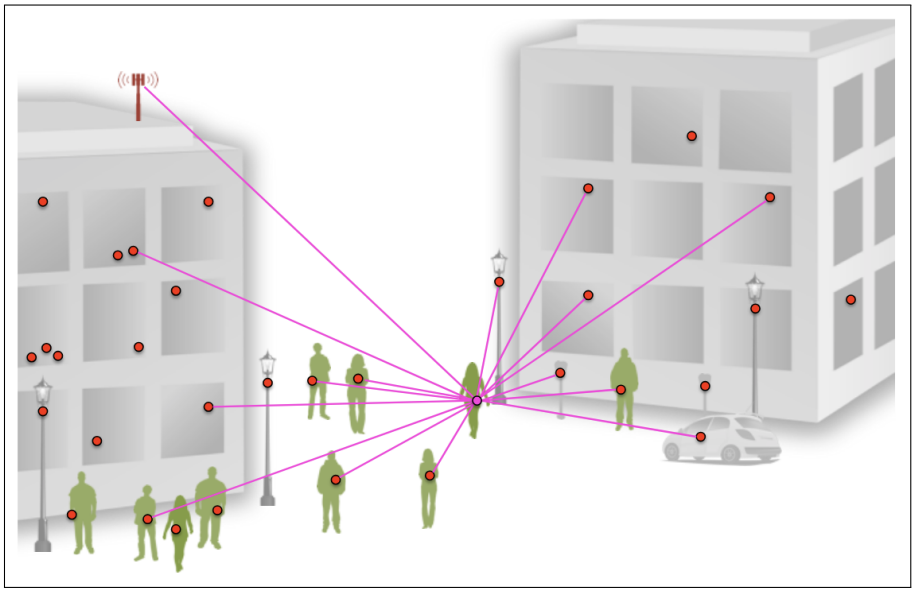

Figure 1.1: A variety of IoT devices communicating with each other in a real world scenario.

1.1.1 Aggregate Programming

Aggregate Programming is a distributed systems paradigm that simplifies programming large networks of devices by focusing on the global, system-level behavior rather than the individual behavior of each device. It shifts the abstraction to view the network as an aggregate, an agglomeration of nodes that collectively exhibit certain behaviors. This model is particularly well-suited for scenarios where devices need to coordinate and collaborate based on local interactions, such as in IoT applications.

As discussed in distributed field programming [VBD+19], there are principles to follow when designing large-scale systems:

- The mechanisms ensuring robust coordination should operate transparently in the background, so that programmers don't need to manage them directly.
- The composition of modules and subsystems should be simple and predictable, allowing developers to clearly understand the consequences of combining components.
- Large-scale distributed systems often consist of multiple heterogeneous sub-

systems, each of which may require different coordination strategies depending on the region or context in time.

Aggregate programming aims to overcome these challenges through three fundamental principles:

1. The computational target is conceived as a region of the environment, with the underlying device-level details abstracted away, potentially even modeled as a continuous spatial domain.

2. The program logic is expressed as the manipulation of data structures that extend spatially and temporally across that region.

3. These computations are executed locally by individual devices within the region, which coordinate through resilient mechanisms and proximity-based interactions.

The foundation of this paradigm lies in *field calculus* [BPV15], a core set of constructs modeling device behavior and interaction. These essential features are captured in a tiny universal language suitable for mathematical analysis. The concept of **field** plays a central role in this context and originates from physics, where it is defined as a function mapping every point in a space–time domain to a scalar value.

According to [VBD+19], the *field value* ϕ is a function

$$\phi : D \to L$$

that maps each device δ in the domain D to a local value ℓ in the range L. This value can change in time with a *field evolution* that maps each point in time to a field value. Figure 1.2 shows how aggregate programming abstracts the complexity of the underlying distributed network and its coordination challenges through a sequence of hierarchical abstraction layers.

The messaging system developed in this thesis was implemented using *Collektive* [Cor], an open-source framework designed to simplify the development of distributed systems through the principles of Aggregate Programming. Collek-

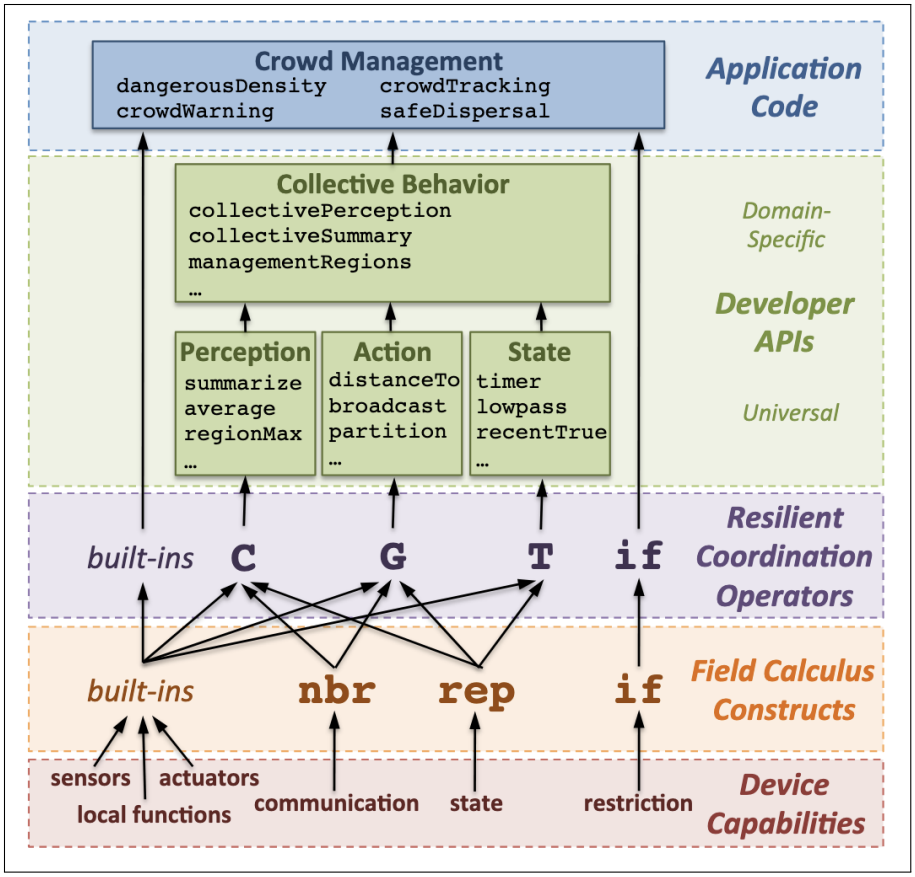

Figure 1.2: Layered structure of aggregate programming, illustrating the abstraction at each level.

tive provides developer-facing APIs (as in Figure 1.2) used to define and execute aggregate applications.

1.1.2 Collektive

Collektive is a framework with the purpose of simplifying the definition of Aggregate Computing systems. To achieve this, Collektive adopts the Field Calculus model, providing an intuitive way to define aggregate behavior without low-level coding. Built with Kotlin Multiplatform (KMP), it can be executed seamlessly on different targets. Moreover, a compiler plugin was introduced to support function alignment, automatically annotating aligned functions to ensure consistent coordination among nodes [Cor].

The main features of this framework can be summarized as follows [Cor25]:

- **High-Level Abstraction**: Collektive streamlines the development of distributed behaviors by enabling developers to define coordination logic succinctly, without explicitly handling individual device states.
- **Declarative Programming**: The framework adopts a declarative paradigm, allowing developers to specify the desired system outcomes, while the underlying runtime manages the execution details.
- **Dynamic Adaptation**: Applications developed with Collektive can automatically adapt to network variations and external inputs, maintaining robustness in dynamic and unpredictable environments.

It provides a minimal Domain Specific Language (DSL) where the developer can specify the collective behaviour of a network of devices and the devices can directly communicate with each other and execute the same program.

1.2 Motivations

In this section motivations for implementing a decentralized proximity-based messaging system will be illustrated.

In recent years, the proliferation of IoT and mobile devices has increased the need for local communication systems capable of operating even without stable network infrastructure. Centralized architectures, although efficient in connected environments, fail to guarantee reliability and resilience in dynamic or disconnected scenarios. As devices are becoming pervasive and ubiquitous, new models are required to enable coordination and information exchange without relying on fixed infrastructures. This need motivates the exploration of decentralized and distributed approaches, which can operate robustly in dynamic and connectivity-limited environments.

1.2.1 Problem statement

Traditional messaging systems and architectures often rely heavily on centralized servers and infrastructures, which are unsuitable and vulnerable for scenarios where connectivity is not guaranteed. These situations include **disaster recovery**, the process for restoring an organization's network infrastructure and operations after a disruptive event, such as a natural disaster, cyberattack, or system failure.

In such environments there are several vulnerabilities and challenges to consider:

- **Single point of failure**: Centralized systems are vulnerable to failures of the authority that makes decisions. This can lead to widespread service disruption and outages.
- **Scalability issues**: As the number of devices increases, centralized systems may struggle to manage the growing volume of data and communication, leading to bottlenecks and delayed responses.
- **Limited resilience**: Centralized architectures often lack the redundancy and fault tolerance needed to maintain functionality in the face of network disruptions or device failures.

The networks that need to overcome the issues above can benefit from Aggregate Computing principles, which provides a distributed architecture where the nodes can directly communicate with the neighbors.

1.2.2 Real world use cases

As mentioned in the previous sections, there are several scenarios where a decentralized proximity-based messaging system can be particularly beneficial.

The main use case of such system is in emergency situations. When natural disasters, or infrastructure failures occur, cellular networks and Internet connections often go down, making it difficult for people to communicate and coordinate rescue efforts. Nearby devices can form ad-hoc Bluetooth or Wi-Fi Direct networks, allowing users in the same area to propagate and share messages without relying on centralized servers.

This architecture can also be useful in large IoT networks like sensors systems or drones swarms, operating in remote areas and with the need to communicate without continuous Internet access.

The system's decentralized nature fits perfectly for:

- Exchanging local environmental data (temperature, humidity, motion).
- Maintaining distributed consensus or context awareness among nearby nodes.
- Avoiding single points of failure.

A practical example includes a network of environmental sensors deployed in a forest to monitor conditions and detect wildfires (Figure 1.3), or sensors in a smart agriculture field that share humidity data to coordinate irrigation locally.

1.2.3 Goal

This thesis can be devided into two main goals:

1. **Messaging algorithm**: The first goal is to design and implement a decentralized, proximity-based messaging algorithm grounded in the principles of Aggregate Computing. The network operates in a fully distributed manner, without any central server managing message exchange. Each node acts autonomously, coordinating with its neighbors through local interactions and collectively achieving self-organization, thereby eliminating single points of failure. Communication is proximity-based: devices can only interact with

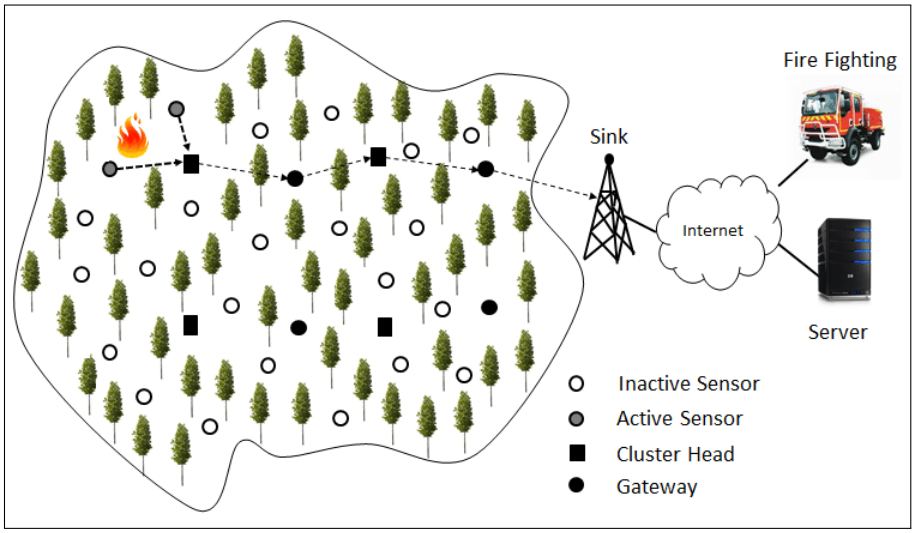

Figure 1.3: Real-world use case of a decentralized proximity-based messaging application built on Aggregate Computing principles. In this scenario, sensors located near the fire detect the event and propagate an alert message to neighboring nodes within the network. [SBEK16]

nearby nodes, and messages are propagated progressively across the network, layer by layer. The algorithm must also handle dynamic network topologies, where devices can join or leave the network at any time, ensuring robustness and adaptability. Message sources can define both an expiration time and a maximum propagation distance, enabling controlled diffusion and limiting unnecessary communication overhead. The system is implemented using the *Collektive* framework [Cor], leveraging KMP to ensure cross-platform compatibility across different targets including JVM, Javascript (browser), Android, iOS, and native versions (for Windows, MacOS, and Linux, both for x86 and ARM CPUs).

2. **Mobile application**: The second goal is to develop a multiplatform mobile application using KMP. This application leverages the messaging algorithm as a core library to send and receive messages from devices in the network. The app must support both Android and iOS platforms providing a simple, user-friendly interface. The User Interface (UI) allows users to compose messages, configure parameters (expiration time and propagation distance),

and view received messages in real-time.

1.3 State of Art

Chapter 2

Contribution

You may also put some code snippet (which is NOT float by default), eg: chapter 2.

2.1 Fancy formulas here

```
public class HelloWorld {
  public static void main(String[] args) {
    // Prints "Hello, World" to the terminal window.
    System.out.println("Hello, World");
  }
}
```

Bibliography

[BPV15] Jacob Beal, Danilo Pianini, and Mirko Viroli. Aggregate programming for the internet of things. *Computer*, 48(9):22–30, 2015.

[Cor] Angela Cortecchia. *A Kotlin multi-platform implementation of aggregate computing based on XC*. PhD thesis.

[Cor25] Cortecchia, Angela. Collektive documentation, 2025.

[SBEK16] Massinissa Saoudi, Ahcène Bounceur, Reinhardt Euler, and Tahar Kechadi. Energy-efficient data mining techniques for emergency detection in wireless sensor networks. 07 2016.

[VBD+19] Mirko Viroli, Jacob Beal, Francesco Damiani, Stefano Montagna, Danilo Pianini, Luca Ricci, and Franco Zambonelli. *Aggregate programming: from foundations to applications*. Springer, 2019.