UNIVERSITAT JAUME • I

An Analytics Platform for Integrating and Computing Spatio-Temporal Metrics in Location-aware Games

Luis Enrique Rodríguez Pupo

Directores: Phd. Sven Casteleyn
Phd. Carlos Granell Canut

Febrero 2021

Programa de doctorado en Informática.
Escuela de Doctorado de la Universitat Jaume I

An Analytics Platform for Integrating and Computing Spatio-Temporal Metrics in Location-aware Games

Memoria presentada por Luis Enrique Rodriguez Pupo para optar al grado de doctor por la Universitat Jaume I

Luis Enrique Rodriguez Pupo   Sven Casteleyn   Carlos Granell Canut

Castelló de la Plana, febrero de 2021

Financiación recibida

Esta tesis doctoral ha sido parcialmente subvencionada por el progama Ramón y Cajal del Ministerio de Ciencia e Innovación, mediante los proyectos con referencia RYC-2014-16606 y RYC-2014-16913.

To my family and friends.

Acknowledgments

I would like to thank Carlos Granell and Sven Casteleyn for their support and unconditional help in the development of this work. Thank you for your help with the research and your invaluable contributions to this work and the required papers. I feel truly indebted to you. Also would like to thank Geotec research group for the many years of sharing your friendship and your contribution to my personal growth. Thanks to Joaquín Huerta for his support and the opportunity of working in Geotec. Starting as a MSc. student in the Geotec group was certainly the biggest opportunity ever given to me.

Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 Motivation and problem statement

Notwithstanding the early promise of location- and context-aware applications two decades ago (see e.g., Chen et al. (2000) for a survey of early systems), only in the last decade we have witnessed the required technological and infrastructural enablers to truly unleash their potential (Bellavista et al., 2012; Pejovic and Musolesi, 2015). For the technological enablers, the increasing availability of a variety of context-capturing machinery, in which embedded sensors, local processing and communication capabilities are combined, allows for large-scale, high-volume capturing and streaming of a broad variety of context data. Examples notably include sensor-packed smart vehicles, mobile hand-held devices (e.g., smart phones, tables) and smart, wearable devices (e.g., smart watches and bracelets, sport trackers, smart clothing), which can effectively collect an individual's location, along with other relevant contextual information (e.g., Rana et al. (2016)). A second technological milestone is the rapid evolution and proliferation of powerful mobile hand-held computing devices, a condition *sine qua non* to run full-fledged, context-aware applications (Hoseini-Tabatabaei et al., 2013).

On the other hand, infrastructure-related enablers are now in place: fully rolled-out communication networks (e.g., 3G and 4G) with resulting ubiquitous internet access, and commercially available, economised cloud-based storage and computing infrastructures (Varghese and Buyya, 2018), provide unprecedented means to build the next-generation of context-aware applications and services, based on multi-user, real-time and high-frequency input streams; real-time data handling, processing and analytics; and real-time, location- and context-based reactiveness (Bainomugisha et al., 2013). Indeed, we see a breakthrough of such applications in various application fields, such as mobility and transportation (e.g., Wan et al. (2014)), health (e.g., Solanas et al. (2014); Chang et al. (2017)), tourism (e.g., Meehan et al. (2013), smart cities (e.g., García et al. (2017); Sagl et al. (2015)), smart homes (e.g., Alirezaie et al. (2017)), gaming (e.g., Pokemon Go (Andone et al., 2017), to name but a few.

Nevertheless, due to their relative new and evolving supportive technologies, building such applications is yet a laborious job. The client-side application needs to deal with and be built around an additional, dynamically changing concern, namely context in general and location specifically, while server-side handling of context data, especially in large-scale multi-user deployments, needs to deal with streaming data, big data issues, spatial analysis and reactiveness.

In this technological context, where context information is highly available, different types of applications leverage and combine sensor data with data regarding social aspects to provide both services and entertainment in unprecedented ways. A type of context-aware, location based application that is gaining popularity worldwide are location-aware games. Location-aware games are games that take place in the real world space, using the location as an important aspect of the game, and often, but not always, require the user to move physically in space (see Chapter 2). Digitally-enabled variants of these games often rely on the capture and analysis of context data produced and processed continuously during the game development, and the user expects the outputs driving the game to be produced in a timely manner. In such location-aware games, stream computing will become a vital feature for multiple reasons. First, it will allow proper management of huge volumes of incoming, varied data that these location-aware games continuously collect. Second, stream computing will permit to analyse players' behavior and other game-related statistics to enable quick, dynamic adaptation of the gameplay to the player's needs, experience, and contextual environment.

If we look at location-aware games from the point of view of geographic information science, an impressive arsenal of techniques, methods and tools for capturing, storing, managing, and processing spatial information has been developed over the past decades (Karimi (2017); De Smith et al. (2018); Worboys and Duckham (2019). Vector-based and raster-based spatial operators are usually arranged together as automated geospatial workflows. Research works (e.g. Granell et al. (2010); Granell (2014); Yue et al. (2016); Zhou et al. (2017); Guo et al. (2018)) have made substantial progress over the past years to go well beyond desktop-based environments to bring geospatial workflows to the cloud and distributed computing environments, contributing to the field of Geoprocessing Web (Zhao et al., 2012; Hofer et al., 2018). To this regard, leading voices in the field of geographic information science recently called for an entirely new brand of geospatial platforms and systems to analyse and process real-time data streams (Batty (2016); Jiang (2015); Miller and Goodchild (2015); Li et al. (2016)). In other words, what served in the past and still serves for scenarios in which real time is not necessary, does not fit well with scenarios that handle data streams such as in location-aware game applications.

Cutting-edge research in analytical platforms to facilitate the measurement, quantification and computation of real-time data related to location and contextual aspects are still in their infancy, especially in support for spatial and temporal dimensions of data streams (Galić, 2016). This thesis partially explores this need in the hope of shedding some light.

1.2 Research objectives

In the context of location-aware games, our research project pursues to address four fundamental research objectives.

The first research objective **(RO1)** is related to the existing support for the spatial aspects in games, as a fundamental requirement in the design and implementation of location-aware games. In particular, the aim is to investigate how (and which) spatial features are taken into account in the design and development of location-aware games, more particularly, to steer game balance and gameplay.

The second research objective **(RO2)** is to identify and classify the measurable spatial features that are relevant in the design of location-aware games. More specifically, the aim is to propose a model that covers the different dimensions of game metrics, particularly to support spatial features.

The third research objective **(RO3)** is to design and develop an operational platform implementing the established theoretical principles (RO2) for game metrics. Hereby, we strive for the platform to exhibit good design and architectural features (i.e., distributed, scalable, asynchrony, good abstractions), and the final implementation to be easy to use for developers (i.e., easy to setup, to develop with, to learn).

The final research objective **(RO4)** is to validate the proposed theoretical model and its practical implementation, by applying it in different real-world scenarios, both within and outside the field of location-aware games, and hereby demonstrating the concrete benefits elaborated in RO3.

1.3 Research methodology

The methodology followed in this work consisted of an exploratory literature review of metrics systems, with particular emphasis on the support for spatial metrics, in order to understand the current state of the art and gain awareness of the features of the different systems. Through this analysis, we discovered the current features supported in systems developed in both commercial and academic contexts. Based on this literature analysis, we divided the features into four functional areas, namely, (i) Data collection and communication, (ii) Data representation, (iii) Data analysis and reaction, and (iv) Data visualisation and reporting. These functional groups were then used to perform a comparison between the systems analysed. As we will present later in Chapter 2, the systems exhibit a wide variety of implementations, targeting games with different characteristics, which allowed us to gain a better understanding of the different aspects of these systems.

The analysis served to identify the common functionalities and lacks encountered in this kind of systems, and at the same time evaluate the trade-offs and challenges of the different implementations in the context of the target application domains. The result of this research showed a clear lack of spatial support, only provided in a few systems, through basic spatial operations. This served as a starting point for designing a metrics platform, based on a model centered in easing the definition of the required data structures and metrics functions, and providing the required tools and functionalities for supporting advanced spatial operations. Spatial support is provided for effectively processing, storing, and analysing spatial data in the context of location-aware games; nevertheless, the model designed is also usable for general-purpose systems, as we will show in Chapter 5.

The platform is implemented using a modern stack of technologies that ensure scalability. As part of this platform, we also provide an SDK for Android and Java for interacting with the platform through a set of RESTful services, covering functionalities for data submission, retrieval, as well as management operations. To gain access to the platform's data processing capabilities, the platform provides APIs and a set of JavaScript based functionalities that allow interacting with the user-provided data model through the metrics.

For validating the metrics model and the implemented supporting platform, we have performed two experiments. The experiments consist of solving spatio-temporal analysis problems in two different use cases using a wide range of functionalities of the platform. The aim is to demonstrate the sufficiency of the metrics model and the tools provided for externalising and supporting the spatio-temporal processing, and evaluating its ease of use from the platform's user perspective. The first experiment focuses on re-implementing and evaluating an existing, fully functional location-aware game (described in (Martí et al., 2012)) from a developer perspective. We compared the original code (without use of the platform) and the re-implementation (with use of the platform), using evaluation features such as learnability, asynchrony, scalability, and abstraction as a basis for comparing the advantages and trade-offs of the proposed platform (further details in Section 5.1). These features are used to demonstrate the ease of setup and implementing commonly included features using the proposed model, and therefore provide a critical perspective of its effectiveness for implementing location-aware games. The second experiment used the proposed platform to collect and process patients' mobile-based location data during a therapy. It serves to show the use of the framework beyond location-based games, in the context of location-aware applications.

1.4 Contributions

In this document we present two main contributions, related with the metrics model defined and the platform implemented for supporting the model. Additional smaller contributions are derived from the process of the implementation of the platform, related to supporting software produced.

1.4.1 Major contributions

Conceptual metrics model

The underlying idea of metrics is to measure monitored phenomena of interest for users, independent of their application domain, e.g., research, urban planning, software development. An essential requirement in doing so is to capture the required data by means of flexible data models, since these phenomena may require the collection of data of diverse nature, concerning different aspects a user wants to monitor. Example of such aspects include application-environment interaction, (location-aware) user interface interaction, user mobility, or any other relevant data that are necessary to quantify the context.

The concept of metrics has been commonly associated with a sole function. Here, we extend this concept to a model composed of three main elements: the data model or structure, the analytic function, and associated action(s). As such, the main strong points of the proposed conceptual model of spatio-temporal metrics, which addresses RO2, are:

- The model can be used in a wide range of location-aware games, allowing the representation of diverse data structures and metrics functions for most cases.

- The model can be shared and reused by different applications and is platform independent, allowing its consumption by different systems.

- Actions definition are part of the model, which allows the notification of interested parties in case information extracted from data is considered relevant.

Analytics platform for location-aware games

We present an analytics platform for defining and computing spatio-temporal, context-aware metrics. The proposed concept of metrics is central to allow application developers to define data requirements that capture relevant spatio-temporal aspects of an observed phenomenon, collect the required (client- generated) data, and execute the associated function to process streams of collected data. Based on the processed data, the analytics platform provides asynchronous notifications for quick reaction in end-user applications, as well as post-hoc programmatic access and data visualisation features. Hereby, the analytics platform acts as a service, allowing application developers to outsource the burden of handling, analysing and interpreting context-related data. It suffices to communicate relevant raw data to the platform to receive context-related event notifications, based on the defined metrics. The platform is based on a cloud-based architecture, and is specifically designed to handle large amounts of data, perform analysis over the collected data, and realise notifications in a decentralised way.

As such, the main strong points of the analytics platform, which addresses RO3, are:

- defined at a sufficient level of abstraction to be able to support multiple application domains, i.e., not application-specific;

- capable of handling spatial and non-spatial metrics; designed for intrinsic support for streaming data collection and processing;

- use of an extended metrics model, including specification of necessary data and its structure, a function to define a relevant context condition, and associated action(s);

- use of an open, declarative specification of metrics to reuse them in different location-aware games (and more broadly, application domains).

1.4.2 Minor contributions

In addition to the process of designing and developing the analytics platform, a variety of general, re-usable supporting software libraries have been also produced. These libraries are general enough to be reused in a broader context, beyond the use cases reported in this dissertation, and thus are important side-effect contributions resulting from this work. The libraries implemented are the following:

- **schemaconverters** (see Appendix 6.2.3): This library was built with the purpose of defining Cassandra tables structures from a JSON Schema. This is achieved by converting the structure defined by the JSON schema to Cassandra DDL statements. This means that by definition, the Cassandra table is able to store schema compliant JSON objects seamlessly.

- **geo-test-utils** (see Appendix 6.2.3): spatial-related utils library used mainly for testing purposes, contains functionalities for working with routes extracted from GPX formatted files, and helps with sample routes useful for different situations. For example, extracting points of a route in a given time interval, or until a condition is met. The conditions can be, for example, until the route enters an area delimited by a point and a radius or when the route points abandon such an area. This library was used for developing the tests of the platform geo-fencing functionalities, using real route data, but can also be used in other contexts.

- **actors** (see Appendix 6.2.3): this library contains several functionalities that can be also leveraged by software developers in other projects. The idea is to provide crosscutting functionalities implemented through different types of Akka actors. Functionalities include a simple abstract API for querying and persisting data in databases. Based on this API, two implementations are provided, one for MongoDB and another for Cassandra, as those are the main databases the analytics platforms works with. Software developers can use the actors library, for example, to interact with the MongoDB database, used in the platform for retrieving the data related to the metrics definition and system configuration. The Cassandra implementation for this actor (in project cassandra-persistence-actor) is used in the metrics data ingestion and metrics data access through the REST API. Besides, the library also include actors for sending push notifications through Google Cloud Messaging platform.

Next to these reusable software libraries, a large amount of software artefacts resulted from this dissertation, which are all freely available to the software community. The links to the source code repositories of these all these artefacts are provided in Section 6.2.3 in Table 6.2.

1.5 Thesis organisation

The thesis is organised in the following chapters.

Chapter 2 addresses RO1. It is dedicated to analyse the state of the art regarding the existing definitions and elements for characterising gameplay through the evaluation of game balance. It also investigates how (and which) spatial features are taken into account in the design and development of location-aware games.

Chapter 3 addresses RO2. It focuses on the identification and classification of measurable spatial features that are relevant in the design of location-aware games. More specifically, the aim is to propose a model that covers the different dimensions of game metrics, particularly to support spatial features. The proposed model takes a central role in the conceptual view of the analytics platform and, consequently, in the implementation described in subsequent chapters.

Chapter 4 addresses RO3. Considering the platform as an ecosystem of applications and supporting tools, we discuss its implementation emphasising the use of state-of-the-art big data processing and analytics technologies, and exhibiting desirable architectural features (i.e., distributed, scalable, asynchrony, good abstractions), and easy to use features for developers (i.e., easy to setup, to develop with, to learn, shareable and reusable metrics). It also provides a discussion comparing the implemented platform with the ones analysed in Chapter 2.

Chapter 5 addresses RO4. Its focus is on experimentation and assessment. The first part describes two experiments that uses the analytics platform. The first consists of a location-aware game implemented with and without the platform, and the second is an application of the platform beyond games, namely in the field of mental health. The second part evaluates and discusses the first experiment in more detail, in order to contemplate advantages and disadvantages of using the platform for defining and computing spatio-temporal metrics in location-aware games. Overall, the chapter aims to validate our findings presented in previous chapters in real-world scenarios.

Finally, in Chapter 6, we present our conclusions, where we critically present the results and possible improvements of the analytics platform.

Chapter 2

BACKGROUND

In this chapter, we explore the fundamental concepts related to location-aware games and the existing metrics classifications, and place of spatial metrics in the metrics classifications proposed by different authors. We establish a common understanding of such concepts and use it for performing a comparative analysis between the existing tools regarding the spatio-temporal support they provide and the features supported. We aim to use these results to collect the desired spatial capabilities needed in the context of location-aware games and tackle the current limitations exhibited by the existing tools.

2.1 Introduction

Within the varied landscape of games genres and applications (Avedon and Sutton-Smith, 1971; Salen and Zimmerman, 2004; Ritterfeld et al., 2009; Johnson et al., 2017; Pedreira et al., 2015; Duggan, 2017), location-aware games are games that take place in the real world space, using the location as an important aspect of the game, and often, but not always, require the user to move physically in space. Since the first variants of digitally-enabled treasure hunting games, such as geo-caching (Schlatter and Hurd, 2005; Duggan, 2017), location-aware games are gaining momentum and popularity. This culminated in the absolute success story of Pokemon Go, which has/had an unprecedented usage and user community (Smith, 2017). In the literature, a wide variety of terminology is used to denote location-aware games, or slightly more specific variants of them, such as location-based games (Schlieder et al., 2006), pervasive or urban games (Duggan, 2017), or geo-games (Schlieder et al., 2005), just to name a few. We discuss definitions, terminology, and differences in terminology in detail in Section 2.2. In this document, we use the term location-aware games, as most general term denoting games whose mechanics and rules are tightly coupled with the location, surrounding context, and/or geography. *Where* these games are being deployed and played clearly matters; the *where* may even turn out to be the most influential factor for the success or failure of location-aware games.

As for regular games, an important success factor for location-aware games is their playability. González Sánchez et al. defined playability as "a set of properties that describe the player experience using a specific game system whose main objective is to provide enjoyment and entertainment, by being credible and satisfying, when the player plays alone or in company" (González Sánchez et al. (2009), p. 67). This multifaceted concept involves usability, user experience and satisfaction, and the ability for users to achieve the game goals. As such, one of the measures of playability is game balance. Keeping a game "balanced" involves finding the right trade-off between important dimensions of a game, such as difficulty, challenges, progress, and incentives, which is ultimately reflected in the level of user engagement with the game. Game balance is especially challenging for game designers as games are getting more complex and sophisticated, both in a technical and narrative (i.e., storytelling) way. Therefore, it requires constant re-evaluation and follow up. Existing research has therefore focused on the evaluation of game balance. Jaffe et al. (2012) explored assessment methods for games in general, while Kiefer and Matyas (2005) put the focus on spatio-temporal design parameters involved in games. Central to the previous and other methods for assessing game balance is to (continuously, on a regular basis or on demand) capture data related to relevant aspects and features of the game. For example, the movement of a player in the game scenario, the players interactions and their characteristics, and game rewards achieved by players can help to investigate and detect potential imbalances and deficiencies in the game. Game data collection is not only relevant at the design stage, but most importantly while the game is being "played" (deployed), to allow game designers to investigate and monitor those game aspects and dimensions that are difficult to assess during design time, such as user engagement level, among others.

The amount of data collected in games can be potentially big, depending on how many aspects are to be tracked and the number of users/players playing. For example, monitoring progress, access to resources, and players' trajectories are some aspects in which game developers might be interested. In order to normalize and make it comparable, gathered data is processed by means of metrics(or game metrics). The term metric has been broadly used in different contexts and fields, such as urban studies (Reis et al., 2016), software engineering (García et al., 2017), ecology (Bhatti et al., 2017), and human computer interaction (Seaborn and Fels, 2015; Nacke and Deterding, 2017), but in all cases, the rationale behind metrics boils down to establishing a standard way of or common practice to describe certain relevant (high level) parameters of the game to be monitored, and allow comparison of the collected data.

The interplay between metrics and location-aware games is the main focus of this chapter. It is worthwhile to note that this research is not about game metrics in general (e.g., El-Nasr et al. (2013b)). As location-aware games are tightly-coupled to spatio-temporal dimensions, metrics for location-aware games will necessarily handle spatio-temporal characteristics of monitored data. Thus, geospatial data needs to be effectively collected, analyzed and computed. Spatial metrics have received little attention so far in the literature (with outstanding exceptions (Drachen and Schubert, 2013b)).

After an overview of the terminology and related concepts in Section 2.2, in order to establish a common understanding about location-aware games and metrics, the main contributions presented in this chapter are covered in Sections 2.3 and 2.4:

- In Section 2.3, we analyze and compare the spatio-temporal support in data handling and metric analysis for different existing commercial and academic game analytics platforms and tools, and their application to location-aware games. The comparative analysis evaluates whether such platforms offer the required features for collecting, processing and visualizing spatio-temporal data. Next, we discuss current limitations of these platforms and tools to set the basis for defining an analytics platform for location-aware games.

- In Section 2.4, we propose a classification of spatially-related metrics, framed in the realm of other non-spatial metric categorizations from the literature. Our intent is to establish the desired spatial capabilities, which would allow game developers to perform analysis that could otherwise not be done with non-spatial game analytics.

Finally, we provide a discussion in Section 2.5 where we emphasize the need of improved models for spatial metrics, and provide our vision of a platform for supporting such models. While the scope of this chapter falls within the two aforementioned contributions, our research objective is to use this analysis to set out the conceptual platform (later analysed in Chapter 3) to enable metrics definition and execution for location-aware games in terms of components, services and required functionality, accompanied with the implementation (provided in Chapter 4) of an operational analytical platform to support the definition, monitoring and computation of metrics for location-aware games.

2.2 Definitions and Terminology

In this section, we introduce related terms and definitions and set out a common ground for subsequent sections.

2.2.1 Gameplay

Gameplay is a popular term used throughout the industry (Fürnkranz, 2011), which "typically refers to the behavior of a game (e.g., the rules, difficulty, consistency, fairness, goals, and constraints), rather than the presentation of the game (e.g., graphics, artwork, and sound)" (Southey et al. (2005), p. 123). The role of gameplay is well known and, as such, game designers have put a considerable effort in designing, testing and refining gameplay. Quality assurance and playtesting, a testing technique during the game design process where players (testers) are asked to "think aloud" about their perception of different game aspects, are crucial parts of the design and development process. The evaluation of gameplay is a continuous process that leads to several cycles of adjustment and refinement during a game's life-cycle.

Prensky takes a motivational viewpoint in that gameplay is "all the doing, thinking and decision making that makes a game, either fun, or not. In a puzzle game, the gameplay is the physical and mental activities in the puzzles. In a shooter, it is the players and the opponents' speed and abilities. In a strategy game it is the available options and tactics" (Prensky (2002), p. 9). For the author gameplay is also a dynamic, evolving concept that needs to be continuously considered as it includes "not only providing engaging activities, characters, and situation but also balancing and constantly adjusting the game so that it continually keeps the player in the 'flow zone' " (Prensky (2002), p. 9).

Other authors took complementary views of gameplay. For example, game designers Rollings and Ernest (2003) defined gameplay as "one or more causally linked series of challenges in a simulated environment". This definition suggests that gameplay also includes the actions that players take to address the challenges and accomplish the game goals. Costkyan (2002) focused on the user/-player perception by stating that "a good gameplay keeps a player motivated and engaged throughout an entire game".

From the aforementioned, and as acknowledged by Kiili (2005), gameplay lacks a precise definition among the gaming community. It seems so broad that related literature works have addressed it from distinct viewpoints and perspectives. Nevertheless, there is a common belief of the importance of gameplay (or overall game experience), and it is undeniable that "its significance should not be underestimate" (Kiili, 2005). Typically, gameplay includes the game's goals and how to achieve them, the time to complete part(s) of the game, the "cost" (e.g., health, game money, resources), and the probability of succeeding. It is considered essential for the game to be "fun" and "enjoyable", and the primary concern for game developers is to improve gameplay. As such, they need to measure and monitor those aspects and features of interest of the game, in order to improve gameplay as a whole. The focus of this document —within the context of metrics for location-aware games—is precisely on those aspects/features of gameplay that either have spatio-temporal connotations or are suitable to be analyzed using spatio-temporal methods and techniques (as part of metric analysis).

2.2.2 Location-Based Games, Location-Aware Games and Geo-Games

Schlieder et al. (2006) defined location-based games as those that "involve body movements beyond figural space –that is, beyond the space of computer screens and small 3D objects". The authors highlighted the importance of locomotion or movement "in vista space, typically a single room or sports field, or in environmental space, such as a neighborhood or city" (Montello (1993) defines vista space as "space that can be visually apprehended from a single place without appreciable locomotion and environmental space as space too large and otherwise obscured to visually apprehend without considerable locomotion"), where the game takes place. Other authors define location-based games differently (Jacob and Coelho, 2011; Coulton et al., 2008). For them, these games do not necessarily require movement; they just need to take into account the user's location and/or environment. Anticipating our argumentation line, we envision a more exclusive definition of location-based games where user's movement is an important but not mandatory aspect of the game.

Kiefer and Matyas (2005) distinguished between location-based games and geo-games, where the latter are a specific case of the former. According to the authors, in geo-games “a fixed number of players move between a fixed number of locations taking up and putting down resources when they reach a new location. Resources cannot move around without any involvement of a player, which is one basic constraint for geo-games”. The authors call this constraint “spatial coherence”, and declare it a defining characteristic of geo-games. Apart from the spatial coherence, Schlieder et al. (2005) identified temporal coherence as a second constraint in geo-games. Temporal coherence asserts that performing an action needs time at least as long as the synchronization interval, a predefined constant for the game. Looking at the particular context of this definition though, the authors were introducing a framework for location-based games covering boards games. This context (board games) explains the (perhaps overly) restrictive definition (i.e., fixed amount of players, focus on picking up or dropping resources, resources that can only be moved by players, turn-based games). While a general appreciation of spatial and temporal coherence as part of the game mechanics makes sense, we do not share the focused view and narrow definition of geo-games, which we think should be open to a larger array of location-based games besides (geospatial) boardgames.

To avoid confusion with the restrictions associated, at least by some authors, with location-based and geo-games, we use the term location-aware games in this document, going beyond of players’ and resources’ position and/or movement to cover an ample range of games where the game, and the players are *aware* of their location and (possibly) surroundings (i.e., their environment and the objects in it), and vice versa, the surrounding environment is *aware* of players. This broader definition includes any games where location plays a role, and thus includes location-based and geo-games, as well as other games where location (and environment) is important, such as pervasive or urban games (Duggan, 2017). In the remainder of this document, we embrace the term location-aware game, unless the contrary is explicitly stated.

2.2.3 Metrics in Location-Aware Games

In its mathematical interpretation, the term “metric” is equivalent to “a nonnegative function describing the distance between neighboring points for a given set” (Weisstein, 2017). Regardless of the target domain or discipline, the recurrent and invariable aspect of a metric is that is a function, i.e., it processes gathered data to produce comparable results (indicators) for evaluation, monitoring and decision-making processes. Applied to the gaming context, we find the following definitions in literature (Table 2.1):

Table 2.1: Key aspects of metric definitions extracted from the literature.

Authors	**Metrics**
Tychsen and Canossa (2008)	“are numerical data obtained from user interaction with games”, “denote a standard unit of measure”, and “are utilized for quantitatively measuring and evaluating processes”.
Drachen and Canossa (2009)	“are instrumentation data about the user behavior and user-game interaction”, and “provide detailed quantitative information about the player (user) behavior”.
Hochleitner et al. (2015)	“are interpretable measures of something, whereas telemetry is the raw data”, and “represent telemetry data that have been transformed somehow”.
Fürnkranz (2011)	“are the aspects of gameplay of interest to the designer”.
Medler et al. (2011)	“are monitoring player behavior (e.g., logging in-game events)”.
Kaner and Bond (2004)	“are measurement functions”.
Stanton et al. (2014)	“are distance functions”.

Table 2.1 shows a mixture of interpretations of what a metric is. In game user research and games telemetry (Tychsen and Canossa, 2008; Drachen and Canossa, 2009; Hochleitner et al., 2015; Fürnkranz, 2011; Medler et al., 2011), a metric is termed both as “data used in” and as “data collected for evaluating” an aspect of the game. As such, the term metric is equated to data collected or data obtained, suggesting that metric (as a function) and (input or output) metrical data are used interchangeably. A possible reason might be that in some cases, raw acquired data is sufficient as a comparable metric, and further data analysis and computation is not required. In such cases, it may be difficult to distinguish between metric and metrical data.

To avoid aforementioned terminological inaccuracies, and establish and delimiting the scope and understanding of what a metric is in the context of location-aware games, we first provide clarity on what we understand under “metric”. We consider the most general definition for metrics is the mathematical definition, which is coherent with the one in the software engineering field (Kaner and Bond, 2004). This definition reflects the fact that collected (metrical) data is raw information, which in most cases is used for further evaluation rather than being the metric itself. To clarify, consider as an illustrative example a metric that provides a ratio of values, e.g., amount of movements per minute. Clearly, such a composite metric needs to be calculated from basic data, i.e., movement derivation from singular sensor values, temporal average computation. Notice that data is dynamic but re-usable (e.g., sensor data), whereas metrics are calculated for a specific purpose (e.g., amount of movements per minute, amount of lateral movements) and often static (i.e., execution of the same function over and over). Furthermore, normalization of data among sessions, or even among games, through metrics is what makes comparison of data possible.

Metrics are related to gameplay in the fact that they serve to measure certain parameters of the game that are considered to, at least partly, assess gameplay. While re-usable metrics are conceivable (e.g., total distance covered), it is unrealistic to think of a set of predefined metrics for measuring and assessing any facet of gameplay, in any game. Especially in location-aware games, where location and spatial context may notably influence which aspects of gameplay are going to be monitored and how. Consequently, the ability to support the design and creation of custom metrics is a quintessential feature for game analysts, designers and developers, and a clear separation of concerns between metrics and metrical data is therefore essential to allow game developers the degree of flexibility and customization required for metric design and creation. In this document, we focus on metrics, and associated geospatial data, which are relevant for games with a spatio-temporal dimension; in other words, metrics for location-aware games.

With the concepts of “metrics” and “metrical data” clarified, and our focus on their spatio-temporal dimension outlined, we subsequently study the support in existing commercial and academic platforms for geospatial game metrics and analysis.

2.3 Geospatial Support in Current Games Analytics Platforms and Tools

Given the importance of measuring different aspects of games during design, development and operational phases, several companies and research institutions have invested significant resources in platforms and tools for metrical data collection and analysis. Nevertheless, these commercial and academic tools are strongly influenced by specific purposes and aims, or are general in scope and cover only main scenarios and use cases. In either case, none of the surveyed tools were specifically designed for location-aware games, that is, spatial and/or spatio-temporal features were not natively included even though some support and workarounds may exist. To the best of our knowledge, there is no tool or platform to specifically handle metrics for location-aware games. In this chapter, we conduct a comparative analysis of popular games analytics tools to examine what is currently covered and, next, discuss limitations and missing features in the realm of location-aware games.

2.3.1 Background for Tool Comparison

The comparative analysis allowed us to identify a set of commonly available features to ideally support game analysts and developers to be able to compute metrics. While analyzing the tools, we identified these features and grouped them into four functional areas, which schematically form a conceptual architecture shown in Figure 2.1 that frames the process for tool comparison.

Typically, most of the surveyed platforms and tools follow a client-server architecture, in which the game clients collect raw data and transmit it to the server platform for data storage and processing (i.e., metric computation). In the context of games analytics, a client-server architecture eases computation and further access to data by keeping data centralized and accessible through an abstraction layer, which typically takes the form of service endpoints or data access APIs. Such centralized data availability is particularly important in the case of multi-player systems, where data synchronization among multiple players needs to be supported. Additional reporting and visualization of processed data (metric outputs) are usually developed as extensions or add-on components to the analytical platform. The general conceptual architecture shown in Figure 2.1, which is not yet intended to hypothesize an operational analytical platform for geospatial metric processing, helps us to delimit the functional areas of existing and desirable features for the comparative analysis.

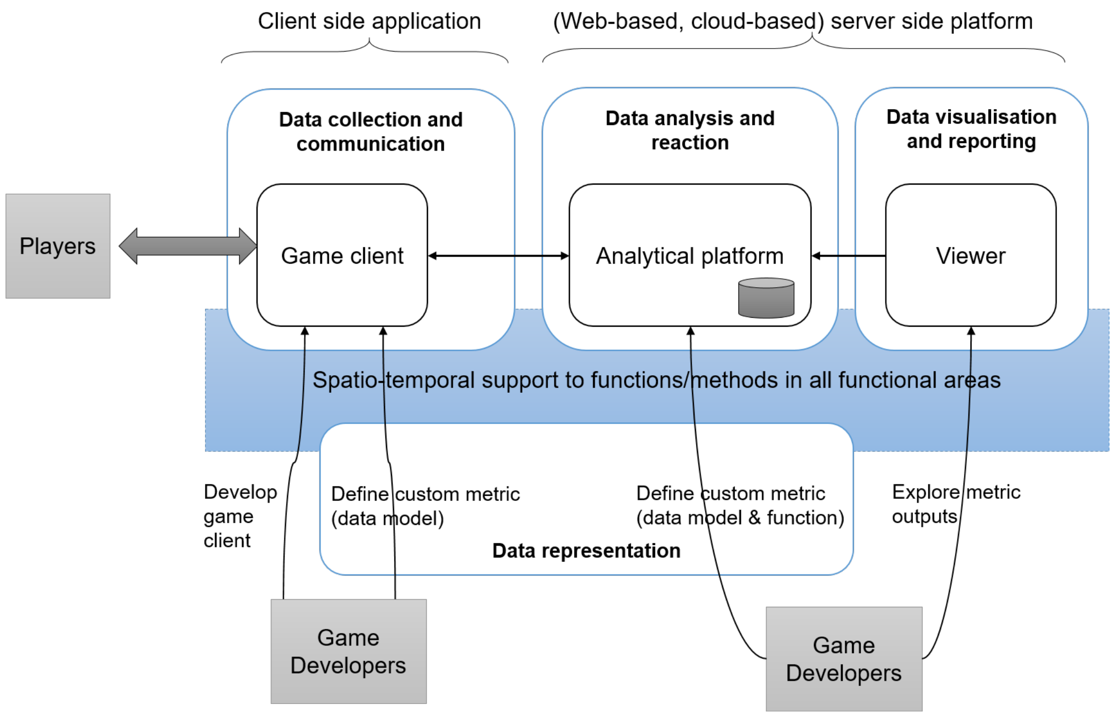

Client side application
(Web-based, cloud-based) server side platform
Data collection and communication
Data analysis and reaction
Data visualisation and reporting
Players
Game client
Analytical platform
Viewer
Spatio-temporal support to functions/methods in all functional areas
Develop game client
Define custom metric (data model)
Define custom metric (data model & function)
Explore metric outputs
Data representation
Game Developers
Game Developers

Figure 2.1: Schematic architecture for (location-aware) games analytical platforms and tools.

The first functional area is concerned with data collection and communication. Both are basic operations of game clients, which collect raw data for monitoring and tracking user interactions and behavior with(in) the game and transfer it (involving data transformation and wrangling if required) to the analytical platform. These two actions together are often termed as telemetry in the game community (Drachen et al., 2013b), whereas the GIS community tends to use single terms: data collection (or data gathering) plus data transferring. Game developers implement game clients coupled to specific game engines (e.g., Unity) or operating systems (e.g., Android, iOS) using custom SDKs (Software Development Kit) associated with the analytical platforms or tools.

The second functional area, data representation, is shared between the client and server side developments of analytics platforms. It deals with the data model to structure the monitored phenomena of the game. Game clients capture raw data and instantiate data models accordingly. Analytical platforms compute metrics based in these shared data models, either as inputs or outputs.

The third functional area, data analysis and reaction, resides at the server side; they are the main tasks of a game analytical platform. The term analytic refers to metrics computation i.e., the execution of distance functions over collected data. Reaction is concerned with the ability of the platform to *react* when certain conditions are met. Examples of reactions are callback and notification mechanisms, or programmed actions that are being triggered subject to metric outputs.

Finally, data visualization and reporting are the focus of the fourth functional area. Game developers are often provided with monitoring and visualization tools to explore processed data in varied forms (visual, table-form etc.) in order to make informed decisions to improve overall gameplay.

All features of the surveyed tools and platforms fall within one of these four functional areas. Beyond feature categorization, these functional areas can be considered as main building blocks for conceptually defining an operational platform for metric computation for location-aware games, in which spatial and/or spatio-temporal support is a cross-cutting layer, thereby ensuring support for handling spatial and/or spatio-temporal data and techniques in any of these functional areas.

2.3.2 Comparative Analysis

In the following Tables 2.3–2.7, we compare popular platforms and tools for game metric analytics, ordered by functional area. Table 2.2 summarises the relation between functional areas and their features. Data representation is shared by the client- and server-side functional areas.

Table 2.2: Functional areas and contained features for game metrics.

Functional Areas	**Features**
Data collection and communication	• Data collection strategies • Data communication strategies • Client-side development support
Data representation	• Default event (data model) types • Custom event types definition • Spatial support (for data modelling)
Data analysis and reaction	*Analysis* • Default metrics and games analysis • Custom metrics definition • Spatial support (for analysis) *Reaction* • Reactive rules • Spatial support (conditions & actions)
Data visualisation and reporting	• (Open) Data Access • Visual analytics • Spatial support (for visual analytics)

With respect to the data collection and communication functional area, the supported features encountered in the surveyed tools are described next, and Table 2.3 summarizes the type of support for data collection and communication per feature:

- **Data collection.** Strategies for data collection can be either time-based or event-based (Jimenez et al., 2011). The differentiation between the two is mainly determined by the fact that data associated with the former strategy is produced at high rates or does not exhibit high variabilities over short periods of time, while on the contrary, data required for event triggering is usually collected and produced "eventually", based on a certain frequency or as a result of an action in the game. The latter is therefore more meaningful to game developers in the context of the game. An example of time-based data collection is capturing user location at a certain frequency, while an example of event-based data collection is the usage of a given weapon, which can only be collected in case the player changes and selects the particular weapon. Game analytics platforms usually do not distinguish between time- and event-based data collection, which is considered a responsibility of the game developer; they only provide facilities for defining event types and communicating captured data (see further on), independent of their data collection strategy. Indeed, all reviewed game analytical tools follow this approach. We thus do not include data collection strategies in Table 2.3. Nevertheless, the way data is collected may influence how it can be processed, and therefore influence the choice of analytical methods. We visit this later on in Section 2.3.3.

Another way of qualifying data collection is based on *who*, rather than *how*. Drachen and Schubert (2013a) categorize collected data into player-derived data and system-derived data. Player-derived data refers to data that, at least partly, captures (some) player behavior. Four dimensions are hereby considered relevant: who, what, when and where. Examples of player-derived data include players' items owned, position, trajectories, etc. We go into more detail on the spatial dimensions of player-derived data when discussing the data representation functional area further on in this section. System-derived data refers to data generated by the game, and is useful for e.g., monitoring technical issues (e.g., network balancing, bug tracking) or the game as a whole. Game analytics engines do not distinguish between player- and system-derived data; it is the responsibility of the game developer to define appropriate event types and communicate the relevant data to track.

- **Data communication**. It indicates how gathered data is imported or fed into the analytical tool. As mentioned earlier, most of the surveyed tools follow a common architectural pattern: a client part collects data, and a server part performs analytical tasks and provides services for getting the collected data and store it permanently into repositories. Mechanisms for data communication, i.e, how gathered data is transferred from the client to server, can be roughly classified in two groups: data streaming or data upload. In case of data streaming, events are immediately streamed to the server, as real-time reflection of events in the game is important (e.g., real-time game strategy) and overall system performance is not relatively affected by the frequency of data transmission. It has the advantage of enabling on-the-fly analysis over the streamed data.

  On the other hand, in the data upload group, bits of data are gathered over a certain (usually short) period of time, packaged together, and sent to the server. It is useful in cases where events occur at high rates and/or involve large amounts of data, and therefore immediate and frequent unitary communication might disrupt system performance. On the downside, the lack of live streamed data limits the possibility for real-time data analysis. Indeed, this approach mainly serves analysis a posteriori, and often includes the use of log files (or databases), and batch bulk updates to a server platform for non-real-time analysis. The main reason for using this strategy for data communication is that the amount of data produced could lead to undesirable levels of network load or decreased performance.

- **Client-side development support**. Supporting tools for game development facilitate the integration of data communication protocols in game clients. This is essential to free game developers from knowing the particularities of the ways to transfer collected data to the analytics back-end platform. Commercial tools mostly provide software development kits (SDKs) for target mobile operating systems (e.g., iOs, Windows, Android) and game engines (e.g., Unity, GameMaker Studio, etc.). REST Application Programming Interfaces (APIs) are another way to support game development, which are supported in varied programming languages (e.g., JavaScript, C#, Java). In general, SDKs and REST APIs are often viewed as secondary features but are extremely useful to make developers' life easier for, e.g., being in compliance with the protocol to validating and submitting collected data.

Table 2.3: Comparison of data collection and communication related features among academic and commercial tools for game analytics. NA/NS = Not Applicable, Not Specified or Unknown.

Tool/Citation	Data Communication	Client-Side Development Support
WebTics (McCallum and Mackie, 2013)	Streamed events	HTTP clients
TRUE (Kim et al., 2008)	Data Upload	NA/NS
DataCracker (Medler et al., 2011)	Streamed events	NA/NS
Skynet (El-Nasr et al., 2013c)	Streamed events	NA/NS
GameAnalytics (gam, 2017)	Streamed events	SDKs for Android, iOs, Xamarin, Unreal Engine, Unity; REST API
HoneyTracks (hon, 2017)	Streamed events	SDKs for Android, iOS, Unity C#; REST API
Xsolla (xso, 2017)	Data Upload	HTTP
GameGuts (Albuquerque et al., 2014)	Streamed events	Java
DeltaDNA (deltadna, 2017)	Streamed events	SDKs Unity, Android, iOs, GameMaker Studio; REST API

Table 2.4 summarises the type of support for the functional area of data representation in the surveyed tools, according to the following features:

- **Default event types**. Most surveyed platforms include the definitions of events as logical data structures (data models), called event types. Some of them provide default event types, of which each has an associated data model and often, but not necessarily, related processing methods and/or predefined metrics. For example, GameAnalytics (gam, 2017) provides a "business" event type that is used to track (and validate) real-money transactions in games. Based on this business event type, GameAnalytics supports various types of revenue metrics, such as average daily revenue per daily active user or per paying user.

  It is worth noting that the way game analytics platforms offer event types influences their extensibility and customizability. The most common logical data model for event types are records, with a fixed number of fields that represent the monitored phenomenon. Record-based event types are useful in many cases, where the data model is relatively static, and a number of event types are predefined for being used in previously known metrics. Another commonly used logical data model is based on arrays of key-value pairs (e.g., DataCracker - see Medler et al. (2011)), which are more versatile and customizable data models than those based on data records mentioned above. From the surveyed game analytics platforms, McCallum and Mackie (2013); El-Nasr et al. (2013c); hon (2017); gam (2017) provide record-based; Medler et al. (2011); Kim et al. (2008); deltadna (2017); Albuquerque et al. (2014) provide key-value pairs [1] like event types; and xso (2017) provides a combination of both.

- **Custom event types**. Contrary to default event types, custom event types allow game developers to define their own data model to handle any type of monitored aspect of the game. Custom event types provide high-level support for game-specific events, and the (composite) data they require. This feature is linked to the definition of custom metrics (see the third functional area below), as the combination of both gives game developers the freedom to extend the capabilities of a game analytics platform to practically monitor and compute any facet of a game. The support for this feature is varied among the analysed platforms, although most of them restrict the creation of new data models to those based exclusively on simple data types (i.e., boolean, string, integer). Support for more complex and nested data structures was practically inexistent. Reasons might be the absence of a flexible processing model and/or rigid storage schemes to handle data structures beyond that of a sequence of simple data types.

- **Geospatial data support**. As mentioned earlier, Drachen and Schubert (2013a) categorised collected data for game analytics into player-derived data and system-derived data. From the four dimensions for player-derived

[1]We consider structured documents like XML, as in the case of Kim et al. (2008), or even unspecified structures that are dynamic (which is not always documented in commercial system) and similar to arrays of key-value pairs for the purpose of this comparison.

data (who, what, where, when), two are particularly relevant from the spatio-temporal point of view: “where is *this* happening?” and “when (at what time) is *that* happening?”, where this/that is any relevant (player-specific) event in the game. For the former (where), the authors pointed to “the spatial position, user location, movement, speed, orientation”. In case of games where no avatars exist, location may refer to the screen coordinates when users touch or swipe on the gaming device. The latter (when) looks for a temporal reference, which may refer to actual or in-game time, point in time, time interval, timespan, etc. As space and time are so tightly integrated in games, custom data models for event types should ideally be able to natively manage and store data structures to support the aforementioned spatio-temporal characteristics.

That is why we envision the support for geospatial data as a cross-cutting layer (see Figure 2.1), i.e., where spatio-temporal and/or spatial characteristics of collected data are supported, either by default or by custom event types. In practice, though, native support is scarce, and game developers are mostly limited to create simple representation of space: pairs of coordinates. To do so, we found two approaches in between the tools that provide some support for capturing geo-spatial data. A first group of tools (McCallum and Mackie (2013); Kim et al. (2008); Medler et al. (2011); El-Nasr et al. (2013c); deltadna (2017) is able to capture coordinates as a pair of simple attributes (location/space attributes), which is far from being able to really handle geospatial data. A second group (gam (2017); xso (2017); Albuquerque et al. (2014) does not provide explicit support, but is possible to store the coordinates by mean of other mechanisms, usually custom attributes. In these cases it is up to the developer to implement the best way to store spatio-temporal information.

Table 2.4: Comparison of data representation related features among academic and commercial tools for game analytics. NA/NS=Not Applicable, Not Specified or Unknown.

Tool/Citation	**Default Event Types**	**Custom Event Types**	**Geospatial Data Support**
WebTics (McCallum and Mackie, 2013)	NA/NS	Yes(simple)	Location/space attributes
TRUE (Kim et al., 2008)	NA/NS	Yes	Location/space attributes
DataCracker (Medler et al., 2011)	NA/NS	Yes (simple)	Location/space attributes
Skynet (El-Nasr et al., 2013c)	NA/NS	Yes	Location/space attributes
GameAnalytics (gam, 2017)	Resource, Business, Progression, Error	Yes (simple)	via custom attributes
HoneyTracks (hon, 2017)	User, Session, Business, Social	Yes (simple)	NA/NS
Xsolla (xso, 2017)	Business, User, System	Yes (simple)	via custom attributes
GameGuts (Albuquerque et al., 2014)	NA/NS	Yes (simple-DSL)	via custom attributes
DeltaDNA (deltadna, 2017)	Many types	Yes (simple)	Location/space attributes

For the third main functional area, data analysis and reaction, we separately discuss data analysis (Table 2.5) and reaction (Table 2.6). Relevant features for data analysis extracted from the survey are:

- **Default metrics and games analysis**. All game analytic platforms support a set of predefined metrics, algorithms and analysis tools that are available to process collected data. Predefined metrics are re-usable and possibly customizable metrics that address commonly measurable game aspects and are to be used out of the box. Most (commercial) platforms focus on monetary aspects (e.g., virtual money spent per session, amount of purchases), yet some also offer other default metrics (e.g., related to user engagement, mean time spent by a user in the game per day, social aspects such as invites and shares, or interaction with interface elements). Next to default metrics, all platforms also provide algorithms and analysis tools that can be used to analyze captured data, and/or to build more complex (custom) metrics with (see next bullet). For example, descriptive statistical methods are commonly offered (measures of central tendency, e.g., average, median; measures of variability, e.g., standard deviation, etc.); as they are common, we do not show these in Table 2.5. Further algorithms and analysis tools include funnels (i.e., a series of steps towards a certain goal, a kind of behavior analysis), predictive analysis, events correlation, frequency analysis, segments analysis (the user base can be segmented to target the analysis on such segments, or make metrics comparison between different segments, i.e., paid users, and trial users), A-B test (a method for comparing two versions of the game, to determine which one performs better regarding some criteria), OLAP Cubes (a technique for analyzing multi-dimensional data), and Funnels (a technique for describing the navigation path followed by users in a system). Note that visual analytics are considered within the next functional area (Table 2.7).

- **Custom metrics**. Custom metrics allow developers to capture game-specific analytics, or extend already available metrics. A custom metric is hereby understood as a custom-defined distance function with an arbitrary level of complexity. Developers have a certain specification mechanism to their disposal (e.g., a restricted programming language), as well as the default game analytics tools and methods (see previous bullet) available to define their custom metric, and may extend default or other custom metrics. It is similar to coding user functions in any programming language. Custom metrics may be using input data captured as default event types, yet often they are based on custom event types as the underlying data model, required to model or structure a particular phenomenon. As such, the definition of custom metrics is linked to the ability to define custom event types. The possibility to specify new event types and custom metrics, beyond the default ones, is obviously a desirable characteristic for an analytics platform.

- **Spatial analysis**. As mentioned when discussing custom event types, we consider geospatial data support, and spatial analysis support as cross-cutting layers. With respect to metrics, this refers to regular metrics supporting geospatial data types, and specific spatial analytical technique or method supported or integrated in the analytics platform. As Table 2.5 shows, native support for spatial analysis is practically inexistent.

Table 2.5: Comparison of data processing and analysis related features among academic and commercial tools for game analytics. NA/NS = Not Applicable, Not Specified or Unknown.

Tool/Citation	**Default Metrics and** ***Game Analysis***	**Custom Metric**	**Spatial Analysis**
WebTics (McCallum and Mackie, 2013)	Simple events correlation, frequency analysis	NA/NS	NA/NS
TRUE (Kim et al., 2008)	Video referencing	NA/NS	NA/NS
DataCracker (Medler et al., 2011)	NA/NS	NA/NS	NA/NS
Skynet (El-Nasr et al., 2013c)	Yes	NA/NS	Yes
GameAnalytics (gam, 2017)	A/B-Tests, Funnels	NA/NS	NA/NS
HoneyTracks (hon, 2017)	Custom segments, comparison, A/B-Tests	Yes	NA/NS
Xsolla (xso, 2017)	Aggregations, etc.	NA/NS	NA/NS
GameGuts (Albuquerque et al., 2014)	Aggregations, custom	Yes	NA/NS
DeltaDNA (deltadna, 2017)	Funnels, Custom segments predictions, etc.	NA/NS	NA/NS

Once metrics or analytical functions are computed, the outputs are usable in immediately, where game developers may program in-game reactions according to observed behavior (e.g., of players), or a posteriori, where output data from analysis is used for inspection, visualization or reporting. We treat the latter in more detail in the next functional area, and focus here on the reaction dimension.

Reactions from the game system often take the form of programmed rules that usually define what actions to take (storage, notification, altering game parameters or mechanics, etc.) when certain conditions, which often depend on the metric outputs, are met. As such, reactive behaviour can be regarded as reactive rules, either programmed or declarative, which are composed of conditions and actions. The ultimate aim for a game developer or analysts is to improve gameplay by provoking and/or incentivizing player behavior change through timely feedback/actions to players. Related features for reactive behaviour are depicted in Table 2.6:

- **Reactive rules**. When certain logical conditions are met over the results of data analysis, more specifically metric outputs, the system has the ability to react by triggering purposeful actions. Typical actions with a specific aim are data storage (where possibly only a subset of the outputs is stored; supported by all surveyed platforms and thus not mentioned in Table 2.6) or notifications (e.g., to game developers or players). Further actions are often required to realize detailed, in-game consequences of observed behavior, for example to perform additional data transformation over the outputs of the metric or to trigger additional analytical tasks in cascade. In such a case, the ability for a game developer to *program* custom reactive actions is essential to optimize the game, and possibly trigger game mechanics updates. Obviously, actual alterations to the game are within the real of the game implementation, not the analytics platforms. If a platform supports some form of reactive rules (i.e., logical conditions and consequent actions), we specify the type of action (storage, notification, or custom) in Table 2.6.
- **Geospatial support**. Output data from metrics may contain geospatial data structures, such as trajectories or polygons (e.g., representing the geographical area covered by a player). Such geospatial data may be subject to meet certain spatial or spatio-temporal conditions (e.g., within, intersect, crosses) to check, for example, whether a player's traveled distance during a game session is greater than 1 km, or the percentage of time a player remained inside a delimited area (often called "fence") is above average. Similarly, triggered actions can also be spatially-enabled. For example, actions

that redistribute monitoring points or recalculate (a metric's) geofencing areas. For convenience, we split this feature into two columns in Table 2.6: geospatial support for conditions and actions, respectively.

Table 2.6: Comparison of reaction related features among academic and commercial tools for game analytics. NA/NS = Not Applicable, Not Specified or Unknown.

Tool/Citation	**Reactive Rules**	**Geospatial Support (*Conditions*)**	**Geospatial Support (*Actions*)**
WebTics (McCallum and Mackie, 2013)	Notification	NA/NS	NA/NS
TRUE (Kim et al., 2008)	NA/NS	NA/NS	NA/NS
DataCracker (Medler et al., 2011)	NA/NS	NA/NS	NA/NS
Skynet (El-Nasr et al., 2013c)	NA/NS	NA/NS	NA/NS
GameAnalytics (gam, 2017)	NA/NS	NA/NS	NA/NS
HoneyTracks (hon, 2017)	NA/NS	NA/NS	NA/NS
Xsolla (xso, 2017)	NA/NS	NA/NS	NA/NS
GameGuts (Albuquerque et al., 2014)	Custom	NA/NS	NA/NS
DeltaDNA (deltadna, 2017)	NA/NS	NA/NS	NA/NS

The fourth and last functional area comprises data visualization and reporting, and includes the following features (Table 2.7):

- **Data access**. Open access to data is useful to allow users to get access and query observed data, metrics results, or both, for further analysis by third-party tools. While this allows to harness the (extended) analytical power of external tools, in case the capabilities of the game analytics platform prove to be too general and/or insufficient, it also hinders integrability for real-time analysis and reactive rules generation. Indeed, access to analytical/output data is usually done for later off-line analysis. We broadly contemplate two strategies for data access: the first is based on on-line interfaces for querying analytical data (e.g., service end-points), and the other to enable a full export or bulk download.

- **Visual analytics**. It differs from games analysis in that the latter is thought to be part of metric computation to process collected/observed data. Visual analytical tools are situated at the end of the analytical pipeline (Figure 2.1), taking metric outputs as input (possibly after data transformations) to carry out in-depth analysis. Visual analytics comprises a large and varied arsenal of data visualisation and inspection techniques (Dill et al., 2012; Fry, 2008), in order to gain new insights and discover new patterns for game developers, which would otherwise be difficult to detect and/or go unnoticed. Most of the surveyed tools and platforms only support basic techniques, such as different types of charts to visually summarise and aggregate data.

- **Geospatial support**. Similar to the geospatial support for game analysis presented earlier, here we refer to specific spatial and spatio-temporal support for visual analytics, commonly referred to in literature as visual geoanalytics (Andrienko et al., 2010). Examples include visualization for moving point datasets (e.g., players position in multiplayer games) or time series (e.g., virtual land ownership or visibility over time). This area of research is quite extensive and developed methods and techniques have been proven to be effective for decision making (Andrienko et al., 2007). Only a few analytical tools and platforms (e.g., El-Nasr et al. (2013c)) offer basic support, in the form of map visualisation of simple data types like points or heatmaps (density maps of spatial frequency).

Table 2.7: Comparison of data reporting and visualization related features among academic and commercial tools for game analytics. NA/NS=Not Applicable,Not Specified or Unknown.

Tool/Citation	(**Output**) **Data Access**	**Visual Analytics**	**Geospatial Support**
WebTics (McCallum and Mackie, 2013)	NA/NS	NA/NS	NA/NS
TRUE (Kim et al., 2008)	NA/NS	Reports	NA/NS
DataCracker (Medler et al., 2011)	NA/NS	Charts, filtered reports	NA/NS
Skynet (El-Nasr et al., 2013c)	NA/NS	Charts, filtered reports	Point maps, heatmaps
GameAnalytics (gam, 2017)	Yes	Charts, filters, comparison	NA/NS
HoneyTracks (hon, 2017)	Yes	Charts, reports, cohorts, comparison	NA/NS
Xsolla (xso, 2017)	Yes	Charts, reports	NA/NS
GameGuts (Albuquerque et al., 2014)	NA/NS	NA/NS	NA/NS
DeltaDNA (deltadna, 2017)	Yes	Charts, reports, funnels, predictions	NA/NS

2.3.3 Limitations

Looking at Tables 2.3–2.7 we identify some limitations of the reviewed analytical tools and platforms with respect to their geospatial support, and hence, to the creation, computation and visualisation of metrics for location-aware games.

A first general observation is that virtually all columns that refer to “geospatial support” are empty, i.e., geospatial characteristics and techniques are poorly supported. This is a direct result of the fact that none of the surveyed tools and platforms were specifically designed for defining and computing spatio-temporal metrics. Nevertheless, while built-in support is largely missing, some platforms allow to treat simple geospatial concepts, such as data models to capture and store location and points. In some platforms (Table 2.4), the definition of a data model for capturing location (e.g., player position) is straightforwardly possible, since a position is a rather simple data structure (i.e., event type) composed of a pair of coordinates. In most cases, default event types suffice as a point data model. Other types of more complex geometries, such as lines, poly-lines, or polygons are not mentioned or not supported. This suggests that the most popular game analytical platforms are merely restricted to native point geometries.

Nevertheless, the ability to structure and manage a player's position does not necessarily imply that this bit of spatial information is properly exploited and analyzed. Beyond of the simple visualization of player position onto a map (Table 2.7), which represents a simple analysis method over point geometries, more advance spatial analysis techniques like topological operators, are absent. This deficiency—the lack of a geospatial toolbox and the support for complex geometries beyond point data models—considerably limits the use of the surveyed analytical platform for defining advanced custom metrics for location-aware games, even though (non-spatial) games analysis and metrics are generally well covered (Table 2.5). We therefore argue that, to the best of our knowledge, spatio-temporal analytical methods for metric computation are not the focus, and thus not properly supported, in the current game analytical tools. Simply put, spatial analysis is passed over in favor of mainstream analytics methods for traditional metric scenarios in games (Table 2.5).

Another observation is whether there exists a relation between data collection strategies and the subsequent type of game analysis supported. For example, if a platform supports streamed events as an strategy for data communication (Table 2.3), does it mean that stream computing (e.g., Garofalakis et al. (2016)) is also covered? Does the platform meet the demand for fast monitoring and storage of huge amounts of data in real-time? Unfortunately, we do not observe a clear pattern with respect to this matter. Indeed, those platforms that claim to support streamed events do not exhibit any type of real-time data analytics. Regardless of the data communication strategies (data streaming, data upload), the type of game analytics supported is quite homogeneous over the tools but constrained to well-known, off-line, and non-spatial techniques for metric analytics (see El-Nasr et al. (2013b)), especially for monitoring and measuring any variable related to the actual behavior of a player (gameplay). We may consequently argue that real-time analytics and stream processing is a concern that requires support throughout all functional areas.

We have just stated that the type of game analytics supported is quite homogeneous, in the sense that most tools and platforms offer pretty much the same core set of game analysis techniques (see Table 2.5). The other side of the coin is the support for user-defined or custom metrics. Here, the degree of personalisation is quite marked, resulting in two opposed groups of tools. One group (McCallum and Mackie (2013); Kim et al. (2008); Medler et al. (2011); El-Nasr et al. (2013c)) does not consider customisation at all, and thereby, custom metrics and associated data representations are not configurable (or the process of customisation introduces a significant complexity and additional effort from the game developer). The other group (gam (2017); hon (2017); xso (2017); Albuquerque et al. (2014); deltadna (2017)) supports both custom event types and custom metrics, which makes sense for the specification of user-defined metrics. Obviously, as we mentioned earlier, custom metrics as the way to specify new event types and custom functions (metrics), beyond of default ones, is a desirable characteristic for game analytics platforms.

In conclusion, what is missing for supporting metrics oriented to location-aware games? Responding to this question is challenging since there is no native analytical platform for location-aware games. Paradoxically, the ever-increasing importance of the where and location in varied game genres (e.g., urban games, adventure) situated in real places, is not being reflected with the same intensity in the research and development of analytical platforms for spatio-temporal metrics computation. Current trends suggest that both commercial and academic game analytical tools are shyly glancing at the geospatial field, but there remains a lot to be done, especially to cover (advanced) spatial analysis throughout all functional areas. In the next section, we attempt to identify what spatial techniques are required, by proposing a categorisation of spatial metrics, in order to facilitate, promote and push for game analytical platforms (gradually) supporting various types of spatio-temporal metric computation.

2.4 Classification of Metrics for Location-Aware Games

From the argumentation of the previous section we can summarise that location-aware games and games in general share many aspects, specifically what concerns measuring gameplay. This makes perfect sense as the improvement of gameplay is a recurrent objective for game developers and analysts regardless of the nature of game. However, we also remarked important differences between the two, especially due to the fact that spatial analysis deals with the real-life dimensions of play, i.e, the actual location, environment and geography are defining characteristics of the mechanics of location-aware games. Logically, spatial analytics techniques are required to compute and monitor spatial-related aspects of the game. Nevertheless, our analysis showed that this is precisely the missing feature in the game analytics platforms and tools considered in Section 2.3.

In this section, we propose a classification of spatially-related metrics for location-aware games and position it with respect to other existing classifications of non-spatial metrics. By doing so, we pursue two main objectives:

1. To make spatial metrics visible by placing them at the same level and embedding them within the larger context of metrics for games. Higher visibility may lead to the next generation of analytics platforms to support native spatial data and spatially-related metrics, increasingly recognising the strategic importance of managing location and spatial concepts in games.

2. To bring the gaming and GIScience research communities' attention to location-aware games as an interesting and relatively unexplored research field, with ample possibility for additional research and practical tool development that may directly impact applications in a wide variety of application fields, such as (smart) cities (e.g., urban games), health (e.g., games to stimulate physical activity) or sociology (e.g., games to stimulate social inclusion).

2.4.1 Existing Metrics Classifications

As emphasised in Section 2.2, we focus on this document on metrics related to gameplay, i.e., to evaluate game design, user experience and user behaviour. Gameplay considers the player as the center of the metrics process. Drachen et al. (2013a) exemplified this vision in their high-level classification for game metrics (top part of Figure 2.2). Looking at the Gameplay box, the authors classified metrics for gameplay into three main groups: Interface, In-game and System. Interface metrics are about interactions of the player with the game interface (e.g., interactions with menus and buttons, mouse sensitivity; In-game metrics cover players' actions and behaviors during the game (e.g., players' position, their interactions with game elements such as objects or resources); and System metrics cover the actions game systems initiate to respond to player actions (e.g., refresh game resources once a player reaches a pre-defined set of conditions).

In the literature, there exist various proposals for game metrics classification mainly related to Interface and In-game groups, as these two accrue most of the existing examples of game metrics. For example, Tychsen and Canossa (2008) proposed four specific game metrics, namely navigation, interaction and narrative metrics, which fall into the In-game group, and interfaces metrics, which are quite similar to the Interface group. Besides, the authors strongly linked these types of metrics to data collection strategies. Navigation metrics often require data collection via time-based strategies (at regular rates), while interaction metrics are well suited for event-based data collection strategies. Nacke et al. (2014) took a more social perspective of in-game metrics and proposed specific types of game metrics such as social metrics (e.g., number of friends, of challenges, etc.) and viral metrics (e.g., number of invitations, shares, kudos, etc.), among others. The focus was to measure social aspects and behaviors of games, which were often based on the Facebook platform, thereby capturing the strong social dimension of this type of games. Bernhaupt and Mueller (2016) classified metrics depending on genre/type of game they are used in. They distinguish between generic gameplay metrics (for features that are common in all games, e.g., game session time, total playing time), genre-specific gameplay metrics (for games in a same category that share common features, e.g., in shooter-based games: time spent using a certain weapon), and game-specific gameplay metrics (metrics that are based on game-specific unique features and that are custom designed, e.g., usage of special abilities of a character). This classification can be considered orthogonal to Drachen et al. (2013a)'s classification, as it provides an alternative view, according to context of use (type/genre of game) rather than functional context (Interface, In-game, System), on the metrics landscape.

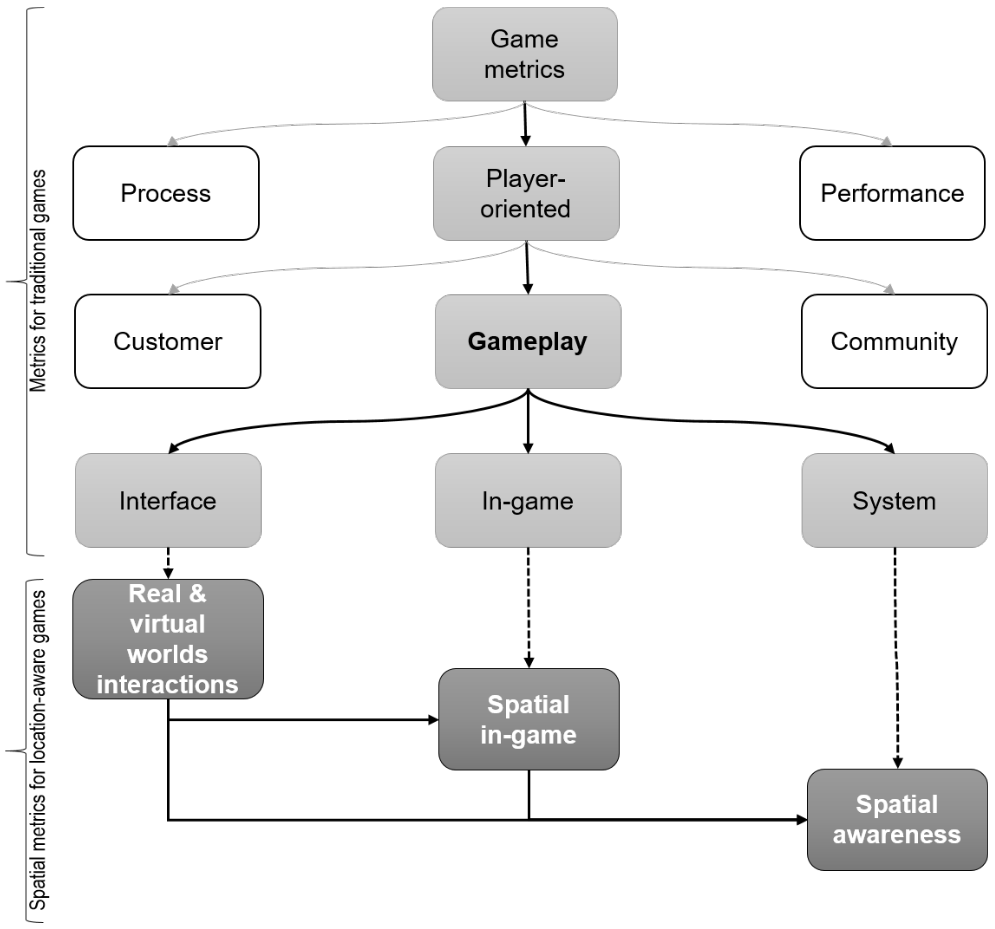


Figure 2.2: Spatial metrics classification embedded into existing classification of game metrics centered on gameplay/player (extended from Drachen et al. (2013a)).

2.4.2 Spatial Metrics Classification

Rather than proposing a new classification, we consider Drachen et al. (2013a)'s high-level, schematic classification as an appropriate starting point to embed spatial metrics within the realm of traditional metrics for gameplay. We hereby emphasise that both types of metrics can and *should* work in cooperation, and we recognize the importance and impact the functional context (interface, in-game, system) of metrics may have in practice (e.g., for game analytics platforms implementations). As such, we extend Drachen et al. (2013a)'s classification to include the spatial notions critical for location-aware games, see bottom part of Figure 2.2. Nevertheless, we note that the inclusion of the spatial context of location-aware games, and the spatial metrics that address them, requires a slight re-interpretation of the three metric groups (Interface, In-game, System).

First, we note that the In-game and Interface categories need to be more broadly interpreted. Furthermore, both are more tightly coupled and influence each other. Indeed, in location-aware games, the real world may be (at least partly) embedded in the game, whereby it influences or becomes the playing field. In other words, the real world becomes part of the In-game context. Therefore, In-game metrics need to address "regular" In-game features, yet also include spatial metrics aimed at monitoring behaviour of a player in the real world, which is possibly influenced by the surrounding environment. Furthermore, when real and virtual worlds blend, metrics mixing virtual features with real-world (spatial) features are necessary.

On the other hand, interactions with the game become tightly coupled with the real world, as e.g., game resources (avatars, enemies, etc.) may be embedded both in the virtual and the real world. For the latter case, they are strongly dependent on the (spatial) characteristics of the real world. Therefore, while metrics for Interface and In-game were treated in relative isolation in traditional games, this is no longer exclusively the case in location-aware games. Notwithstanding this tighter coupling between Interface and In-game contexts, both also still exhibit distinctive features which may be studied independently. For example, game developers may utilise location-based technology (e.g., augmented reality, context-aware mobile interfaces), and as such, how players interact with the user interface of the game application is still important and subject to monitoring within the realm of Interface metrics.

The resulting classification in Figure 2.2 shows the newly added metrics groups, addressing geospatial features of location-aware games, along with their relation to each other and to the original groups. The geospatially related groups refer to what the target of the analysis of the metrics is, namely “Real & virtual world interaction”, “Spatial In-game” and “Spatial awareness”. Arrows among them denote the interaction between these metrics, which can not always be seen in isolation. For example, player’s actions (Real & virtual worlds interaction) in location-aware games leave spatial footprints that can help to better explain player behaviour in the game (Spatial in-game) than when that latter aspect is studied without spatial context.

We hereby note that, from a GIScience point of view, the different types of metrics defined in the well-known classification of de Smith et al. (2015) based traditionally on univariate, bivariate, and multivariate spatial analysis, are applicable. Choosing the right type of spatial analysis, based on the number of spatial variables involved, is up to game developers/analysts and the scope of the metric. For example, trajectory-related metrics can measure movement of a player, or a group of players (clustering), or be used in combination with event data (number of deaths, picked up resources, etc.). In these cases, the process of computing trajectories may involve one, two or more variables and thus require different spatial techniques (see de Smith et al. (2015)), even though all these examples belong to the same logical group of metrics (Spatial in-game). In other words, our metrics are grouped based on functional context, just as in Drachen et al. (2013a)’s original classification, rather than on the technicality of the used analysis technique, as in de Smith et al. (2015).

2.4.3 Spatial Metrics

Spatial event types, i.e., (logical) data models that capture geospatial concepts, along with geospatial processing algorithms and analysis tools, are essential to support spatial metrics. Spatial metrics are particularly important for the Spatial in-game group, where they use spatial properties of the player collected data to compute and reason about the spatial behavior of players. Nevertheless, as discussed throughout Section 2.3, support for geospatial event types and processing is equally relevant for the Interface and System metrics groups, where they allow to take geospatial properties of interface- (e.g., *where* do game interactions take place?) and system-related (e.g., *where* are system-related actions taking place?) actions into account. In the remainder of this section, we discuss different types of spatial metrics according to the spatial properties they utilize (summary in Table 2.8), and primarily relate them to the most important metrics group, Spatial in-game metrics.

- **Location-based metrics (point-based metrics)**. This category includes metrics that take into account specific locations (i.e., most often represented by *coordinates*), for example, the location of a player or resource. Point-based metrics are the simplest, yet they may reveal interesting properties of the player, interface and game as a whole. For example, point-based metrics may determine stationary behavior (e.g., an ”ambush” in a shooter-based game); proximity or line of sight among and between players, resources, and environmental features; and altitude difference or altitude-related computations. Point-based metrics may reveal relevant game properties, such as significant locations (e.g., locations that attract players) or imbalances (e.g., locations which provide an (unfair) advantage or disadvantage). Example papers supporting this type of metrics include Coulton et al. (2008), where location-based metrics are employed to analyze players behavior in concrete location-based games. In particular, the distance between players as they moved around the game area, and also the mean distance from the average position, is used as a measure of “how adventurous each player was during the game”. Some other examples are presented in Drachen and Canossa (2011), in that location is used for analyzing where deaths occur in the game, or where the users request help, for determining those places where possible imbalances might be present. In Drachen and Canossa (2009) the speed is used for determining spots in the game where the player was less challenged.

- **Trajectory metrics**. This category includes metrics that take the trajectory of the players (or resources) into consideration, for example, the distance between the trajectories of two players; the time spent in a trajectory; rhumb; the convergence or divergence of trajectories; trajectory patterns;

etc., can be candidates for metrics, or elements to consider in metrics. Example papers supporting this type of metric include Hoobler et al. (2004), where the player trajectory analysis can provide information about the players or team strategies, as well as resources (e.g., trajectories of fires/shots). In Drachen and Schubert (2013a) the importance of trajectory analysis is highlighted in order to detect bots, analyzing different types of user behaviors (cooperation, flock, etc.), and detection of user strategies that can be indicators of game imbalance problems. Coulton et al. (2008) also used trajectories as a basis for analysis of players behavior (and unusual behaviors) during the game.

- **Space-related metrics (area-related metrics)**. This category includes metrics which perform calculations based on areas, for example, in the centre, forest, sea; overlapping/disjoint/contained areas (e.g., action radius); inclusion/exclusion of a player/resource in an area (i.e., geofencing); explored/-covered area; exploration speed; area exploitation; spreadness/distribution of resources (i.e., rewards); and so on. Example papers supporting this type of metrics include Hoobler et al. (2004), who suggested clustering techniques for evaluating the distribution of game resources or for design validation ("planned for affordance"). This is especially important for location-aware games, where access to resources usually involves a physical effort. In Wallner and Kriglstein (2012) clustering techniques are used for identifying areas where the users behave more fiercely in the game. Similarly, Wallner and Kriglstein (2014) used clustering techniques for providing insights about the distribution of changes in states of the game. In Drachen and Canossa (2009) the analysis is based on comparing player's behavior in selected areas of the game. Finally, geofencing techniques are exemplified by Map Attack (ESRI, 2017), while Martí et al. (2012) discussed its application for noise pollution monitoring.

Table 2.8: Summary of examples of spatial metrics.

Type	Examples	Example Papers
Point-based	*Distance*: between players' position; mean distance from the average position	Coulton et al. (2008)
	Location: where (point cloud) deaths occur, players request help; where players are less challenged	Drachen and Canossa (2011), Drachen and Canossa (2009)
Trajectory	*Analysis*: distance, convergence, etc. of either players' or resources'	Hoobler et al. (2004)
	Patterns: trajectory patterns for user behaviour	Drachen and Schubert (2013a); Coulton et al. (2008)
Area-related	*Clustering*: clustering techniques for resources/players distribution	Hoobler et al. (2004); Wallner and Kriglstein (2012)
	Geofencing: inclusion/exclusion of players/resources in a area (fence); virtual urban zoning	ESRI (2017); Martí et al. (2012)

2.5 Discussion

2.5.1 Spatial Metrics to Improve Gameplay in Location-Aware Games

As it can be expected and was reasoned throughout this chapter, many of the metrics used in regular games can also be used in location-aware games. Indeed, it can be argued that general metrics in games are a subset of location-aware games metrics. For example, metrics related to user engagement (daily game usage, actions performed, levels played, time spent per session) or monetary metrics (statistics about money spent in digital items) can be used in location-aware games, as they are general enough to be used in most kind of games.

Nevertheless, the particularities of location-aware games demand the inclusion of spatial properties, both for existing metrics (e.g., for monetary metrics, statistics about money spent at a specific location) and for specific (spatial) metrics (e.g., for game balance, the spatial distribution of player actions/events, such as deaths, time spent, etc.).

Given the nature of the playground in location-aware games (real, physical world), the changing conditions, and the characteristics of the terrain (i.e., type of terrain, altitude, slope) can influence gameplay. Spatial metrics (see Section 2.4.3) come into play to measure these characteristics, and encompass the three previously mentioned types of spatial metrics. Although the conditions might be the same for all players, in cases where the game can be played in different places or times, the playground conditions can be determinant with respect to the players' performance and/or be a source of imbalance, e.g., assessing the "effort" needed for reaching the resources (given the physical component that may be involved with location-aware games), determining valid routes, etc. Terrain type, altitude, weather conditions and so on can be monitored by spatial metrics to extract useful information, such as mean slope of the terrain, light conditions (day, night), terrain visibility (e.g., open field, a city, a mountain, forest), humidity, temperature, wind speed, altitude etc. For example, Herold et al. (2005a) evaluated the role of spatial metrics in the context of urban land use change modeling. The spatial metrics are defined as "measurements derived from thematic-categorical maps, exhibiting spatial heterogeneity at a specific scale, and resolution."

Player's characteristics such as physical attributes and conditions (Jacob and Coelho, 2011) (i.e., weight, locomotion characteristics), and geography awareness/knowledge of the place where the game is taking place, can also be considered as contributing factors for the "effort or difficulty" the player has to employ to participate in the game, and regardless of other types of games, are aspects to consider when evaluating the playability and the engagement of the users.

The spatial awareness between players and the actual spatial dimension and configuration of the game is significant in location-aware games. Again, this encompasses the three previously mentioned spatial metric types, and quantifies the relation of the player with the geographical space, such as interaction with the map (map visibility, capacity for exploring/obtaining information about the map), previous knowledge of the map/place, availability of the location and the location device. For example, Kremer et al. (2013) empirically discussed how the spatial decisions of a player (i.e., deciding which point of interest to visit next, or where to start searching) have an impact in the overall learning experience.

Finally, on the downside, there exist difficulties and open issues in designing location-aware games. Jacob and Coelho (2011) provide a concise account of these difficulties: adapting the game to the player's location; lack of availability of location-based information; player's fitness to game physical requirements; hardware limitations; and data protection and privacy concerns. Besides, we add software challenges for materialising location-aware games. With an increased amount of (spatial) data to consider, location-aware games face the challenge of managing, processing and taking into account such data in real time. While massive on-line multi-player games have successfully coped with the challenge of handling data of millions of players, the increased strain of the additionally monitoring, processing and taking into account geospatial data provides a significant software challenge, which yet needs to be addressed.

2.5.2 Implication for Future Geospatial Game Analytics Platforms

In this chapter we have explored the literature for existing support, and categorized and characterized spatial metrics. As further steps, we plan to develop a conceptual platform that addresses all required features to measure spatio-temporal aspects of location-aware games, as uncovered in this chapter, and develop the actual platform. As such, let's shortly discuss the implications of the analysis and findings presented on the architecture and implementation of geospatial analytics platform presented in this document.

As a first observation, scalability is an important factor to take into account, as location-aware games may attract a potentially large number of players (e.g., Pockemon Go had thousands of daily active users (Smith, 2017)), with huge amounts of data gathered. Evidently, the same is true for other classes of games, such as (regular) massively multiplayer online games (MMOGs), yet the computational complexity significantly increases for location-aware games as geospatial data and computations come into play. In addition, the desired reactive capabilities of the analytics platform, (e.g., perform notifications based on conditions applied to the results of spatial metrics computations), puts an additional computational strain on future geospatial analytics platform. All of this has direct implications on the architecture of the system, which necessarily needs to deploy techniques for big data and computational decentralization (e.g., data distribution, distributed computing, etc.) with associated technologies (e.g., distributed NoSQL databases, such as Hadoop HBase or Cassandra, cluster computing frameworks, such as Apache Spark).

Secondly, a rich built-in support for geospatial data representation and event types, beyond simple point-based coordinates (e.g., spatio-temporal trajectories, polygon, 3D objects, etc.), with associated geospatial computation methods, is required. Offering such native support allows performance gains due to internal optimizations in storing and handling geospatial data, and easier use and exploitation of geospatial features (by developers) through pre-defined geospatial functions and metrics. A wealth of work in this respect is available in the Geographical Information Science (GIScience) community (e.g., data storage optimizations, spatial algorithms), which can be re-used in the future geospatial metrics framework. Furthermore, compatibility with existing standards may facilitate interfacing with and use of existing open source and commercial software specialized in geospatial data storage and computations.

Thirdly, in extension to the previous point, extensibility is essential to support a broad variety of location-aware scenarios (including those using potentially new or upcoming technologies, such as augmented reality). While predefined geospatial event types are important (e.g., for optimization purposes), a lack of extensibility limits the flexibility of the system to handle unforeseen or changing requirements. The envisioned platform should be flexible for modelling any type of data and ideally provide convenient mechanisms to handle and propagate it to other platforms components so that such information about data model can be purposed for metric definition, computation, and validation. This implies support for custom metrics definition, both native and custom event types and geospatial computations. This level of integration necessarily requires a flexible architecture, which allows data models and metric definitions to be accessible all over the platform (i.e., both at client and server side). While this level of flexibility might carry some negative implications (e.g., storage of arbitrary complex data models may impede performance), the benefits would overpass by far the limitations incurred by limited extensibility, as generally found in the current state-of-the-art.

Finally, and hand in hand with custom event types as discussed in the previous point, the future geospatial analytics platform should support custom geospatial computation and custom metrics specification and execution at the server side, accompanied by convenient tools and APIs to support developers in handling the full metrics life-cycle (i.e., definition, execution, spatial analysis, interpretation of results). In designing such tools and APIs, openness of data and analytics services hereby need to be balanced with data privacy, as these may be suspect to location-based data mining attempts/attacks (e.g., see Narain et al. (2017); Jia et al. (2015)).

Chapter 3

CONCEPTUAL DESIGN

In this chapter we present a high-level conceptualization and design of the spatial analytics platform, using as a starting point the evaluation of the state of the art presented in Chapter 2. We describe the components and services necessary for the implementation of the platform functionality required to support the definition and computation of (spatial) metrics for location-aware applications. Here we offer a conceptual view of the analytics platform (Section 3.1) and the conceptual model behind the spatio-temporal metrics (Section 3.2). Both serve as the guidelines for the implementation of the platform described in Chapter 4.

3.1 Architectural view of the analytics platform

The analytics platform for computing spatio-temporal metrics presented here grew out of our previous work in location-aware games, a specific class of location-based applications (Karimi, 2013). In the previous Chapter 2, we have analysed existing game analytic platforms to assess their level of support of spatio-temporal features for analytic processes. Results suggested that little support is currently available, and existing solutions lack generality and re-usability. Although a plethora of location- and context-aware systems can be found in the literature, existing solutions focus only on limited facets of a context-aware system (Alegre et al., 2016). For example, some frameworks offer programming-level support for context-aware applications (e.g., Kulkarni and Tripathi (2010), Guo et al. (2011)). Other solutions are mostly designed for one particular application field such as ambient assisted living (Forkan et al. (2014)), multimedia services (Zhou et al. (2010)) and mobile social networks (Arnaboldi et al. (2014)). Therefore, the common denominator is a lack of solutions to support both spatial and non-spatial analytic computation for location- and context-aware applications. In response to this lack, we conceptualise an analytics platform to enable the definition and execution of spatio-temporal metrics as part of location-aware applications. Therefore, in this section, we overview the conceptual architecture of the proposed analytics platform at a high level of abstraction, while we delve further into the implementation details and the technology stack on which the platform is built in Chapter 4.

Figure 3.1 shows the architecture of the analytics platform, viewed as an ecosystem of client- and server-side applications. The central part of the figure shows the server side of the platform that is designed as a set distributed applications organised as (micro)services and backed by big data processing methods (Chen and Zhang, 2014). Both sides of the figure illustrate the client side as a set of Web and mobile applications, communicating with the server-side through service interfaces.

The conceptual logic of the server-side platform follows the lambda architecture to integrate stream- and batch-processing at the same time. Lambda architecture refers to a big data processing architecture pattern that combines batch-processing and fast- or stream-processing methods (Marz and Warren, 2015). Shown in Figure 3.1 in the central box labelled as *Microservices*, the stream-processing method is used in the *Data ingestion* microservice for storing collected data. It depicts the flow of client-side collected data by various client applications (right side of Figure 3.1), their buffering and routing, and finally their storage into a (distributed) database (*Data persistence* microservice). This stream processing pathway, which is represented in Figure 3.1 (in yellow) as a single circle but indeed contains distinct types of microservices, is logically decoupled from the com-

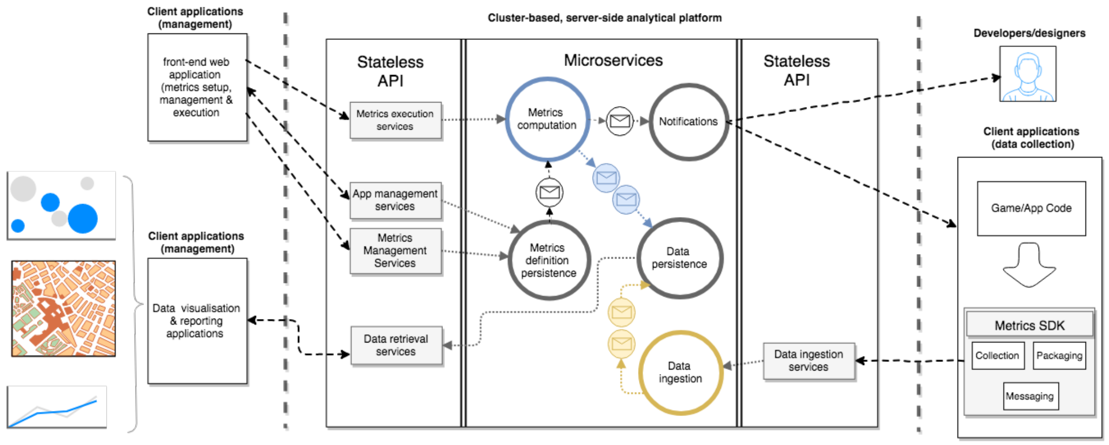

Client applications (management)
front-end web application (metrics setup, management & execution
Client applications (management)
Data visualisation & reporting applications
Cluster-based, server-side analytical platform
Stateless API
Metrics execution services
App management services
Metrics Management Services
Data retrieval services
Microservices
Metrics computation
Notifications
Metrics definition persistence
Data persistence
Data ingestion
Stateless API
Data ingestion services
Developers/designers
Client applications (data collection)
Game/App Code
Metrics SDK
Collection
Packaging
Messaging

Figure 3.1: Layered architecture of the analytics platform for computing spatio-temporal metrics. The server-side platform is placed in the central part, whereby the client side applications are on either side and communicate with the server-side through stateless Application Programming Interfaces (APIs). Yellow denotes stream-processing methods, and blue denotes batch-processing methods.

putational pathway (in blue), which utilizes the batch-processing model in order to compute metrics and perform data analysis (*Metrics computation* microservice).

The *Metrics computation* microservice uses stored metrics definitions (*Metrics definition persistence* microservice), generated by a client-side application (see below), and includes other types of specific microservices (not depicted in Figure 3.1), such as metrics scheduler, evaluator and execution. This microservice also produces notifications which are separately handled through *Notifications* microservices to end users (e.g., developers, designers). The decoupling between data ingestion and metrics computation (coloured circles in Figure 3.1) promotes a distributed big data architecture, while ensuring continuous data storage capabilities, independently of the data analytical operations.

The client side includes two types of applications: management applications (left side in Figure 3.1), which allow configuring metrics and visualizing metrics results, and end-user applications (right side), which contribute application data and receive metrics notifications. For the former, the first management application is a one-stop Web (front-end) application [1] to set up, access, and manage all

[1] Available at https://gganalytics.geotecuji.org

running applications and metrics. Through a visual interface, application developers define, test and update metrics, including their defining characteristics such as application and run-time scope, data requirements, and resulting actions. The second management application is a Web-based visualisation & reporting application, which allows the visualisation and inspection of metrics data and results. Both management applications are targeted to application developers who decide to rely on spatio-temporal metrics for their location-aware applications. The latter type of client applications plays the role of both (raw) spatio-temporal data collectors, necessary as input data for metrics computation, and data consumers, for (push-based) notifications of the results of the computation of metrics.

In subsequent sections and chapters, we go into more details in the defining features of the analytics platform, namely the conceptualization of the underlying metrics model (Section 3.2), and the implementation of the platform (Chapter 4).

3.2 Conceptual model of spatio-temporal metrics

The central concept of the analytics platform is the notion of metrics. In general, metrics are used to monitor or characterise natural or artificial phenomena. A metric can be considered a function that takes input data and produces comparable outputs, i.e., for comparing or contrasting similar phenomena and, thereby characterising them. In economy, an illustrative example is the GNP (Gross National Product) metric that is broadly used to compare economic development across countries. In urban sciences, urban landscape metrics (Herold et al., 2005b) and urban sprawl metrics (Sudhira et al., 2004) help urban planners and researchers characterise urban dynamics to better understand how cities operate. In videogames, metrics are must-have tools for game developers and designers to monitor aspects of the game (e.g., game mechanics, strategies, user interface, player behaviour) and consequently make informed decisions to improve overall gameplay experience (Sedig et al., 2017). Specifically in location-aware games, the role of spatio-temporal metrics is as important as other metrics (Rodríguez-Pupo et al., 2017), and more generally, in location- and context-aware applications, spatio-temporal metrics can help developers to quantify and better understand the phenomena and dynamics that occur in real-world applications (El-Nasr et al., 2013a; Nijholt, 2017).

The concept of metrics has been commonly associated with a sole function. Here, we extend this concept to a model composed of three main elements: the (input/output) data model or structure, the analytic function, and associated action(s). The specification of the data model of the phenomenon to be monitored is the only mandatory requirement to obtain a functional but minimal metrics model, which can only be used to store incoming data compliant to the specified data model (data ingestion in Figure 3.1). To fully harness the analytic power of the platform, i.e., the metric computation phase in Figure 3.1, the other two elements of the metrics model are needed too. Next, we describe each element in detail (Sections 3.2.1 through 3.2.3) and compare our proposal with the literature (Section 3.2.4).

3.2.1 Data model

The underlying idea of metrics is to measure monitored phenomena of interest for users, independent of their application domain, e.g., researchers, urban planners, software developers. An essential requirement in doing so is to capture the required data through flexible data models, since these phenomena may require the collection of data of diverse nature, concerning different aspects a user wants to monitor. Examples of such aspects include application-environment interaction, user interface, user mobility, or any other relevant data that are necessary to quantify the context.

Context matters

Because the interpretation of context varies depending on many factors (discipline, view or dimension being analysed), establishing a clear meaning of context is necessary to better understand what we mean by "quantifying the context". Context is "any information that can be used to characterise the situation of an entity", where an entity is "a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" (Dey, 2001, p. 5). One way to look at context is taking a social perspective. In this view, context emerges as a dynamic construct associated with the user's current activity (Greenberg, 2001). Another way to examine context is from a computational point of view, in which context is often seen as an informational representation of an entity, and as such, developers know the possible contexts beforehand, i.e., the possible range of potential situations, so that associate actions can be *a priori* codified into a system. As in most context-aware systems (Alegre et al., 2016), we prefer the computational view of context. We take an informational representation of what might happen when monitoring a phenomenon, i.e., the recognition of an event of interest and foreseen actions in response to it. The metrics model allows application developers to design customised data structures to capture the bits of interest (contextual data) of the monitored phenomenon. Consequently, by defining custom functions over the captured data (Section 3.2.2), these data can be processed, along with other contextual data, to deliver actionable information (see Section 4.1.5).

For example, many location-aware games distinguish collected data from different users, play sessions and, even applications, in case various (game) applications coexist at the same time. In the proposed metrics model, the meaning of user, session, and application is intentionally left to the application designer, who is in charge of defining how these bits of contextual data are best used to meet his/her needs. For example, a session can be defined as each time a player opens the application (e.g., for sports monitoring applications) or when a user performs a new search (e.g., in location-based recommender systems). Sessions can also be applied to groups of users, for example, when various users accomplish a certain goal (e.g., when a complete coverage is reached in participatory sensing applications). In this case, all users share the same session. Therefore, these contextual views can be adjusted in a way to favour data access, aggregation, partitioning, and retrieval at metrics computation time, thus being a determinant mechanism for selecting, contextualising or filtering potentially large volumes of input data.

Equally important are the spatio-temporal features of context. Ensuring temporal order of collected data is vital for phenomena related to mobility, for example. Monitoring the movement of users in location-aware applications to determine travelled distance or trajectory implies necessarily capturing contextual data in the form of spatio-temporal features of movement (Andrienko et al., 2013). As distance can be distorted in case collected points are temporally shuffled, spatio-temporal features are key for input data consistency and ensuring reliable computations of spatial-aware metrics. Next, we describe how specific data models capture contextual data.

Variables and dimensions

Each phenomenon can be explored or studied from distinct viewpoints, depending on what a user wants to monitor. In location-aware applications, one of the monitored phenomena is typically human location and movement, and the outputs of the desirable metrics must be comparable over space and time for being able to process and discover mobility patterns (Andrienko et al., 2013). As any real-life phenomenon, human movement is a complex and multifaceted phenomenon that requires multidimensional data. Accordingly, the proposed data model is driven by variables and dimensions. A *variable* indicates the phenomenon of interest (e.g., movement), while a *dimension* defines the required data model to capture an aspect of that variable. For example, each dimension (`orientation`, `temporal`, `location`, `steps`) of the variable `movement`, as represented in Figure 3.2, refers to complementary aspects of the phenomenon of movement. Therefore, the proposed data model is composed of multidimensional variables.

While variables refer to high-level views of the phenomenon to be monitored, dimensions define the data structure of a variable. Dimensions can be classified into default and custom (Figure 3.2). Default dimensions are data structures already provided by the analytics platform and ready to be used in the definition of variables. Their intent is to handle contextual data, as for example spatio-temporal features, partition and/or filtering of collected data. Examples are the `temporal`, `orientation` and `location` dimensions to capture spatio-temporal features, and the application, session, and user dimensions (grouped under `application` in Figure 3.2), to enable data selection, filtering, and contextualisation (see below). One key advantage of default dimensions is that their data is automatically collected by the analytics platform.

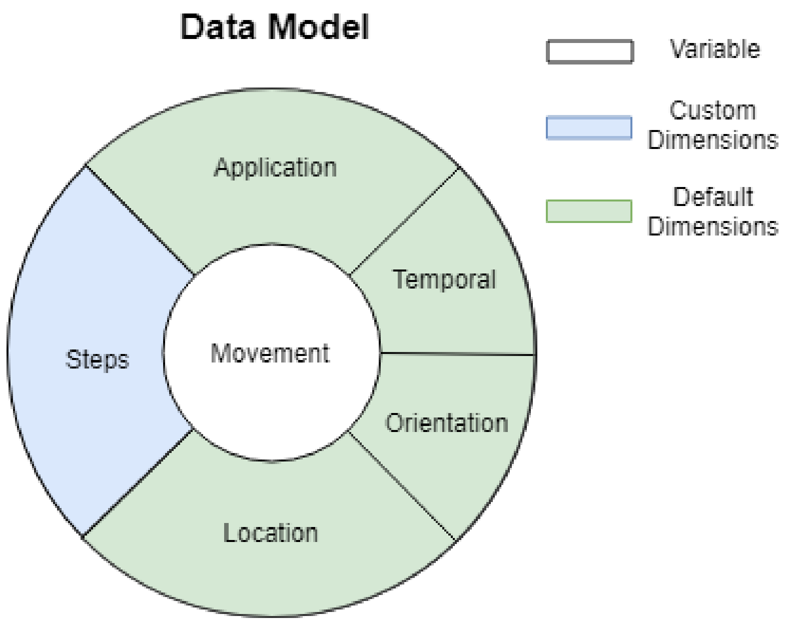


Figure 3.2: Representation of the relationship between the variable `Movement` and default and custom dimensions.

While default dimensions in terms of data structures are given as such and always included in new variables, custom dimensions are up to developers to define. Therefore, developers also need to specify (and implement) how to collect data for custom dimensions, because the platform cannot do so automatically. Nevertheless, it provides convenient methods to help developers collect and manage data according to the custom dimension's data structure. Custom dimensions can be based on any combination of predefined data types (e.g. string, number, etc.), allowing more complex data structures such as nested or hierarchical data structures. Implementation details about dimension definition and data collection methods, and the supporting metrics SDK, are described in Section 4.1.2.

Default dimensions related to data selection and contextualisation (e.g. user, session, application) allow to efficiently filter out and select collected data during the analysis and processing phase. That's why we state that a dimension guides the definition of analytic functions, and not the other way around. The definition of dimensions (data model) for structuring collected data comes first, and this determines the type of analytic functions needed. This aspect is extremely important in stream processing because input streams are dynamic, while functions remain static, i.e. a continuous flow of input data is processed over and over by the same function (Galić, 2016). As we explain next, a default dimension has associated default/predefined functions that are well suited to operate over its own dimension's data model, providing handy methods and utilities for developers to handle, manage and process collected data. The following rules summarise the semantics of variables and dimensions, and set the relationships to the other two elements of the metrics model:

- A valid metrics model must contain at least one variable.
- A variable is composed of one or more dimensions.
- A dimension's data model defines the data types and data structure of input (collected) and output (processed) data.
- A dimension's data model is queryable.
- A dimension's data model is customisable and extensible to allow characterisation of any phenomena of interest.
- A dimension can have associated default (or built-in) functions that operate over the dimension's data model.

3.2.2 Analytics functions

An analytics function takes captured data as input, structured according to the dimension's data model, and computes output data, which are also structured according to the metrics' output data model. Similar to default dimensions, default analytics functions help developers to handle dimensions' data structure. Default dimensions come with default functions that remain at the developer's disposal for creating custom functions. For example, default functions pertinent to the default dimensions `application`, `session`, and `user` are tasked with querying or filtering out input data. Other default functions are optimised to handle dimension's data models, such as built-in functions related to the `temporal` dimension for filtering and aggregating temporal data (data between dates range, time intervals, etc.), and for spatial data filtering and aggregation as well as for computing topological operations (e.g. data within an area, buffering, union, intersect), associated to the `location` dimension.

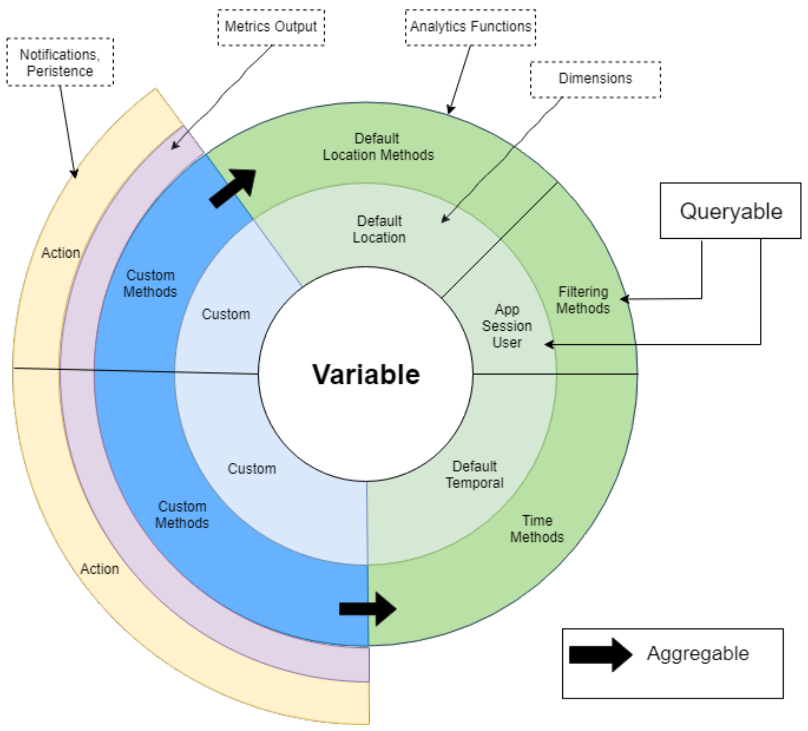


Figure 3.3: Representation of the relationships between dimensions, analytics functions, and actions. Black arrows denote that custom functions may combine (compose) default methods. Actions are only connected to the results of custom analytics functions.

Returning to the example of human movement as the monitored phenomenon, the variable `movement` aggregates default dimensions (`location`, `orientation`, `temporal`, `application`, `session`, and `user`) and other custom dimensions to capture specific aspects of movement (e.g., number of steps) and/or application context (e.g., player progress in the game). While a dimension's data model is queryable (i.e., filtering or selecting data subsets), functions are composable, meaning that custom functions can aggregate default functions to define more sophisticated computations, as the black arrows illustrate in Figure 3.3. For example, to compute the travelled distance based on the number of steps, the custom function will call existing spatial and temporal functions (from the respective `location` and `temporal` default dimensions) to calculate linear distance over a sequence of temporally ordered spatial points (`steps`). Thus, the metrics model represents an elegant but extremely powerful approach to reuse dimensions and associated analytics functions to characterise spatio-temporal phenomena without introducing new conceptual constructs or computational artefacts. In summary, when creating a custom analytics function, developers have at their disposal functionality available for:

- accessing and navigating through the data fields and nested structures of the dimensions' data models, making input data of each declared variable accessible from the execution environment of an analytics function. This requires data access methods in place to get to both input data and output data, being the latter specially useful to get access to the history of metrics outputs, for example, to calculate statistics over historical data or compare current results to past results;

- contextualising input data through the `application`, `session` and `user` default dimensions; Slightly different combinations of operators from these default dimensions permit to conveniently select, query, and filter data relevant to the current execution context and needs of an analytics function (see Section 4.1.5);

- operating and filtering over temporal data through the `temporal` default dimension. Temporal filtering can be contextualised, for example, to retrieve "data belonging to variable `movement` for the *current* user and *current* session of the application over the last five minutes". This functionality is crosscutting to any declared variable as the proposed metrics model always includes the `temporal` dimension and associate functions;

- operating and filtering over spatial data through the `location` default dimension. This is important for enabling geospatially related functionality in the analytics platform. Built-in spatial functions are based on spatial algorithms and methods from external libraries (e.g. Turf.js as we will see in Section 4.1) to, for instance, calculate measurements (area, distance, centroid, etc.) and transformations (convex hull, simplification, etc.). Spatio-temporal operations for trajectory analysis are also included to compute average/max/min speed, travelled distance, number of points within a trajectory, spent time to travel it, among others;

- performing descriptive statistics measures of tendency that are traversal to any metrics definition such as average, maximum, minimum, as so on. These functions are not tied to any dimension but taken for granted as built-in operations embedded in the analytics platform; and
- composing and reusing existing analytics functions in the definition of new analytics functions.

Built on top of the variable and dimensions rules (Section 3.2.1), the following statements summarise the semantics of the analytics functions in the context of the metrics model:

- An analytics function's return type must conform to the data model representing output data.
- An analytics function is reusable and composable.
- An analytics function is customisable and extensible to allow computation of any phenomena of interest.

3.2.3 Actions

An action defines what to do with the result of the execution of an analytics function. An action's input is the return value of an analytics function. An explicit definition of a custom action is optional in the metrics model, but a default action is always data persistence, since the analytics platform always stores the results of the analytics function for future use.

Notification is another type of supported action. Using simple reactive rules (if-then rules), users can specify when an action is triggered (Rodríguez-Pupo et al., 2017). For example, a user gets notified only if the output is greater than a given threshold. Notification actions are especially interesting because they allow the platform to be in continuous interaction with users, the physical environment and other systems that interact with it. Notification actions are also customisable, i.e., a developer can choose the delivery method (e.g., push [2], post[3]), the target audience (i.e., the current user, current session or the entire application, for broadcasting notifications), and the content or payload to be sent out (e.g., containing the result of the calculation or informational messages). Note that the target audience of the notifications is in coherence with the context, which is a cross-cutting concept across the system.

As in the previous two elements of the metrics model, we add a new rule to complete the set of statements that characterise the proposed metrics model:

- A metrics model is the sum of variables-dimensions, analytics functions, and actions.

3.2.4 Situating the analytics platform in the literature

In this section, we compare the innovative aspects of the proposed metrics model and the analytics platform with related literature. Indeed, context modelling and representation have been an extensively researched topic. For example, Kaenampornpan et al. (2004) established conceptual key elements that have an influence on a user's activities in a ubiquitous computing world. The focus is on modelling user's activity into the context. This is also a defining characteristic of the proposed model, as the model is particularly designed to capture spatio-temporal features of contextual data, which are relevant in location-aware applications, activity theories and ubiquitous computing. Bolchini et al. (2007) analysed 16 context models in relation to supported characteristics for context representation and formalisation. The focus of this review of approaches for context modelling and representation is indeed heavily biased towards semantic formalism and schema, much in line with the active research lines at that moment.

In Alegre et al. (2016), the authors reviewed methodologies and techniques for developing context-aware systems, and proposed a two-axis classification based on the modalities of interaction with context-aware systems: active/passive *vs.* execution/configuration. Next, we take the active/passive configuration axis for delimiting the scope of the proposed conceptual model, while below we situate

[2]Mobile push notifications

[3]HTTP POST to a defined url

the analytical platform regarding the active/passive execution axis. In an active configuration interaction, a system "is able to learn from the user preferences in order to autonomously evolve his rules for future behavior" (Alegre et al., 2016, pp. 58). In the passive configuration, a user "is involved in the manual personalisation of his/her preferences, likes, and expectations of the system" (Alegre et al., 2016, pp. 58). The proposed model falls into the passive configuration type, which allows developers to interact with the context-aware system (see for example Section 4.1.1) to manually set up any aspect of the model, from the variables and dimensions to custom analytical functions and actions. The configuration effort required is drastically reduced since a developer can reuse a wide range of default dimensions, analytical functions and actions already implemented and integrated into the analytics platform. Rather than being a *pure* passive configuration, the proposed conceptual model indeed defines a *guided* passive configuration interaction with the system, as the developer is supported in the configuration process by the platform's built-in functionality (e.g., use of default dimensions, see section 3.2.1).

According to the active/passive execution axis, active execution means that "the system acts autonomously depending on the context", while in passive execution users "specify how the application should change in a specific situation" (Alegre et al., 2016, pp. 58). The analytics platform belongs to the passive execution type since developers (users) specify the analytics functions and actions that best fit the aspect that's being monitored and, therefore, the platform will take the actions that the developer wants. As said above, between the two edges –active and passive–, there is a wide range of possibilities. The proposed analytics platform tends to be a passive execution in terms of interactive levels but incorporates many *active* elements such as methods for automated data collection pertinent to default dimensions (see Section 4.1.2). In this way, no specialised developer knowledge is needed to manage and collect data associated with the default dimensions, as the analytics platform smoothly manage them. On the other hand, the passive nature of the platform provides other benefits, since developers understand better how the platform works, as they specify their own analytical functions and, therefore, the confidence towards the platform also increases.

Chapter 4

IMPLEMENTATION

In this chapter, we describe the implementation of the conceptualisation of the analytics platform defined in Chapter 3. A substantial part of the chapter is devoted to the data workflow through the components and microservices to enable the computation of spatio-temporal metrics (Section 4.1). As the assessment of the analytics platform is vital, we cover it on two different levels. In Section 4.2, we perform a conceptual comparison with the literature, considering the key features of the platform, which were initially listed in Chapter 2. Next, we will compare two different implementations of the same location-aware game, one with the platform and one without the platform, to assess the differences between them in Chapter 5.

4.1 Data Workflow and implementation of the analytics platform

This section puts the conceptual metrics model, explained in Chapter 3, into practice. It explains the implementation of the analytics platform, bringing a set of state-of-the-art software technologies, components, services, tools, and underlying big data processing and analytic systems (e.g., Spark, Kafka) together to realise an analytics platform for integrating and computing spatio-temporal metrics. The rest of the section deepens relevant features of the platform that are especially significant for developing large-scale, context-aware architectures and applications. Subsequent subsections expands specific parts of the conceptual architecture (Figure 3.1) initially described in Section 3.1. In particular, Section 4.1.1 details the definition of a metric according to the metrics model. Sections 4.1.2 and 4.1.4 describe how the first element (data model) of the metrics model is being used for data collection and data ingestion into the analytics platform, respectively. Collected data is transferred to the analytics platform through standardised data services (Section 4.1.3). In Section 4.1.5, we describe the metrics computation phase, which uses the three elements of the metrics model. Lastly, in Section 4.1.6, we highlight a supporting tool to retrieve and visualise the input and output data of metrics.

4.1.1 Metrics definition specification

Metrics definition is a core concept of the platform and is extensively used by components both in the back-end and front-end. The metrics definition specification aims to express, in a declarative way, both the data model of the metrics model (variables and dimensions) and the computational behavior (metrics functions and actions). However, some components of the platform use the metrics model for distinct purposes and so each one pays attention to specific elements of the metrics model. This varied usage imposes the following design requirements to design the metrics definition specification:

- Sharing: The specification must be declarative to facilitate sharing, both between different components of the platform or across applications, but also between designers and developers. This implies a trade-off needs to be found between a machine readable and human readable format.
- Validation: The specification format must be easily validated regardless of the chosen technology stack (client libraries, third party systems, etc.).
- Completeness: The specification must be "complete", i.e. it must contain all the required elements (variables definitions, actions, metrics functions,

scopes) for the platform to operate, without the need for additional information or artefacts (beyond configuration aspects that are not reusable by its nature, such as passwords and identity-related data).

- Flexible: The specification must be flexible enough to represent the most varied data models to meet the various computational requirements of the user problem domain.

To fulfill the previous requirements, we use the JSON open standard file format and the JSON schema vocabulary for validation. JSON is a widely used text-based representation format for lightweight data exchange among several public services on the Internet. It counts with excellent support in most platforms and technology stacks. It also has a fair degree of (human) readability, which facilitates sharing and editing.

Regarding validation, the JSON schema allows to specify a set of rules for validating the represented data structures related to variables and dimensions, functions, and actions. As the JSON Schema is supported by several libraries, programming languages and platforms (i.e., Java, C# [1]), a way for external, third-party systems (i.e., tools) and services to validate the metrics definition JSON-formatted documents is implicitly provided. Yet, as metrics functions are coded in a programming language, they can not be validated directly against a JSON schema, and therefore need to be executed and interpreted by processing components of the platform.

With respect to completeness, beyond the cross-platform features of the format, this aspect has a strong relation to the design of the analytics platform, i.e., it must be able to deal with the metrics definition specification as a whole, meaning that no dependencies are kept behind the scenes that might restrain the use of the specification document in a different context (using the analytics platform). For this reason, all of the elements required by the system must be included (specified) in the schema.

As already mentioned, the popularity and abundance of supporting tools for JSON further motivated its choice. For example, several libraries exist for visu-

[1] Some implementations are listed at https://json-schema.org/implementations.html

ally representing JSON Schemas (JsonForms.IO [2], SchemaForm [3], RactJson-SchemaForm [4] and JsonForm [5]; we use the latter in the implementation of the metrics definition tool). JSON is widely used not only by front-end libraries but also by back-end ones, where several data storage technologies are "aligned" with and support it. Examples are MongoDB, that accepts JSON as input data, and Apache Cassandra that has good support for JSON-encoded data input and output. Both database systems are used in the analytics platform; MongoDB is used for storing metrics definition files, while Cassandra is used for keeping the collected metrics data. In addition, JSON is used for defining the data structures (through JSON schema) and the actual data exchange with the REST services.

Metrics definition schema

The metrics definition schema should provide the flexibility for the user to specify the data structures needed to solve the problem at hand (see Chapter 3). As a consequence, a metrics definition data model must be able to represent different types of data structures, with different levels of complexity (i.e., data structure depth) to address a wide range of problem domains. Flexibility also favors the integration with existing systems, as the outputs of other systems can be (potentially) more easily integrated or converted to the metrics definition schema.

The JSON schema of the metrics definition provides a formal specification to structure the data. Figure 4.1 focuses on the top-level properties of the metrics definition schema as explained below:

- **Variables**: are used to represent high-level views of the phenomenon to be monitored (Section 3.2.1) and are composed by dimensions. For example, a variable *movement* can be composed by dimensions *location*, *direction* and *time*.

- **Metrics**: allow to specify and describe metrics functions (Section 3.2.2). They take as input datasets based on the data structures defined in the

[2]https://jsonforms.io/
[3]http://schemaform.io/
[4] https://github.com/mozilla-services/react-jsonschema-form
[5]https://github.com/jsonform/jsonform

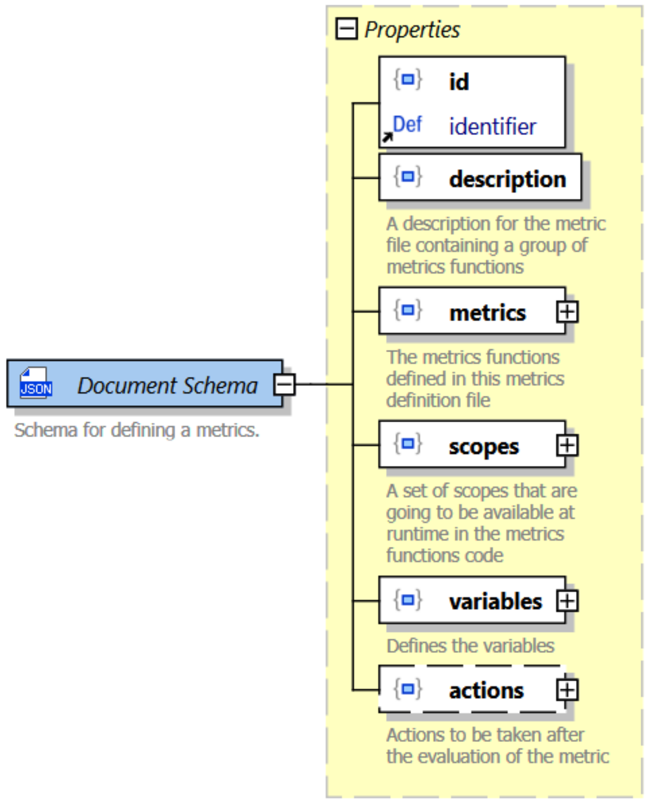

Properties
id
Def identifier
description
A description for the metric file containing a group of metrics functions
metrics
The metrics functions defined in this metrics definition file
JSON
Document Schema
Schema for defining a metrics.
scopes
A set of scopes that are going to be available at runtime in the metrics functions code
variables
Defines the variables
actions
Actions to be taken after the evaluation of the metric

Figure 4.1: Representation of the top-level properties of the metrics definition schema.

variables. An example of metrics could be in this case the speed.

- **Scopes**: are used to represent a logical partition of the data, that can be reused in different metrics functions. More specifically, scopes are used as a convenient utility to define frequently used data across different metrics functions. For example, a scope could be defined for selecting the data of users with an age over 60 years.
- **Actions**: allow to specify actions to perform upon the computation results of the metrics functions. As mentioned in Section 3.2.3, they also involve the evaluation of the results to assess whether or not an action should trigger. An example of action could be to send a push notification in case speed is over 5 km/h.

Variables and dimensions definition

Variables are conceptualised as a set of dimensions, each describing a facet of the data. For example, the variable *movement* of the player contains dimensions such as his *location* and his *orientation*. This structure has implications in the static representation and dynamic behavior of the platform, for example, at data collection time, variable instances (i.e., objects containing the dimensions as attributes) are validated according to the their specification (schema) and sent to the back-end as a data object (or record in some systems). Other components of the platform use variables to produce the corresponding specification in the back-end storage. For example, to define and create tables (also called column families), and data types (user-defined types or UDTs) in Cassandra.

In the specification of the dimensions we have addressed structural aspects such as their type (either complex or simple), representational aspects such as their description and title, and we have also included information related to data collection. For data collection, dimensions are categorized in those provided by a user (or produced as a result of user interactions) and those supplied by a specific provider component and, therefore, produced by a client application system. The implementation of the dimensions takes into account the previous aspects by permitting certain specialisations to simplify the definition of variables:

- **Provided dimension** (provided-dimension): Its value is not expected to be obtained as a result of an interaction with the user, but otherwise supplied by software artifacts, such as sensors and system components. Besides, it also allows defining which software component will provide the values of the dimensions at run-time. Its type must be specified as a JSON Schema.

- **Provided default dimension** (provided-default-dimension): It is a specialization of the previous one to declare dimensions such as location and orientation, which do not require to define a type. Its representation and, ultimately, interpretation is up to the user (i.e., the developer), although defaults are provided for convenience.

- **Complex Custom dimension** (complex-custom-dimension): It is provided by the user or developer (i.e., possibly coming from an interaction with the user or as an output of a running task producing data). Its type must be specified as a JSON Schema.

- **Simple Custom dimension** (simple-custom-dimension): It is mainly used to ease the specification of dimensions of simple types, for example, integer and string. It does not require a JSON schema to specify its type.

In contrast to the previous types of dimensions, which are explicitly defined, there is a special type of dimensions that is implicitly defined in each variable and therefore is always included when submitting data to the platform. Implicit dimensions can be categorized into two groups. A first group is related to the context (see Section 3.2.1), and includes the following dimensions: `application`, `user` and `session`, which are of type string (due to its versatility). They define the context associated with the data. Indeed, the context is a crosscutting concept used in the whole platform: computation, data, and notifications are all driven by context. Later, in Section 4.1.5, we will show how we use it in the computation model and its role in partitioning the data.

The second group of implicit dimensions includes the `time` (ISO-8601 formatted), a global unique identifier (GUID) `timeid` used as the identifier the data object, and `previd`, which refers to the identifier of the previous data object produced or collected, if any. The identifiers are especially relevant for the algorithms that deal with location, where the order and sequence of data objects sent to the platform are crucial. Although the time dimension could have been used for identifying the data object, in general, it is not possible to guarantee to be unique in a context (given that the context could involve several players) and, therefore, it would not be of much help for identifying consecutive locations.

Metrics functions

The metric functions specify how to process the data collected (e.g., speed, total distance, pace). They are specified through the *metrics* property in the metrics definition schema. The metrics functions properties include an identifier, used for invoking the function, the programming language for implementing the metric function, and the code itself. To define the output type, the specification requires to declare the schema of the output of the function, which is used for preparing the database for storing the results. At the moment only JavaScript is supported for implementing metric functions.

The specification does not allow the definition of input parameters of the metric functions. The input data of the functions must be variables (data) defined in the metric definition file, results of metric functions, or scopes, which the platform treats in the same way as a variable.

Scopes

As mentioned earlier, scopes are a mechanism to subset data in order to be reused across different metrics functions, for example, data for players with low performance, can be filtered and reused through the scope “lowPerformanceUsers”. To define scopes, it is required to provide through the *code* property a function for querying data, possibly from one or more of the variables defined. Besides, the user must provide a *name* (e.g., “lowPerformanceUsers”), which can be used later to refer to the specific scope from the code of metric functions.

Actions

Actions are used to act upon the completion of metric functions evaluation. In actions, the output of the metric functions are used for two purposes: as data for evaluating conditions for triggering notifications, and for obtaining the payload of such notifications. This process involves executing code[6], which is specified through the property *filter*, to determine whether the action should be triggered. The actions also need to specify a target for resulting notifications. The supported targets include the current user, the application, and the session (as defined by the values `current_user`, `current_session` and `current_application`). As can be noted, available targets are defined in coherence with the context. That is, there is a correspondence between the context setup during the configuration of client applications (e.g., a mobile application) and the target defined in actions. This allows referring to the clients (e.g. a running mobile application) belonging to the current context. For example, if a group of players is in a game session, notifying the `current_session`, would send a notification to the users (e.g. players) in this game session, excluding other users not involved in that specific session, although they might have the game or application installed.

In this regard, two types of notifications are supported, push notifications, widely used in the context of mobile applications, and URL post notifications[7], which are intended for a broader range of use cases (for example, while interacting with third party systems). Finally, the notification payload, which contains additional information related to the conditions that originated the notification, is extracted from the return value of the action *filter* property. Examples of how to define and use actions are provided in subsequent sections.

In summary, the metrics definition specification has a central role in the platform. It is used as a single source of information about data involved in the metrics computation, the processing, and the actions to be taken upon metrics execution completion. Besides, as we will see in Section 4.1.2, the metrics definition is also used in the client side for dynamically loading data providers, and checking the completeness and validity of the collected data to be submitted to the data inges-

[6]As in the case of metric functions only JavaScript is supported at the moment

[7]An HTTP post message with a body carrying the notification payload.

tion services. In the back-end, the metrics definition is used as the specification of the code to be executed, the actions, and the storage structure. Being encoded as a JSON formatted file, all these aspects of the platform can be tracked and versioned in a single place, which provides a single source of changes in the definition of an application.

Tools for working with the metrics definition schema

To define metrics, a user (developer) defines a metrics definition file that is compliant with the metrics definition schema. Although JSON format is human-readable, it is far from ideal for editing fairly complex structures that are required to be compliant with a certain schema. To assist users in this task, the *metrics definition tool* was developed to ease the creation and edition of metrics definition files.

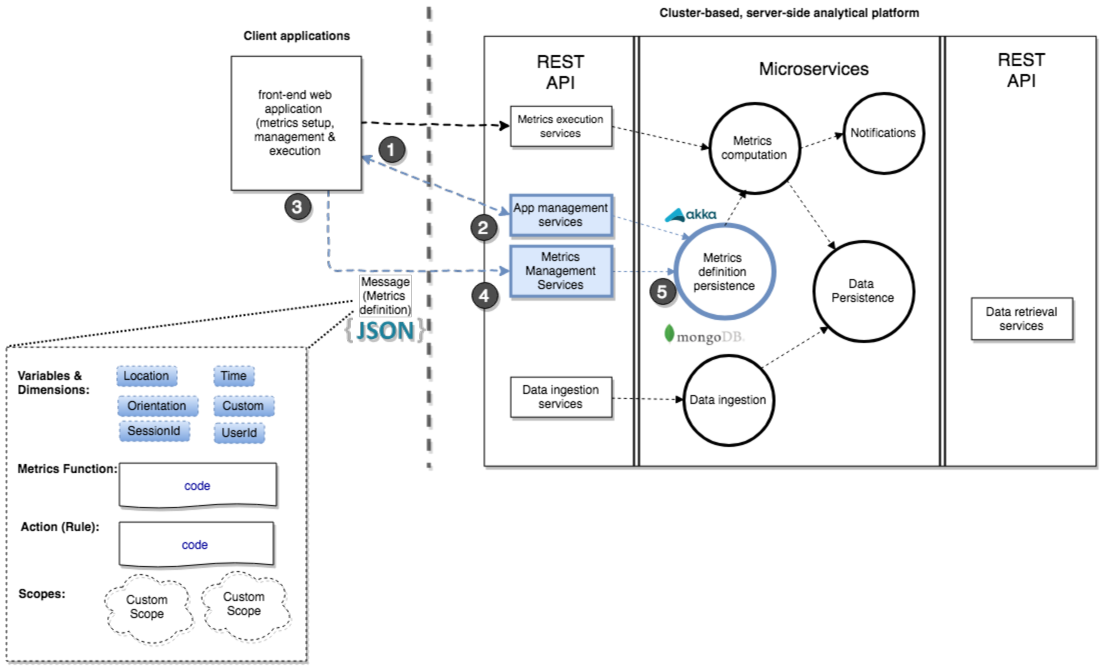


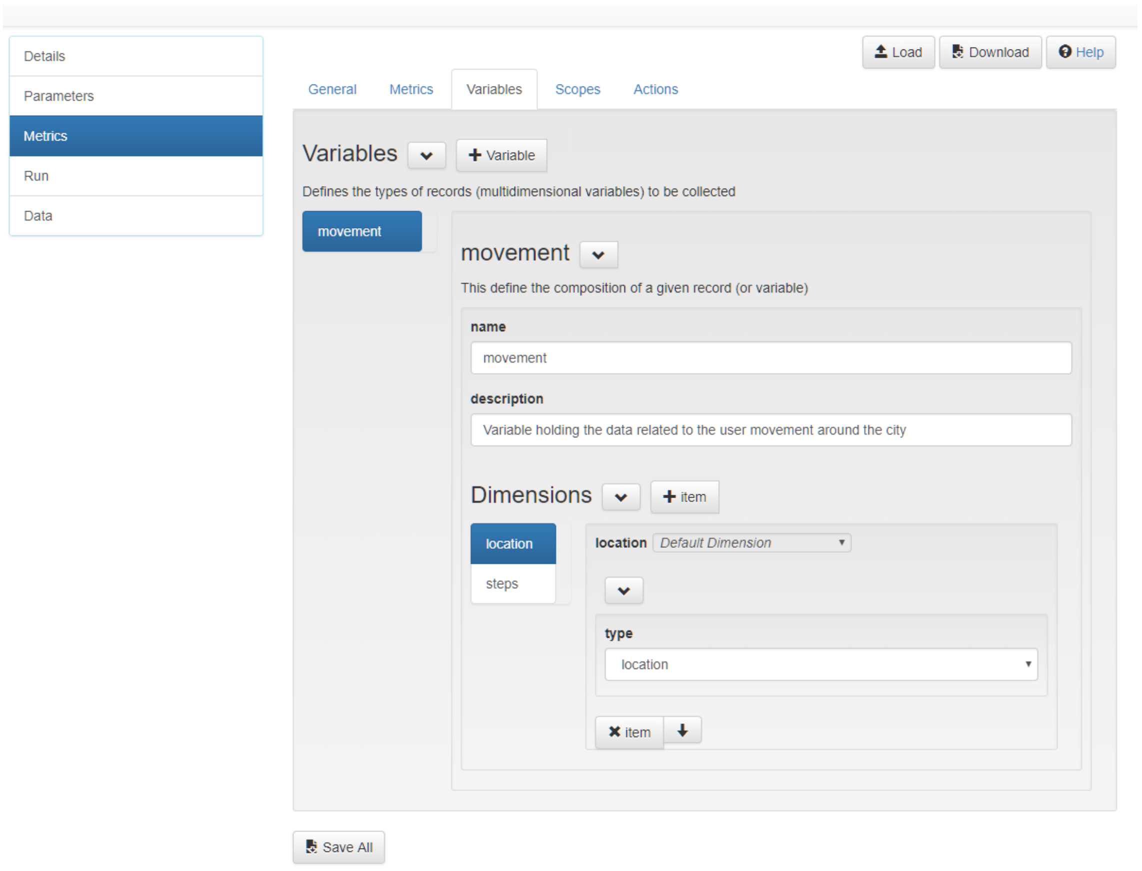
Figure 4.2: Elements in blue are involved in the definition, set-up, and storage of an instance of the metrics model using the metrics definition (front-end) tool.

The metrics definition tool[8] is part of a web-based application that plays a dual role. On the one hand, it is a one-stop catalogue to access the metrics

[8]Available at https://gganalytics.geotecuji.org.

definitions for the different applications. This function is reflected in steps 1-2 in Figure 4.2. In step 1, a user sets up a few application-level, general configuration parameters, including, for example, the configuration needed for enabling push notifications. The metrics definition tool interacts with a layer of RESTful web services which expose convenient functionality for application management (step 2). These services store the configurations and the metrics definition in a MongoDB database (step 5), separated from the actual metric data. This enables the separation of concerns, as the configuration data and the data collected by the client not only have different purposes, but also will be involved in processes with different computational requirements.

These RESTful services are explained in more detail in Section 4.1.3. Besides the general, application-wide configurations, the web-based application also allows users to define a new instance of the metrics model using the metrics definition tool, which is the focus of this section. This second functionality is reflected through steps 3-5 in Figure 4.2. In step 3, a user specifies the main elements of the metrics model (namely, variables, dimensions, metrics functions, and scopes). The metrics definition tool (Figure 4.3) helps users in this task. A resulting metrics definition file is then forwarded to public RESTful services for metrics management (step 4) and subsequent persistence into the back-end MongoDB database (step 5). As we described above, the metrics definition file and collected/processed data are stored separately. While metrics definition files (and other application-related configuration data) are stored in a MongoDB database, flows of input data and processed data (metrics results) are persisted into a Cassandra distributed database. The choice for each is motivated by a different set of requirements. MongoDB provides high flexibility concerning the model stored (in contrast to traditional relational databases, which pose a more rigid schema), which is what we pursue to store and access a variety of application-level data. On the other hand, Cassandra supports high throughput to handle and store streams of incoming data, which is the goal of the data ingestion phase (see Section 4.1.4). The difference in objectives of both databases are clearly visible in the experimental configuration, where we run a single instance of the MongoDB database, but three instances of the Cassandra database to be able to cope with rapidly and massively incoming data.



Defines the types of records (multidimensional variables) to be collected

movement

This define the composition of a given record (or variable)

name: movement

description: Variable holding the data related to the user movement around the city

Dimensions

location: Default Dimension

type: location

Figure 4.3: Screenshot of the metrics definition tool to specify variables and dimensions.

The metrics definition tool's options faithfully reflect the intention and structure of the metrics model (Section 3.2). It consists of a *Variables* tab for defining variables and dimensions; the *Metrics* tab for specifying the code of the custom metric functions; and the *Action* tab to specify the code corresponding to actions based on the results of the metric function (see Figure 4.3).

Besides the three main elements of the model, the tool includes additional features for exploring the data, both the collected data and the output of the metrics computation. Specifically, it offers two types of views, that are intended to show the general aspects of the data (i.e., time, application, session, user): one for presenting the data in a tabular form, and another that allows to display the same data on a map if location data is available, thereby allowing to visualize the spatial properties of the data. The tool accesses the data through services, which are exposed for all client applications (including potential third party, better tailored applications).

Within the *Scopes* tab, the tool provides support for defining scopes, which permits users to pre-define queries over the collected input data and function outputs to generate subsets of data to be reused by the metrics functions, during the metrics computation process. Scopes are intended to define more sophisticated queries than the filtering methods accessible through `userSession`, `session`, and `application` as seen in Section 3.2.2. The resulting JSON document completely specifies and documents metrics, and can be seen as a novel way to share and reuse metrics, as the contained dimensions (data models), metrics functions and actions are openly available so that these metrics definitions may be re-purposed, reproduced, modified, or replicated in other applications and scenarios. This is a contribution towards the establishment of comparable metrics for mobility, for example, confirming recent open calls: “[there is a need for] measurable metrics for making comparisons over place and time.” (Miller and Tolle, 2016, p. 452).

4.1.2 Data collection through Metrics SDK

A fundamental role of a client application is to gather (contextual) data, validate it, package it according to the data structures defined through the metrics model, and transfer it to the server-side analytics platform. Figure 4.4 summarises the elements of the platform directly involved in the data collection task. Only steps 1-4 are concerned with data collection; steps 5 and 6 are involved in the data ingestion phase explained in Section 4.1.4.

Data collection is carried out in two phases (Figure 4.4). The first phase is explained here, while the second phase is described in Section 4.1.3. The first phase occurs in the client application, represented by steps 1-3, where the *Metrics SDK* is central. The *Metrics SDK* is a convenient API library to help developers integrate data collection functionality into their client applications. It provides methods to collect and validate data against the metrics model (step 1, Figure 4.4), provides supporting methods to package and encode captured data into JSON-formatted metrics messages according to the metrics schema (step

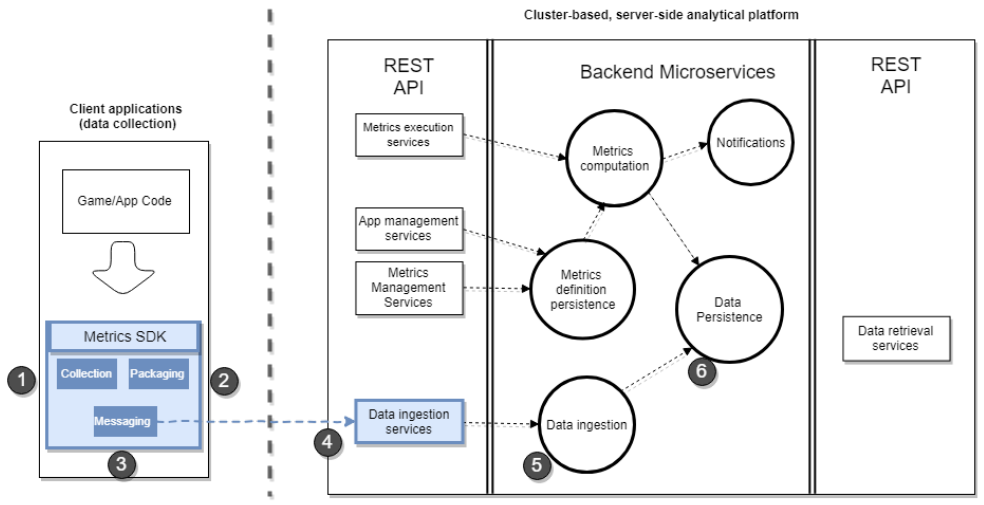


Figure 4.4: Metrics SDK embedded in client applications is used for data collection and transfer to the server-side platform. Coloured in blue the elements involved in the data collection phase.

2), allows to transfer them to the server-side platform (step 3) through data ingestion RESTful services (step 4). For Android[9], the Metrics SDK also takes care of persistence and re-trial of data collection requests in case of failure.

All these tasks only require the first element of the metrics model (variables and dimensions) and the initial configuration setup of the client application (i.e., a valid specification of `user`, `session` and `application` as seen in Section 3.2.1). It is worth recalling here the distinction between default and custom dimensions (Section 3.2.1). In contrast to default dimensions, whose data structures are automatically filled by the Metrics SDK (e.g., GPS-based positioning, or other mobile phone sensor data), the data model of a custom dimension necessarily has to be populated by the developer (using custom code). For example, in the case of the `movement` variable, which contains the custom dimension `steps`, its value must be provided by the developer, by implementing interaction with a step counter sensor (Nagpal, 2016).

[9]At the time of writing, the Metrics SDK only supports the Android platform, but as it encapsulates and abstracts the interaction with the supporting implementation language through RESTful services, it can be easily expanded to other platforms (e.g., iOS) and implementation languages.

Listing 4.1 shows different ways of setting up and using the Metrics SDK. It is necessary to provide the context to a singleton object which injects it to the data objects before sending them to the back-end services. In the listing below, configuration is included during the initialization of the Android application, but it is not a mandatory requirement. Nevertheless, initialization is always necessary before sending data to the back-end.

```
public class MapTestApplication extends Application {

@Override
public void onCreate() {
  super.onCreate();
  // Set up user, session and application.
  final String application = "app-47f2e1b5a3c44404";
  final String session = "session7";
  final String user = "user1";

  AndroidMetricsApi.init(this, new MetricsConfig() {
    public String getUser() {
        return user;
    }

    public String getSession() {
        return session;
    }

    public String getApplication() {
        return application;
    }
  });
 }}
```

Listing 4.1: Configuring and initializing the AndroidMetricsApi instance.

The configuration consists of indicating the application, session, and user the data belongs to (i.e., defining the context). Once configured, the Metrics SDK can send data along with the appropriate context, which serves as coordinates for the data in the platform (see Section 3.2.1). The application id is obtained through the metrics definition tool (Figure 4.3) and is generated by the platform when a new application is created. It is a globally unique identifier that serves to identify all the data collected in the context of an application. The user and the session values are left to the developer to decide as they might depend on the game using the SDK. As we have already mentioned in Chapter 3, the session value can be used for the purpose of partitioning the collected data.

After initialization, it is possible to start collecting and sending data to the back-end services. The Metrics SDK takes care of the details of data collection by automatically obtaining the provided dimension's data. For submitting the data, the Metrics SDK API provides the notion of *tracker*. A tracker is an object that allows sending metrics data from the client and enforces the developer to provide the custom dimensions values required.

```
Tracker tracker = AndroidMetricsApi.getInstance().getDefaultTracker
    ();
// Fill custom dimensions with data
Map<String, Object> dimsValues = new HashMap<>();
dimsValues.put("noiseStrength", 20);
dimsValues.put("noiseLevel", "Level1");
tracker.send("noise", dimsValues);
```

Listing 4.2: Data submitted through a *Tracker*. Note that Trackers are requested to a singleton instance of *AndroidMetricsApi*.

Listing 4.2 shows how to use the *Tracker* to send data of the variable `noise` to the back-end services. This variable is composed of two custom dimensions: `noiseStrength` and `noiseLevel`. All the data related to the custom dimensions must be provided at once; otherwise, the Metric SDK raises an error indicating that data are not complete. The tracker provides a low-level API that is completely dynamic, which means that the API does not provide hinting assistance for the actual dimensions to be supplied. The Metrics SDK does not know how to directly send `noise` because variables are defined dynamically by the user. To illustrate this, for example, an expression like *tracker.sendNoise(noiseStrength, noiseLevel)* would not be valid, as the tracker API is provided for the general case.

The actual implementation is illustrated by the code in Listing 4.2 (last line), which is a general approach but also has the inconvenient of providing less support to the developer.

As indicated in Section 3.2.3, actions can be configured to trigger push notifications to client applications. Notifications are associated with context's properties (i.e., user, application, session), so the receiver has access to the context. The bottom line is that context is a global concept and applies from the computation to the client applications. Thus, the results of a computation in a specific context are notified to instances of client applications running in that same context. This permits fine-grained feedback based on contextual properties.

Notifications contextualized with application, session, and user data, will reach different instances or groups of instances of client applications (i.e., Android game or application). For example, if an action triggers a notification for a specific user, only the instances of client applications associated with that user get notified. The same applies to the session and application. The Metrics SDK handles the details of subscribing to the channels needed to receive notifications and provides the necessary functionality for developers to attach handlers containing application's logic code. The Metrics SDK receives back-end notifications by an event-based subscription mechanism. The events supported are, unsurprisingly, application, session, and user events. These events are mapped to the notification configuration of the actions created during the metrics definition (through the metrics definition tool, Figure 4.3).

For event subscription, the method `registerReceiver` takes an input parameter of type `MetricsEventsReceiver`, which provides different types of event handlers depending on the scope of the events to be handled. The Metrics SDK has three interfaces implementing `MetricsEventsReceiver`, which are designed to limit the scope (i.e., application, session, and user) of the event handled. Listing 4.3 shows the subscription process to the different types of events.

```
ApplicationMetricsEventsReceiver appEventReceiver = new
    ApplicationMetricsEventsReceiver (){
     @Subscribe
     void onMetricsApplicationEvent(ApplicationEvent event) {
        //application's logic code for application event
    }
 }

SessionMetricsEventsReceiver sessionEventReceiver = new
    SessionMetricsEventsReceiver(){
    @Subscribe
    void onMetricsSessionEvent(SessionEvent event) {
        //application's logic code for session event
    }
}

UserMetricsEventsReceiver userEventReceiver = new
    UserMetricsEventsReceiver() {
    @Subscribe
    void onMetricsUserEvent(UserEvent event) {
        //application's logic code for user event
    }
}

// subscribe trough the API singleton object
AndroidMetricsApi.registerReceiver(appEventReceiver);
AndroidMetricsApi.registerReceiver(sessionEventReceiver);
AndroidMetricsApi.registerReceiver(userEventReceiver);
```

Listing 4.3: Registering for notifications in the Android Metrics SDK.

Besides the functionality for handling notifications, the Metrics SDK also offers notifications when data is submitted. This event is registered in a similar way to that of back-end notifications, as illustrated in Listing 4.4. Here, the `event` object (an input parameter of the function `onMetricsDataSubmitted`) contains details about the data submitted and its status. For example, it includes information indicating if the data submission was successful, if failed, or if it was retried. This provides a developer a handler to take proper action in each case. By default, the Metrics SDK internally manages the failure by storing failed requests and retrying them later. It is worth mentioning that the singleton object `AndroidMetricsApi` is an assembly of components (using Dagger [10]) providing features such as serialization, default data providers, communication and client-side local storage, which can be tailored using new implementations.

```
// create the event receiver
MetricsEventsReceiver.SubmitActionReceiver submitDataEventReceiver =
    new MetricsEventsReceiver.SubmitActionReceiver () {
     @Subscribe
     @Override
     public void onMetricsDataSubmitted(AndroidDataSubmitter.
    SubmissionDataEvent event) {
         if (event.getStage() == AndroidDataSubmitter.SubmissionStage.
    SENT) {
             DataUtils.Location location = event.getLocation();
             Log.d(TAG, String.format("Location: %s", location));
         }
     }
}

//register the receiver
AndroidMetricsApi.registerReceiver(submitDataEventReceiver);
```

Listing 4.4: Registering for data submission event.

In summary, the analytics platform provides a set of comprehensive functions for data collection. Collected items may be either observational data, contextual data, or any combination thereof, to enable the enrichment of raw collected data with contextual data for different purposes such as data validation at collection time or for supporting posterior analysis. Default context data is provided by the Metrics SDK, including the configured context and extra data for identifying uniquely the data[11] (an id, previously collected data id, and a timestamp).

Furthermore, the Metrics SDK's methods help developers perform data validation and data enrichment tasks before packaging the resulted data into the metric's data model (Step 2, Figure 4.4). While data validation, against the data

[10]https://github.com/google/dagger

[11]Globally unique identifiers (GUID) are used

definition specification, is available by default, even for custom dimensions, the developer may implement additional and custom validation tasks. These data collection and validation tasks should be viewed as initial steps of the data workflow to transform raw data into actionable information derived from metrics execution.

4.1.3 Back-end Services (REST) API

While Section 4.1.2 refers to the client side functionality for data collection, such data need to be submitted to a set of back-end services. The second phase just presents more details about the server-side REST API services, illustrated by the REST API box in Figure 4.4.

Client applications operate and interact with the analytics platform through a set of RESTful web services [12]. These services are grouped into well-delimited functional categories. *App(lications) management services* (Table 4.2) and *Metrics management services* (Table 4.1) are used for managerial tasks, for initial set-up and for managing new or existing metrics definition specifications respectively, as seen in Section 4.1.1. Another type of services are the *Metrics execution services* (Table 4.5), which give users full control on scheduling metrics execution on demand (Section 4.1.5) since event-based scheduling can only be configured through the *Scheduler* actor (Section 4.1.5). *Data retrieval services* (Table 4.4) are public end-points to allow third-party tools to download and/or access input and processed data for visualization and reporting purposes (Section 4.1.6). Finally, *Data ingestion services* (Table 4.3) are of interest for data collection (step 4, Figure 4.4). These services are also documented through an endpoint by means of the OpenApi swagger specification [13]. Note the use of $\{$ and $\}$ for denoting the URL path parameters used in the URLs of the services in next tables.

[12]Available at `https://metrics-api.geotecuji.org/docs`
[13]Open API specification `https://swagger.io/docs/specification`

Table 4.1: Metrics management services.

Method	*Url*	*Description*
POST	/api/v1/create-metrics/{appid}	Creates the metrics definition of the specified application.
POST	/api/v1/delete-metrics/{appid}	Deletes the metrics definition of the specified application.
POST	/api/v1/invalidate-metrics-cache	Invalidates the metrics cache for all the applications.
POST	/api/v1/invalidate-metrics-cache/{appid}	Invalidates the metrics cache for the specified application.
GET	api/v1/metrics-exist/{appid}	Allow checking if the metric definition exists for an application.
GET	/api/v1/metrics-schema	Returns the metrics schema. This is useful mainly for tools (i.e., editing the schema definitions files).
GET	/api/v1/metrics/{appid}	Gets the metrics definition of the specified application.
GET	/api/v1/test-submit/{appid}	Tests that the submission functionality is working properly.
GET	/api/v1/test/{appid}	Test method to help checking the state of services for the application.
POST	/api/v1/update-metrics/{appid}	Updates the metric definition of a given application (specified by appid path segment).
POST	/api/v1/submit-data/{appid}/{variable}	Allow the submission of data for the specified variable and application id.

Table 4.2: Application management services.

Method	*Url*	*Description*
GET	/api/v1/application-exist/{appid}	Checks if the application with the specified appid (i.e., application id) exists.
GET	/api/v1/applications	Gets the applications definition data.
GET	/api/v1/applications-editor-template	Returns a template 'json-chema' useful for editing applications. This is intended to be used by tools for easing applications configuration.
GET	/api/v1/applications/{appid}	Gets the application by its appid
POST	/api/v1/create-application/	Creates an application.
POST	/api/v1/delete-application/{appid}	Deletes a given application, identified by the application id.
POST	/api/v1/update-application/{appid}	Updates the data associated with a given application.

Table 4.3: Data ingestion services.

Method	*Url*	*Description*
POST	/api/v1/submit-data/{appid}/{variable}	Allow the submission of data for the specified variable and application id.

Table 4.4: Data retrieval services.

Method	*Url*	*Description*
GET	/api/v1/data/{appid}/{datasource}	Gets the data of a given datasource (i.e., variable or metric output), for the application specified, the datasource name, and filters. The filters allowed (through query string) at this time are, *session*, and *user*, as the application can be extracted from the url path.
GET	/api/v1/datasources/{appid}	Shows the datasources available for a given application.
GET	/api/v1/metrics-data/{appid}/{metric}	Gets the data associated to the specified metric, given the application, the metric name, and some filters. The filters allowed (through query string) are *session*, and *user*, as the application can be extracted from the url.
GET	/api/v1/variable-data/{appid}/{variable}	Gets the data belonging to the specified variable given the application, the variable name, and some filters. As in other services, the filters allowed (through query string) at this time are, the session and the user.

Table 4.5: Metrics execution services.

Method	*Url*	*Description*
POST	/api/v1/create-exec-params/{appid}	Creates or updates the execution parameters for the application specified.
POST	/api/v1/delete-exec-params/{appid}	Deletes the configuration associated to the application specified.
GET	/api/v1/exec-params-editor-template	Returns a template 'json-chema' useful for editing execution parameters. This functionality is mostly intended to be used by tools.
GET	/api/v1/exec-params-exist/{appid}	Checks if the execution parameters for the application specified are already defined.
GET	/api/v1/exec-params/{appid}	Gets the configured execution parameters for the application specified.
POST	/api/v1/update-exec-params/{appid}	Updates the execution parameters configured for the specified application.
GET	/api/v1/run-details/{appid}	Returns some details about metrics definition of the applications specified (i.e., the variables, and the metrics).
GET	/api/v1/run-params-editor-template/{appid}	Returns a template 'json-chema' useful for tools related to editing run parameters
POST	/api/v1/run/{appid}	Triggers the execution of the metrics of the specified application, based on the configuration provided in the body.

Apart from services used to interact with the back-end and submit data, additional "utilities" services have been developed to help render and edit associated service's parameters (e.g., configuration data, other REST endpoints parameters). Examples of such services, which follow the same naming convention with the suffix *template*, are *exec-params-editor-template* and *run-params-editor-template*. These two services return a JSON schema that indicates how to automatically build the user interface[14] and to provide input parameters of the services

[14]This can be achieved by using a JSON Schema editor library.

update-exec-params/{*appid*} and *run/*{*appid*}, respectively. Besides the properties required, the schema includes representational elements such as title and the rendering format. Such representational elements are used, for example, in the labels and descriptions of the properties editors or validators for the input properties.

4.1.4 (Fast) Data ingestion

Steps 4-6 in Figure 4.4 cover the data ingestion phase to ensure streams of collected data reach and are properly stored in the back-end clustered database. In this section, we look into the *data ingestion* microservice (step 5, Figure 4.4) to examine the pair of contained components as depicted in the central part of Figure 4.5 below.

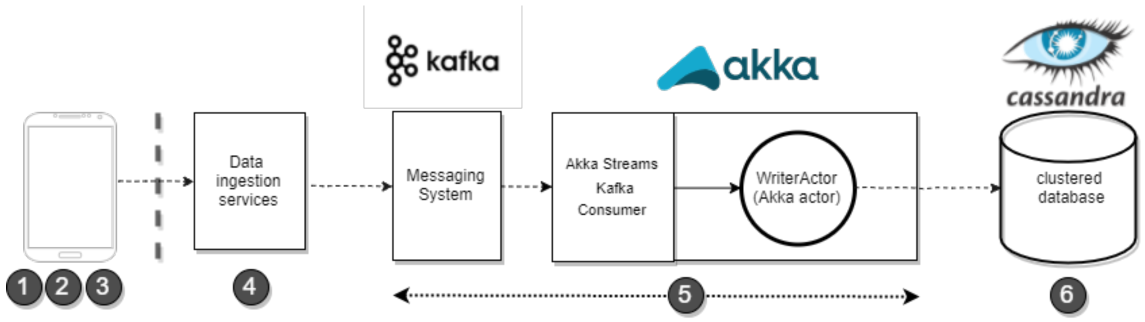


Figure 4.5: Data flow, implemented tools and involved micro-services in the Data ingestion phase. Step numbers are the same than in Figure 4.4.

The first component is the (Kafka) *messaging system*, which ensures reliable handling (receive, buffer, route) of the incoming flow of data. The message system emits and exchanges messages, which contain the metrics data payload, through a Kafka topic to which connector applications can subscribe in order to handle/process the data stream. One such connector application is the self-developed persistence component, which connects to the (Kafka) messaging system and persists the messages in the (Cassandra) database. It is implemented as an Akka actor-based application (Roestenburg et al., 2015) and internally contains two main elements. The first element is an (Akka) Streams Kafka Consumer [15] that subscribes to the (Kafka) *messaging system*, and forwards messages (i.e., metrics data) to the second element, an (Akka) actor called *WriterActor* for performing data persistence into the back-end Cassandra database. This latter actor is aware of the database structure and selects the appropriate table(s) for persistent storage. The data, through the context, includes the information needed to decide where the data should be stored in the back-end database. Before inserting data originating from an application for the first time, the *WriterActor* retrieves the metrics definition specification with the application's variables schema and converts it to the appropriate Cassandra statements that permit the insertion of (JSON-encoded) data messages. This process is based on the fact that the entire JSON schema of all incoming data to the platform is known and included during the design phase of the metrics definition. For creating such structures, the persistence actor uses the names of the variables and the application so that it is linked by a naming convention, which also facilitates the subsequent inspection and querying of the data.

The aforementioned *connector application* furthermore doubles for other purposes besides data persistence; it also has the responsibility to notify changes in the data belonging to different variables. For example, when data associated to a variable, changes, the *connector application* emits notifications with messages (in a ”data-changed” Kafka topic), which include the coordinates of the data changed (i.e., variable name, application, session, and user). This allows (additional) data consumers, for example, to account for the data entering to the platform or for the overall monitoring of the platform.

The back-end database (step 6, Figure 4.5) stores both (collected) input data and output data (i.e., results of the metrics functions). It consists of a Cassandra database configured as a cluster of (experimentally) 3 nodes (for replication), where each node holds both data related to variables and output data. The data is kept in separate tables for the different variables and metrics functions, and named in a way that avoids clashing between applications. Hereby, it allows to seamlessly scale up, adding additional nodes, as the stress on the data persis-

[15] `https://doc.akka.io/docs/akka-stream-kafka/current/home.html` (accessed on 9 January 2019)

tence increases with the increasing amount of supported client applications, the number of variables, and/or the number of end-users. Cassandra (Carpenter and Hewitt, 2016) is the distributed database management system of choice for fast data ingestion because of its high availability, reliability, and performance traits for storing high volumes of data. Besides, it also offers good integration with big data processing frameworks like Apache Spark, as we discuss next.

Performing computation on data where the order is important, such as location data associated with activities, imposes several challenges. This is particularly important in the context of big data applications, where the computation usually takes place in a distributed environment, and it is therefore necessary to deal with partitioned data. Most programming models and their implementations for processing big data rely on parallel computing. This is the case of MapReduce (Dean and Ghemawat, 2008). It is based on two functions (which can be considered phases): execution of the map function in parallel, and execution of the reduce function over the results of the map function. In such a context, the sort operation is considered to have a high cost, due to the potential data shuffle necessary by the algorithm. Order is an important property of location data, and especially for activity or trajectory data, where information is based on the order of the locations that are collected and then processed. For example, to compute the distance covered by an activity, the order in which the locations are considered is crucial, because a different order would produce different results. In the analytics platform, we solve this problem guaranteeing that data is inserted in order, by using the *clustering order* feature of Cassandra [16]. Through this technique, and with other features related to Spark connector for Cassandra [17], we can overcome some of the problems related to the order of data. Besides, we also include the properties `previd` and `timeid` (Section 4.1.2) integrated in our metrics data model for providing information about the continuity of the data. With these properties, we can determine during the computation the order in which sequences of locations are processed.

[16]https://www.datastax.com/dev/blog/we-shall-have-order

[17] How the Spark connector works https://academy.datastax.com/units/how-spark-cassandra-connector-reads-data?resource=how-spark-cassandra-connector-reads-data

4.1.5 Metrics computation

The metrics computation phase comprises a set of microservices that work collaboratively. Figure 4.6 highlights the main microservices involved in this phase, for comparison with previous figures; Figure 4.7 shows the configuration of the metrics computation phase in more detail, whereby numbered elements correspond in both figures.

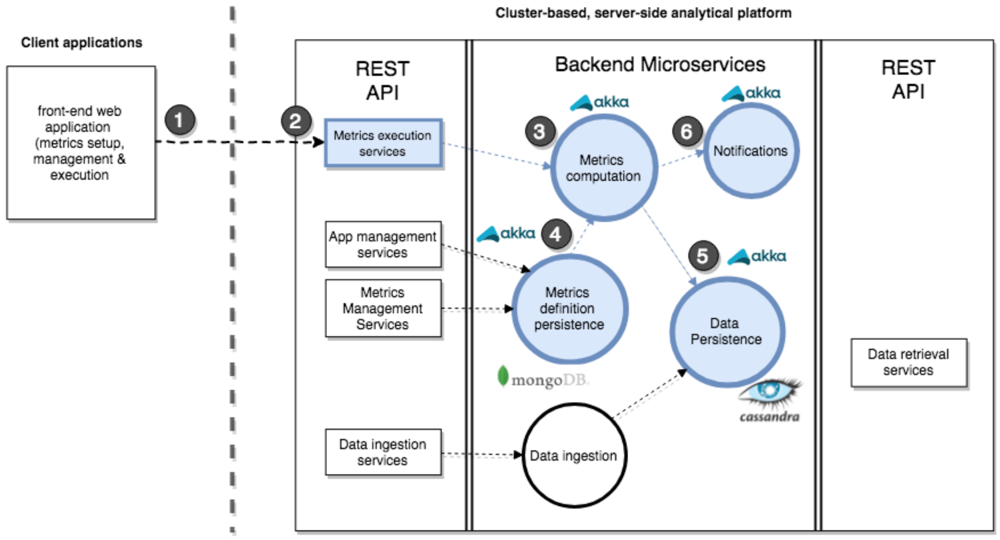


Figure 4.6: Coloured in blue microservices involved in the metric computation phase.

The *scheduler* is responsible for triggering metrics executions. It can be set programmatically through the *Metrics execution services* (step 2, Figure 4.7), or can also be called from the one-stop front-end management application. The scheduler supports event-based and on-demand scheduling. Event-based scheduling triggers metrics execution based on the occurrence of an event, e.g., the arrival of (new) data into the database. In practical terms, a metrics execution is not performed every time an event occurs, but a minimum (but configurable) time is set between executions to avoid degraded execution performance as a result of a sudden peak of events occurring simultaneously. On-demand scheduling allows to immediately execute a metrics based on a manual request. This way, client applications can force a metrics calculation, regardless of the execution of event-based scheduling.

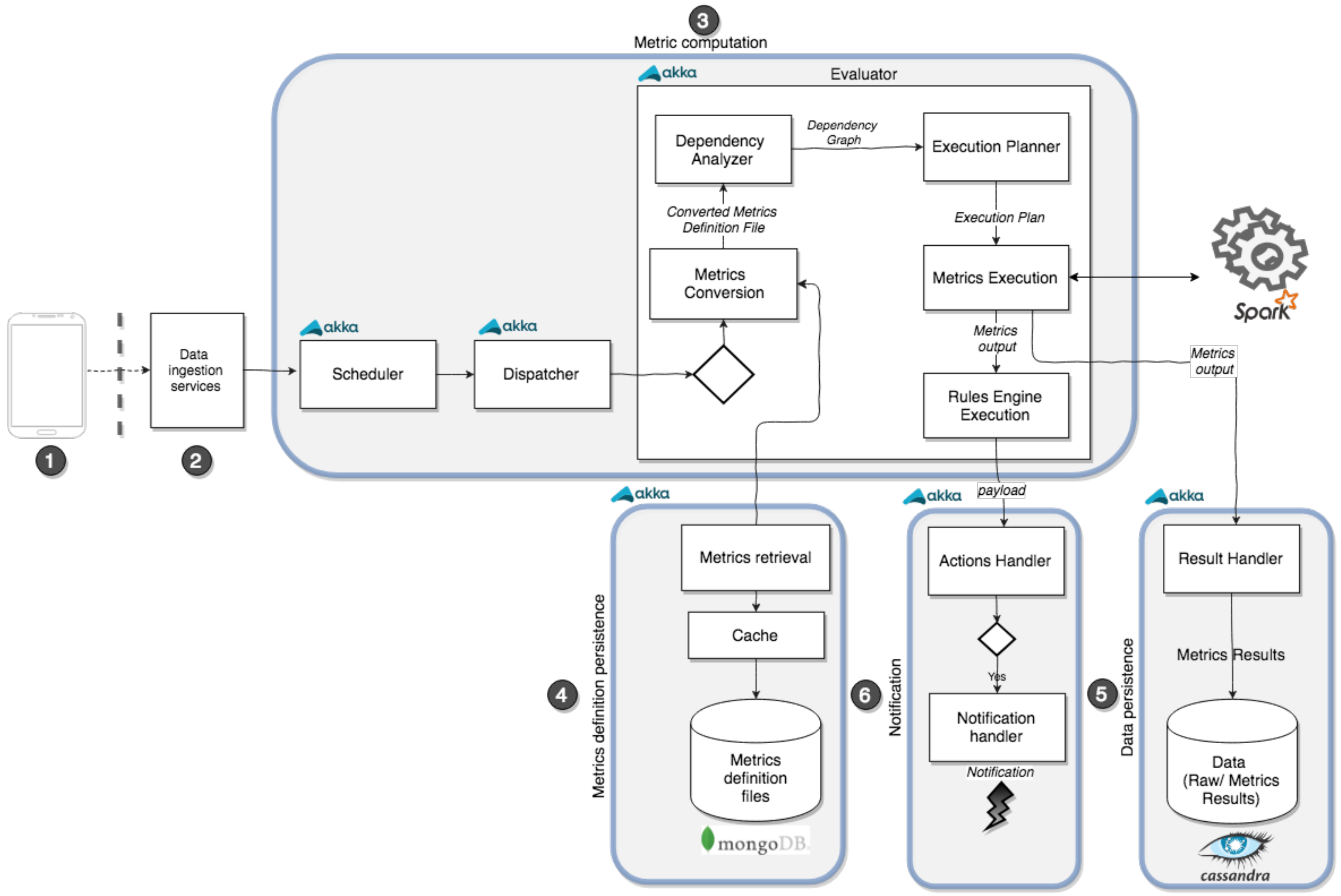


Figure 4.7: Data flow and involved micro-services in the Metrics computation phase.

The *dispatcher* is a router aimed to send metrics execution tasks, as scheduled by the scheduler, to (a set of) evaluator(s). The dispatcher performs message buffering and load balancing tasks, selecting and dispatching evaluators to perform evaluation tasks based on the current load and statistical (historical) performance information.

The *evaluator*(s), the central element of the metrics computation box in Figure 4.7, is responsible to coordinate and carry out the execution of metrics according to the metrics model introduced in Section 4.1.1. It is implemented as a set of components (actors), which work together and are duplicable to allow distributed execution. The evaluator handles incoming metrics execution requests from the dispatcher. First, *Metrics Conversion* takes place: the relevant metrics definition specification is retrieved from the persistent storage (or from cache, for efficiency), as reflected in step 4, Figure 4.7. The declarative metrics specifications are then converted into executable (JavaScript) code. This is a *partial* conversion, because the metrics descriptions already contain executable code representing the metrics function and/or action function. It also prepares and sets up the current context, that will drive the execution and data accessed during metrics function computation. Next, the *Dependency Analyser* takes the JavaScript code and analyses dependencies to generate a directed dependency graph, in which functions and datasets are nodes, and an edge represents a dependency between a function node and dataset node or between two function nodes (e.g., when a custom function *composes* or calls another custom function). This dependency information is useful, for example, for a variable-driven execution, e.g., the execution of all the functions depending on the variable `movement`. Prior to the metrics execution, the *Execution Planner* transforms the dependency graph into an execution plan and runs it (using EclairJs [18] on the Spark platform).

Finally, once the results of the metrics execution (more accurately: execution of metrics functions) are in place, two actions are carried out with the results. First, the results are persisted by the *Data Persistence* (step 5, Figure 4.7): the *Result Handler* inserts data into or retrieves data from the corresponding backend database. This handler abstracts from the concrete database implementa-

[18] https://github.com/EclairJS/eclairjs

tion to prepare data manipulation statements (e.g., inserts), which are executed by specialized components (actors), allowing abstraction from specific database parlance and distributed / redundant persistence.

Second, results are also passed to the *Rules Engine Execution* component to check whether an action needs to be executed. Actions are specified as if-then rules (in the metrics definition specifications), based on the results of metrics functions execution. In case one or more rules trigger, the associated action(s) is executed by the *Action Handler*, and the *Notification Handler* is notified to carry out notification task(s) (step 6, Figure 4.7) based on the chosen notification configuration (see Section 3.2.3).

All of the components mentioned above are actor-based microservices implemented as Akka actors using Apache Spark to exploit its Resilient Distributed Dataset (RDD) model and inherent processing capabilities (Zaharia et al., 2016b).

Server-side computation model and supporting functions

So far, we have shown the basic mechanisms of the engine to execute the code defined in the metrics definition. Nothing has been said about how to access and use data during metrics computation. In this section, we explore the server-side computation model and provide usage examples.

Metrics functions are allowed to access data belonging to the current application (i.e., through the variables defined, and using the mechanisms for accessing the partitioned data) as well as calling/invoking other metrics. At its core, the metrics functions use the EclairJs API, a port of Apache Spark API that runs on top of Java Scripting API [19]. This means that although the code is implemented in JavaScript, it is actually running on top of the Java virtual machine, the native environment for Apache Spark applications.

For server-side programming, the platform provides a JavaScript library containing a series of functions grouped in three libraries: *coreutils*, *metricsutils*, and *geoutils*. Most of these functions aim to ease the access to the structure of data, while others allow the use and manipulation of data (see Figure 3.3). For exam-

[19] https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/prog_guide/api.html

ple, through these functions, a developer can easily access location data and the variables and dimensions according to the metrics definition.

In particular, the *coreutils* library provides core functions to access the internal structures at a low level and helps debug applications, for example, providing logs and basic assertions functionalities. The *metricsutils* library, probably the most important, encapsulates all the functions related to the interaction with data, metrics functions, results, and context (see next section).

Finally, the *geoutils* library provides core geo-related functions, including geofencing and geostatistics (basic route analysis), among others. For advanced geo-spatial analysis tasks, we rely on TurfJs library [20], which implements a variety of geospatial analysis methods. TurfJs methods can be used in the context of *metricsutils* and EclairJs (Spark API) for performing advanced spatio-temporal analysis over the collected data.

Data access in metrics functions

The implementation of metrics functions often requires the access to application data; for example, the value of a particular variable declared in the metrics definition or the results of a metrics function. The back-end microservices provide ways to do this. They internally create RDDs (Spark Resilient Distributed Datasets) for the variables declared in the metrics definition; they also partition the data of variables by filtering the properties of the current context. Listing 4.5 shows how to count the elements of the variable `movement`. The single code line remarks two key aspects of coding metrics. The first is that the very names of the variables defined in the metrics definition are used directly for coding. The second is that the methods of the function libraries provided as part of the Spark library can be invoked on the variables as they are converted to Spark datasets.

```
var count = movement.count();
```

Listing 4.5: Usage of variable `movement`.

[20] https://turfjs.org/

Listing 4.6 shows the use of the EclairJs API for low-level access to the dimensions declared in the metrics definition. Note the use of the functions *map* and *fieldIndex* that belong to RDD and Row Spark classes. In Listing 4.6, *userSession.movement* is a Spark dataset and, therefore, the invocation of the method *toRDD* is necessary to use the RDD functionalities. Any other dimension (also known as field in the Spark terminology) can be accessed similarly. Take the example of the variable `movement`, with the default dimension `location` and the custom one `steps`. To access the dimension `steps` is then required the code in Listing 4.7. Similarly, the custom dimension `time` is accessible by the convenient methods *MetricsUtils.getTimeDefault*, and *MetricsUtils.getTimeStr* (Listing 4.8).

```
var locations = userSession.movement.toRDD().map(function (row){
    var latlon = MetricsUtils.getAsLatLon(
        row, row.fieldIndex("location"));
    return latlon;
});
```

Listing 4.6: Using the RDD functionalities on a variable.

```
var onlySteps = userSession.movement.toRDD().map(function (row){
    var steps = row.get(row.fieldIndex("steps"));
    return steps;
});
```

Listing 4.7: Accessing dimensions on a variable using RDD row functionalities.

```
var userSessionTimes = userSession.movement.toRDD().map(function (row){
    var time =  MetricsUtils.getTimeDefault(row); // As JS Date obj
    var timeStr = MetricsUtils.getTimeStr(row);   // As ISO string
    return time;
});
```

Listing 4.8: Accessing time dimension through `MetricsUtils.getTimeDefault`, and `MetricsUtils.getTimeStr` functionalities.

As the computation model is based on EclairJs, the Spark API (more specifically, the ported functionalities) is available and accessible from the metrics functions. Listing 4.9 shows how to obtain basic statistics about the dimension `steps` of the variable `movement` by using the Spark Dataset `describe` function:

```
var movStats =  movement.describe('steps');
```

Listing 4.9: Using *describe* (Spark Dataset) function on a variable.

Metrics context usage in metrics

As already referred to in the realm of location-aware games in Section 3.2.1, *context* is a crosscutting concept in the entire analytics platform and is materialized in several ways across the platform. For example, in the data collection phase, context is used to establish links between parts of data to further extract information about the underlying relationships or characteristics of such data. These relationships can be used to investigate the behaviour of users in a given activity within a common environment. For example, users who share the same game session can have a different behavior pattern than two users playing separately in a context of a game. In a location-aware game for marketing purpose, sharing a session (depending on the location-aware game design) can imply that a given marketing promotion reached both users who belong to a certain age group, share some common interests, etc.

Therefore, across the platform, we enable functionality to refer to and access data, in order to later perform notifications based on the contextual information extracted. In the process of defining metrics functions code, the metrics context is accessible through the variable *mc*. Listing 4.10 shows how to access the context properties of the variable *mc*.

```
// mc contains the metrics context.
var application = mc.application;
var user = mc.user;
var session = mc.session;
var execid = mc.execid;
```

Listing 4.10: Accessing the context on the server side.

Besides the known context properties, Listing 4.10 includes *execid* (i.e., an execution id). This property of the metrics context is a globally unique identifier (GUID) assigned by the `evaluator` at the beginning of the metrics computation (Section 4.1.5) and is later kept as a property of the results of the metrics computation. In consequence, all metrics outputs derived from the execution of a metric function, and other metrics which depend on it, share the execution id.

Internally, the context variable is set up at the beginning of the computation. It is extracted either from the context of the variable triggering the computation or, in case the execution is triggered manually, from the context specified by the user through the user interface. It is, therefore, the programmer's choice to operate with the data referring to the context or not (i.e., work with contextualized data). This decision depends on the problem at hand; in any case, all the data of the application is available in the metrics functions.

The analytics platform also includes specific objects to access data filtered by the properties of the metrics context (which we could think of as contextualized data). To define them we use a simple naming convention: `application`, `session` ,and `userSession`. These objects resemble different scopes of the data. The `application` object contains the datasets for all the variables filtered by the application property of the current metrics context (*mc*). The `session` object contains the data belonging to the current session (i.e., filtered by *mc.session* and *mc.application*). The `userSession` object contains the data of the variables filtered by the application, session, and user in the metrics context. The syntax pattern is <*context-data-object*>.<*variable*>. We chose this nomenclature to provide a straightforward way to refer to a partition of the collected data that naturally matches different data "views" of the current context.

Listing 4.11 shows some examples of filtered-based access to context data. For example, *movementsInSessionForUser* is initialised by filtering the current application, session and user context of the variable *movement*, while *movementsInApp* is only filtered by the current application context.

```
// Filtering by current application, session and user context.
var movementsInSessionForUser =  userSession.movement;

// Filtering by current application and session context.
var movementsInSession =  session.movement;

// Filtered by current application context.
var movementsInApp =  application.movement;
```

Listing 4.11: Accessing variables filtered by context.

Filtering

Filtering is an important aspect of handling data. In the analytics platform, a set of methods are provided to ease filtering tasks, mainly related to the temporal aspect of the data. Built on top of the ad-hoc filtering capabilities of the Spark/EclairJS API, we add a layer for filtering data by time, based on the knowledge of the structure of the variables and the context data. Time-related filtering methods include:

- *MetricsUtils.last()* filters data by a time period (e.g., 3 hours, 25 minutes) immediately before the time of the current metrics context (mc).
- *MetricsUtils.range()* filters data by a time range.
- *MetricsUtils.dataForLast()* filters data by a time period before a reference time.
- *MetricsUtils.centered()* filters data in a time period centered on a reference time.

Listing 4.12 shows some examples of the previous methods.

```
var rdd = userSession.movement;
// Last 5 hours
var datasetLast5Hours = MetricsUtils.last(rdd, 5, "hours");
```

```
// Between timeBegin and timeEnd
var  timeBegin = ...; var  timeEnd = ...
var datasetInRange = MetricsUtils.range(rdd, timeBegin, timeEnd);

// Time window of 30 hours centered on referenceTime
var referenceTime = ...;
var datasetCentered = MetricsUtils.centered(rdd, referenceTime, 30, "
    hours");
```

Listing 4.12: Examples of time-based filtering.

Note that the previous time filtering methods apply to any dataset (RDD) in the metrics function, including datasets associated to all the variables and metrics functions defined, using the metrics context. This eases the filtering computation referred to a specific time window. Besides, these methods are useful for defining scopes (see Section 4.1.1), which can be reused through different metric functions.

Location handling and geo-related functions

As we have mentioned in previous chapters, it is possible to include a spatial default dimension of type `location` in variables. To help access location data declared in a location dimension, the platform provides the method *MetricsUtils.getLatLonsRDD()*. This method returns a new RDD containing location information necessary to perform a variety of geo-spatial analyses. The order in which location data is processed is important in most cases. For example, order is critical while handling trajectory data, because it is necessary to ensure that the locations to be processed are consecutive in time. The analytics platform provides convenient methods for extracting location data from variables, and verify that data points are consecutive based on the properties `previd` and `timeid`.

- *MetricsUtils.isNextLocation()* determines if a given location is the next location of another.
- *MetricsUtils.isPrevLocation()* determines if a given location is the previous

location of another.

The *MetricsUtils* library also includes methods for performing spatial analysis over location data, as well as key conversion methods to the GeoJson format. Listing 4.13 shows examples of the next methods.

- *MetricsUtils.isInside()* computes if a point is inside a geometry.
- *MetricsUtils.lineDistanceLocal()* computes the Euclidean distance between two points of the trajectory.
- *MetricsUtils.lineStringFromLocationsRDDLocal()* returns the RDD list of points of a trajectory as a GeoJson lineString object.
- *MetricsUtils.geoStats()* provides statistics about sequences of locations or trajectories.

Note that the *"Local"* suffix in some of the above method names denotes the operation executes locally instead of in parallel. In local mode, the RDD is collected (which means the RDD will be materialized[21]) before performing the computation and, therefore, this operation should be used with caution.

```
var rdd = userSession.movement;
var locations = MetricsUtils.getLatLonsRDD(rdd);
print("Count of filtered locations: " + locations.count());

// Distance
var lineStringDistance = MetricsUtils.lineDistanceLocal(locations);

// List of points (trajectory)
var line = MetricsUtils.lineStringFromLocationsRDDLocal(locations);

// GeoStats
var stats = MetricsUtils.geoStats(rdd);
print("Number of points: " + stats.count);
```

[21]This is the term used in Spark for the process of executing the RDD and later collecting the data in one node.

```
print("distance: " + stats.distance);
print("mean speed: " + stats.meanSpeed);
```

Listing 4.13: Using GeoStats in the metrics functions.

In the context of location-aware games, the concept of geo-fences is frequently used. Geo-fences are virtual areas of different shapes (circles, rectangles, etc.), which are often virtually linked to physical features/places of the urban environment like a landmark or building. Events can be triggered when tracked objects (cars, bikes, pedestrians) cross, enter, or exit those geo-fences, opening the possibility for enabling new types of interaction between the urban feature or place and nearby users (e.g., Fechner et al. (2016); Törnros et al. (2016)).

As part of the analytics platform, the function *MetricsUtils.geoFences* returns a list of states (e.g., inside, outside) that express the relationship between a sequence of locations and a geo-fence. This method also returns the pattern discovered, i.e., entering or exiting the geo-fence. Listing 4.14 clarifies the use of geo-fences.

```
var rdd = userSession.movement;
var locations = MetricsUtils.getLatLonsRDD(rdd);

// Geo-fence computation
var result = MetricsUtils.geoFences(rdd, MetricsUtils.LatLonTime(
    geoutils.getLatLon(lat, lon), time), radius);

print("Points analyzed in trajectory", result.count);
print("Last state:" + result.state); // inside/outside
print("Number of location in last state:" + result.state_count);
print("Previous state:" + result.prev_state); // inside/outside
print("Pattern:" + result.pattern); // just_entered/just_exited
```

Listing 4.14: Use of geo-fences in metrics functions.

Metrics results

Metrics functions must return a value that complies with the output schema declared in the metrics definition (Section 3.2). Metrics results are stored once they are returned and kept for evaluation during the actions execution. The analytics platform supports two ways of returning results: either the calculated value is directly returned or by calling the method *MetricsUtils.resultWith*. This function binds the context with the payload. Listing 4.15 shows an example of the latter case, while Listing 4.16 contains the associated output schema.

```
var metricsResult = { "sampleProperty": "propertyValue",
                        "sampleProperty1": "propertyValue1"};
return  MetricsUtils.resultWith(metricsResult, mc);
```

Listing 4.15: Returning metrics results.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
     "sampleProperty": {"type": "string" },
     "sampleProperty1": {"type": "string" }
  },
  "additionalProperties": false
}
```

Listing 4.16: Corresponding schema to the result returned in Listing 4.15.

Actions

As introduced in Section 3.2.3, actions perform notifications based on the results of the metrics function computation. The implementation of actions comprises a condition (filter), a target, and a type. The type can be either PUSH or POST, which indicates the method used for the notification. The target aims at defining the scope of the action such as application, session, or user. For example, in the context of the location-aware games, a target could be all the players of the application, a group of users playing in a game session, or even a specific user.

Actions are evaluated and executed once the scheduled metrics functions are evaluated (Section 4.1.5). As a consequence, action filters can use the metrics results while evaluating the conditions for triggering notifications. For doing this, the value of the last execution of the metrics function is obtained by accessing the property `<metricsFunctionName>.<value>`.

To illustrate the previous idea, Listing 4.17 shows a filter defined on a threshold of the result of `simpleMetric` metrics function. Note that the first parameter of `MetricsUtils.actionResult` is a Boolean value specifying if a notification should be sent, while the `actionPayload` parameter indicates its payload. The context passed with the variable `mc` (namely, metrics context) permits to deliver notifications to the right recipients. Otherwise, it is indicated by calling `MetricsUtils.noAction()`. Accessing the value of metrics results is also possible by simply using the expression `complexMetric.value`, for example.

```
if (simpleMetric.value > 100) {
  var actionPayLoad = { "sampleProperty": "propertyValue" , "
   sampleProperty1": "propertyValue1"};

  // Notify if fist parameter is true
  return  MetricsUtils.actionResult(true, actionPayload, mc);
} else {
  return MetricsUtils.noAction();
}
```

Listing 4.17: Example implementation of a filter function for an action.

4.1.6 Metrics output visualisation

Visual analytics tools are key to support decision making processes (Andrienko et al., 2010). The same is true for metrics evaluation. In the conceptual architecture of the analytics platform (Figure 3.1), data visualisation and reporting tools are situated at the end of the analytical pipeline, taking metrics outputs as input to potentially carry out in-depth analysis. These visualisation tools gain access to metrics outputs through the *data retrieval* services introduced in Section 4.1.2. In this case, these RESTful services access the clustered database, which implies that both input and output data are available to this type of client applications.

Rather than building sophisticated visual analytical tools, the focus has been on making data accessible and easily reachable through public endpoints, so developers can download data and create custom visualisations with their preferred environments and tools. Notwithstanding, we still developed a default, generic visualisation tool. The tool supports both map-based (Figure 4.8) and tabular-based visualisations (Figure 4.9) of collected data and metrics outputs, and permits the download of data in various open formats (e.g., CSV, JSON, GeoJSON).

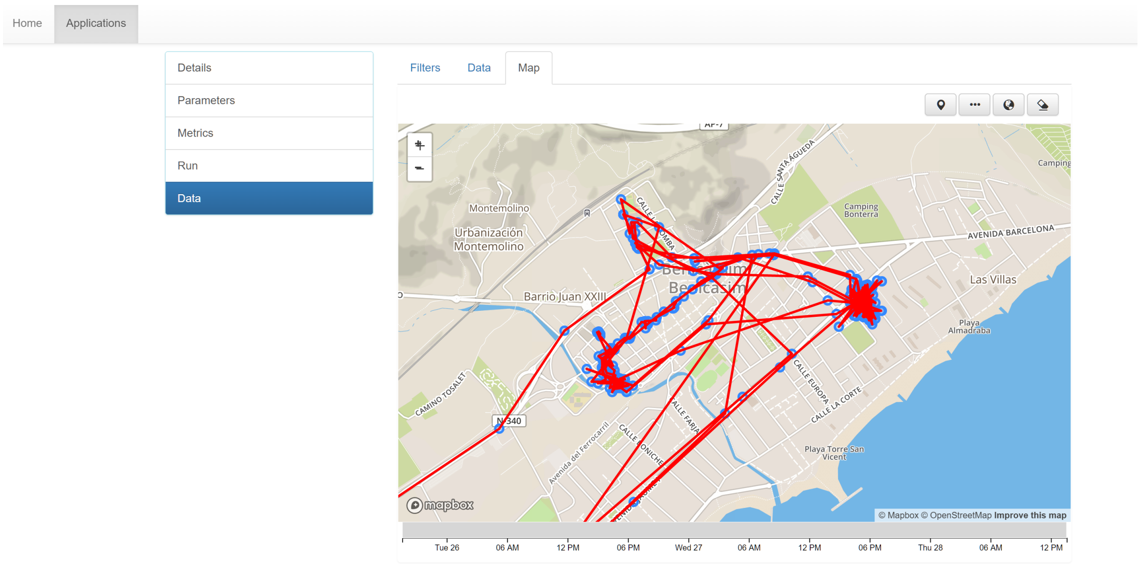
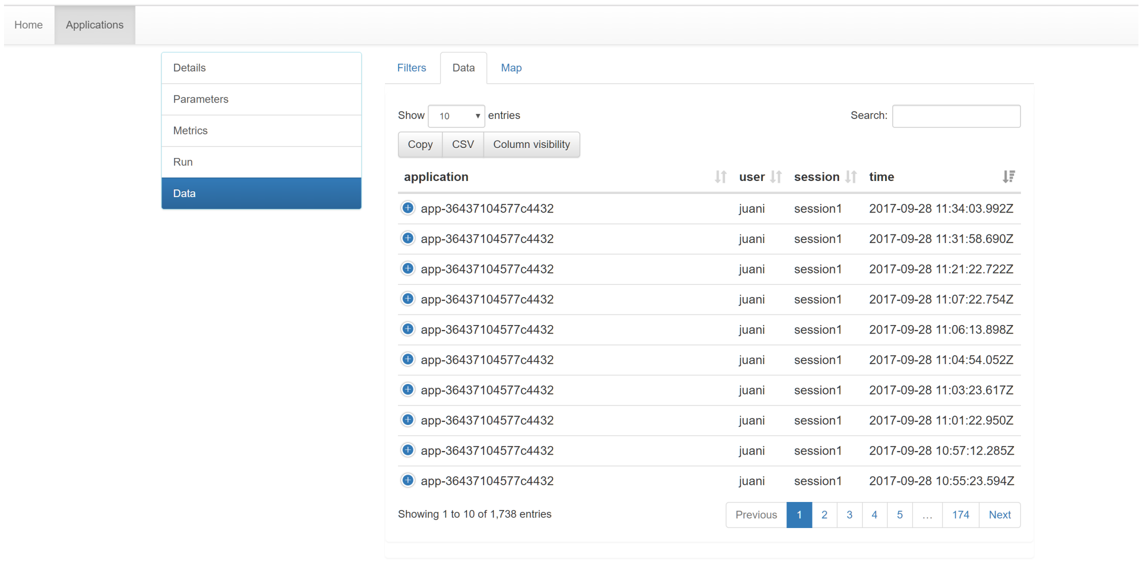


Figure 4.8: Map-based visualisation of collected trajectory data stored in the analytics platform.

4.2 Discussion and comparison with existing platforms

In Chapter 2, we surveyed a varied set of academic and commercial tools for game analytics based on a set of functional areas and key features per area that



application	user	session	time
app-36437104577c4432	juani	session1	2017-09-28 11:34:03.992Z
app-36437104577c4432	juani	session1	2017-09-28 11:31:58.690Z
app-36437104577c4432	juani	session1	2017-09-28 11:21:22.722Z
app-36437104577c4432	juani	session1	2017-09-28 11:07:22.754Z
app-36437104577c4432	juani	session1	2017-09-28 11:06:13.898Z
app-36437104577c4432	juani	session1	2017-09-28 11:04:54.052Z
app-36437104577c4432	juani	session1	2017-09-28 11:03:23.617Z
app-36437104577c4432	juani	session1	2017-09-28 11:01:22.950Z
app-36437104577c4432	juani	session1	2017-09-28 10:57:12.285Z
app-36437104577c4432	juani	session1	2017-09-28 10:55:23.594Z

Showing 1 to 10 of 1,738 entries

Figure 4.9: Tabular-based visualisation of the data in Figure 4.8.

helped us to perform the comparative analysis. Both the functional areas and the features are presented globally in Table 2.2 and in detail per each functional area in Tables 2.3–2.7. In this section, we analyse the analytics platform presented in this thesis based on these same functional areas and features as in Chapter 2. Table 4.6 summarises the characterisation of the implemented analytics platform regarding these functional areas and features, which allows us to compare it against the tools seen in Chapter 2. The following discussion is organised per functional area.

Regarding the data communication and collection, we implemented an event-based data collection strategy and supported streamed events as communication approach. The platform provides an SDK that eases external client applications to interact with it and helps in the handling of communications and event messages conforming with the event types defined through the metrics definition variables in the back-end. As we showed earlier in this chapter, the SDK functionalities significantly simplify the client-side development of Android applications. Besides, managerial and data access services are described through the Open API specification, which allows a well-defined interaction with the platform. The Open API specification provides a support mechanism to allow developers to generate a wide variety of client applications based on different programming languages and

Table 4.6: Functional areas support in the game metrics analytics platform.

Functional Areas	Features	Support
Data collection and communication Table 2.3	Client-side development support	REST API available for data communication. Described through an Open API specification. SDK provided for Android.
	Data communication strategies	Streamed Events.
	Data collection strategies	Event based.
Data representation Table 2.4	Default event (data model) types	Partially, all event types must be defined, but default dimension types are provided.
	Custom event types definition	Yes, through Variables and dimensions. Custom dimensions specified through JSON Schema.
	Spatial support (for data modelling)	Yes, mainly location and space attributes.
Data analysis and reaction Tables 2.6 and 2.5	Reactive rules	Storage, Notification and Custom.
	Spatial support (conditions & actions)	Yes, several operators available for conditions. Actions supported through the data submission API.
	Default metrics and games analysis	Yes, some default metrics are provided.
	Custom metrics definition	Yes, using Javascript and EclairJs.
	Spatial support (for analysis)	Yes, TurfJs functionalities available.
Data visualisation and reporting Table 2.7	(Open) Data Access	Yes, REST API.
	Visual analytics	Tables.
	Spatial support (for visual analytics)	Routes and Point Maps.

third-party platforms (e.g., JavaScript, Powershell, Ruby, to mention a few [22]). This Open API specification differs from the functionalities provided by the SDK in that the latter provides support not only to interact with the backed-end services, but also for the handling of notifications, communication, and data collection related functionalities such as data submission failure handling, and default dimensions providers.

In comparison with other tools in Table 2.3, the features of the analytics platform pertinent to the support of client-side development clearly stand out. Features such as the REST API, the provided SDK and the description through Open API, provides improved client-side development support when compared with other systems. We attribute this difference to the design requirements of the systems analysed, which in some cases targeted data collection and processing for specific, in-house systems, and had no intention to cover a wider audience. It should be also noted that some tools in the analysis were designed and implemented in a different "technological" context, where some of the features and supporting technologies were not fully developed or popularized at the time and, therefore, their acceptance was not as wide as it is today. This conditions the development of tools (i.e., libraries, frameworks), languages, and protocols used, as a basis for the construction of software in general.

[22]Example code generation tools: https://swagger.io/tools/swagger-codegen/

When it comes to the data representation functional area, our analytics platform offers significant flexibility for data representation, by allowing the developers and designers to specify versatile data structures through a custom event type model. Considering that the custom event types can be defined dynamically, application re-build and deployment is not necessary while defining the event types. A set of default types of dimensions helps with the custom event types definitions. The spatial support in this model is provided in different ways, for example with the inclusion of spatial dimensions in the variables (both location and orientation are supported), and through the use of custom dimensions to encode spatial data. In this context, complex spatial data is supported in the back-end platform through the use of Turf.js. The tools discussed in Chapter 2 provided spatial support through location/space attributes, and to some extent, our platform uses a similar approach. The difference lies in that our platform is more flexible and extensible because other spatial attributes can be included through custom dimensions of the GeoJson encoded data. In addition, our platform employs standard formats like GeoJson, which is fully supported by the spatial capabilities of the platform.

In regards to the data analysis, we have included default metrics that range from general purpose statistics to more spatial-related ones. All these metrics have in common, on one hand, a shared model that can be adopted by others (as data structures) and, on the other hand, computations are described and defined in a portable, human-readable format that does not imply the need of re-building the application while incorporating new changes. Most of these features are driven by the design decision to provide the ability to define custom metrics through the use of JavaScript, and offer functionalities that "understand" the underlying data model, as described through variables and dimensions. Besides, concerning spatial support, we have provided tools for supporting spatial analysis, a feature that has received scarce attention in the tools for game analytics seen in Chapter 2, see Table 2.5. In our platform, we developed spatial functionalities by including *Turf.js*[23] as a base library, which provides to some extent a level of spatial support (GeoJson format, build-in spatial operations) that covers most functionalities widely used in the context of location-aware games.

[23]https://turfjs.org/

Reaction is supported using notifications through widely used channels such as HTTP and PUSH notifications which eases the integration with third-party tools and systems. Notifications can be conditioned through rules that can involve non-spatial and spatial data produced by computation tasks. As presented in Chapter 2 and summarised in Table 2.6, these features are barely provided in most of the tools analysed, as they only support notifications in some cases.

The implemented analytics platform is intended to be a general-purpose system, yet with specific support for spatial analysis too. In the functional area of data visualisation and reporting, we followed this approach and provided general-purpose visualization tools, both for spatial and non-spatial data. These tools allow data inspection, basic data filtering by the properties of the context (i.e., user, session, application). The platform provides ways to export and access collected and processed data (metric results) to be ingested by sophisticated third-party data visualization tools. Data access for external tools is based on a REST API that is documented through Open API (Swagger) specification. Regarding data access, the features included in our platform are similar to the ones provided by some systems, as summarised in Table 2.7, while in the field of visual analytics, we have not implemented advanced tools (as in the case of some of the systems compared). We have only including basic support for visualizing tables and, in the spatial aspect, only basic visualisation of locations and routes.

In summary, the analytics platform has implemented an architecture for supporting a large number of features related to data collection, processing and storage. It inherently promotes scalability, such as through streamed data collection, distributed data storage, and a scalable computation framework for enabling distributed data processing. Although the platform is able to handle any type of user-collected data, the geospatial aspect of the data is particularly important in the context of location-aware games. We have implemented a set of features to ease the collection, storage, and processing of geospatial data. The platform handles both general-purpose data and spatial data, by providing a model based on variables and dimensions, which are crosscutting concepts with materialization in different aspects of the platform. These high-level abstractions make the platform re-usable for various applications domains, requiring only the definition of the data model and the metrics through the built-in tools and support services of the platform. In comparison with the systems analysed in Chapter 2, the platform stands out in two ways: the back-end services provide spatial support while using a scalable data processing framework, and include spatial related functionalities (such as geofencing ones) that are commonly used in the context of location-aware games. Besides, the platform incorporates features related to reaction that, in most aspects, are not supported in the analysed platforms and tools.

Chapter 5

EXPERIMENTATION AND ASSESSMENT

It is commonly known that the evaluation of a framework or platform is inherently difficult (Jogalekar and Woodside (2000); Neuman (1994); Duboc et al. (2007)), as it needs to assess subjective properties such as scalability, code abstraction, ease of use, etc. In this chapter, we embark on a multi-faceted assessment of the analytics platform. First, we qualitatively discuss the global platform's design and architecture towards technically desirable features and ease of use (Section 5.1). Second, we evaluate the platform's applicability in two real-world cases in two different domains, to demonstrate its use in different application areas, and to assess the aforementioned features and the platform's use from a developer perspective. The real-world case studies consist of a game to acquire noise measurements and an application for mental health. For the former, an experiment is set up, whereby the game was implemented with and without the analytics platform, followed by a qualitative comparison and discussion. The latter case study shows the versatility of the analytics platform and its application beyond location-aware games.

5.1 Discussion of Features of the Analytics Platform

In Section 1.2, as part of the research objectives R03 and R04, we indicated the main features for the platform to address. For this discussion, we grouped them into those belonging to implementation and architectural related characteristics of the platform and those referring to the ease of use of the platform from a developer perspective. Our purpose in this section is to present how the analytics platform, as a whole, addresses each of these features. In what follows, we first define the implementation and architectural related features (asynchrony, scalability, and the use of high-level abstractions) and discussed them in the context of the platform. Next, we dive into the ease of use features encompassing ease of setup, ease of development and learnability, and discussed how they globally applied to the analytics platform.

Asynchrony refers to the ability of the platform to handle and run requests asynchronously. Unlike synchronous requests, these are treated in independent execution threads and processes (in distributed systems, as it is the case of our platform implementation) and do not block the client application (or other internal components in the server side of the analytics platform) while the requested process is being executed. The analytics platform supports asynchronous requests by providing services that handle incoming requests (e.g., metrics execution and data submission requests) and deliver other requests to other microservices where the actual processing is performed, as explained in Sections 4.1.4 and 4.1.5. At a low level, this feature is supported by the asynchronous nature of actors provided by the Akka framework. Both RESTful services and backend microservices (see Figure 4.6) implementations are based on actors performing processing tasks in an Akka cluster. In addition, asynchrony is assisted by the notification services (also shown in Figure 4.6) that asynchronously deliver responses (through the defined actions) for metrics computation requests (e.g., notifications in case of the metrics processing).

Another feature is scalability (Duboc et al., 2007), which refers to the ability to handle an increasing volume of work (e.g., requests, computation, etc.) without degrading performance of the analytics platform. The platform tackles it by being implemented as a distributed system, where data ingestion and metrics processing tasks are separated and these services form a cluster that support multiple service instances. Thereby, it is possible to configure the platform for running several data ingestion services (note that these services can be part of different processes and even be deployed in different hosts) and metrics processing microservices. This confers horizontal scalability to the platform to adapt to different request volumes. Internally, specialised Akka actors handle messages between actors (belonging to the services) in a cluster. These actors use routing[1] and balancing mechanisms for handing requests to the different services instances (i.e., actors in services running in the cluster). Database scalability is also achieved by using Cassandra distributed data storage (Abramova et al., 2014). Finally, from the perspective of data processing, we have used Apache Spark in our platform to leverage Spark's big data processing capabilities (Inoubli et al., 2018) and processing model (Zaharia et al., 2016a). Apache Spark has become one of the most important framework for implementing big data applications with applicability in several fields, as some papers show (Nothaft et al., 2015; Freeman et al., 2014).

Abstraction is the last implementation and architectural related feature. It refers to the extent to which the underlying metrics model is either specific and geared to a particular domain or application (e.g., the Noise Battle game), or it can be widely applied to a variety of applications domains. In the analysis provided in Chapter 3, we presented the concept of variables and dimensions (see Section 3.2.1) as a data representation model for the platform, metrics and actions for processing and results handling. This model fits well in most problems in the context of location-aware games. During the implementation of the platform, these abstractions have materialized in different components of the platform, and have proven to be useful to support advanced aspects of the platform (e.g., storage, data processing, client side data validation, etc.).

Besides the implementation and architecture related features, one of our objectives was to create a platform that would be easy to use for developers, favor-

[1]More information at: `https://doc.akka.io/docs/akka/current/cluster-routing.html`

ing the reuse of the definitions created in the application domain of location-aware games and applications. Through the metrics definition, we designed a representation of the abstractions provided that can be easily created and maintained by using the tools provided and that, at the same time, is readable and shareable between different systems. We chose a representation based on widely used formats for favoring specifically the reuse of the model for varied purposes. The platform itself does intensive use of the metric representation (the metrics definition files). Data ingestion, storage, the client Metrics SDK, and processing services leverage different aspects of the metrics representation for data structure definition, validation, processing and scheduling.

The second group of features falls into the category of ease of use. It comprises a set of features to alleviate, from the perspective of the developer, the complexity of development of a solution: ease of setup, ease of development and learnability.

Ease of setup is related to how easy would be for a developer to setup an application on top of the platform. It comprises the necessary steps to make the platform usable by a client application for data collection and analysis. For using the platform, it is necessary to create a new application through the metrics definition tool provided. This tool also allows to setup aspects related to notifications, such as the keys for using the Google Cloud Messaging (GCM) services and the metrics itself. These steps are assisted by the metrics definition tool. Clients applications setup is also facilitated as the Metrics SDK for Android, which is published in a repository to facilitate task of including the library in the Android applications. The setup needed in the client application is related to the configuration of the notifications (which requires a key provided from Google services). Besides, it is also necessary to configure the application id (provided when creating the application in the metrics definition tool), the session and user information, during the SDK library initialization.

From the programming point of view, the platform provides access to the data collected in a way that is coherent with the metrics model proposed, easing the adoption of the platform. Therefore, ease of development is supported by a set of functionalities that are related to the creation of location-aware games, easing the implementation of spatial related features. Adding new functionalities to the applications is achieved by including new metrics functions, a process that does not require redeployment of the application, as the metrics functions are evaluated dynamically by the platform. For the development of client applications, we provide an SDK for Android. It offers a set of functionalities that are also coherent with the metrics model proposed. This way we have tried to keep a common set of concepts through all the components of the platform. For example, variables are the main data structure for the data collection functionalities offered and the context must be established for the SDK before submitting data to the platform. The SDK loads and keeps the metrics definition in the client applications at startup and upon data submission, the data is then validated based on the metrics definition specification. Besides, the Android SDK (see Section 4.1.2) simplifies the development by offering functionalities that reduce the amount of boilerplate code needed for an application to interact with the metrics platform such as notifications handling, automatic sensor data collection and data submission handling (i.e., error and retrial).

Learnability is the last ease-of-use feature. As a key element of usability (Grossman et al., 2009), learnability alludes to the capacity of the analytics platform and the metrics definition tool to allow the developer to learn how to use/code on it. Note that we do not treat usability as a criterion due to the broad scope of the term. In the platform, we favor learnability by providing users/developers with tools for creating the different elements of the metrics, as well as providing a few abstractions for defining the data model and the processing requirements of the problem at hand. The simplest problems that do not require specialized data types, for example, can be modeled through the metrics definition tool with ease. More complex scenarios can be implemented through the same tools by using open, well-known formats (e.g., using JSON schema). Regarding processing, the learnability is favored through the use of JavaScript as the base language for defining the metrics functions, and Apache Spark API, which since few years has become popular and widely adopted as a platform of choice for big data processing. Due to its popularity, Apache Spark programming model is well known and documented in the field of big data processing. Furthermore, it is worth noting that we have kept a common set of concepts (variables, dimensions) through the platform, which can ease the understanding of different aspects of the systems. For example, the notion of context is reused in the client side in aspects such as the notifications handling and client application setup.

5.2 The Noise Battle Experiment

5.2.1 Design and setup

Rather than monitoring and recording internal indicators of the analytics platform, we focus attention on how the analytics platform support developers in the creation and development of client applications, i.e., location-aware games. In other words, if the platform improves and eases the tasks of development and monitoring of location-aware games. The strategy was to implement the same client application with and without the analytics platform and to observe differences between the two. For this, we rescued a previous game development prior to the analytics platform, which did compute spatial metrics too. Logically, the next step was to adapt it to be compatible with the analytics platform. We considered developers as the primary end users. This means that the platform was tested and evaluated in terms of how easy it was to create useful client applications and spatio-temporal metrics on top of it.

In what follows, we report on two variants of the same experiment. The first one is the original Noise Battle location-aware mobile game, which has been published elsewhere (Martí et al., 2012). Implemented by the author, the app computes spatial metrics as part of the gameplay. As we will show later, that implementation is pre-platform. Subsequently, we came up with the concept of spatial metrics and the analytics platform as explained in the previous chapters. Once the analytics platform was implemented, two post-platform experiments followed. The fist one was the relaunch of the Noise Battle game in connection with the analytics platform. The before and after of the Noise Battle app is described in Section 5.2.2. The second experiment is only post-platform (Section 5.3). It aims to validate the versatility of the analytics platform to cover cross-domain applications and experimental use cases. In section 5.2.3, we compare the two versions of the Noise Battle game and discuss differences.

5.2.2 Experiment

The Noise Battle game (Martí et al., 2012) is a multiplayer location-based game for the collection of noise measures using a mobile phone's microphone in a city. It was created in the context of a workshop held in the University Jaume I of Castellón de la Plana, Spain. The workshop's goal was to propose and discuss research ideas and prototype applications for participatory collection of urban noise measurements to contribute to the University of Münster's Open Noise Map platform (Schweizer et al., 2011), much like the Open Street Map platform for noise pollution.

The general idea of the game is to engage players in "battle" for conquering the city while collecting noise measurements and rewards around the city. For this purpose, the city is divided into grid cells of equal size, called blocks, with multiple rewards distributed in each block (see Figure 5.1). For collecting rewards, the player must take noise measurements within a few meters around the location of the reward. Noise measurements (in dB) are taken with the microphone sensor of the mobile devices. Importantly, the action of taking measurements also implies that the player obtains the ownership of the block (regardless of obtaining a reward or not), or could appropriate it from the previous owner, and therefore "conquer" it. At the same time, the player accumulates points that can be used to send noises or sounds to enemies' phones.

Depending on the state of the block in which the user takes the measurements, the game rewards with different scores. The block's state is configurable as part of the gameplay. By default, if a player takes a measurement with a "better" value than the previous used to conquer the block, the player is awarded 15 points. If the block was not previously conquered, the number of points awarded is 10. In contrast, if the taken measurement is "worse" than the existing one, only 3 points are awarded. The criteria for determining if a measurement is better or worse is based on the noise level (dB) in the block. That is, greater levels of noise is considered better. Noise level is normalized by the underlying noise

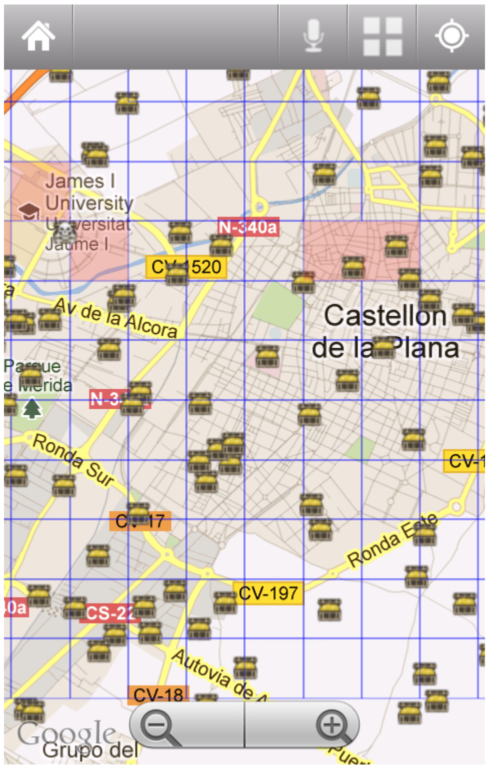


Figure 5.1: Original interface of the Noise Battle location-based game. The city map denotes the battle field and each grid cell the block to conquer.

collection library called NoiseDroid[2]. A player can lose ownership of a block if no measurements are taken in a certain amount of time (e.g., a few days).

Figure 5.2 shows the game in action. Each player has assigned a unique color. When a player conquers a block, he gets a better noise measurement than the actual one, that block is coloured with the player's colour. Not all blocks have got the same importance, geographically speaking. Special blocks for noise monitoring are strategically distributed over the city in points of interest such as schools and hospitals due to scientific, social or urban planning reasons. There, players can collect special rewards (shown in Figure 5.2 with a chest icon), which give 20 points to the player.

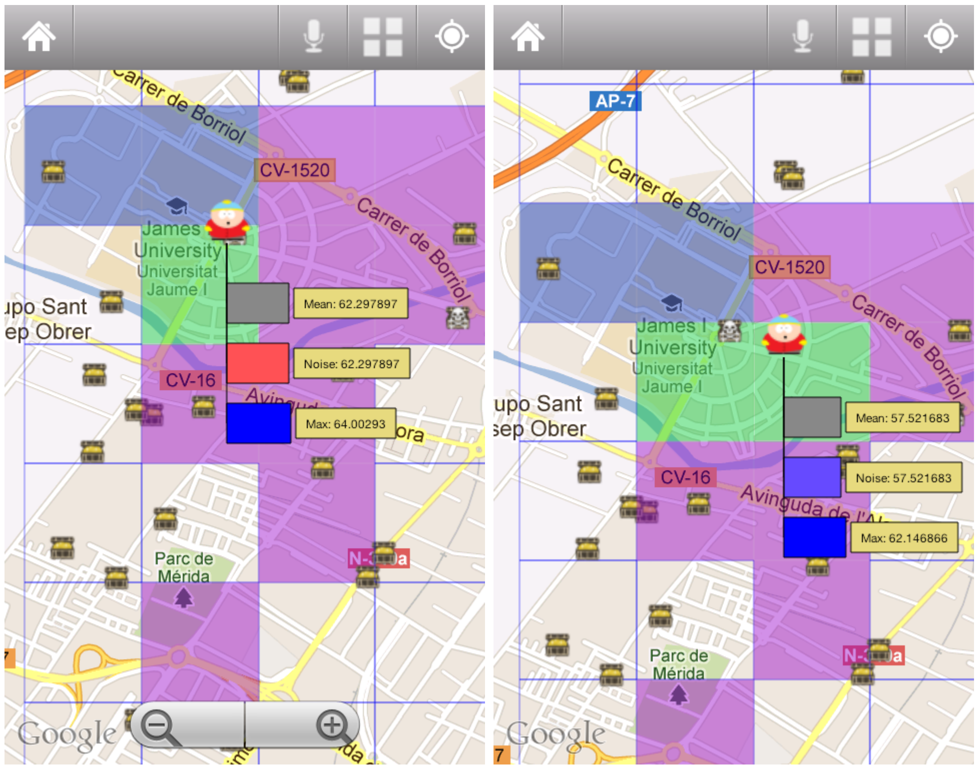


Figure 5.2: NoiseBattle in action. Coloured cells represent blocks conquered by different players. Players are assigned unique colors at the beginning of the game. Chest icon denotes relevant points of interest for noise monitoring.

The rewarding system aims to encourage regular and up-to-date measure-

[2]NoiseDroid source code can be found in the project noise-battle in Section 6.2, under the folder *noisedroidlib*

ments taken from different players and encourages spatial distribution of the measurements over the city area. During the game, notifications are essential. The game notifies the players whenever one of their blocks is under attack because other players are collecting noise measurements there, and therefore that block is most likely to be conquered. Besides, notifications serve as a mechanism for sending (sharing) noise/sounds between the players. We refer to Martí et al. (2012) for additional description of the gameplay of the Noise Battle game.

Pre-platform implementation

The original implementation of the Noise Battle game uses a service layer of SOAP services[3], which in turn rely on PostgreSQL[4] (with PostGIS[5]) and Hibernate spatial[6] for storing the games' geotagged noise measurements. The game's architecture is illustrated in Figure 5.3. Spatial computation required by the game setup is necessarily performed by the spatial extensions of the back-end PostgreSQL/PostGIS database. These spatial extensions support a great deal of spatial data types for storing and handling spatial data, such as Geometry and its specializations (i.e., Polygons, Polylines). Besides, these extensions include an extensive set of spatial methods, ranging from simple methods to create geometries to more complex operations such as geometry simplification or topological relations between geometries.

For the implementation of the game setup, we used spatial operations, for example, to create the battlefield (i.e., city grid cell), to determine the block a player is taking a measurement in, and to check the proximity of a player to a rewarding item. Each of these operations was represented by a spatial query against the database. Listing 5.1 shows an example of a spatial query to determine the block in which a player has measured the noise level.

The game client was implemented as an Android application. It displays a grid over a city map, and uses the GPS sensor and microphone of the mobile device to collect spatially-referenced noise measurements. From an implementa-

[3]Implemented using Apache Axis
[4]https://www.postgresql.org/
[5]https://postgis.net/
[6]http://www.hibernatespatial.org/

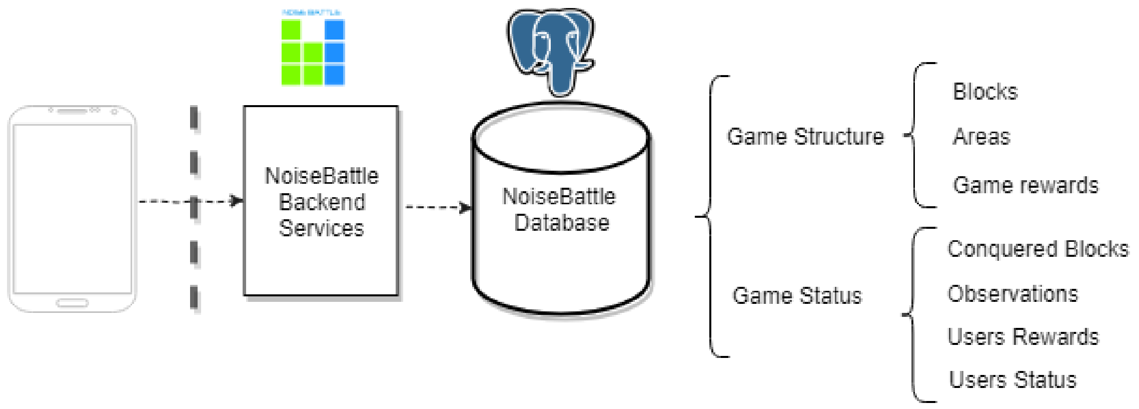


Figure 5.3: Architecture of the pre-platform Noise Battle game.

```
select b.* from observation o, block b, game g
  where o.game_instance = :instance
    and o.game_id = :gameId
    and g.game_id = o.game_id
    and b.area_id = g.area_id
    and st_Contains(b.bbox, o.coordinates)
    and o.observation_id = :observation
```

Listing 5.1: SQL Query using the spatial extensions to obtain the block where the observation was collected.

tion perspective, the game setup is straightforward. When a player collects a new observation in a block, the back-end services determine which block is affected and searches for the owner (if any). If the block is not owned (conquered) yet (i.e., no measurements registered yet), the game immediately assigns it to the current player, updating his score according to the rewarding system. In case the block was already conquered by another player, new and old noise measurements are compared. If the new noise measurement beats the old one, the block's ownership is assigned to the new player (updating his/her score accordingly). Otherwise, the ownership of the block in battle does not change.

Both gathered measurements and the state of the game (gameplay) were stored in a set of relational tables in the back-end database (Figure 5.4). Most of the tables are used to support the *game setup*, such as users, (game) areas, rewards, and blocks. Another group of tables, though, covers the *game state* such as the tables status, status of rewards, noise levels sent, observation, and conquered blocks. Each group of tables is briefly described below.

Game setup related tables are:

- **game:** Is the actual instance or game session in which players can enter for the battle. Games have an associated area, and as a consequence, a set of blocks.
- **area:** Defines a template of the game where the instances of the game (game sessions) can take place. This is usually one per city, although there are no restrictions for the number of areas in the same place.
- **block:** Defines a block as the minimal space unit which users can conquer by taking measurements. Blocks are subdivisions of an area.
- **game_rewards:** Defines a template for the distribution of rewards associated with an area.
- **observation:** Keeps the information about noise measurements collected by the players in a game session and the different elements of the measurement (e.g., *observation_id*, *user_id*, *game_id*, *coordinates*, *time_stamp*, *game_instance*, *morale*, *avg_morale*, *noise_value*, *min_noise*, *max_noise*, *avg_noise*, *other_data*).
- **users:** Keeps the players involved in the game.

Game state related tables are:

- **status:** Represents the state of the game, which is defined as *conquered_blocks, observation, noise_send,* and *status_rewards*.
- **conquered_blocks:** Represents the state of the blocks, containing the current player holding the ownership of the block.
- **noise_send:** Contains information about the noises/sounds exchanged between the players.

- **status_rewards:** Contains the rewards obtained by the players in the game.
- **game_statistics:** Stores general game statistics of the game sessions.

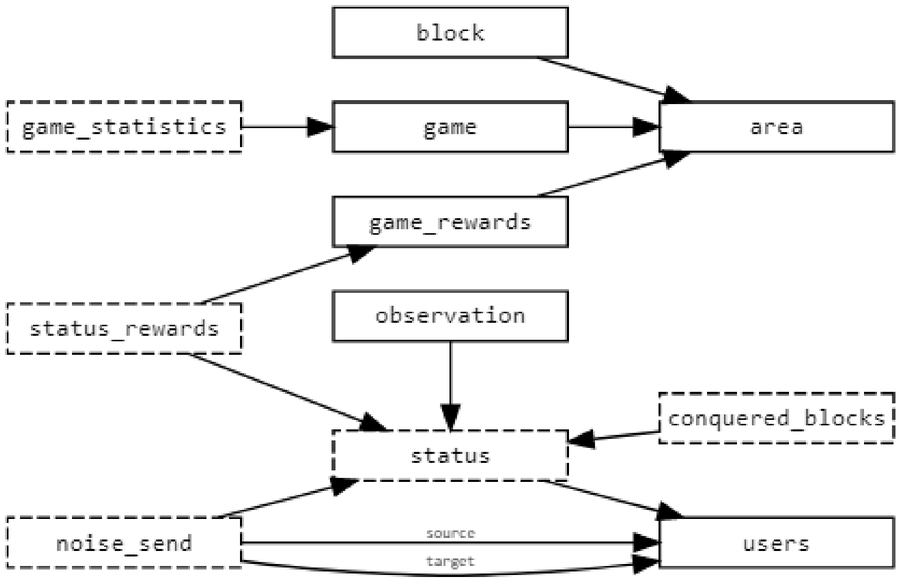


Figure 5.4: Table relations of the pre-platform Noise Battle game. Dotted border denotes game state related tables. Solid border denotes game setup related tables.

Post-platform implementation

The goal of the Noise Battle experiment is to compare the original implementation with an implementation of the game using the analytics platform. For a fair comparison, the new version must completely support the game setup of the original one. As the game setup does not change, the focus here is on the implementation approach of the post-platform Noise Battle game.

The key to the implementation is to replace all (spatial) analytics of the original Noise Battle game with equivalent functionality offered by metrics of the analytics platform. To achieve this, the back-end services to perform score calculations every time a new noise observation is submitted were replaced by metrics services in the analytic platform. Therefore, as a first step, we mapped (or replicated) the structure of the database tables (Figure 5.4) to variables in the analytics platform. Secondly, we created metrics for calculating the scores that a player obtained while collecting/submitting noise observations. As a result, the services layer of the pre-platform implementation served as a bridge for consuming the metrics services exposed by the analytics platform, while maintaining the same service interfaces to interact with the original Noise Battle mobile application (Figure 5.5). Therefore, this approach does not utilise the Metrics SDK seen in Section 4.1.2 for data collection. Instead, it uses an intermediate layer of REST services to make the pre-platform data structures compatible with the new ones, requiring in practice fewer modifications.

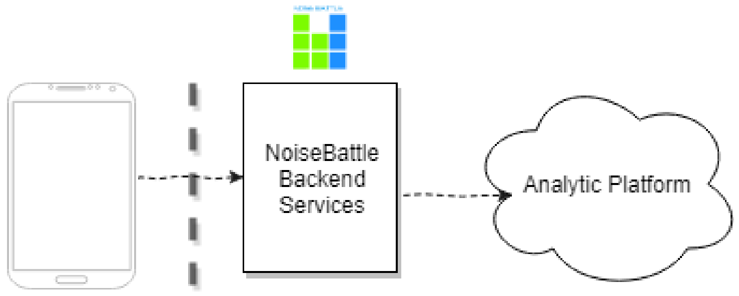


Figure 5.5: Post-platform implementation of the Noise Battle game uses only the analytics platform, leaving aside the Metrics SDK.

For the first step of the mapping, we observe the fact that the values in the state-related tables in the original implementation (dashed border boxes in Figure 5.4) were the result of the execution of the score evaluation algorithms over an incoming observation. For example, the conquered blocks are the result of jointly evaluating the level of accuracy of the submitted noise observation, the current value (if any), and the game state in terms of game objects (i.e., blocks, players, rewards, etc.), which are stored in the game setup related tables (solid border boxes in Figure 5.4). In the context of the analytics platform, this means that a new state of the game can be understood as the result of a metric computation and, consequently, the output definition of the metrics should contain the same information present in the relevant state-related tables of the original implementation. Thus, we created the variables *observations*, *areas*, *blocks*, *gameRewards* and *users* resembling the structure of the tables in Figure 5.4.

Listing 5.2 shows the declaration of the *observations* variable with multiple *dimensions*. In particular, an *observation* has got the default dimension `location` that refers to the position where the noise was collected. By using and declaring location in this way, an hypothetical implementation of a mobile client through the Metrics SDK (for Android) would fill automatically it by capturing GPS coordinates directly. Note that, as said earlier, we did not implement a new game client but used the old client implementation, replacing all (spatial) analytics of the original Noise Battle game with equivalent by metrics running on the analytics platform. Returning to Listing 5.2, some of the remaining dimensions represent a range of noise parameters such as *maxNoise*, *minNoise* and *morale*, while others are pertinent to the game setup (*areas*, *blocks*, *gameRewards*). In summary, all of these dimensions can be defined either using the metrics definition tool (Figure 4.3), or writing manually a declarative metrics-schema compliant JSON file, as shown in Listing 5.2.

```
"variables": [{
  "name": "observations",
  "description": "Variable for collecting the noise data",
  "dimensions": [
    {"type": "location"},
    {"name": "observationId", "type": "integer", "description": ""},
    {"name": "userId", "type": "integer", "description": ""},
    {"name": "gameId", "type": "integer", "description": "" },
    {"name": "gameInstance", "type": "integer", "description": ""},
    {"name": "morale", "type": "double", "description": ""  },
    {"name": "avgMorale", "type": "double", "description": ""},
    {"name": "noiseValue", "type": "integer", "description": ""},
    {"name": "minNoise", "type": "double", "description": ""},
    {"name": "maxNoise", "type": "double", "description": ""},
    {"name": "avgNoise", "type": "double", "description": ""},
    {"name": "otherData", "type": "string", "description": "" }
   ]},
  ...
```

Listing 5.2: Metric schema specification of the variable *observations*.

For the second step of the mapping process, modelling the back-end methods of the original Noise Battle game, we have created four metrics for calculating the new state of the different aspects of the game: *calculateStatus* that internally calls the metrics *calculateConqueredBlocks* and *calculateStatusRewards*, which update the state of the conquered blocks and the state of rewards in the game, respectively. The last metric function *calculateGameStatistics* summarizes the information of the game and the players. Both metrics functions, *calculateConqueredBlocks* and *calculateStatusRewards*, take as input a noise observation to determine the new state in the game; the data structure of the outputs (See Listing 5.3) is similar to the original game's tables *conquered_blocks* and *status_rewards*.

```
{
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "conqueredBlockId": {"type": "integer"},
    "userId": {"type": "integer"},
    "gameId": {"type": "integer"},
    "blockId": {"type": "integer"},
    "numTimesConq": {"type": "integer"},
    "measuresTaken": {"type": "integer"},
    "gameInstance": {"type": "integer"},
    "wasConquered": {"type": "integer"},
  },
  "required": [
    "blockId",
    "conqueredBlockId",
    "gameId",
    "gameInstance",
    "measuresTaken",
    "numTimesConq",
    "userId",
    "wasConquered"
  ]
}
```

Listing 5.3: Output schema of the conquered blocks metric function.

Listing 5.4 shows the code of the metric function *calculateConqueredBlocks* that calculates the values of the current state as output.

```
function wasConquered(allObservationsInAffectedBlock) {
    var sameBlockObs = allObservationsInAffectedBlock.take(2);
    var firstObs = sameBlockObs[0];
    var secondObs = sameBlockObs[1];
    return MetricsUtils.getFieldValue(firstObs, "noisevalue")
                < MetricsUtils.getFieldValue(secondObs, "noisevalue"
    );
}

var blocks = session.blocks; // Observation for this game instance
var lastObservations = MetricsUtils.getRDD(session.observations).
    take(1); // Ordered, current and prev work as expected

var location = GeoUtils.getLatLon(MetricsUtils.getRowData(
    lastObservations[0]).latlon);
var affectedBlocks = blocks.filter(function (row, location) {
    var bboxFieldData = MetricsUtils.getFieldValue(row, "bbox");
    var bbox = bboxFieldToBbox(bboxFieldData);
    var point = GeoUtils.createPoint(location);
    var value = GeoUtils.isInside(point, turf.bboxPolygon(bbox));
    return value;
}, [location]);

var affectedBlock = affectedBlocks.take(1)[0];
var bboxFieldData = MetricsUtils.getFieldValue(affectedBlock, "bbox
    ");
var affectedBlockBBox = bboxFieldToBbox(bboxFieldData);

var allObservationsInAffectedBlock = session.observations.filter(
    function(row, bbox) {
    var data = MetricsUtils.getRowData(row);
    var point = GeoUtils.createPoint(data.latlon);
    var inside = GeoUtils.isInside(point, turf.bboxPolygon(bbox));
    return inside;
}, [affectedBlockBBox]); // Ordered, current and prev work as
    expected
    // Get the conquered block
var conqueredBlocks = calculateConqueredBlocks.history.filter(
    function (row, session, blockId) {
         var blockGame = MetricsUtils.getFieldValue(row, "session");
         return blockGame == session;
     },[mc.session, affectedBlock.blockId]).collect();
var conqueredBlock = conqueredBlocks.length ==0? null:
    conqueredBlocks[0];

var measuresTaken = conqueredBlock!= null ? conqueredBlock.
    measuresTaken + 1: 1;
var score = 0;

// If already mine (user mc.user), then include measures taken
var mine = conqueredBlock!=null? conqueredBlock.userId == mc.user:
    false;
var numTimesConq = conqueredBlock!= null? conqueredBlock.
    numTimesConq: 1;

if (!mine) {
     var conquered = wasConquered(allObservationsInAffectedBlock)
     if (conquered)  { numTimesConq ++;}
}

var result = {
     "conqueredBlockId": conqueredBlock.blockId,
     "userId": mc.user,
     "gameId": mc.session,
     "blockId": conqueredBlock.blockId,
     "numTimesConq": numTimesConq,
     "measuresTaken": measuresTaken,
     "gameInstance": mc.session,
     "wasConquered": conquered?1:0
  };

return MetricsUtils.resultWith(result);
```

Listing 5.4: Metric function *calculateConqueredBlocks* calculating the new state of the game.

Code in Listing 5.4 relies on a few assumptions. The context is modeled in such a way that the `context session` property refers to the *gameInstance* in the original Noise Battle game (the instance of the game taking place in an area). This makes sense as the players choose to play (battle) in a given area/city (i.e., a game instance), and this can be seen as a “game session” which several players share. Besides, the context’s `user` property is the actual *userId* in the game, which is a natural equivalence. As the variable *blocks* contains the blocks of the games, the blocks for the current game instance are accessible through *session.blocks*. Similarly, the observations of all players in the current game instance are accessible through *session.observations*.

Following this argumentation line, the access to the location of an observation is done by calling the utility method *GeoUtils.GetLatLon()*, and the affected block is determined by computing the blocks that contain the observation’s location (note the use of *GeoUtils.isInside(...)* in Listing 5.4, line 18). After selecting the affected block, the method finds all the observations in the game instance to determine whether or not it is better than the *lastObservation* (Listing 5.4, lines 26-32). Whether a block is conquered is determined by selecting the metrics outputs for the current affected block and the current session.

```
if (calculateConqueredBlocks.value.wasConquered) {
    var actionPayLoad = calculateConqueredBlocks.value
  // To do a notification, the first parameter below must be True
  return  MetricsUtils.actionResult(true, actionPayload, mc);
} else
    return MetricsUtils.noAction();
```

Listing 5.5: Actions filter sample code.

Finally, regarding notifications, the analytics platform can set up the rules for notifying the players in the different situations we mentioned earlier. Listing 5.5 shows how an action sends a notification using the output of the metric function as the notification’s payload. In particular, this action notifies all users about a newly conquered block, including in the payload convenient information such as *blockId*, *conqueredBlockId*, and the number of *measuresTaken*. This information can be used for updating the game playground representation. The entire code in Listing 5.5 is specified in a JSON fragment as Listing 5.6 illustrates. Note how the action *target* property is set to the *current_session*, which refers to the current game instance, and the action *type* is a push notification.

```
"actions": [{
        "id": "notifyConqueredBlock",
        "label": "notifyConqueredBlock",
        "description": "",
        "filter": {
            "code": "...",
            "language": "javascript"},
        "target": "current_session",
        "type": "push_notification"
    }]
```

Listing 5.6: JSON definition of an action to notify conquered blocks.

Our previous discussion in this section has focused on changes made in the back-end implementation, that may not affect the client application implementation directly. To this respect, though, there are important changes related to the way the requests for data submission are handled in the post-platform implementation. A major difference is that the post-platform implementation handles the requests for data submission asynchronously (see Section 4.1.2). Therefore, when the client application sends a new noise observation, the platform does not wait until the calculation of the new game state (i.e., the metrics results) is ready and available to be included in the response. This approach for asynchronous request handling in the analytics platform is incompatible with the original version of the client application, which needs modifications for the post-platform implementation of the Noise Battle game. Instead, results containing the new state is received through notifications.

For the client to work properly, two strategies can be implemented. The first one is to get the new state from the notification payload, once the metrics computation is finished, which is the most convenient as the client application waits for the notification to render the new state. The other alternative includes pooling the back-end data access services (Table 4.4) for collecting the latest results, which is less convenient since there is no guarantee that the metrics computation is complete and ready to return the results. In this particular experiment, however, the client application of the Noise Battle game remained intact. Changes were only required in the back-end side of the implementation as explained above. Therefore, the two alternatives mentioned above apply to general situations in which a client application would have been rebuilt.

5.2.3 Assessment

For the analytics platform, we established a set of features to adhere to, both from a implementation point of view and from the developer's perspective in Section 1.2 (RO3), and we globally discussed these features for the analytics platform in Section 5.1. In this section, the assessment strategy is based on the concrete case study of the Noise Battle game (Section 5.2.2), whereby we compare the pre-platform and post-platform implementations and discuss them driven by the aforementioned desirable features.

Unlike the pre-platform implementation, when it comes to asynchrony, the post-platform implementation does handle asynchronous requests. Even though the original application could have been implemented asynchronously, at additional cost of the developer, this comes as an inherent feature with the analytics platform (Chapter 4), free of additional burden to the developer. Asynchrony also has a positive impact on the scalability of the application (server-side), since memory-demanding resources such as network connections are not occupied during the computation of multiple game states. This is particularly important in scenarios in which a large number of users concurrently play in different game sessions, as is typically the case in multi-player location-aware games.

An example of inherent asynchrony in the analytics platform is the use of Metrics Execution Services (Section 4.1.3). When a request is received to execute a metrics through the front-end (REST) Metrics Execution Services, it forwards the request to an actor system to process it, and immediately returns a response to the client (original requester) with information on the progress of the task and how to retrieve the results once they are ready (e.g., logs containing the progress). The data produced can be later accessed through the data access APIs or received through notifications.

Regarding scalability, the post-platform implementation relies on a technology that is intrinsically scalable by design. It permits to scale out (horizontally) and scale up (vertically). The analytics platform exhibits the former by means of adding more nodes to augment the computational resources as additional strain is put on the system. For example, as the data storage of the platform (Section 4.1.4) is built upon the Cassandra distributed database (Lakshman and Malik, 2010), it offers features for large scale data handling over multiple nodes. For the latter case, scaling up, it refers to adding resources to a single node. Regarding processing, for example, the analytics platform achieves high levels of scalability through Apache Spark[7] at the expense of using the Spark programming model. This means that a developer must use the Spark API for data processing to benefit from the parallelism and scalability Spark offers. The JavaScript processing APIs in Section 4.1.5 uses the Spark API to access and process collected data.

In terms of abstraction, the analytics platform is based on a conceptual architecture, with a well-founded conceptual model of spatio-temporal metrics (see Chapter 3). The fundamental abstractions of this model are *variables, dimensions and the metric context*, used throughout the platform, and powerful enough to represent a wide range of virtual and real-world phenomena. The implementation of the model is combined with boilerplate code that involves the main methods that are often required in any client implementation, reducing thus the complexity of developing on top of the analytics platform and providing great flexibility. For example, the use of variables and dimensions allows even to code in different languages (i.e., database DDL commands, data transfer objects, etc.) and abstracting from the programming language of choice. For example, variables and dimensions eases the creation of database entities, the validation of the data in the back-end REST services, and the creation of the dimension providers the required in the client application (by the Metrics SDK). The data access services are also consistent with the variables and dimensions, and hence fully compliant with

[7]https://spark.apache.org/

the conceptual model of spatio-temporal metrics (Section 3.2). For example, the analytics platform offers a service to access a certain application variable or metrics result. The later contrasts dramatically with the pre-platform implementation, in which there is a service for accessing the game status.

If we look at the easy-of-use features, ease of setup may be measured by the number of artifacts needed to get a working implementation done, the pre-platform implementation required to deploy the binaries for the back-end services, which in the post-platform implementation are unnecessary. In addition, the post-platform implementation does not require a back-end database, as it is already supplied by the analytics platform and can be shared between different client applications.

If we pay attention to the ease of development, the analytics platform provides the developer with a simple yet powerful conceptual data model, flexible and customisable metrics logic, and a range of utilities (a software development kit, libraries, methods and functions) to facilitate development. Particularly, the platform offers support for seamlessly representing, handling and storing data related to monitored phenomena, and advanced spatial analysis support for processing the acquired data streams (i.e., through the underlying Turf.js library).

The analysis functions in the original implementation, for example to evaluate score and to compute whether a block was conquered, included code for updating the tables in the database. If, as a result of evaluating a new observation, a new block was conquered, a function added the data of the block conquered in the *conquered_blocks* table and updated accordingly the score in the *status* table. All of this is implemented seamlessly in the post-platform game as a result of the way the problem is modeled, using the abstractions and utilities of the analytics platform.

Regarding the complexity of developing the algorithm to calculate the new state in the post-platform implementation, it is somehow comparable, if not more complex. This extra complexity might be due to the use of “unfamiliar” APIs. For example, in the original implementation, a standard way for data access was implemented, namely using an Object Relational Mapping framework (Lorenz et al., 2017) (i.e., Hibernate), which eased the interaction of the algorithm with entities stored in the database. This is a “natural” way of accessing and interacting with data, since ORM data are treated as objects and collections. On the other hand, the new implementation requires knowledge of the Spark concepts and API, and the analytics platform’s API and model for data access and processing. Although Spark API eases the distributed processing, its model inevitably imposes programming assumptions while developing. One of such assumptions is that it needs to be considered whether data is locally available or processed in a different JVM during implementation[8].

A key aspect to consider is the availability of development and debugging tools. In the original implementation, mostly based on Java, an ecosystem of mature, third-party tools was available to support debugging, code edition, code assistance, etc. In contrast, the post-platform implementation still lacks advanced development assistance such as code completion. In the new platform implementation, despite the fact that the language of choice is JavaScript and a heap of supporting tools for code development are available, we have to consider that JavaScript is used through the Java scripting API (i.e., on top Java JVM) and debugging is then supported through a small set of powerful tools, compared with the browser debugging tools and traditional JavaScript tools. On the positive side, as in other development environments, the analytics platform provides basic testing capabilities (i.e., custom sample data generation capabilities, location data generation based on GPX, a GPS exchange format for track files[9]). Moreover, we have also developed tools for helping with testing such as the submission of test data to the platform (See Appendix 6.2.3). Also, the analytics platform incorporates integrated tools to support development. An example is the metrics definition tool (Section 4.1.1), which provides basic syntax highlighting and de facto validation of variables and dimensions against the metrics-definition schema.

Finally, when it comes to learnability, the original back-end implementation involved several technologies. For example, the list of REST services was created using the Jersey framework[10] and the Java programming language, which

[8]See example at `http://spark.apache.org/docs/latest/rdd-programming-guide.html#understanding-closures-`

[9]`https://www.topografix.com/gpx.asp`

[10]`https://eclipse-ee4j.github.io/jersey/`

facilitated routing and data serialization of input and output parameters. For data access, the back-end service layer was implemented with Hibernate[11] and the Hibernate Spatial extension, together with PostGIS[12], which in turn requires advanced knowledge in spatial databases and spatial operators. The combination of varied technologies in the same tool, and the need to interact directly with them, placed a great barrier for the developer in the sense of having advanced skills in all of these technologies.

The post-platform implementation keeps the set of technologies to a minimum and, instead, provides a set of high-level concepts and abstractions. These high-level concepts refer to the conceptual model of spatio-temporal metrics seen in Section 3.2. Variables, dimensions, metrics context, and so on, are crosscutting concepts in the sense that they are used coherently throughout the analytics platform (e.g., data structures design, data access). A few specific conceptualisations related to the methods provided for the metrics implementation (Section 4.1.5) are necessary too. The rest of elements such as back-end services and notifications are provided by the analytics platform and, for the most part, they only require a minimum configuration.

This approach allows the developer to concentrate on the development of the metrics functionality rather than on the surrounding technology. On the downside, greater abstraction implies that a developer must have profound knowledge about these abstractions, the mapping of these abstractions and conceptual artifacts of the analytics platform into a particular gaming domain. In other words, the developer needs to get familiar with the analytics platform, which may represent a steep learning curve for developers at the beginning, though arguably milder compared to the myriad of technologies required to achieve similar functionality.

5.2.4 Summary

Table 5.1 summaries the assessment features in terms of asynchrony, scalability, abstraction, ease of setup, ease of development and learnability, with respect to the original pre-platform implementation and the post-platform implementation of

[11] http://hibernate.org/

[12] https://postgis.net/

the Noise Battle game experiment.

5.3 Beyond games: Mental health experiment

The second experiment was developed in cooperation with a team of psychologists of the LABPSITEC research group of the Universidad Jaime I[13]. It consisted of a Web and mobile application as an intervention for patients diagnosed with agoraphobia disorder. In the experiment, users (emulating patients) were provided with a mobile app that tracked their position in order to study their daily activities in three different periods: morning, afternoon, and night. Collected data, including user's location, was sent to the analytics platform on a regular basis (5 minutes). The following behavior was studied: time spent outside and at home, the travelled distance when going out and the number of times the user enters and exits home.

Before starting the treatment, the therapist configured the application by adding the user's home location (coordinates) and a threshold radius. The resulting circular area represented the user's home surroundings, where, for the treatment, the user was considered to be at home. The user was also asked to do additional configuration in the mobile application once he/she was at home. During this configuration (at users' home), the app collected Wi-Fi fingerprints (i.e., measures of the "visible" Wi-Fi endpoints, identified by their SSID (Group et al., 2007) and their signal strength) to improve the accuracy of detecting when a user was at home (i.e., reducing false positives). During the treatment, the data collected included GPS coordinates, an identifier of the experiment, and a value called "coincidence" (see below).

To study the desired behavior using the analytics platform, two variables were defined to store the necessary collected data: *userLocation*, and *userConfig*. The *userLocation* variable is used to store the user's current location and his location history, and the coincidence value. The *coincidence* dimension denoted the "visibility" or "presence" ratio between the visible Wi-Fis at any moment and the ones collected during initial setup by the user. In practice, this ratio was obtained by

[13] http://www.labpsitec.uji.es/

Table 5.1: Pre-platform *vs* post-platform implementation of the Noise Battle game.

Features	**Post-platform (analytics platform)**	**Pre-platform (original)**
Asynchrony	Asynchronous model for handling requests: Requests are immediately handled and processed by the backend microservices (backed by distributed actor model system). Clients do not receive immediate results for the requests but through notifications mechanisms. .	Synchronous requests. Once the request is submitted to the server, it synchronously handles it, which involves requesting data from the database, the client has to wait for response.
Scalability	Scalable underlying systems: Apache Spark, use of a Distributed Database (Cassandra) and distributed actor model system (with o) in the back-end for handling incoming data and distributing the workload in different nodes. Separated data ingestion and data processing pipelines.	Non-scalable underlying systems setup: Java-based back-end services, Postgres database, synchronous handling of incoming data. Equivalent support requires additional effort from developer.
Abstraction	High-level concepts are used for modeling the problems (e.g., variables, dimensions, notifications). Abstracts aspects of the such as storage, data exchange, and formatting (how is data sent to the system), as well as validation and formatting in the client through the use of an SDK. Provide high-level guidelines regarding sensor data collection (e.g., location and orientation) throughout the use of default providers.	Ad-hoc implementation of entities for the game. Ad-hoc tables for storage. Custom implementation for data transmission and collection. In summary, low-level, limited reusable abstractions are used.
Ease of setup	Requires setting up the SDK library, including elements such as application, session and user. Integration with Google Cloud Messaging is required.	Only Google Cloud Messaging is required to be configured. Other elements belonging to the application.
Ease of development	Computation model based on Spark API (through Eclairjs), and a platform custom metrics library. Support for several spatial-related functionalities: geofences and Turf.js for advanced spatial analysis support. More complexity, as Spark API is inherently more complex. Less support for debugging the metrics included by users though the tools provided (i.e., the metrics definition tool). Included testing support using sample data generation and location data mocking through GPX data consumption.	Computation model based on Java back-end services. Support for several spatial related functionalities is available through PostGis. Development might be less complex due to the more ”familiar” client-server model. Support for debugging is available through existing Java tools (i.e., IntelliJ, Netbeans, Eclipse, etc.).
Learnability	Declarative, and crosscutting data model (in the sense that it is used across the system), using a formal specification (metrics definition JSON schema). Script-based computation model using Apache Spark API. It is not necessary to compile or package to deploy new metrics. The use of external tools is possible as a REST API is provided for submitting both data model and processing specifications. Access to results through Data access services is provided.	Ad-hoc data model, including the tailored database entities and data transfer objects. Java-based back-end services computation model. Rebuild and redeployment is needed when changes occur in the computation or the data model. Ad-hoc REST API is implemented for submitting and accessing the data.

evaluating a distance function that defined the level of similarity between the Wi-Fi fingerprint gathered in real time and the reference Wi-Fi fingerprints taken during configuration. This mechanism helped identify when the user was at home because GPS signal shows high inaccuracy in indoor environments. A third dimension included in the *userLocation* variable is a unique identifier for labelling the experiment. The second variable *userConfig* was aimed to register configuration data of the experiment. It included several dimensions such as the experiment id, the user's home location and radius, the number of days of the experiment and of the baseline period (a calibration phase prior to the treatment itself).

Using these variables, a single (custom) metric function was defined to extract the desired information (i.e., time inside/outside home, distance walked, number of times exiting/entering home) about the users' daily behaviour from their collected data (i.e., GPS coordinates and Wi-Fi fingerprints). This metrics function is summarized (in natural language) in the following steps:

1. Divide the collected data by the morning-afternoon-night periods (from 00:00 to 8:00, from 8:00 to 16:00, and from 16:00 to 00:00).
2. For each period, process the sequence of user's locations to determine:
   (a) Whether the user was inside or outside home.
   (b) The desired output values, i.e., time inside/outside, distance walked and number of times the user enters and leave home are calculated.

```
sqlContext.udf().register("session_batch", function (timeCol, location)
    {
    var result = "";
    //timezone calculation
    var timeStr = MetricsUtils.getFieldValue(location, 'time');
    var timeZone = moment.parseZone(timeStr).utcOffset();
    var m_time = moment(Date.parse(timeCol)).utcOffset(timeZone);
    var h = m_time.hour();

    //classification
    if (h >= 0 && h < 8) result = "0-8";
    else if (h >= 8 && h < 16) result = "8-16";
    else result = "16-24";
    return result;
}, DataTypes.StringType);
```

Listing 5.7: Analytic function to divide data into three periods.

To divide the collected location data (step 1) in periods of 8 hours (i.e., 0h-8h, 8h-16h, and 16h-24h), a Spark SQL query was used[14], as shown in Listing 5.7. To determine if the user was at home (step 2 (a)), we analysed the location data and classified it "at home" or "not at home". Listing 5.8 shows the function implementation to transform users' location data into trajectory data. In this case, both geographical proximity and fingerprints-based coincidence parameters are used. A location belongs to the user's home when the collected point is within the configured home area (through the function *MetricUtils.isInsidePointRadius()*) or the coincidence had to be greater than a threshold of 0.8. Note that even if the function *MetricUtils.isInsidePointRadius()* indicates that the location is not within the defined area, which could be the case when users are inside their home with poor GPS signal, the coincidence can indicate the contrary, and the user would be considered at home.

```
sqlContext.udf().register('locDetails', function (locationCol, timeid,
    previd, time, userHomeLocation, radius, coincidence) {
     var latLon = (locationCol == null) ? null : MetricsUtils.
    structAsLatLon(locationCol);
     latLon.timeid = timeid;
     latLon.previd = previd;
     latLon.time = MetricsUtils.toTime(time).getTime();
     var latLonRadius = {"point": MetricsUtils.structAsLatLon(
    userHomeLocation), "radius": radius};
     var atHome = true;
     if (!coreutils.isUndefinedOrNull(latLonRadius)) {
          atHome = MetricsUtils.isInsidePointRadius(latLonRadius, latLon)
      || (coincidence > 0.8) ;
     }
```

[14] https://spark.apache.org/sql/

```
    latLon.atHome = atHome;
    return JSON.stringify(latLon);
}, DataTypes.StringType);
```

Listing 5.8: Determining the location of the user (at home/not at home).

After partitioning and classifying the location data, the algorithm calculates the desired output values (step 2 (b)). In Listing 5.9, each conditional statement calculates output variables taking into account if the user was at home (and its previous state) based on the sequence of locations. For example, when the user was at home, the number of locations (indoors), the total distance (indoors), and the time at home were increased accordingly.

```
if (isInside && (prevState == inside)) {
    //was inside and remains inside.
    lastSegmentTimeInside += lastDiff;
    timeInside += lastDiff;
    totalDistanceInside += lastDistance;
} else if (!isInside && (prevState == outside)) {
    //was outside and remains outside
    lastSegmentTimeOutside += lastDiff;
    timeOutside += lastDiff;
    totalDistanceOutside += lastDistance;
}
else if (!isInside && (prevState == inside)) {
    //exited
    maxInside = Math.max(maxInside, lastSegmentTimeInside);
    lastSegmentTimeInside = 0;
    totalDistanceInside += halfDistance;
    totalDistanceOutside += halfDistance;
    timeInside += halfTime;
    timeOutside += halfTime;
    countExiting++;
} else if (isInside && (prevState == outside)) {
    //entered
    maxOutside = Math.max(maxOutside, lastSegmentTimeOutside);
    lastSegmentTimeOutside = 0;
    totalDistanceInside += halfDistance;
    totalDistanceOutside += halfDistance;
    timeInside += halfTime;
    timeOutside += halfTime;
    countEntering++;
}
```

Listing 5.9: Determining the resulting properties when user at home/not at home.

The results of executing the previous metric (Listing 5.9) was encoded as a string containing data encoded as a JSON object. The Listing 5.10 depicts the results as a JSON array. Note that in this case, the metrics function calculates various output values, and the output data is a complex structure, represented by a string containing JSON object(s), rather than an object. This represents a trade-off, where performance was gained at the expense of simplicity. Indeed, all calculated output values require the same input data, which only needs to be iterated once by the single metrics function, yet on the other hand, the output data structure is generic, which prevents the use of ad-hoc data processing capabilities supported by the analytics platform (i.e., filtering). Alternatively, we could have used three metrics functions, one for each output to calculate, with corresponding simple output structures. Also in other cases, it may be needed to create a string of JSON object(s) as output structure, for example, when the exact structure of the output is unknown beforehand or when the output data structure depends directly on the metrics calculation.

```
[{
"session": "session1",
"maxTimeConsecutiveOutside": 0,
"timeInside": 0,
"repeatedLocations": 2,
"maxTimeConsecutiveInside": 0,
"experiment": "experiment2",
"experimentDate": 1,
"countLocationsInside": 0,
"max_time": "2018-01-09T19:22:26.093Z",
"countExiting": 0,
"totalDistance": 1867.9506215357012,
"minSpeed": 0.007252697434780406,
"timeOutside": 2513.3579999999997,
"totalDistanceInside": 0,
"countLocationsOutside": 11,
"countEntering": 0,
"batch": "16-24",
"countLocations": 11,
"maxSpeed": 5.332015396697649,
"totalDistanceOutside": 1867.9506215357012,
"min_time": "2018-01-09T18:29:09.709Z",
"application": "app-608724ac9ae3b56d",
"user": "user1",
"locations": [{
    "acc": 5,
    "timeid": "fbc1ccea-f56a-11e7-82ff-580203040506",
    "alt": 110.53965081833303,
    "atHome": false,
    "lon": -0.07102740882230436,
    "time": 1515522549709,
    "lat": 39.99290583105257
    }]
}, {...}]
```

Listing 5.10: Metrics result example as a JSON array containing elements properties

To monitor and assess the evolution of users, therapists were provided with a web-based tool to visualise the results derived from the execution of the previous metrics functions. Among the visualisations provided, two stand out. Figure 5.6 shows the number of collected locations per period (morning-afternoon-night) and per category for multiple users. Some bars are low because of the lack of collected data due to distinct impediments such as restrictions on the execution of background services [15] [16] in Android-based devices, and dying battery, out of cov-

[15] Background Execution Limits: https://developer.android.com/about/versions/oreo/background

[16] Android background location limits: https://developer.android.com/about/versions/oreo/android-8.0-changes

erage, etc. Figure 5.7 illustrates in detail the behavior of a user during the day, where colored bars represent when a user was at home (blue) or away (orange) during one day.

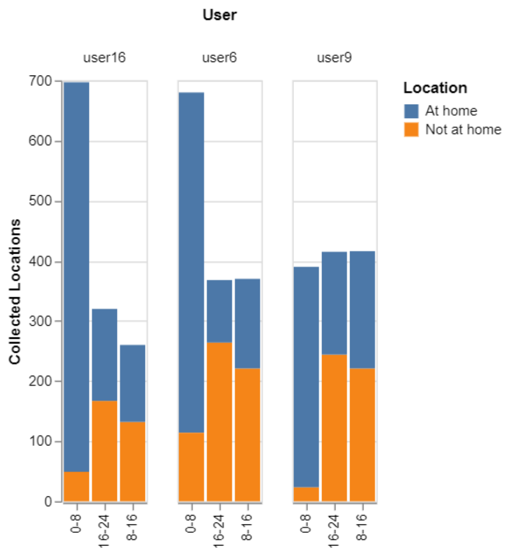


Figure 5.6: Locations collected per periods for three users.

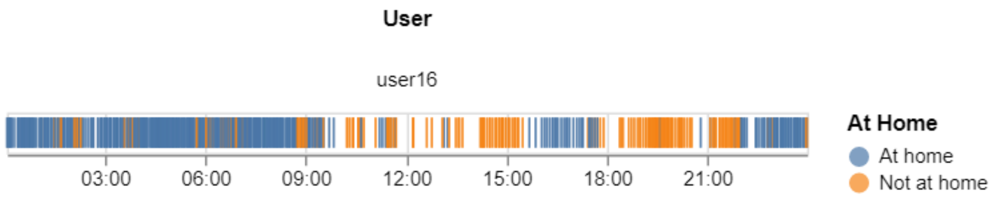


Figure 5.7: Categorised locations during the day for a sample user.

When it comes to the implementation, the metrics functions used a different set of Spark API methods and utils methods provided in the analytics platform than the ones used in the Noise Battle experiment (Section 5.2.2). For example, methods such as geofencing related functionalities to calculate trajectory properties (the distance and relation of a trajectory with the patient's home) and Spark SQL functionalities.

In summary, this second experiment, a real-world case study in mental health, demonstrated the versatility of the analytics platform, the underlying metrics concepts, and its applicability beyond the field of location-aware games, namely, in the context of an application for the treatment of patients with agoraphobia. A variety of geo-data was collected (i.e., GPS coordinates and WiFi-fingerprints), and geo-related methods were used to implement metric functions over these data to check certain spatial properties of users with respect to their home.

Chapter 6

CONCLUSIONS

In this chapter we provide the conclusion of this work by answering the research questions proposed in Chapter 1. We also summarize the contributions and expose the limitations of the framework implemented.

6.1 Summary

In games, monitoring and understanding player-game interaction, in-game actions and events are crucial to ensure a balanced game and enjoyable gameplay. In location-aware games, the physical location of the player(s) and game artefacts are key components of the game and, therefore, location- and context-related phenomena need to be taken into account. To convert these collected data into relevant, actionable information, metrics are used. Several research works and commercial platforms exist that allow game developers to address the different aspects involved in metrics definition and deployment: data specification and collection, metrics definition and calculation, analysis and visualization. Throughout this document, we have focused specifically on spatio-temporal metrics for location-aware games.

To unambiguously set the context, we first discussed definitions of location-aware games, metrics, and related terms in Chapter 2. We also conducted a literature review regarding the support of spatio-temporal features in existing game analytics platforms. We characterised the surveyed platforms in four functional areas: data collection and communication, data representation, data analysis and reaction, and data visualization and reporting. For each area, we discussed and analysed their defining features, paying special attention to the spatial characteristics as required by location-aware games. As a result, our analysis revealed a clear lack of support for spatio-temporal metrics in existing platforms.

To fully understand the needs of spatial metrics, we subsequently analysed in Chapter 2 the spatial metrics in the realm of game metrics, and proposed a classification for spatial metrics integrated in existing taxonomies of game metrics: real and virtual world interactions (related to user interface, regarding both the virtual and real-world interactions), spatial in-game (related to player's actions and behavior in the game, both in the virtual and real-world) and spatial awareness (regarding spatially-related decisions and actions the game initiates). With our proposal of an extended classification, we underlined the importance of geospatial features in location-aware games on one hand, and emphasise their relation with regular game metrics, and throughout all aspects of games in general. Indeed, the specific nature of location-aware games requires spatial and temporal dimensions that may justify extending traditional metrics (e.g., tracking the location of monetary transactions) or defining new ones (e.g., difficulty with respect to physical terrain features).

In this setting, we next elaborated three types of spatial metrics, namely point-, trajectory- and area-based metrics, which are particularly relevant for in-game behavior, provided examples and discussed difficulties. In addition to general difficulties in implementing location-aware games as reported by Jacob and Coelho (2011) (taking into account players' location, availability of location-based data, players' fitness, data protection and privacy concerns), we identified one key issue related to the lack of suitable (scalable) platforms to support geospatial features in the integration and computation of game-related metrics.

Finally, in Chapter 2, we discussed the implications of the conducted analysis of existing game analytics platforms in the design of a new generation of platforms to support geospatial features required by location-aware games. The main observations and implications were:

- Scalability, both with respect to the data and computational requirements, suggesting support for big data handling and processing.

- Need for built-in data representation and event type support, with associated compatibility with existing methods and tools.
- Platform extensibility, requiring an open and flexible platform architecture.
- Custom metrics support and supporting APIs and analysis tools.
- Balancing analytical strength with privacy concerns, as the latter are equally important and sensitive in the design and deployment of location-aware games.

In Chapter 3, we presented the architectural view of the analytics platform based on the insights from the previous Chapter 2. We discussed the conceptual and architectural decisions made to address the requirements for the integration and computation of spatio-temporal metrics. A central aspect was the definition of the conceptual model of spatio-temporal metrics. That conceptual model is composed of three parts. First the data model in which variables and dimensions allow designers to create personalised, versatile and sophisticated data structures to account for spatial and temporal data needs. The second part are the metrics functions, which operate over and compute game-relevant information from the data collected structured according to the data model definition. The last part of the conceptual model are the actions, which allow designers to trigger actions (e.g., notifications) when certain conditions are met. All in all, the proposed data model encapsulates all the necessary elements that a metric needs, allows designers to compose simple metrics into more complex metrics, and allows designers to share and reuse the definition of metrics since metrics are described in an open metrics specification.

In Chapter 4, we described the implementation of the analytics platform which takes into account the features identified and discussed in Chapter 2 and puts in practice the conceptual model of spatio-temporal metrics proposed in Chapter 3. The platform allows application developers to define data requirements, to collect required (client-generated) data, and to define and execute metrics that capture relevant non-spatial, spatial and temporal aspects of the monitored phenomena.

It thus performs analysis over the collected data, and carries out actions and notifications. Technically, the cloud-based, distributed platform is specifically designed to handle large amounts of (streaming) data, spatial and non-spatial data, and to scale well under increasing amount of data and metrics computations.

in Chapter 5, we presented an assessment of the analytics platform. First, we described an experiment where a location-aware game for collecting noise data in a city was developed with and without the analytics platform, and both implementations were qualitatively discussed from a developer's point of view, considering several implementation- and usability-related aspects. Second, we showed that the analytics platform can be regarded as domain agnostic, and is applicable beyond location-aware games, by implementing a mobile application as part of a mental health treatment, calculating various metrics relevant in this context.

6.2 Contributions

6.2.1 Scientific contributions

As we anticipated in Section 1.4, the main contributions of this work are twofold: a metrics model proposed for describing context-aware, spatial enable metrics; and the implementation of the analytics platform to collect and process spatio-temporal data based on the metrics model.

The main features of the proposed conceptual model of spatio-temporal metrics are:

- The model can be used in a wide range of location-aware games, allowing the representation of diverse data structures and metrics functions for most cases.
- The model can be shared and reused by different applications and is platform independent, allowing its consumption by different systems.
- The model includes the definition of actions which to react (notify) under (custom) circumstances related to collected or processed data.

In summary, the proposed model allows to abstract all facets related with spatio-temporal metrics from the particular application domain, by providing the required data structures (i.e., input data required), the processing specifications (i.e., how to extract the information required), and ultimately, how to evaluate and act upon the findings on the information extracted. This proposed model, in conjunction with the tools and libraries provided, can be used for rapidly collecting and processing data compliant with the model.

The analytics platform as a software artifact, using state-of-the-art technologies and exhibiting important features such scalability and spatio-temporal processing capabilities, is the second main contribution of this dissertation. Its main features are summarised as follows:

- The platform handles an extended metrics model for defining and computing spatio-temporal data, metrics functions, and actions.
- The platform supports data intensive applications that require computations of spatio-temporal metrics by offering data handling, storage, analysis, interpretation, and notification as a service.
- The platform provides tools (e.g., Metrics SDK, the metrics definition tool, among other developed tools) and mechanisms (REST APIs) for the integration of third-party components for data collection and processing.
- The platform taps into the growing amounts of contextual information streams, to provide meaningful contextual information at the application level.
- The platform applies a big data oriented architecture, uses recommended programming models and techniques (e.g., actor-based programming, publish/subscribe and reactive programming) and extensively uses big data technologies (Kafka, Akka, Spark, Cassandra). This architecture is inherently designed for separation and decoupling of concerns (e.g., data ingestion and storage, data processing, metrics computation), scalability and high volume data handling and processing.

6.2.2 Scientific publications

The scientific publications directly derived from the thesis are listed below, the most recent ones at the top.

- RODRÍGUEZ-PUPO, LUIS E.; GRANELL, CARLOS and CASTELEYN, SVEN (2019). An Analytics Platform for Integrating and Computing Spatio-Temporal Metrics. *ISPRS International Journal of Geo-Information*, **8(2)**. ISSN 2220-9964. 10.3390/ijgi8020054.
  https://www.mdpi.com/2220-9964/8/2/54

  We described the analysis and implementation of the analytics platform, and discussed the core conceptual features implemented in the platform. Besides, we showed the supporting applications and tools implemented as part of the platform. The article entirely links to Chapter 4 and part of Chapter 3.

- RODRÍGUEZ-PUPO, LUIS E.; CASTELEYN, SVEN and GRANELL, CARLOS (2017). On Metrics for Location-Aware Games. *ISPRS International Journal of Geo-Information*, **6(10)**. 10.3390/ijgi6100299.

  We analysed and discussed how existing game analytics platforms address spatio-temporal features of location-aware games. We also proposed a classification of spatial metrics, embedded in an existing categorization in the literature, and discussed three types of spatial metrics-point-, trajectory- and area-based metrics. The article entirely links to Chapter 2 and part of Chapter 3.

- MIRALLES, IGNACIO; GRANELL, CARLOS; RODRÍGUEZ-PUPO, LUIS E.; CASTELEYN, SVEN and HUERTA, JOAQUÍN (2017). Games, Health and the City: Developing Location-Aware Games for Leveraging the Most Suitable Places for Physical Activity. En: *Extended Abstracts Publication of the Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '17 Extended Abstracts, pp. 239–245. ACM, New York, NY, USA. ISBN 978-1-4503-5111-9. 10.1145/3130859.3131313.
  http://doi.acm.org/10.1145/3130859.3131313

We described the technical development of a location-aware game for promoting physical activity in relation to the urban environment, and designed an experiment with real users. This work informed the development of the analytics platform.

- KHOI, N. M.; RODRÍGUEZ-PUPO, L. E. and CASTELEYN, S. (2017). Citizense A generic user-oriented participatory sensing framework. En: *2017 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT)*, pp. 1–8. ISSN null. 10.1109/MoWNet.2017.8045954.

  We analysed the requirements of a multi-purpose participatory sensing frameworks to ease the generation of context-aware, multi-purpose participatory sensing campaigns. This work informed the (client-side) SDK Metrics tool described in Chapter 4.

- MENDOZA-SILVA, GERMÁN MARTÍN; RODRÍGUEZ-PUPO, LUIS ENRIQUE; TORRES-SOSPEDRA, JOAQUÍN and HUERTA-GUIJARRO, JOAQUÍN (2016). Solutions for signal mapping campaigns of Wi-Fi networks. En: *JIIDE 2016 Barcelona (27-30/09/2016)*, .

  We studied software solutions to enable users to participate in campaigns to collect WiFi signal samples, and explored signal intensities of the detected WiFi access points. This work informed the development of the mental health experiment described in Section 5.3.

- GARCÍA-MARTÍ, IRENE; TORRES-SOSPEDRA, JOAQUÍN and RODRÍGUEZ-PUPO, LUIS ENRIQUE (2014). A comparative study on VGI and professional noise data. En: Joaquín Huerta-Guijarro; Sven Schade and Carlos Granell-Canut. (Eds.), *Connecting a Digital Europe through Location and Place. Proceedings of the AGILE'2014 International Conference on Geographic Information Science.*, AGILE Digital Editions. ISBN 978-90-816960-4-3.
  http://repositori.uji.es/xmlui/handle/10234/98489

  We showed the results of an experiment in which user-gathered noise observations were comparable in quality to professional data. This work is directly related to the development and testing of the Noise Battle game reported in

Section 5.2.2.

- MARTÍ, IRENE GARCIA; RODRÍGUEZ, LUIS E; BENEDITO, MAURICIA; TRILLES, SERGI; BELTRÁN, ARTURO; DÍAZ, LAURA and HUERTA, JOAQUÍN (2012). Mobile application for noise pollution monitoring through gamification techniques. En: *International Conference on Entertainment Computing*, pp. 562–571. Springer, Berlin, Heidelberg.

  We presented an approach for gathering noise pollution data that combined mobile devices and gamification techniques. This work links directly to the design and development of the Noise Battle game in Section 5.2.2.

- GARCÍA-MARTÍ, IRENE; RODRÍGUEZ-PUPO, LUIS ENRIQUE; BENEDITO-BORDONAU, MAURICIA; TRILLES, SERGIO; BELTRÁN-FONOLLOSA, ARTURO; DÍAZ-SÁNCHEZ, LAURA and HUERTA-GUIJARRO, JOAQUÍN (2012). Aplicación móvil para la monitorización de la contaminación acústica en entornos urbanos a través de técnicas de Gamificación. En: *JIIDE 2012: III Jornadas Ibéricas de Infraestructuras de Datos Espaciales. Madrid, Octubre 2012*, .

  We presented an approach for gathering noise pollution data that combined mobile devices and gamification techniques, and the architecture of a location-aware game implementation. This work links directly to the design and development of the Noise Battle game in Section 5.2.2.

6.2.3 Software artefacts

Table 6.1: Software projects.

Project	**Description/URL**
actors	Implementation and base classes for defining messages types and actors used in persistence, notifications, etc.
cassandra-persistence-actor	Implementation of an Akka actor for persistence in Cassandra.
geo-test-utils	Common classes including a set of geofences-related utils. Includes reference and sample trajectories, algorithms for producing different geofencing situations etc., used for testing.
common-gpx	Utils and classes used for loading GPX formatted sample tracks.
k8s	Artifacts related to deployment of platform (Services databases, documentation).
labsitec-project	Source files related to Labpsitec experiment.
metrics-android-sdk	Metrics Android SDK library.
metrics-common	Common utils for working with metrics definitions
metrics-core	Library containing the core functionalities, core data model (i.e metrics definition), used in Android SDK and backend services.
metrics-data-ingestion	Project containing the data ingestion services.
metrics-engine	Project containing the backend services (metrics executions, metrics result data storage, etc.).
metrics-engine-frontend	Project for the web tool for editing metrics definitions, application configuration and visualization tools.
metrics-integrations	Repository containing commands used in different areas, Cassandra, Kafka, etc.
metrics-server-common	Common functionalities shared between server side projects (i.e metrics-services and metrics-engine-frontend projects).
metrics-services	Project containing Rest services interface of the system.
metrics-tools	Client tool created for interacting with the services.
noisebattle	Set of projects containing all NoiseBattle related applications (Android App, backend services, data, commands etc.).
schemaconverters	extension of the library json-schema-core[1] for converting a JSON schema definition to a DDL command in Cassandra.

Table 6.2: Software projects source code.

Project	Description/URL
actors	https://bitbucket.org/metricsappsteam/actors
cassandra-persistence-actor	https://bitbucket.org/metricsappsteam/cassandra-persistence-actor
geo-test-utils	https://bitbucket.org/metricsappsteam/geo-test-utils
common-gpx	https://bitbucket.org/metricsappsteam/common-gpx
k8s	https://bitbucket.org/metricsappsteam/metrics-k8s
labsitec-project	https://bitbucket.org/metricsappsteam/labpsitec-project
metrics-android-sdk	https://bitbucket.org/metricsappsteam/metrics-android-sdk
metrics-common	https://bitbucket.org/metricsappsteam/metrics-common
metrics-core	https://bitbucket.org/metricsappsteam/metrics-core
metrics-data-ingestion	https://bitbucket.org/metricsappsteam/metrics-data-ingestion
metrics-engine	https://bitbucket.org/metricsappsteam/metrics-engine
metrics-engine-frontend	https://bitbucket.org/metricsappsteam/metrics-engine-frontend
metrics-integrations	
metrics-server-common	https://bitbucket.org/metricsappsteam/metrics-server-common
metrics-services	https://bitbucket.org/metricsappsteam/server-metrics
metrics-tools	https://bitbucket.org/metricsappsteam/metrics-tools
noisebattle	https://bitbucket.org/metricsappsteam/noisebattle_commons https://bitbucket.org/metricsappsteam/noisebattle-commons-geo https://bitbucket.org/metricsappsteam/noise-battle-db https://bitbucket.org/metricsappsteam/noisebattle-server https://bitbucket.org/metricsappsteam/noisebattle
schemaconverters	https://bitbucket.org/metricsappsteam/shemaconverters

6.3 Research objectives answered

We posed the four research objectives (RO) of the thesis (Section 1.2) and organised this document in such a way that each of the central chapters addresses one of the RO (Section 1.5). Next, we briefly summarise to what extent we have addressed each of the RO.

RO1 was aimed to investigate how (and which) spatial features are taken into account in the design and development of location-aware games, more particularly, to steer game balance and gameplay. In Chapter 2 we addressed RO1 by conducting an exploratory literature review of existing game analytic platforms and tools to study in deep their most important features and their spatial support. We found that the analysed platforms and tools exhibited poor or no support for spatial features. Based on the review, we identified four functional areas, 1/ data collection, 2/ data representation, 3/ data analysis and reaction, and 4/ data visualization and reporting. For each functional area, we identified and grouped different features and measurable spatial aspects of metrics evaluation in the context of location-aware games. As a result, we established a classification of the spatial metrics to help designers in the process of designing and implementing location-aware games. Figure 2.2 summarised the proposed spatial metrics classification which is built on top of existing classification of game metrics centered on gameplay/player perspective.

RO2 was aimed to identify and classify the measurable spatial features that are relevant in the design of location-aware games. More specifically, the aim was to propose a model that covers the different dimensions of game metrics, particularly to support spatial features. Chapter 3 attempted to answer RO2. Through the literature examined in Chapter 2, we realised that different authors had presented distinct definitions of what a metric is. Some authors understood metrics as “raw data” collected for measuring a given phenomenon where others used a mathematical definition. We took the latter approach and defined metrics in the scope of our research as a “mathematical function” and extended it in the context of location-aware games for including both data structure aspects, and actions related to the metrics functions. The result was the conceptual model of spatio-temporal metrics whose main benefits are that it accounts for spatial features and it allows to be reused and shared in order to promote knowledge sharing.

RO3 was aimed to design and develop an operational platform implementing the establish theoretical principles (RO2) for game metrics, while addressing typical challenges in location-aware games, such as scalability and ease of use. Chapter 4 attempted to meet RO3. There, we designed and implemented an analytics platform that uses as a basis the proposed metrics model (RO2), addressing the functional areas extracted from the literature review and analysis (RO1), with specific strategies to face challenges in current location-aware games such as scalability and ease of use.

RO4 was aimed to validate the proposed theoretical model and its practical implementation, by applying it in different real-world scenarios. Chapter 5 puts the focus on RO4 by demonstrating the feasibility of the analytics platform through two different case studies. In the first case study, an experiment was set up, whereby we compared the pre-platform and post-platform implementation of Noise Battle, a location-aware game aimed to collect noise samples throughout the city (Section 5.2.2). We established a set of evaluation criteria from the developer's perspective, and then discussed both implementations according to each of the evaluation criteria (see Table 5.1). In the second case study, we applied the analytics platform in a mental health application for patients suffering agoraphobia. The two real-world applications show the generality of the analytics platform, and its feasibility to be used in different domains.

6.4 Limitations

Main limitations in this research are related to the implementation of the analytics platform, due mainly to the extent of the project and the evolution of software artefacts it relies on. Next, we overview the most important problems we faced during the development of the platform, grouping them into implementation and validation limitations.

6.4.1 Implementation limitations

The open source big data technologies we used to implement the analytics platform (e.g., Akka, Kafka, Spark, Cassandra, Docker) are at the forefront of technological developments in the field and, therefore, constantly evolve, which imposes many integration challenges. These technologies are inherently complex, due to the distributed nature of some of them and the interactions required for the platform to work. Most of the problems we found were related to stability issues with publicly available Kafka images in our on-premises environment, apparently related to networking issues. Regarding data storage, we faced and diagnosed issues while retrieving data as JSON in Cassandra due to an old JSON handling library (after reporting[2], a fix was provided by the Cassandra maintainers, and we needed to update the libraries in our docker images[3]). Regarding the usage of EclairJs, several compatibility issues arose during the implementation, as shared libraries between the different technologies (i.e., Spark client library and EclairJS library were not in the same version). We also found compatibility issues related to the usage of JavaScript libraries (i.e., Turf.js, and Moment.js) in our setup, mostly related to the Turf.js library, due to the restricted support of the JavaScript features provided by the Java Scripting API.

The analytics platform was developed with Spark deployed and running in local mode. The Spark local mode is intended for testing and demonstration purposes, and the execution of programmes in local mode seamlessly runs in a cluster environment. However, given the complex technology stack we used, the configuration and setting up of the cluster mode was complicated, and we did experience difficulties switching from local to cluster mode.

The data specification of variables and outputs through JSON schema has limitations. Although the platform supports the specification of complex structures, it is mainly limited to nested structures. This is due to the fact that the conver-

[2]The issue report can be found at: https://issues.apache.org/jira/browse/CASSANDRA-13949

[3]Details about the issue were to maintainers though the repositories https://github.com/lrodriguez2002cu/cassandra-issue-images and https://github.com/lrodriguez2002cu/cassandra-issue-tests

sion library (see project *schemaconverters* in Appendix 6.2.3) implemented for translating the JSON-based data schema to Cassandra DDL statements cannot support varying structures[4] as supported by the JSON schema specification.

Supporting schema evolution would mean to have several running instances of metrics models depending on slightly different variations of variables and dimensions, which will further complicate the implementation. For the time being, schema evolution is not yet supported. This means that once the variables and dimensions are set and the supporting structures in the database are created, introducing modifications in such structures are currently not possible. Schema evolution remains future work.

As mentioned in Section 4.1.5, the execution of metrics functions can be driven by data changes, in which case *evaluators* only need to execute such functions depending on (one or more) changes in data. An early implementation of this feature exists, but it is under development. Pending aspects are related, for example, to the high frequency of the data sent, since the activated metrics functions must be limited to avoid overloading the platform. Ideally, the time at which data changes are accelerated for a given variable must be configurable, since different variables may have different rates of data change and a delay applied to all variables may affect those that do not change as frequently.

Debugging is supported by the centralized logs of the platform, which can be accessed through the web-based visualisation tool (Section 4.1.6) during metrics trials. Besides, developers can run on-demand, user-defined metrics to test their functionality and check logs in case of failure. Tests, both unit tests[5] and integration tests[6] are key mechanisms for debugging too. As part of the software repositories provided, we include tests examples for validating the metrics execution that can be used by developers, but using it requires including the test in the code and rebuilding for executing the test. Despite the support for metrics debugging, it is still insufficient and metrics executions remain hard to debug. Therefore,

[4]For example, JSON schema supports structures of type ”any of” and ”one of” when referring to the fields an object can have. See JSON schema reference: `https://json-schema.org/understanding-json-schema/reference/combining.html`

[5]A definition can be found at `https://martinfowler.com/bliki/UnitTest.html`

[6] `https://martinfowler.com/bliki/IntegrationTest.html`

as a future enhancement, some of the debugging and testing functionalities developed could be put together in a separate library for helping developers implement, test and maintain authored metrics more easily.

6.4.2 Validation limitations

In the implementation of the platform we are using systems and libraries that support scalability at various levels. For example, Cassandra for storing data, and Apache Spark for computing. We use these systems to provide scalability to the analytics platform, as a feature that we identified as necessary in the context of location-aware applications. Nevertheless, the example applications we have implemented do not allow to asses the scalability to a full extent. A set of experiments targeting specific aspects such as storage, data ingestion, and metrics evaluation should have been implemented for validating the scalability of the system.

While developing the platform, several tests have been implemented across the different projects (see Appendix 6.2.3). For example, unit tests have been focused on JavaScript functionalities provided by the analytics platform (e.g., time handling and data access). Besides, we have used test frameworks specifically targeting these systems (e.g., Akka TestKit[7]) and implemented libraries (e.g., geo-test-utils and geofences-common project), which provide utils and sample routes for testing, using real trajectory data. We have used this data to test metrics functions with sample spatial data in the analytics platform, mainly for geofences and spatial capabilities. Despite the effort made in software testing, many more tests are still needed given the complexity of the platform. Specifically, given the different systems involved in the platform, more integration tests should be included, for testing the system end to end.

6.5 Future work

While several smaller technological improvements are planned on the platform, we foresee to continue the development of the analytics platform in two main lines.

[7]https://doc.akka.io/docs/akka/current/testing.html

One is to truly support stream processing (Garofalakis et al., 2016). While data streams are partially covered during the data ingestion process, stream processing is a current limitation of the proposed analytics platform during the metrics computation phase. The other line is related to performance analytics tools. Data regarding the platform's performance should be more rigorously collected. Performance data can be shown in visual real-time dashboards to give an overview of the system resource usage such as per-application usage and per-user usage so that administrators can detect critical performance bottlenecks and possibly update the platform's configuration to mitigate them (e.g., adding more hosts and nodes to the cluster). Besides, users (e.g., developers implementing the metrics) would benefit from performance data as they would have the necessary information to pinpoint and improve the metrics implementations.

On a final note, future improvements related to the limitations exposed above are possible. While metrics schema evolution is difficult to address properly in the current implementation of the platform, we believe it is feasible to find a solution that offers a fair trade-off between backward compatibility and the addition of new fields in the data models. Besides, the issues related to data schema complexity require the extension of the implemented schema conversion library (project *schemaconverters*, Section 6.2.3) to deal with complex data schema (e.g., JSON schema of the metrics functions output). We also believe that data access limitations can be overcome by implementing tools that allow bulk-copy data to cloud-enable, big data storage providers (e.g., Google Cloud storage[8] or Microsoft Azure Storage Accounts[9]).

[8]Google Cloud Storage: https://cloud.google.com/storage/

[9]Azure Storage Accounts: https://docs.microsoft.com/en-us/azure/storage/common/storage-account-overview

Appendices

A Abbreviations

The following abbreviations are used in this manuscript:

API	Application Programming Interface
CSV	Comma separated value
DDL	Data Definition Language
GeoJSON	Geo related schema and types for using in JSON
GPS	Global Positioning System
GPX	GPS Exchange format
GUID	Globally Unique Identifier
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
NoSQL	Term used to denote non relational databases
POST	Request method belonging to HTTP protocol
RDD	Resilient Distributed Dataset
REST	Representational State Transfer
SDK	Software Development Kit
UDT	Cassandra user defined type
WIFI	A trademarked term related to IEEE 802.11x family of standards

Bibliography

(2017). GameAnalytics web site. http://www.gameanalytics.com/ (accessed on 9 August 2017).

(2017). HoneyTracks web site. https://honeytracks.com/ (accessed on 9 August 2017).

(2017). Xolla Analytics web site. https://xsolla.com/modules/analytics/ (accessed on 9 August 2017).

ABRAMOVA, V.; BERNARDINO, J. and FURTADO, P. (2014). Testing Cloud Benchmark Scalability with Cassandra. En: *2014 IEEE World Congress on Services*, pp. 434–441. ISSN 2378-3818. doi: 10.1109/SERVICES.2014.81.

ALBUQUERQUE, M. T. C. F.; RAMALHO, G. L.; CORRUBLE, V.; SANTOS, A. L. M. and FREITAS, F. (2014). Helping Developers to Look Deeper inside Game Sessions. En: *Proceedings of 2014 Brazilian Symposium on Computer Games and Digital Entertainment*, pp. 31–40. ISSN 2159-6654. doi: 10.1109/SBGAMES.2014.28.

ALEGRE, UNAI; AUGUSTO, JUAN CARLOS and CLARK, TONY (2016). Engineering context-aware systems and applications: A survey. *Journal of Systems and Software*, **117**, pp. 55–83.

ALIREZAIE, MARJAN; RENOUX, JENNIFER; KÖCKEMANN, UWE; KRISTOFFERSSON, ANNICA; KARLSSON, LARS; BLOMQVIST, EVA; TSIFTES, NICOLAS; VOIGT, THIEMO and LOUTFI, AMY (2017). An ontology-based context-aware system for smart homes: E-care@ home. *Sensors*, **17(7)**, p. 1586.

Andone, Ionut; Blaszkiewicz, Konrad; Böhmer, Matthias and Markowetz, Alexander (2017). Impact of location-based games on phone usage and movement: a case study on Pokémon GO. En: *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services*, p. 102. ACM.

Andrienko, Gennady; Andrienko, Natalia; Bak, Peter; Keim, Daniel and Wrobel, Stefan (2013). *Visual analytics of movement*. Springer Science & Business Media.

Andrienko, Gennady; Andrienko, Natalia; Demsar, Urska; Dransch, Doris; Dykes, Jason; Fabrikant, Sara Irina; Jern, Mikael; Kraak, Menno-Jan; Schumann, Heidrun and Tominski, Christian (2010). Space, time and visual analytics. *International Journal of Geographical Information Science*, **24(10)**, pp. 1577–1600.

Andrienko, Gennady; Andrienko, Natalia; Jankowski, Piotr; Keim, Daniel; Kraak, M-J; MacEachren, Alan and Wrobel, Stefan (2007). Geovisual analytics for spatial decision support: Setting the research agenda. *International Journal of Geographical Information Science*, **21(8)**, pp. 839–857.

Arnaboldi, Valerio; Conti, Marco and Delmastro, Franca (2014). CAMEO: A novel context-aware middleware for opportunistic mobile social networks. *Pervasive and Mobile Computing*, **11**, pp. 148–167.

Avedon, Elliott M. and Sutton-Smith, Brian (1971). *The study of games*. John Wiley, New York, NY.

Bainomugisha, Engineer; Carreton, Andoni Lombide; Cutsem, Tom van; Mostinckx, Stijn and Meuter, Wolfgang de (2013). A Survey on Reactive Programming. *ACM Comput. Surv.*, **45(4)**, pp. 52:1–52:34. ISSN 0360-0300. doi: 10.1145/2501654.2501666.
http://doi.acm.org/10.1145/2501654.2501666

Batty, Michael (2016). Big Data and the City. *Built Environment*, **42**, pp. 321–337.

BELLAVISTA, PAOLO; CORRADI, ANTONIO; FANELLI, MARIO and FOSCHINI, LUCA (2012). A Survey of Context Data Distribution for Mobile Ubiquitous Systems. *ACM Comput. Surv.*, **44(4)**, pp. 24:1–24:45. ISSN 0360-0300. doi: 10.1145/2333112.2333119.
http://doi.acm.org/10.1145/2333112.2333119

BERNHAUPT, REGINA and MUELLER, FLORIAN "FLOYD" (2016). Game User Experience Evaluation. En: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pp. 147–148. ACM. ISBN 978-1-4503-4082-3. doi: 10.1145/2851581.2856683.
http://doi.acm.org/10.1145/2851581.2856683

BHATTI, SAAD SALEEM; REIS, JOS PEDRO and SILVA, ELISABETE A. (2017). Spatial Metrics: The Static and Dynamic Perspectives. En: *Reference Module in Earth Systems and Environmental Sciences*, Elsevier. ISBN 978-0-12-409548-9. doi: https://doi.org/10.1016/B978-0-12-409548-9.09604-4.
http://www.sciencedirect.com/science/article/pii/B9780124095489096044

BOLCHINI, CRISTIANA; CURINO, CARLO A.; QUINTARELLI, ELISA; SCHREIBER, FABIO A. and TANCA, LETIZIA (2007). A Data-oriented Survey of Context Models. *SIGMOD Records*, **36(4)**, pp. 19–26. ISSN 0163-5808. doi: 10.1145/1361348.1361353.
http://doi.acm.org/10.1145/1361348.1361353

CARPENTER, JEFF and HEWITT, EBEN (2016). *Cassandra: The Definitive Guide: Distributed Data at Web Scale*. "O'Reilly Media, Inc.".

CHANG, JINGKUN; YAO, WENBIN and LI, XIAOYONG (2017). A Context-Aware S-Health Service System for Drivers. *Sensors*, **17(3)**, p. 609. ISSN 1424-8220. doi: 10.3390/s17030609.
http://dx.doi.org/10.3390/s17030609

CHEN, CL PHILIP and ZHANG, CHUN-YANG (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*, **275**, pp. 314–347.

CHEN, GUANLING; KOTZ, DAVID et al. (2000). A survey of context-aware mobile computing research. *Technical report*, Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College.

COSTKYAN, G (2002). I have no words & I must design: toward a critical vocabulary for games. En: *Proceedings of the computer games and digital cultures conference, Finland*, .

COULTON, PAUL; BAMFORD, WILL; CHEVERST, KEITH and RASHID, OMER (2008). 3D Space-time visualization of player behaviour in pervasive location-based games. *International Journal of Computer Games Technology*, **2008**, p. 2.

DE SMITH, MICHAEL J; GOODCHILD, MICHAEL F and LONGLEY, PAUL A (2015). *Geospatial Analysis. A Comprehensive Guide to Principles, Techniques and Software Tools*. The Winchelsea Press, Winchelsea, UK, 5thedition.

DE SMITH, MICHAEL JOHN; GOODCHILD, MICHAEL F and LONGLEY, PAUL (2018). *Geospatial analysis: a comprehensive guide to principles, techniques and software tools*. The Winchelsea Press, 6thedition.

DEAN, JEFFREY and GHEMAWAT, SANJAY (2008). MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, **51(1)**, pp. 107–113. ISSN 0001-0782. doi: 10.1145/1327452.1327492.
http://doi.acm.org/10.1145/1327452.1327492

DELTADNA (2017). deltaDNA web site. https://deltadna.com/ (accessed on 9 August 2017).

DEY, ANIND K (2001). Understanding and using context. *Personal and ubiquitous computing*, **5(1)**, pp. 4–7.

DILL, JOHN; EARNSHAW, RAE; KASIK, DAVID; VINCE, JOHN; SALEN, PAK CHUNG WONGKATIE and ZIMMERMAN, ERIC (2012). *Expanding the Frontiers of Visual Analytics and Visualization*. Springer-Verlag London, 1edition.

DRACHEN, A. and SCHUBERT, M. (2013a). Spatial game analytics and visualization. En: *Proceedings of the IEEE Conference on Computational Intelligence in Games*, pp. 1–8. ISSN 2325-4270. doi: 10.1109/CIG.2013.6633629.

DRACHEN, ANDERS and CANOSSA, ALESSANDRO (2009). Towards gameplay analysis via gameplay metrics. En: *Proceedings of the 13th international MindTrek conference: Everyday life in the ubiquitous era*, pp. 202–209. ACM.

—— (2011). Evaluating motion: spatial user behaviour in virtual environments. *International Journal of Arts and Technology*, **4(3)**, pp. 294–314. doi: 10.1504/IJART.2011.041483.

DRACHEN, ANDERS; EL-NASR, MAGY SEIF and CANOSSA, ALESSANDRO (2013a). *Game Analytics - The Basics*. pp. 13–40. Springer, London.

DRACHEN, ANDERS and SCHUBERT, MATTHIAS (2013b). *Spatial Game Analytics*. pp. 365–402. Springer, London.

DRACHEN, ANDERS; THURAU, CHRISTIAN; SIFA, RAFET and BAUCKHAGE, CHRISTIAN (2013b). A comparison of methods for player clustering via behavioral telemetry. En: *Proceedings of the Foundations of Digital Games*, .

DUBOC, LETICIA; ROSENBLUM, DAVID and WICKS, TONY (2007). A Framework for Characterization and Analysis of Software System Scalability. En: *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC-FSE 07, p. 375384. Association for Computing Machinery, New York, NY, USA. ISBN 9781595938114. doi: 10.1145/1287624.1287679. https://doi.org/10.1145/1287624.1287679

DUGGAN, EDDIE (2017). *Squaring the (Magic) Circle: A Brief Definition and History of Pervasive Games*. pp. 111–135. Springer Singapore, Singapore. ISBN 978-981-10-1962-3. doi: 10.1007/978-981-10-1962-3_6. http://dx.doi.org/10.1007/978-981-10-1962-3_6

EL-NASR, MAGY SEIF; DRACHEN, ANDERS and CANOSSA, ALESSANDRO (2013a). *Game analytics*. Springer.

—— (2013b). *Game Analytics: Maximizing the Value of Player Data*. Springer, London.

—— (2013c). *Game analytics: Maximizing the value of player data*. Springer Science & Business Media.

ESRI (2017). Map Attack An Urban Geofencing Game. http://web.mapattack.org/ (accessed on 13 September 2017).

FECHNER, THORE; SCHLARMANN, DOMINIK and KRAY, CHRISTIAN (2016). Facilitating citizen engagement in situ: assessing the impact of pro-active geofenced notifications. En: *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pp. 353–364. ACM.

FORKAN, ABDUR; KHALIL, IBRAHIM and TARI, ZAHIR (2014). CoCaMAAL: A cloud-oriented context-aware middleware in ambient assisted living. *Future Generation Computer Systems*, **35**, pp. 114–127.

FREEMAN, JEREMY; VLADIMIROV, NIKITA; KAWASHIMA, TAKASHI; MU, YU; SOFRONIEW, NICHOLAS J.; BENNETT, DAVIS V.; ROSEN, JOSHUA; YANG, CHAO-TSUNG; LOOGER, LOREN L. and AHRENS, MISHA B. (2014). Mapping brain activity at scale with cluster computing. *Nature Methods*, **11(9)**, pp. 941–950. ISSN 1548-7105. doi: 10.1038/nmeth.3041.
https://doi.org/10.1038/nmeth.3041

FRY, B (2008). *Visualising Data*. O'Reilly Media, Inc.

FÜRNKRANZ, JOHANNES (2011). Machine learning and game playing. En: *Encyclopedia of machine learning*, pp. 633–637. Springer.

GALIĆ, ZDRAVKO (2016). *Spatio-Temporal Data Streams*. Springer.

GARCÍA, FÉLIX; PEDREIRA, OSCAR; PIATTINI, MARIO; CERDEIRA-PENA, ANA and PENABAD, MIGUEL (2017). A Framework for Gamification in Software Engineering. *Journal of Systems and Software*, **132**, pp. 21–40.

GARCÍA, ÓSCAR; ALONSO, RICARDO; PRIETO, JAVIER and CORCHADO, JUAN (2017). Energy Efficiency in Public Buildings through Context-Aware Social

Computing. *Sensors*, **17(4)**, p. 826. ISSN 1424-8220. doi: 10.3390/s17040826. http://dx.doi.org/10.3390/s17040826

GARCÍA-MARTÍ, IRENE; RODRÍGUEZ-PUPO, LUIS ENRIQUE; BENEDITO-BORDONAU, MAURICIA; TRILLES, SERGIO; BELTRÁN-FONOLLOSA, ARTURO; DÍAZ-SÁNCHEZ, LAURA and HUERTA-GUIJARRO, JOAQUÍN (2012). Aplicación móvil para la monitorización de la contaminación acústica en entornos urbanos a través de técnicas de Gamificación. En: *JIIDE 2012: III Jornadas Ibéricas de Infraestructuras de Datos Espaciales. Madrid, Octubre 2012*, .

GARCÍA-MARTÍ, IRENE; TORRES-SOSPEDRA, JOAQUÍN and RODRÍGUEZ-PUPO, LUIS ENRIQUE (2014). A comparative study on VGI and professional noise data. En: Joaquín Huerta-Guijarro; Sven Schade and Carlos Granell-Canut. (Eds.), *Connecting a Digital Europe through Location and Place. Proceedings of the AGILE'2014 International Conference on Geographic Information Science.*, AGILE Digital Editions. ISBN 978-90-816960-4-3. http://repositori.uji.es/xmlui/handle/10234/98489

GAROFALAKIS, MINOS; GEHRKE, JOHANNES and RASTOGI, RAJEEV (2016). *Data Stream Management*. Springer-Verlag Berlin, 1edition.

GONZÁLEZ SÁNCHEZ, JOSE LUIS; PADILLA ZEA, NATALIA and GUTIÉRREZ, FRANCISCO L. (2009). From Usability to Playability: Introduction to Player-Centred Video Game Development Process. En: *Proceedings of the 1st International Conference on Human Centered Design*, pp. 65–74. Springer-Verlag, Berlin, Heidelberg. ISBN 978-3-642-02805-2. doi: 10.1007/978-3-642-02806-9_9. http://dx.doi.org/10.1007/978-3-642-02806-9_9

GRANELL, CARLOS (2014). Robust Workflow Systems+ Flexible Geoprocessing Services= Geo-enabled Model Web? En: *Geographical Information Systems: Trends and Technologies*, pp. 172–204. CRC Press.

GRANELL, CARLOS; DÍAZ, LAURA and GOULD, MICHAEL (2010). Service-oriented applications for environmental models: Reusable geospatial services. *Environmental Modelling & Software*, **25(2)**, pp. 182–198.

GREENBERG, SAUL (2001). Context as a dynamic construct. *Human-Computer Interaction*, **16(2)**, pp. 257–268.

GROSSMAN, TOVI; FITZMAURICE, GEORGE and ATTAR, RAMTIN (2009). A Survey of Software Learnability: Metrics, Methodologies and Guidelines. En: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI 09, p. 649658. Association for Computing Machinery, New York, NY, USA. ISBN 9781605582467. doi: 10.1145/1518701.1518803.
https://doi.org/10.1145/1518701.1518803

GROUP, IEEE 802.11 WORKING et al. (2007). IEEE 802.11-2007: wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE 802.11 LAN Standards 2007*.

GUO, BIN; ZHANG, DAQING and IMAI, MICHITA (2011). Toward a cooperative programming framework for context-aware applications. *Personal and ubiquitous computing*, **15(3)**, pp. 221–233.

GUO, DANHUAI; ZHU, YINGQIU and YIN, WENWU (2018). OSCAR: a framework to integrate spatial computing ability and data aggregation for emergency management of public health. *GeoInformatica*, **22(2)**, pp. 383–410. ISSN 1573-7624. doi: 10.1007/s10707-017-0308-z.
https://doi.org/10.1007/s10707-017-0308-z

HEROLD, MARTIN; COUCLELIS, HELEN and CLARKE, KEITH C (2005a). The role of spatial metrics in the analysis and modeling of urban land use change. *Computers, Environment and Urban Systems*, **29(4)**, pp. 369 – 399.

HEROLD, MARTIN; COUCLELIS, HELEN and CLARKE, KEITH C. (2005b). The role of spatial metrics in the analysis and modeling of urban land use change. *Computers, Environment and Urban Systems*, **29(4)**, pp. 369–399. doi: 10.1016/j.compenvurbsys.2003.12.001.

HOCHLEITNER, CHRISTIN; HOCHLEITNER, WOLFGANG; GRAF, CORNELIA and TSCHELIGI, MANFRED (2015). *A Heuristic Framework for Evaluating User Experience in Games*. pp. 187–206. Springer International Publishing, Cham.

ISBN 978-3-319-15985-0. doi: 10.1007/978-3-319-15985-0_9.
http://dx.doi.org/10.1007/978-3-319-15985-0_9

HOFER, BARBARA; GRANELL, CARLOS and BERNARD, LARS (2018). Innovation in geoprocessing for a Digital Earth. *International Journal of Digital Earth*, **11(1)**, p. 36. doi: 10.1080/17538947.2017.1379154.
http://dx.doi.org/10.1080/17538947.2017.1379154

HOOBLER, NATE; HUMPHREYS, GREG and AGRAWALA, MANEESH (2004). Visualizing competitive behaviors in multi-user virtual environments. En: *Proceedings of the IEEE conference on Visualization*, pp. 163–170. IEEE Press.

HOSEINI-TABATABAEI, SEYED AMIR; GLUHAK, ALEXANDER and TAFAZOLLI, RAHIM (2013). A Survey on Smartphone-based Systems for Opportunistic User Context Recognition. *ACM Comput. Surv.*, **45(3)**, pp. 27:1–27:51. ISSN 0360-0300. doi: 10.1145/2480741.2480744.
http://doi.acm.org/10.1145/2480741.2480744

INOUBLI, WISSEM; ARIDHI, SABEUR; MEZNI, HAITHEM; MADDOURI, MONDHER and NGUIFO, ENGELBERT MEPHU (2018). An experimental survey on big data frameworks. *Future Generation Computer Systems*, **86**, pp. 546 – 564. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2018.04.032.
http://www.sciencedirect.com/science/article/pii/S0167739X17327450

JACOB, JOÃO TIAGO PINHEIRO NETO and COELHO, ANTÓNIO FERNANDO (2011). Issues in the development of location-based games. *International Journal of Computer Games Technology*, **2011**.

JAFFE, ALEXANDER; MILLER, ALEX; ANDERSEN, ERIK; LIU, YUN-EN; KARLIN, ANNA and POPOVIC, ZORAN (2012). Evaluating Competitive Game Balance with Restricted Play. En: *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 26–31.

JIA, Y.; DONG, X.; LIANG, Z. and SAXENA, P. (2015). I Know Where You've Been: Geo-Inference Attacks via the Browser Cache. *IEEE Internet Computing*, **19(1)**, pp. 44–53. doi: 10.1109/MIC.2014.103.

JIANG, BIN (2015). Geospatial analysis requires a different way of thinking: The problem of spatial heterogeneity. *GeoJournal*, **80(1)**, pp. 1–13.

JIMENEZ, EDUARDO; MITCHELL, KENNY and SERON, FRANCISCO (2011). Capture and analysis of racing gameplay metrics. *IEEE software*, **28(5)**, p. 46.

JOGALEKAR, P. and WOODSIDE, M. (2000). Evaluating the scalability of distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, **11(6)**, pp. 589–603. ISSN 2161-9883. doi: 10.1109/71.862209.

JOHNSON, DANIEL; HORTON, ELLA; MULCAHY, RORY and FOTH, MARCUS (2017). Gamification and serious games within the domain of domestic energy consumption: A systematic review. *Renewable and Sustainable Energy Reviews*, **73**, pp. 249 – 264. ISSN 1364-0321. doi: https://doi.org/10.1016/j.rser.2017.01.134.
http://www.sciencedirect.com/science/article/pii/S1364032117301478

KAENAMPORNPAN, MANASAWEE; ONEILL, EAMONN and AY, B BA (2004). An integrated context model: Bringing activity to context. En: *Proc. Workshop on Advanced Context Modelling, Reasoning and Management*, .

KANER, C. and BOND, W. (2004). Software engineering metrics: What do they measure and how do we know. En: *Proceedings of the 10th International Software Metrics Symposium*, pp. 1–12.

KARIMI, HASSAN A (2013). *Advanced location-based technologies and services*. CRC Press.

—— (2017). *Big Data: Techniques and Technologies in Geoinformatics*. CRC Press.

KHOI, N. M.; RODRÍGUEZ-PUPO, L. E. and CASTELEYN, S. (2017). Citizense A generic user-oriented participatory sensing framework. En: *2017 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT)*, pp. 1–8. ISSN null. doi: 10.1109/MoWNet.2017.8045954.

KIEFER, PETER and MATYAS, SEBASTIAN (2005). The Geogames Tool: Balancing spatio-temporal design parameters in location-based games. En: *Proceedings of Conference on Computer Games, Angoulême, France*, .

KIILI, KRISTIAN (2005). Digital game-based learning: Towards an experiential gaming model. *The Internet and Higher Education*, **8(1)**, pp. 13–24.

KIM, JUN H; GUNN, DANIEL V; SCHUH, ERIC; PHILLIPS, BRUCE; PAGULAYAN, RANDY J and WIXON, DENNIS (2008). Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems. En: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pp. 443–452. ACM.

KREMER, D.; SCHLIEDER, C.; FEULNER, B. and OHL, U. (2013). Spatial Choices in an Educational Geogame. En: *Proceedings of the 5th International Conference on Games and Virtual Worlds for Serious Applications*, pp. 1–4. doi: 10.1109/VS-GAMES.2013.6624243.

KULKARNI, DEVDATTA and TRIPATHI, ANAND (2010). A framework for programming robust context-aware applications. *IEEE Transactions on Software Engineering*, **36(2)**, pp. 184–197.

LAKSHMAN, AVINASH and MALIK, PRASHANT (2010). Cassandra: A Decentralized Structured Storage System. *SIGOPS Oper. Syst. Rev.*, **44(2)**, pp. 35–40. ISSN 0163-5980. doi: 10.1145/1773912.1773922.
http://doi.acm.org/10.1145/1773912.1773922

LI, SONGNIAN; DRAGICEVIC, SUZANA; CASTRO, FRANCESC ANTÓN; SESTER, MONIKA; WINTER, STEPHAN; COLTEKIN, ARZU; PETTIT, CHRISTOPHER; JIANG, BIN; HAWORTH, JAMES; STEIN, ALFRED et al. (2016). Geospatial big data handling theory and methods: A review and research challenges. *ISPRS journal of Photogrammetry and Remote Sensing*, **115**, pp. 119–133.

LORENZ, MARTIN; RUDOLPH, JAN-PEER; HESSE, GÜNTER; UFLACKER, MATTHIAS and PLATTNER, HASSO (2017). Object-Relational Mapping Revisited

- A Quantitative Study on the Impact of Database Technology on O/R Mapping Strategies. En: *HICSS*, .

MARTÍ, IRENE GARCIA; RODRÍGUEZ, LUIS E; BENEDITO, MAURICIA; TRILLES, SERGI; BELTRÁN, ARTURO; DÍAZ, LAURA and HUERTA, JOAQUÍN (2012). Mobile application for noise pollution monitoring through gamification techniques. En: *International Conference on Entertainment Computing*, pp. 562–571. Springer, Berlin, Heidelberg.

MARZ, NATHAN and WARREN, JAMES (2015). *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications Co..

MCCALLUM, SIMON and MACKIE, JAYSON (2013). *WebTics: A Web Based Telemetry and Metrics System for Small and Medium Games*. pp. 169–193. Springer London, London. ISBN 978-1-4471-4769-5. doi: 10.1007/978-1-4471-4769-5_10.
http://dx.doi.org/10.1007/978-1-4471-4769-5_10

MEDLER, BEN; JOHN, MICHAEL and LANE, JEFF (2011). Data cracker: developing a visual game analytic tool for analyzing online gameplay. En: *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 2365–2374. ACM.

MEEHAN, KEVIN; LUNNEY, TOM; CURRAN, KEVIN and MCCAUGHEY, AIDEN (2013). Context-aware intelligent recommendation system for tourism. En: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*, pp. 328–331. IEEE.

MENDOZA-SILVA, GERMÁN MARTÍN; RODRÍGUEZ-PUPO, LUIS ENRIQUE; TORRES-SOSPEDRA, JOAQUÍN and HUERTA-GUIJARRO, JOAQUÍN (2016). Solutions for signal mapping campaigns of Wi-Fi networks. En: *JIIDE 2016 Barcelona (27-30/09/2016)*, .

MILLER, HARVEY J and GOODCHILD, MICHAEL F (2015). Data-driven geography. *GeoJournal*, **80(4)**, pp. 449–461.

MILLER, HARVEY J and TOLLE, KRISTIN (2016). Big data for healthy cities: Using location-aware technologies, open data and 3D urban models to design healthier built environments. *Built Environment*, **42(3)**, pp. 441–456.

MIRALLES, IGNACIO; GRANELL, CARLOS; RODRÍGUEZ-PUPO, LUIS E.; CASTELEYN, SVEN and HUERTA, JOAQUÍN (2017). Games, Health and the City: Developing Location-Aware Games for Leveraging the Most Suitable Places for Physical Activity. En: *Extended Abstracts Publication of the Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '17 Extended Abstracts, pp. 239–245. ACM, New York, NY, USA. ISBN 978-1-4503-5111-9. doi: 10.1145/3130859.3131313.
http://doi.acm.org/10.1145/3130859.3131313

MONTELLO, DANIEL R (1993). Scale and multiple psychologies of space. En: *European Conference on Spatial Information Theory*, pp. 312–321. Springer.

NACKE, LENNART E. and DETERDING, SEBASTIAN (2017). The maturing of gamification research. *Computers in Human Behavior*, **71**, pp. 450–454. ISSN 0747-5632. doi: https://doi.org/10.1016/j.chb.2016.11.062.
http://www.sciencedirect.com/science/article/pii/S0747563216308111

NACKE, LENNART E; KLAUSER, MATTHIAS and PRESCOD, PAUL (2014). Social player analytics in a Facebook health game. En: *Proceedings of HCI Korea*, pp. 180–187. Hanbit Media, Inc..

NAGPAL, VARUN (2016). *Android Sensor Programming By Example*. Packt Publishing Ltd.

NARAIN, S.; VO-HUU, T. D.; BLOCK, K. and NOUBIR, G. (2017). The Perils of User Tracking Using Zero-Permission Mobile Apps. *IEEE Security Privacy*, **15(2)**, pp. 32–41.

NEUMAN, B. CLIFFORD (1994). Scale in Distributed Systems. En: *Readings in Distributed Computing Systems*, pp. 463–489. IEEE Computer Society Press.

NIJHOLT, ANTON (2017). *Playable Cities: The City as a Digital Playground*. Springer.

NOTHAFT, FRANK AUSTIN; MASSIE, MATT; DANFORD, TIMOTHY; ZHANG, ZHAO; LASERSON, URI; YEKSIGIAN, CARL; KOTTALAM, JEY; AHUJA, ARUN; HAMMERBACHER, JEFF; LINDERMAN, MICHAEL and ET AL. (2015). Rethinking Data-Intensive Science Using Scalable Analytics Systems. En: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD 15, p. 631646. Association for Computing Machinery, New York, NY, USA. ISBN 9781450327589. doi: 10.1145/2723372.2742787.
https://doi.org/10.1145/2723372.2742787

PEDREIRA, OSCAR; GARCA, FLIX; BRISABOA, NIEVES and PIATTINI, MARIO (2015). Gamification in software engineering A systematic mapping. *Information and Software Technology*, **57**, pp. 157–168. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/j.infsof.2014.08.007.
http://www.sciencedirect.com/science/article/pii/S0950584914001980

PEJOVIC, VELJKO and MUSOLESI, MIRCO (2015). Anticipatory Mobile Computing: A Survey of the State of the Art and Research Challenges. *ACM Comput. Surv.*, **47(3)**, pp. 47:1–47:29. ISSN 0360-0300. doi: 10.1145/2693843.
http://doi.acm.org/10.1145/2693843

PRENSKY, MARC (2002). The motivation of gameplay: The real twenty-first century learning revolution. *On the horizon*, **10(1)**, pp. 5–11.

RANA, RAJIB; HUME, MARGEE; REILLY, JOHN; JURDAK, RAJA and SOAR, JEFFREY (2016). Opportunistic and context-aware affect sensing on smartphones. *IEEE Pervasive Computing*, **15(2)**, pp. 60–69.

REIS, JOSÉ P.; SILVA, ELISABETE A. and PINHO, PAULO (2016). Spatial metrics to study urban patterns in growing and shrinking cities. *Urban Geography*, **37(2)**, pp. 246–271. doi: 10.1080/02723638.2015.1096118.
http://dx.doi.org/10.1080/02723638.2015.1096118

RITTERFELD, UTE; CODY, MICHAEL and VORDERER, PETER (2009). *Serious Games: Mechanisms and Effects*. Routledge, UK.

RODRÍGUEZ-PUPO, LUIS E.; CASTELEYN, SVEN and GRANELL, CARLOS (2017). On Metrics for Location-Aware Games. *ISPRS International Journal of Geo-Information*, **6(10)**. doi: 10.3390/ijgi6100299.

RODRÍGUEZ-PUPO, LUIS E.; GRANELL, CARLOS and CASTELEYN, SVEN (2019). An Analytics Platform for Integrating and Computing Spatio-Temporal Metrics. *ISPRS International Journal of Geo-Information*, **8(2)**. ISSN 2220-9964. doi: 10.3390/ijgi8020054.
https://www.mdpi.com/2220-9964/8/2/54

ROESTENBURG, RAYMOND; BAKKER, ROB and WILLIAMS, ROB (2015). *Akka in action*. Manning Publications Co..

ROLLINGS, ANDREW and ERNEST, ADAMS (2003). *Andrew Rollings and Ernest Adams on game design*. Indianapolis, Ind.: New Riders.

SAGL, GÜNTHER; RESCH, BERND and BLASCHKE, THOMAS (2015). Contextual sensing: Integrating contextual information with human and technical geo-sensor information for smart cities. *Sensors*, **15(7)**, pp. 17013–17035.

SALEN, KATIE and ZIMMERMAN, ERIC (2004). *Rules of Play. Game Design Fundamentals*. MIT Press, Cambridge, MA.

SCHLATTER, BARBARA ELWOOD and HURD, AMY R (2005). Geocaching: 21st-century hide-and-seek. *Journal of Physical Education, Recreation & Dance*, **76(7)**, pp. 28–32.

SCHLIEDER, CHRISTOPH; KIEFER, PETER and MATYAS, SEBASTIAN (2005). Geogames: A conceptual framework and tool for the design of location-based games from classic board games. En: *International Conference on Intelligent Technologies for Interactive Entertainment*, pp. 164–173. Springer.

—— (2006). Geogames: Designing location-based games from classic board games. *IEEE Intelligent Systems*, **21(5)**, pp. 40–46.

SCHWEIZER, IMMANUEL; BÄRTL, ROMAN; SCHULZ, AXEL; PROBST, FLORIAN and MÜHLÄUSER, MAX (2011). NoiseMap-real-time participatory noise maps. En:

Second international workshop on sensing applications on mobile phones, pp. 1–5. Citeseer.

SEABORN, KATIE and FELS, DEBORAH I. (2015). Gamification in theory and action: A survey. *International Journal of Human-Computer Studies*, **74**, pp. 14–31. ISSN 1071-5819. doi: http://dx.doi.org/10.1016/j.ijhcs.2014.09.006. http://www.sciencedirect.com/science/article/pii/S1071581914001256

SEDIG, KAMRAN; PARSONS, PAUL and HAWORTH, ROBERT (2017). PlayerGame Interaction and Cognitive Gameplay: A Taxonomic Framework for the Core Mechanic of Videogames. *Informatics*, **4(1)**. doi: 10.3390/informatics4010004.

SMITH, CRAIG (2017). 75 Amazing Pokemon Go Statistics. http://expandedramblings.com/index.php/pokemon-go-statistics/

SOLANAS, AGUSTI; PATSAKIS, CONSTANTINOS; CONTI, MAURO; VLACHOS, IOANNIS S; RAMOS, VICTORIA; FALCONE, FRANCISCO; POSTOLACHE, OCTAVIAN; PÉREZ-MARTÍNEZ, PABLO A; DI PIETRO, ROBERTO; PERREA, DESPINA N et al. (2014). Smart health: a context-aware health paradigm within smart cities. *IEEE Communications Magazine*, **52(8)**, pp. 74–81.

SOUTHEY, FINNEGAN; XIAO, GANG; HOLTE, ROBERT C.; TROMMELEN, MARK and BUCHANAN, JOHN W. (2005). Semi-Automated Gameplay Analysis by Machine Learning. En: *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference*, pp. 123–128. AAAI Press.

STANTON, MATT; HUMBERSTON, BEN; KASE, BRANDON; O'BRIEN, JAMES F; FATAHALIAN, KAYVON and TREUILLE, ADRIEN (2014). Self-refining games using player analytics. *ACM Transactions on Graphics*, **33(4)**, p. 73.

SUDHIRA, H.S.; RAMACHANDRA, T.V. and JAGADISH, K.S. (2004). Urban sprawl: metrics, dynamics and modelling using GIS. *International Journal of Applied Earth Observation and Geoinformation*, **5(1)**, pp. 29–39. doi: 10.1016/j.jag.2003.08.002.

TÖRNROS, TOBIAS; DORN, HELEN; REICHERT, MARKUS; EBNER-PRIEMER, ULRICH; SALIZE, HANS-JOACHIM; TOST, HEIKE; MEYER-LINDENBERG, ANDREAS and ZIPF, ALEXANDER (2016). A comparison of temporal and location-based sampling strategies for global positioning system-triggered electronic diaries. *Geospatial health*, **11(3)**.

TYCHSEN, ANDERS and CANOSSA, ALESSANDRO (2008). Defining personas in games using metrics. En: *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, pp. 73–80. ACM.

VARGHESE, BLESSON and BUYYA, RAJKUMAR (2018). Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems*, **79(Part 3)**, pp. 849 – 861. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2017.09.020.
http://www.sciencedirect.com/science/article/pii/S0167739X17302224

WALLNER, GÜNTER and KRIGLSTEIN, SIMONE (2012). A Spatiotemporal Visualization Approach for the Analysis of Gameplay Data. En: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1115–1124. ACM. ISBN 978-1-4503-1015-4.

WALLNER, GÜNTER and KRIGLSTEIN, SIMONE (2014). PLATO: A visual analytics system for gameplay data. *Computers & Graphics*, **38**, pp. 341–356.

WAN, JIAFU; ZHANG, DAQIANG; ZHAO, SHENGJIE; YANG, LAURENCE and LLORET, JAIME (2014). Context-aware vehicular cyber-physical systems with cloud support: architecture, challenges, and solutions. *IEEE Communications Magazine*, **52(8)**, pp. 106–113.

WEISSTEIN, ERIC W. (2017). Metric. MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/Metric.html (accessed on 19 July 2017).
http://mathworld.wolfram.com/Metric.html

WORBOYS, MICHAEL F and DUCKHAM, MATT (2019). *GIS: A Computing Perspective*. CRC Press, 3rdedition.

YUE, SONGSHAN; CHEN, MIN; WEN, YONGNING and LU, GUONIAN (2016). Service-oriented model-encapsulation strategy for sharing and integrating heterogeneous geo-analysis models in an open web environment. *ISPRS Journal of Photogrammetry and Remote Sensing*, **114**, pp. 258–273.

ZAHARIA, MATEI; XIN, REYNOLD S.; WENDELL, PATRICK; DAS, TATHAGATA; ARMBRUST, MICHAEL; DAVE, ANKUR; MENG, XIANGRUI; ROSEN, JOSH; VENKATARAMAN, SHIVARAM; FRANKLIN, MICHAEL J. and ET AL. (2016a). Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM*, **59(11)**, p. 5665. ISSN 0001-0782. doi: 10.1145/2934664.
https://doi.org/10.1145/2934664

ZAHARIA, MATEI; XIN, REYNOLD S.; WENDELL, PATRICK; DAS, TATHAGATA; ARMBRUST, MICHAEL; DAVE, ANKUR; MENG, XIANGRUI; ROSEN, JOSH; VENKATARAMAN, SHIVARAM; FRANKLIN, MICHAEL J.; GHODSI, ALI; GONZALEZ, JOSEPH; SHENKER, SCOTT and STOICA, ION (2016b). Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM*, **59(11)**, pp. 56–65. ISSN 0001-0782. doi: 10.1145/2934664.
http://doi.acm.org/10.1145/2934664

ZHAO, PEISHENG; FOERSTER, THEODOR and YUE, PENG (2012). The geoprocessing web. *Computers & Geosciences*, **47**, pp. 3–12.

ZHOU, LIANG; NAIXUE, XIONG; SHU, LEI; VASILAKOS, ATHANASIOS and YEO, SANG-SOO (2010). Context-aware middleware for multimedia services in heterogeneous networks. *IEEE Intelligent Systems*.

ZHOU, YUANCHUN; ZHANG, YANG; GE, YONG; XUE, ZHENGHUA; FU, YANJIE; GUO, DANHUAI; SHAO, JING; ZHU, TIANGANG; WANG, XUEZHI and LI, JIANHUI (2017). An efficient data processing framework for mining the massive trajectory of moving objects. *Computers, Environment and Urban Systems*, **61**, pp. 129 – 140. ISSN 0198-9715. doi: https://doi.org/10.1016/j.compenvurbsys.2015.03.004.
http://www.sciencedirect.com/science/article/pii/S0198971515000393