Bruno Miguel de Lemos Ribeiro Pinto Cardoso

Two Steps Towards Kairos-Awareness

Dissertação para obtenção do Grau de Doutor em Informática

Orientador: Professora Doutora Teresa Isabel Lopes Romão,
Professora Auxiliar,
Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa

Júri:

Presidente: Professor Doutor Nuno Correia

Arguentes: Professora Doutora Celine Latulipe
Professor Doutor Luis Carriço

Vogais: Professora Doutora Teresa Romão
Professor Doutor António Coelho
Professor Doutor Fernando Birra

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Dezembro, 2016

Two Steps Towards Kairos-Awareness

To my parents,

António Cardoso and Izilda Cardoso,

may you always take Kairos by the forelock.

Acknowledgments

I would like to convey my heartfelt thanks to Professor Teresa Romão for having been the supervisor of my undergraduate and Master studies, and for inviting me to do a PhD. For the trust in me and for the freedom to follow my research interests – it has been quite the journey! Tiring and draining, yes… yet challenging and fantastically worthwhile and I am grateful to you for that.

To NOVA-LINCS, the Department of Informatics of Universidade Nova de Lisboa and Fundação para a Ciência e Tecnologia for funding my research through grant SFRH/BD/73177/2010.

To the Professors of the Department of Informatics who have helped me, shown interest in my research (even if from research areas far from HCI) and kindly agreed to let their students (and sometimes family) participate in my experiments. Namely, Professor Nuno Correia, Professor João Magalhães, Professor Fernando Birra and Professor Carmen Morgado – thank you for your availability and inspiring insights.

To Professor Nuno Correia for keeping everything running smoothly at the P3/13, and for always being available to discuss details of my work – specially in those many times when all the warning I gave was a knock on your office door and an urgent "*Professor, may we discuss something now?*".

To Professor Osvaldo Santos for being a true tutor in all things psychology and for always considering me for his technological endeavours – thank you for your patience and willingness to share your knowledge.

To my PhD colleagues, for their curiosity in research so far from their own interests, and for their availability for participating in my experiments – even when snakes were (virtually) involved or when the experiment time approached the "*…is this going to take much longer?*" threshold, you endured!

To everyone from the P3/13, for making that laboratory such a great place to work, for your interest and time. A special word of acknowledgment to Rui Madeira for letting me conduct my experiments at the Instituto Politécnico de Setúbal – I can't thank you enough for your generosity. Another special thank

you to André Sabino, Filipa Peleja, Rui Nóbrega and Rui Madeira - for putting up with my bad moods – "*…not accepted!*"; for your interest and support; for our scientific, technological and movie discussions; for our road trips, dinners, *D. Teresa*, *A Redentora*, and squash.

To Rute Rodrigues, for your unwavering support and faith in me, cheerful reviewing, uplifting attitude, incredible capacity for understanding technology, tireless dedication and companionship – and who would say that Bilbao is such a great (and *Singular*) place for thesis-writing?

Finally, to my family, António Cardoso, Izilda Cardoso and Susana Cardoso. Thank you for your all-around support and unconditional love. Words are not enough to thank you. But thank you.

Abstract

This thesis describes a research inspired by a concept of the classical discipline of rhetoric: k*airos*, the right moment to deliver a message in order to maximize its effect. The research followed two threads that, ultimately, lead to the same ending: the maximization of the potential of technology to deliver the right interaction at the right time.

The first research thread is an operationalization of the concept of *kairos*. It entailed the development of EveWorks and EveXL, a framework for capturing daily life events in mobile devices and a domain-specific language to express them, respectively. The largely extended use of mobile devices and their proximity with their owners offers exceptional potential for capturing opportunity for interaction. Leveraging on this potential, the EveWorks-EveXL dyad was developed to allow mobile application programmers to specify the precise delivery circumstances of an interaction in order to maximize its potential, i.e., to specify its *kairos*.

Contrasting to most event processing engines found in the literature that implement data-based event models, the EveWorks-EveXL dyad proposes a model based on temporality, through the articulation of intervals of time. This is a more natural way of representing a concept as broad as "daily life events" since, across cultures, temporal concepts like duration and time intervals are fundamental to the way people make sense of their experience. The results of the present work demonstrate that the EveWorks-EveXL dyad makes for an adequate and interesting way to express contextual events, in a way that is "closer" to our everyday understanding of daily life.

Ultimately, in user centered applications, *kairos* can be influenced by the user's emotional state, thereby making emotion assessment relevant. Addressing this, as well as the growing interest in the topic of emotions by the scientific community, the second research thread of the present thesis led to the development of the CAAT, a widget designed to perform quick and reliable assessments of affective states – a paramount task in a variety of scientific fields, including HCI. While there are already a number of tools for this purpose, in psychology, emotion assessments are largely conducted through the use of pen-and-paper questionnaires applied after the affective experience has occurred. As emotional states vary significantly over time, this entails the loss of important details, warranting the need for immediate, in situ, measurements of affect. In line with this requirement, the CAAT enables quick emotion assessment in a reliable fashion, as attested by the results of the

validation studies conducted in order to assess its overall viability along relevant dimensions of usability and psychometrics. As such, aside from being a good fit for longitudinal studies and applications whenever the quick assessment of emotions is required, the CAAT has the potential to be integrated as one of EveWorks' sensors to enhance its ability to find that sometimes elusive opportunity for interaction, i.e., their *kairos*. In this way, it becomes apparent how the two threads of research of the current work may be intertwined into a consolidated contribution to the HCI field.

Keywords: Mobile Application Development; Context Awareness Framework; Domain-specific Language; Event Detection; Time Interval Algebra; Affective Reaction Assessment; Widget; Validation; Psychometry.

Resumo

Esta tese descreve uma investigação inspirada num conceito da disciplina da retórica: *kairos*, o momento certo para entregar uma mensagem por forma a maximizar o seu efeito. A investigação seguiu duas linhas de trabalho que, em última análise, conduzem ao mesmo destino: a maximização do potencial da tecnologia para apresentar ao utilizador a interacção certa no momento certo.

A primeira linha de investigação consistiu numa operacionalização do conceito de *kairos*, através do desenvolvimento de uma *framework* para a captura de eventos da vida quotidiana dos utilizadores de dispositivos móveis – EveWorks –, e de uma linguagem de domínio específico para a expressão destes – EveXL. A ubiquidade dos dispositivos móveis, como os *smartphones*, e a proximidade destes aos seus utilizadores conferem a estes um potencial excepcional para a detecção e captura de oportunidades para interacção. Aproveitando este potencial, o binómio EveWorks-EveXL foi desenvolvido para permitir a aplicações móveis definir as circunstancias precisas em que determinada interacção será apresentada ao utilizador, ou seja, especificar o *kairos* dessas interacções.

Contrastando com o que sucede com a maioria dos motores de processamento de eventos descritos na literatura, que implementam modelos de eventos orientados a dados, o binómio EveWorks-EveXL propõe um modelo baseado no tempo. Esta é uma forma mais natural de representar conceitos tão abrangentes como "eventos da vida quotidiana" visto que, mesmo entre diferentes culturas, conceitos temporais como "duração" e "intervalos de tempo" são essenciais à forma como se tira sentido da experiência. Os resultados do trabalho apresentado demonstram que o binómio EveWorks-EveXL oferece uma forma conveniente e adequada de expressão de eventos contextuais, mais próxima do modo como entendemos a nossa vida quotidiana.

Por outro lado, no contexto de aplicações centradas no utilizador, o momento oportuno de uma interacção pode ser influenciado pelo estado emocional do seu receptor. Dessa forma, e respondendo também ao interesse crescente da comunidade científica no tópico das emoções, a segunda linha de investigação seguida levou ao desenvolvimento da CAAT, um componente de interfaces gráficas de utilizador desenhado para a avaliação rápida de estados emocionais – uma tarefa fulcral em vários campos científicos, incluindo *HCI*. Apesar de existirem já algumas ferramentas para este propósito no campo da psicologia, estas são geralmente questionários em papel construídos para serem aplicados depois da experiência emocional ter ocorrido. Visto que os estados

emocionais e a sua memória variam significativamente ao longo do tempo, isto traduz-se na perda potencial de detalhes importantes pelo que se tornam importantes métodos de avaliação rápidos e passíveis de serem executados no local. Respondendo a isto, a CAAT permite avaliações rápidas e, não obstante, fidedignas, conforme revelado pelos estudos de validação feitos. Desta forma, além de ser uma solução interessante a aplicar em estudos longitudinais em que a avaliação de reacções emocionais é relevante, a CAAT é passível de ser integrada como um dos sensores da EveWorks, assim incrementando com informação emocional o potencial desta última para detecção do melhor momento para apresentar uma interacção. Torna-se assim aparente como as duas linhas de investigação seguidas podem ser interligadas num contributo consolidado para o campo da investigação das interfaces homem-máquina.

Palavras-chave: Desenvolvimento de Aplicações Móveis; Framework de Computação de Contexto; Linguagem de Domínio Específico; Detecção de Eventos; Algebra de Intervalos de Tempo; Avaliação de Reacções Afectivas; Validação; Psicometria.

CONTENTS

List of Figures

List of Tables

Thesis Publications

The research described in this thesis is available in the following publications:

- *Cardoso, B., Santos, O., and Romão, T. 2015. On Sounder Ground: CAAT, a Viable Widget for Affective Reaction Assessment. In Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15), ACM, New York, NY, USA, 501–510. DOI: 10.1145/2807442.2807465*
- *Cardoso, B., and Romão, T. 2015. Avoiding "…too late!" - Expressing and Detecting Opportunity with EveWorks and EveXL. In Proceedings of the 13th International Conference on Advances in Mobile Computing and Multimedia (MoMM 2015), ACM, New York, NY, USA, 293–302. DOI: 10.1145/2837126.2837139*
- *Cardoso, B., and Romão, T. 2015. Making Sense of EveXL , a DSL for Context Awareness. In Proceedings of the 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services on 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, 291–292. DOI: 10.4108/eai.22-7-2015.2260303*
- *Cardoso, B., and Romão, T. 2014. Presenting EveWorks, a Framework for Daily Life Event Detection. In Proceedings of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems (EICS '14), ACM, New York, NY, USA, 289–294. DOI: 10.1145/2607023.2610279*
- *Cardoso, B., and Romão, T. 2014. The Timeline as a Programming Interface. In CHI '14 Extended Abstracts on Human Factors in Computing Systems (CHI EA '14), ACM, New York, NY, USA, 1423–1428. DOI: 10.1145/2559206.2581303*
- *Centieiro, P., Cardoso, B., Romão, T., and Dias, A.E. 2014. If You Can Feel It, You Can Share It!: A System for Sharing Emotions During Live Sports Broadcasts. In Proceedings of the 11th Conference on Advances in Computer Entertainment Technology (ACE '14), ACM, New York, NY, USA, Article 15. DOI: 10.1145/2663806.2663846*
- *Cardoso, B., Romão, T., and Correia, N. 2013. CAAT - A Discrete Approach to Emotion Assessment. In CHI '13 Extended Abstracts on Human Factors in Computing Systems (CHI EA '13), ACM, New York, NY, USA, 1047–1052. DOI: 10.1145/2468356.2468543*

1

General Introduction

Who is the sculptor and from where? He's from Sikyon.
What is his name? Lysippus.
Who are you? Opportunity the omnipotent.
Why have you come on tiptoe? I always run.
Why do you have two wings on your feet? I fly with the wind.
Why do you hold a razor in your right hand? To show men that I am keener than any blade.
Why does your hair cover your eyes? For those I meet to grab on to.
Good heavens, why are you bald at the back? So that no one who still wants to can get hold of me from the back after I have run past just once on my winged feet.
Why did the sculptor fashion you? For your sake, my friend. He set me up in the entrance to be a lesson.

Epigram by Posidippus, describing a statue of Kairos authored by Lysippus (in [47]).

The broad Human-Computer Interaction (HCI) research field has been defined by the Association for Computing Machinery as "(…) *a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them*" [43]. It lies at the intersection of a number of bodies of knowledge, some of them with a clear focus on technological challenges – e.g., computer graphics and computer vision – and others deeply rooted in human factors – like psychology and sociology, just to name a few. This interdisciplinarity is well justified since HCI is ultimately the study of the communication between people and machines or, to put it metaphorically, the exploration of the fuzzily-bounded surface that separates and connects users and machines.

To be sure, just as there are many forms of communication between people, there are also many different ways for people to interact with machines. Of particular relevance for this work is a special form of communication that, as many topics of HCI, has a reflection in both Humanities and Technology – persuasive communication. Indeed, computers can persuade – this was the groundbreaking idea upon which a whole field of research was founded. The topic of Persuasive Technology was introduced by Fogg [37], at CHI'97; however, it was the publishing of his book *Persuasive Technology: Using Computers to Change What We Think and Do*, in 2003, that laid the necessary sounder foundations for research. This field of study, which Fogg coined *Captology* by acronyming the expression "Computers as Persuasive Technologies", is entirely dedicated to the study of the theories, principles, designs and the analysis of technological agents of persuasion. However, before delving deeper into the domain of persuasive technologies, it may be elucidating to consider the phenomenon of persuasion itself.

Persuasion has been eliciting academic interest for millennia, with its study predating that of persuasive technologies by far. Indeed, we can find it figuring prominently in Aristotle's classical work *Rethorica* (book 1, Chapter 2) [49], where a definition for the concept of rhetoric is formulated based on it: "*(rhetoric) may be defined as the faculty of observing in any given case the available means of persuasion*". Interestingly enough, in spite of the long time that persuasion has been studied, the definition of the term itself is still not a settled matter among philosophers and academics. Reardon [77], for instance, defines persuasion as "*the activity of attempting to change the behavior of at least one person through symbolic interaction*", whereas Zimbardo and Leippe [95] propose persuasion to be the changing of one's "*behaviors, feelings or thoughts about an issue, object, or action*". Many other definitions and theories have been proposed and can be found on the body of literature produced since Classical Antiquity, each with its own set of subtle differences. Nonetheless, in spite of this academic schism, all definitions appear to agree on one point: persuasion is about provoking changes as a consequence of adequate communication. Supporting this conclusion is Fogg's own definition [37], in which he states that "*for purposes of captology, persuasion is defined as the attempt to change attitudes, behaviors or both (without using coercion or deception)*".

Because it is a special form of communication – one with the purpose of provoking changes –, persuasion is subject to certain rules that, when followed or violated, can respectively increase or decrease the efficacy potential of a persuasive attempt. One of those rules is related to the moment in which the persuasive message is delivered. In fact, for what concerns persuasive interventions, there appears to be such a thing as a "right moment". The ancient Greek rhetoricians have recognized the importance of seizing these opportune moments to such an extent that they had both a word and a homonymous divinity for it: Kairos (Figure 1-1).

Figure 1-1. Il Tempo come Opportunità (Kairòs), by Francesco de' Rossi [30]**.**

As a curiosity, in ancient Greek mythology, Kairos was the spirit of the favourable moment and due measure, usually depicted as a young athletic figure, with wings on his back and feet - hinting at the fleeting nature of circumstances -, and a rather unusual hair, having only a lock of hair on his forehead - suggesting the urgency to grasp opportunities by the time they present themselves or otherwise lose them, as Kairos has no hair on the back of his head and, therefore, nothing to grab after his back is turned (see Figure 1-1). To conclude, there are many other artistic details to be found on ancient Kairos depictions and statues, each being a metaphor for various aspects of his fluid, transitory nature [26,47].

Besides the name of a divinity, *kairos* was also one of the two words that ancient Greeks used for time, the other being *chronos*, which also was the name of a mythological personification, the titan Chronos. The conceptual difference between *chronos* and *kairos* is that the former refers to linear, measurable time, while the latter suggests situational, circumstantial time, the time of opportunity, and should be perceived in all of its multidimensionality, with chronological time being just one of its facets. In fact, the temporal dimension of *kairos* can span anything from an exact moment to a wide time interval. It is a "window" of time during which action is most advantageous [26]. While *chronos* is quantitative, *kairos* has a qualitative nature.

Within the scope of Persuasive Technologies the prime importance of *kairos* to empower persuasion has also not passed unnoticed; in fact, Fogg has based a principle on it, the *Principle of Suggestion*: "a computing technology will have greater persuasive power if it offers suggestions at opportune moments" [37]. At this point, a natural question would be how to identify these – for lack of a better word – "*kairotic*" moments? Even though the ancients deeply recognized the importance of *kairos* in persuasion, they did not leave behind many practical guidelines on how to recognize those moments [37]. Nonetheless, researchers in the field of Psychology have identified some characteristics that may contribute to the correct identification of persuasion favourable moments. For instance, people appear to be more inclined to compliance when the time to act is close at hand [42,45,76]. Indeed, the suggestion of alternative paths through the creation of decision points just

before action takes place appears to have a significant impact on people's behavior. By way of illustrating this claim, imagine an application on a mobile device that, upon detecting that someone is nearing his/her home at the end of a day, suggests that he/she takes the stairs instead of the elevator, highlighting the myriad of benefits that will come from such an action (assuming, of course, that the application is aware that the user's living place has stair access).

From the previous discussion, one can safely deduce that any application designed to trigger interactions (persuasive or not) at specific moments must be aware of its context, as well as reactive. In this sense, from a technological point of view, it may be fair to surmise that a *kairos* moment can be interpreted as a change in state, i.e., an event or a conjugation of events. As Alberti et al. [1] mentioned, an event may be defined as a combination of observable states in a system that may occur at some points in space and time, in known configurations. It follows that, in order to detect them, it is imperative to observe the system where the events may occur and try to match the observations against the available knowledge. As it can be easily imagined, in terms of persuasive technologies, the system to observe and react to is a very complex composition of smaller systems, and the set of variables of interest is very comprehensive. In fact, it may encompass time, the surrounding environmental, geographical location, the users themselves and anything else that may be used to characterize a situation or entity – this description, in approximate terms, corresponds to the broad definition that Dey [32] has proposed for the concept of context. The successful acquisition and operationalization of that information is one of the central challenges of research on context awareness (more details on this topic can be found in Chapter 2).

Although a necessity – and a challenge in itself –, the detection of *kairos* is only a part of successful persuasive efforts. Indeed, even if an application has the potential to offer timely and adequate interaction, it is obviously of no use if it finds itself being physically or otherwise unable to deliver it. An answer to this issue may be found on mobile technology as it is, nowadays, a pervasive media that is well intertwined in our daily lives. The case of smartphones, in particular, holds an unparalleled promise for the deployment of context-aware,

persuasive applications. In fact, on top of their considerable computational power, smart phones have a privileged relation with their owners, being carried almost everywhere - Dey et al. [32] have found that smartphones are in the same room as their owners for about 90% of the time. Furthermore, if we note the growing plethora of sensors that are progressively being integrated into these platforms, we can further understand their potential for the deployment of context-aware applications and the delivery of persuasive content. Acknowledging the strong potential of mobile technologies for the detection of opportunity, Fogg proposed another principle which he named the *Principle of Kairos*: "*mobile devices are ideally suited to leverage the principle of kairos – offering suggestions at opportune moments – to increase the potential to persuade*".

Aside from the situational, external aspect of *kairos*, there are also relevant variables that are intrinsic to the end-receiver of a persuasive effort, namely affective states. Indeed, research suggests that people are more likely to be susceptible to persuasion when they are in a good mood, and this claim appears to find support in research both in the fields of Psychology [10,55] and Human-Computer Interaction [48,50]. Additionally, recent research suggests that boredom is also a positive factor for persuasion. Indeed, acknowledging that boredom is a common human emotion which may lead to an active search for stimulation, Pielot et al. [59,69] have designed a system that recommends multimedia content to its users upon detecting they are likely to be bored – i.e., looking for stimulation on their smartphones. Their results suggest that people are more likely to engage with recommended content when they are bored, as inferred by their machine learning model of user boredom. The assessment of affective states, however, is a very challenging task. While the field of Psychology has proposed a number of assessment tools, there is not yet a universal standard method for allowing computer applications to assess people's affective states reliably.

Summarizing, the discussion so far started by clarifying the concept of persuasion and introduced that of *kairos*, or opportunity, which is the main source of inspiration for the work presented in this thesis. Then, it has proposed the possibility of having *kairos* modelled as a conjunction of context events and signalled the promise held by mobile technologies for detecting and seizing

those events. It was also noted that, among the myriad of variables that are relevant to the definition of these events, the literature indicates that the user's own emotional states are of prime importance.

1.1 Research Questions

As per the above, the two intertwining research questions that guided this work are hereby formulated:

1. *Can opportunity, in all its inherent complexity, be operationalized in a simple and straightforward manner, in order to empower mobile devices to deliver interactions at meaningful moments of their users' lives?*

2. *Can quick and reliable assessments of emotions be seamlessly integrated into user interfaces, thereby leveraging the potential of technology for opportunity detection?*

1.2 Research Contributions

Addressing the preceding research questions, the work described herein proposes two major contributions, each on its side of the HCI human-computer dichotomy: (1) the development of a way of expressing and detecting contextual events in mobile devices and (2) a validated way of assessing user's emotional reactions.

The first contribution is EveWorks (an acronym of "Event Framework"), an engine for the detection of contextual events. To simplify deployment and use, the engine has been embedded into a prototype Android application bearing the same name: EveWorks. Additionally, much in the same way that applications interact with relational Database Management Systems (DBMS) through statements written in Structured Query Language (SQL), so does EveWorks expose an Application Programming Interface (API) to other applications through its own language, EveXL (another acronym, this time of "event expression language"), that proposes a new way of programming reactive behaviors – programming through intervals of time. The soundness of this approach is supported by the research work of Núñez and Cooperrider

[65], in which they state that everyday temporal concepts like duration, sequence, past, present and future are fundamental to the way we cognitively process and make sense of experience, and these constructs are present in culture after culture. The topics of EveWorks, EveXL and related subjects are further detailed in Chapter 2.

The second major contribution of this thesis is the Circumplex Affect Assessment Tool (CAAT) a graphical control that enables emotional reactions to be captured in a self-assessment approach. The CAAT and its respective user interaction have been designed based on Robert Plutchik's Circumplex model of emotions which, naturally, also serves as the tool's underlying theoretical model of affect. The CAAT's development and validation studies can be found in Chapter 3.

To conclude this thesis' introduction, emphasis should be placed on the broad range of promising applications of its contributions. Indeed, even though the inspiration for this research has been drawn from the rhetorical concept of *kairos*, the potential applications of the contributions proposed in this thesis extend beyond the domain of Persuasive Technology. As it is, applications of different sorts, with or without persuasive intents, may leverage the meaningfulness of their interactions with the user by means of the contextual event detection services offered by EveWorks. Additionally, studies and applications that require comprehensive user models or the assessment of affective data may benefit from using the CAAT's reliable and pleasant-to-use assessment method.

2

EveWorks and EveXL, Programming with Time Intervals

2.1 Introduction

Due to an increasing proximity to the general public in the last decade [51,67], smartphones have become a privileged platform for the deployment of meaningful interactions. Indeed, Dey et al. [32] have discovered that smartphones are in the same room as their owners for about 90% of the time. Their almost ubiquitous presence in our daily lives, along with the growing computational power and the comprehensive array of integrated sensors that smartphones are being shipped with, translates into a valuable potential for context-aware applications running on these devices to grab advantageous opportunities for interaction. We can find many references in the literature to the importance that the "right moment" has for interaction (e.g., [50]), especially in the field of Persuasive Technology – i.e., technology designed to change attitudes and/or behaviors through persuasion and social influence, but not coercion or deception (see Chapter 1). As previously mentioned, the founder of this field, B. J. Fogg [37], has based one of Persuasive Technology's principles on it, the Principle of Suggestion: "*a computing technology will have greater persuasive power if it offers suggestions at opportune moments*".

Nonetheless, the development of software and architectures targeting the detection of opportunities in the users' daily lives is not a trivial task, especially because what constitutes an opportune moment to a given application or interaction may be far from ideal to another. For instance, an application designed to provide support for the adoption of healthy habits may have a good opportunity to exhort a user to exercise by the time he/she enters home after a working day, whereas an application intending to announce the main

news headlines of the day, would likely not. Therefore, the programming of flexible enough architectures to support the implementation of application- or interaction-specific reactive behaviors is as much of an asset, as it is a challenging task. On top of the code for sensor reading, it is also necessary to develop the logic for processing gathered data and to detect and react to changes in context. Carlson and Lisper [19] propose a systematic approach based on the separation between the mechanisms for event detection and the rest of the application logic, thereby facilitating the tasks of designing and analysing reactive systems. The research community has already presented a number of frameworks (see Section 2.4.1) intended to address this challenge. However, most of these software artefacts are libraries whose sources are intended to be included and compiled with their interacting application code. This means that, unless considerable programming effort is involved, an application's reactive behavior is statically defined at compile time. While this approach is valid, it is mostly appropriate to applications with static reactive behaviors as, by default, it does not provide direct answers to applications requiring dynamic reactive behavior, i.e., changes in their reactive behavior at runtime.

To address these challenges, a framework for contextual event detection was developed: EveWorks – an acronym of Event Framework. For prototyping purposes and to facilitate integration into the ecosystem of mobile devices' operating systems, the framework was encapsulated into a standard Android application that offers event detection services to other applications, under the overarching client-server software architecture. By exposing the functionalities of the framework it embeds as services, the application encapsulates the complex mechanics of contextual data acquisition and event matching. Being merely a wrapper for the framework, the application inherits the same name, EveWorks.

This design brings many practical advantages; for instance, it facilitates code separation and promotes efficient use of resources like memory and battery since, by definition, the server component (the EveWorks application) will run as a singleton (see further details in Section 2.2.1).

However, contrasting with most of the approaches found in the literature, external applications submit their events of interest through expressions written in EveXL (an acronym of Event Expression Language), a novel, runtime-interpreted scripting language, developed in the context of this thesis' work. EveXL's runtime interpretation enables a client application to change the set of events it is listening for, after it has been deployed to a user's smartphone – thereby leveraging the adaptation potential of context-aware applications (the topic of EveXL is covered in Section 2.3).

2.2 EveWorks, a Framework for Kairos-Awareness

2.2.1 Architecture

Originally, EveWorks (the framework) was implemented as an Android library – a development module that holds shared source code and resources. That design allowed for direct interactions between EveWorks and the code of any application that would use it: as the library's methods were exposed as an Application Programming Interface (API), they were directly referenced in the application's sources. Therefore, the application code that implemented the application-EveWorks interaction was compiled and included in the resulting Android Application Package (*.apk* file).

However, there were some limitations inherent to this initial implementation. Since EveWorks' core mechanics are implemented as an independent routine service that periodically reads the necessary sensors and runs the event matching procedure (see Section 2.2.2), if two applications were to use the EveWorks library module simultaneously (one per application), there would be two routine threads running in parallel, potentially querying the same sensors simultaneously. This could result in inefficient use of resources like CPU, memory and battery – something that is all the more undesirable in resource-constrained mobile platforms. Additionally, as EveWorks stores sensor readings in a database, running an EveWorks instance per application implies that there would be as many databases on a system as there would be EveWorks-using applications. This scenario, like Kramer et al. [51] have stated, warrants the need for "*single customizable context acquisition mechanism which*

would monitor, manage and disseminate contextual information to context aware applications running on the same mobile device".

We have, therefore, encapsulated the EveWorks library within a regular Android application, applying the client-server software architecture pattern to the relationship between applications and EveWorks. The advantages of this refactoring are many. For instance, because EveWorks is now a singleton entity in a given Android environment, there is no database replication and sensors will not be concurrently read by two different EveWorks instances. Of course, resorting to the client-server pattern is not the only way to address these redundancy problems. Nonetheless, it amounts to the simplest and, therefore, more elegant solution.

To facilitate the interface between client applications and the EveWorks application, a Software Development Kit (SDK) was developed, exposing a minimalist API, while abstracting and simplifying the details of the communication between applications and EveWorks. Therefore, Applications wanting to take advantage of EveWorks' event detection services have to include this SDK and, thereafter, directly invoke its API methods.

For example, in order to notify EveWorks of the interest in a particular event – say, that the user has arrived home –, an application registers a new event in EveWorks by calling the appropriate SDK method which requires the following arguments: (1) the event identifier, (2) the EveXL expression that defines the event itself , (3) the name of the handler class that will be notified when the event is detected, and (4) another handler that will be notified of the results of the event submission process. When EveWorks detects that a submitted event has taken place, it will communicate the occurrence back to the SDK that has originally submitted it, by passing it the event's identifier. In turn, the SDK will instantiate through reflection the handler class whose name had been previously passed as one of the three arguments of the event-submitting method and, afterwards, invoke its respective handling method. This procedure is depicted in Figure 2-1, and Figure 2-2 shows an Android Studio print screen exemplifying how an application can submit an (hypothetical) EveXL event into EveWorks (please refer to Section 2.3 to learn about EveXL and its syntax).

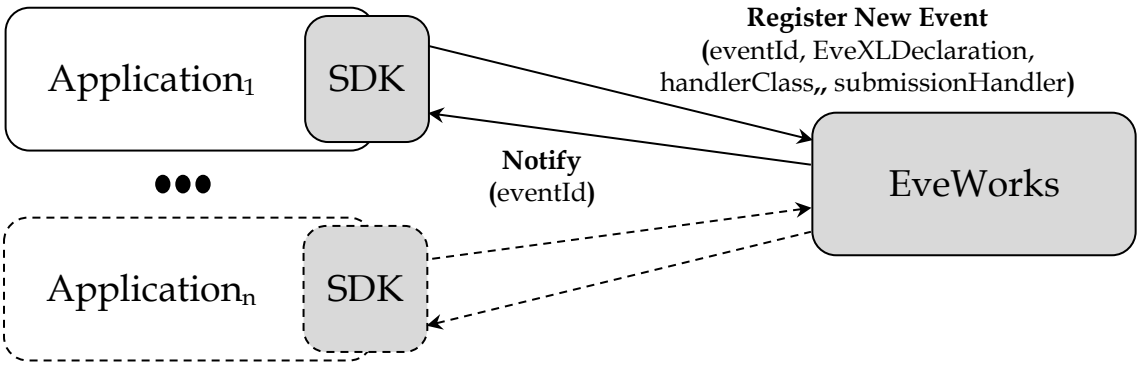


Figure 2-1. EveWorks' interface with other applications.

```
EveWorks.registerNewEvent(MainActivity.this,
    "EXAMPLE EVENT IDENTIFIER",                    (1)
    "[A] AS [color = 'red'] " +
    "[B] AS [color = 'black'] " +                  (2)
    "WHERE [A] MEETS [B]",
    EventReceiver.class,                           (3)
    new EveWorksFeedbackReceiver() {
        @Override
        protected void onParsingError(Context context, String eventId,
                                      String parsingFeedback) {
            // ...Code that will be run when EveWorks
            // reports back a parsing (syntactic) error...
        }
        @Override
        protected void onEventRegistered(Context context, String eventId) {
            // ...Code that will run when EveWorks reports that
            // the event has been successfully parsed and registered...
        }
    }
);
```

Figure 2-2. Android Studio print screen, showing the Java (Android) code for submitting a new event to EveWorks. Argument (1), a string, corresponds to the event identifier; (2) is also a string and is the event's EveXL expression; (3) is the local application class that will be notified when the event occurs; finally, the last argument is the instance of EveWorksFeedbackReceiver that will be notified if the event registration fails or succeeds.

2.2.2 Operation

The core mechanics of EveWorks are implemented as a periodically running routine, with three basic steps: *reading*, *matching* and *cleaning*. On the first step, *reading*, the probes referenced by the currently active events are read for data. This, of course, saves the running device's battery, since it keeps the framework from activating and reading unrequested sensors. At the end of this phase, the gathered context data is stored in the database tables, the trace tables which contain a limited history of chronologically ordered sensor data. On the *matching* step, the trace tables are queried and the retrieved data is used to create or update the currently active events' temporal models, in order to assess if any has occurred (these details are covered in Section 2.3.5). Finally, during the *cleaning* step, EveWorks automatically eliminates outdated readings from the trace tables (although this is configurable, the trace tables will store timestamped readings from the last 24 hours by default). Since this routine runs periodically, sensors are read intermittently – a feature that, in combination with the exclusive reading of directly referenced sensors, allows for important battery savings. Naturally, the amount of saved battery is directly correlated to the (also configurable) frequency of the EveWorks routine.

2.2.3 EveWorks Sensors

Because EveWorks required careful evaluation and tests, three sensors were developed and integrated into its prototype. They are: *time*, *Near Field Communication* (NFC) and *indoor location*. These sensors do not correspond to an exhaustive list, as they were developed for prototyping purposes and are discussed here for the purpose of providing context for the readers of this thesis.

The implementation of both the *time* and *NFC* sensors is very straightforward. Indeed, the former's output values are direct transformations of readings from the Android system's internal clock, and the latter's are direct readouts of the Android framework's native NFC sensor.

The *indoor location* sensor, on the other hand, operates by applying a distance metric to the Wi-Fi networks available in the environment. In more detail, the location sensor manages locations by having users fingerprint and

name their current locations. EveWorks will then store those locations – pairs of Wi-Fi fingerprints and names – in its internal database. Thereafter, locations can be directly referred to by name in EveXL statements (see Section 2.3). When trying to detect if a user is currently in a given location, the sensor will perform a scan of the available Wi-Fi networks, create a fingerprint and return the location whose fingerprint is the "closest" in its location database. The metric used to compare these fingerprints is the Bray-Curtis divergence, which was (empirically) proven to be the most reliable for the conditions of indoor, Wi-Fi-based localization.

Besides through submitted events, external applications interfacing with EveWorks can also make direct calls to its sensors. To do so, they just have to call a specific asynchronous method of the SDK (see Section 2.2.1), passing the sensor name as a parameter. Therefore, once EveWorks has read the specified sensor, it will return the read value to the SDK which, in turn, will supply it to the calling application. This approach lends flexibility to the use of EveWorks, as applications can directly use its sensors and react to changes in context without having to formally compose EveXL expressions.

2.2.4 Extensibility

In the same way that different applications are interested in different events, the sensors used in context assessments may also vary across applications. Therefore, to provide an extensible response to this requirement EveWorks allows client applications to define their own sensors. In technological terms, this means that developers may implement their own sensor class by extending the specific sensor class included in the EveWorks SDK and declaring the new sensor to EveWorks (the Android framework offers a convenient answer to this in its Application class, whose methods will be called once, every time an application is started). The specific SDK method for declaring a sensor to EveWorks requires the name of the external sensor and the keyword that will be used to reference this sensor in future EveXL expressions.

For instance, imagine a smartphone application using weather data to advise its users, when they leave home, about the best sun protection factor to

apply, according to daily ultraviolet (UV) radiation levels. In this setting, on the lack of a built-in UV sensor, an application could resort to querying remote sources (e.g., an external website) for this information. If this functionality was to be implemented in a new sensor class, when declaring it to EveWorks one could associate it with, for instance, the "*uvindex*" keyword. Afterwards, the new sensor would be accordingly referenced in EveXL expressions, thereby implementing new application context-reactive behaviors (see the following Section).

2.3 EveXL, a DSL for Kairos

As previously stated (see Section 2.2.1), communication between applications and EveWorks is accomplished through expressions written in a special Domain-Specific Language (DSL): EveXL.

However, it may be fitting to present here a clear definition of a DSL before discussing the language itself. According to Martin Fowler [39], DSLs are programming languages of limited expressiveness focused on particular domains. He distinguishes two different types of DSL: external and internal. While external DSLs are languages different from the main programming language of an application, that are interpreted or translated into a program in the main language, internal DSLs are transformations of the main programming language. The modern JavaScript library jQuery is an example of an internal DSL, whereas SQL is a good example of an external one. Indeed, although jQuery programming is subject to the same syntactic rules of its base language (JavaScript), it uses the method chaining technique to create a new programming interface, effectively changing the way one writes jQuery/JavaScript programs. On the other side, SQL has its own, independent syntax and an interpreter that translates queries and statements into operations on relational databases.

Regarding Martin Fowler's [39] definition, EveXL is an external DSL, a language with its own syntactic rules and semantics. The application-EveWorks communication process is somewhat analogous to the way that Relational Database Management Systems (RDBMS) rely on SQL expressions to allow external applications to access and manipulate the data they contain. In

EveWorks' case, however, the expressions describe events of interest rather than data management operations. Also, in the same way that SQL is based upon Edgar Codd's Relational Algebra [21], EveXL is based upon James Allen's Interval Algebra [3,4], an interval-based calculus for reasoning about temporal descriptions of events (see further details in Section 2.3.2).

As a programming script, EveXL is nonprocedural in that its statements describe the situations that should be captured, not the procedures to do so. Indeed, EveWorks receives EveXL statements, parses and interprets them, and thereafter automatically generates the routines that capture and store the necessary sensor data, while periodically checking until the received events have either occurred or expired.

In itself, EveXL is unique in many aspects, but the foremost is its underlying, time interval-based model of temporality – of "daily life". Indeed, EveXL proposes a different way of formalizing daily life events and, to an extent, a whole new way of programming the reactive behavior of mobile applications: programming through intervals of time. Events of interest are, therefore, expressed in the form of EveXL statements that, in a nutshell, correspond to temporal articulations of time intervals (more details can be found in the next sections). This model is closer to our sense of reality and the way we make sense of our daily experiences [65], than is the more common data-oriented event models usually found in research and industry (see Section 2.4).

2.3.1 Basic Constructs - Intervals of Time

As mentioned before, one of the most distinguishing aspects of EveXL is that everything revolves around the concept of time interval. An interval of time I is, hereby, defined as a pair of endpoints, a starting point I_α and an ending point I_ω, such that $I_\alpha < I_\omega$ (two lower case letters of the Greek alphabet are used here to designate the beginning and the ending of an interval, the alpha and the omega, which, in some traditions, have long been connoted with the concepts of starting and ending).

In EveXL, there are two types of time interval: (1) pure-time and (2) data-invariant intervals.

The former are the simplest, representing time periods in the course of which the only thing that happens is the passing of time. In a nutshell, pure-time intervals represent precisely that – quite literally, the passing of time.

On the other hand, data-invariant time intervals represent periods of time during which the variation of some specific dimensions of context comply with predefined constraints, i.e., some data-invariants are held. Therefore, data-invariant time intervals are defined through combinations of data invariants, i.e., sets of conditions that must hold in the course of the said interval. Because EveWorks' core routine – which includes sensor reading and event detection – is run periodically, assumptions must be made about the state of the context between readings, when intervals of time are built out of sparse sensor data. EveWorks addresses this by assuming that context does not change when the same data invariants are held in two consecutive sensor readings.

This allows intervals to be built retroactively. For instance, if a given interval's data invariants are true at current time, EveWorks will span this interval from the current time, to the time of the last reading. In turn, if the same invariants are also true at the time of the last reading, the interval will then span from the current time to the penultimate reading and so on, until reaching the point in time when the invariants no longer hold. To exemplify this interval building mechanism, Table 2-1 displays the iterative building of an interval defined through a simple "`S = 'A'`" invariant, that requires a hypothetical sensor "`S`" to read the value "`A`". For simplification, a reading period of 1 "time unit" is assumed here, and that time starts counting at timestamp 1.

Table 2-1. The evolution of interval I's range, defined as S = 'A'.

Timestamp	Sensor S reading	Range of Interval I
1	`A`	[0, 1]
2	`A`	[0, 2]
3	`B`	[0, 2]

By timestamp 1, the time of the first reading, interval `I` already ranges from 0 to 1, as 1 corresponds to the current time and 0 to the time of the previous reading (assuming a reading period of 1). Likewise, since the "`S = 'A'`" invariant is also held at timestamp 2, the range of `I` "stretches" to

assimilate the timestamp 2 as its new ending time. Finally, the invariant is no longer held at the time of timestamp 3, and the range of interval `I` is not updated.

2.3.2 Relations between Intervals of Time

In isolation, data invariant-based time intervals have limited power in expressing the myriad of events that applications may be interested in. Consequentially, EveXL allows the articulation of relations between intervals of time by providing an implementation of the thirteen operators of James Allen's Interval Algebra [3,4]. The operators are: *is before*, *meets*, *overlaps*, *starts*, *during*, *finishes*, plus their respective converses *is after*, *is met by*, *is overlapped by*, *is started by*, *contains*, *is finished by* and, finally, the thirteenth relation *equals*, which is its own converse. A timeline illustration of these operators can be found in Table 2-2.

By definition, all of these basic relations are *distinct*, because no pair of fully defined intervals can be related by more than one; *exhaustive*, because any pair of definite intervals can be described by one of the relations; and they are *qualitative* – rather than quantitative - because no numeric time spans are considered [4,5].

Table 2-2. The operators of James Allen's Interval Algebra, their respective converses, timeline illustrations and relations between their endpoints.

Operator	Converse	Timeline Illustration	Endpoint Relations
A **IS BEFORE** B	B **IS AFTER** A		$A_\omega < B_\alpha$
A **MEETS** B	B **IS MET BY** A		$A_\omega = B_\alpha$
A **OVERLAPS** B	B **IS OVERLAPPED BY** A		$(A_\alpha < B_\alpha)$ && $(A_\omega < B_\omega)$
A **STARTS** B	B **IS STARTED BY** A		$(A_\alpha = B_\alpha)$ && $(A_\omega < B_\omega)$
B **CONTAINS** A	B **IS CONTAINED BY** A		$(A_\alpha > B_\alpha)$ && $(A_\omega < B_\omega)$
A **FINISHES** B	B **IS FINISHED BY** A		$(A_\alpha > B_\alpha)$ && $(A_\omega = B_\omega)$
A **EQUALS** B	B **EQUALS** A		$(A_\alpha = B_\alpha)$ && $(A_\omega = B_\omega)$

2.3.3 EveXL Syntax

EveXL has been designed with simplicity in mind. As previously stated, it is a runtime-interpreted script, whose syntax was mainly inspired by that of SQL and the timeline visual representation of temporality (an example of a timeline can be found in Figure 2-3).

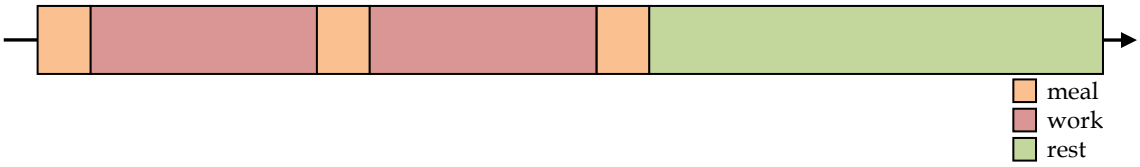


Figure 2-3. Example of a timeline, representing a common working day, with the hours for meals, work and rest (conventionally, time progresses from left to right).

This section will explore EveXL from the syntactic perspective and will also present some comprehensive examples of EveXL statements. When appropriate, syntax (or railroad) diagrams will be used to convey the syntactic rules. The diagrams have been generated out of EveXL's grammar using the publicly available Railroad Diagram Generator tool [75].

2.3.3.1 Literals

EveXL supports text, numeric and time and timespan literals that are used in a number of places and roles.

Text literals

Text literals are encased in single quotation marks and all characters are allowed with the exception of single quotes and new line characters. This notation is represented next, in Figure 2-4. "`'K41r0s-is-now!'`" and "`'TeMpUs FuGiT'`" are examples of text literals (quotes included).

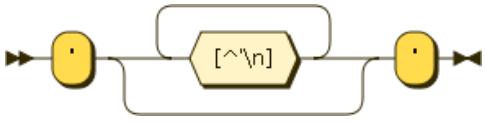


Figure 2-4. The syntax of EveXL's text literals.

Numeric literals

Numeric literals are used to specify integers and floating point numbers, and the notation to use can be seen in Figure 2-5. "`3`" and "`123.123`" are examples of numeric literals.

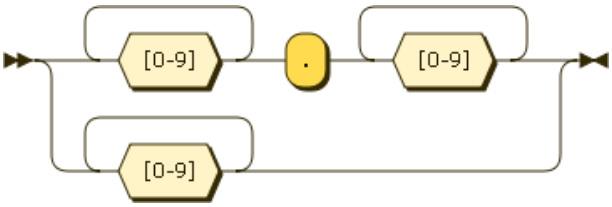


Figure 2-5. The syntax of EveXL's numeric literals.

Time literals

Time literals represent single points in chronological time. The temporal model used in EveWorks is the 24-hour clock convention and, currently, has no default support for dates (i.e., the specification of day, day of week, month or year is not possible by default, although it could be implemented in a new time sensor, if so desired). The specification of seconds is optional. The notation for time literals is represented in Figure 2-6. "`18:00`" and "`12:01:15`" are examples of time literals.

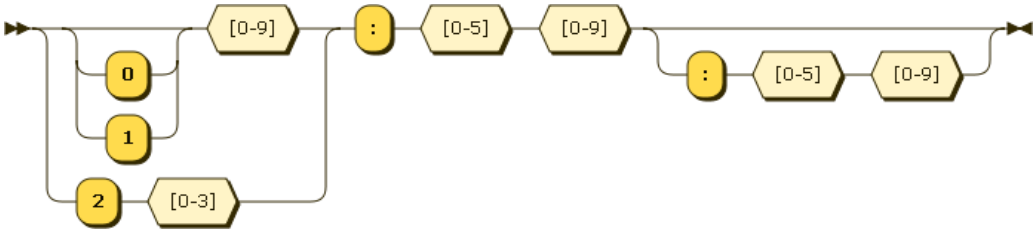


Figure 2-6. The format of EveXL time literals.

Duration literals

Duration literals specify time lengths – or, very literally, the duration of time intervals. They allow time lengths to be represented in hours, minutes, seconds or combinations of them. A duration literal always has a leading field and one or two optional trailing fields, with the only restriction being that the leading fields must be more significant than the trailing, like explicitly represented in Figure 2-7. Examples of valid durations would be "`1m30s`", "`3h30m35s`" and "`48h`", respectively meaning "one minute and thirty seconds", "three hours, thirty minutes and thirty-five seconds" and "forty-eight hours".

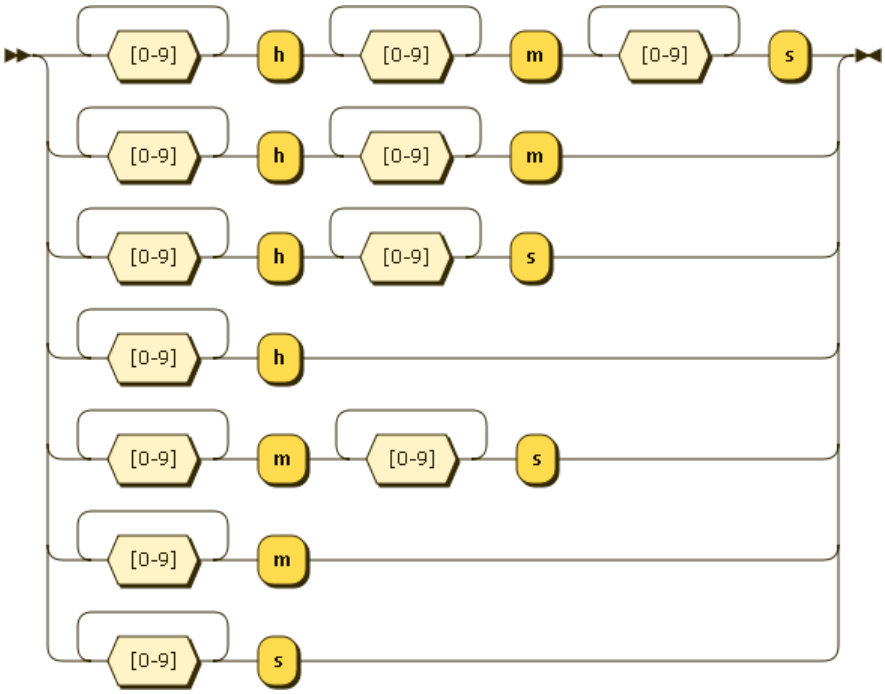


Figure 2-7. EveXL notation for duration literals.

Identifiers

In EveXL, identifiers are used to unambiguously identify intervals of time and sensors in data invariants. Identifiers are case-sensitive, unpunctuated strings that begin with a letter. While the underscore character is permitted after the first letter, blank space is not. They abide to the syntactic rule

represented in Figure 2-8. "`Interval1`" and "`sleepSensor_3`" are examples of valid EveXL identifiers.

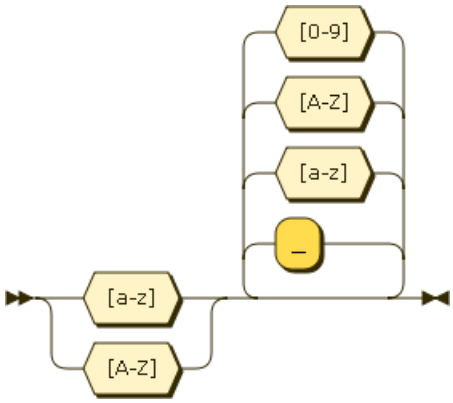


Figure 2-8. The syntax of EveXL identifiers.

Comparators

When programming events with EveXL, it is often necessary to compare values. To this end, EveXL has an assortment of comparison operators which are used to compare sensor data to text and numeric data, according to the nature of the sensors involved and they work in precisely the same way as their SQL counterparts. EveXL's textual and numerical comparison operators can be seen in Figure 2-9.

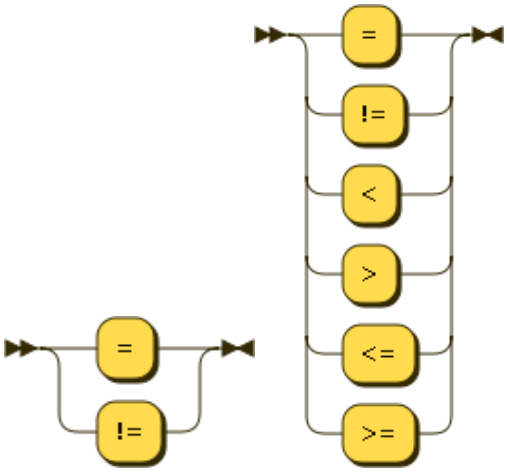


Figure 2-9. EveXL's textual (left) and numerical (right) comparison operators. From top to bottom: *equal*, *not equal*, *less than*, *greater than*, *less than or equal* and *greater than or equal*.

2.3.3.2 *Timespans*

Timespans represent the length of time between two specific points in time. Therefore, a timespan is expressed through its two endpoints, separated by the `AND` keyword. Besides the restrictions covered in Section 2.3.3.1 (*Time literals*), the only restriction a timespan enforces on its two endpoints is that they must be different from one another. This means that, since the EveWorks implementation of chronological time follows the 24-hour clock convention and dates are not represented, the first endpoint may precede the second. For instance, "`11:00 AND 10:00`" is a valid timespan, even if, at first sight, 11:00 comes after 10:00. EveWorks addresses this apparent incoherence by assuming that, in a timespan, the first point in time always comes before the second. Consequently, the "`11:00 AND 10:00`" timespan is assumed to represent the 23-hour interval that separates the 11:00 and 10:00 hours of two consecutive days. The syntax of timespans is represented in Figure 2-10, referencing the terminal *Time* (see Section 2.3.3.1, *Time literals*). Both "`10:00 AND 11:00`" and "`12:30 AND 15:30:45`" are examples of valid timespans.

Figure 2-10. The syntax of timespans.

2.3.3.3 *Interval References*

EveXL's syntax features some visual cues intended to create an analogy between references to time intervals and visual representations of intervals on a timeline (this subject is further explored in the study presented in Section 2.5.1). Therefore, all references to intervals of time within an EveXL's statement are enclosed within square brackets, so as to create the visual feeling of a "block" under the metaphor of an enclosed and labelled box of time during which something happens. Inside the square brackets is the identifier itself, the label of the interval as an *Identifier* literal as described in Section 2.3.3.1, *Identifiers*.

The syntax for this notation can be seen in Figure 2-11. For example, "`[I1]`" and "`[thisIsAnInterval]`" are valid references to intervals of time, and should be read as "*the time interval named I1*" and "*the time interval named thisIsAnInterval*", respectively.

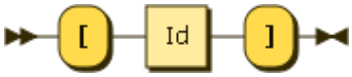


Figure 2-11. The syntax for the interval identifiers, enclosed within square brackets (an analogy to block-based, visual timeline representations).

2.3.3.4 *EveXL statements*

As previously stated (see introductory text of Section 2.3), EveXL is a DSL for the expression of daily life events. These events are expressed as statements that, given the nonprocedural nature of EveXL, describe the situations in which applications are interested and not the way to capture them.

Every EveXL statement is composed of three (or two, as the first is optional) different clauses: an optional *activity clause*, the *interval declaration clauses* and the *predicate clause*. The activity clause is a list of activity timespans that tell EveWorks when the event is to be processed; the interval declaration clauses declare the time intervals that will be used to formalize the event and,

finally, the predicate clause articulates and imposes constraints on the previously declared intervals. EveXL statements follow the general rule represented in Figure 2-12, and comprehensive examples of statements can be found in Section 2.3.4.

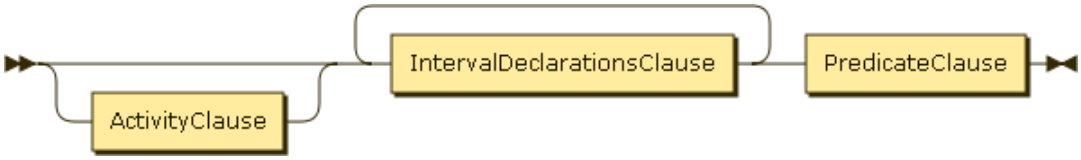


Figure 2-12. The general syntax of EveXL statements.

2.3.3.5 *Activity Clause*

The activity clause is an optional component of a statement that starts with the "`ACTIVE BETWEEN`" reserved words, followed by a comma-separated list of timespans whereupon the event is to be processed by EveWorks. The syntactic rule for this clause is represented in Figure 2-13.

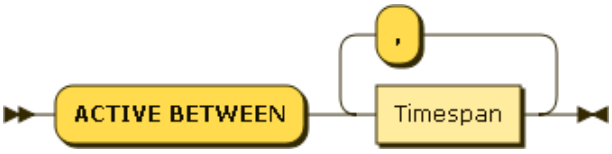


Figure 2-13. The format of the activity clause.

More than an essential part of an event description, activity timespans are a way of explicitly optimizing battery consumption by telling EveWorks that a given event should only be processed while the current time (as given by the system's clock) is within any of its activity timespans. This logically implies that, while the current time is out of an event's activity timespans, the sensors used in its intervals will not be read, data will not be collected and neither will EveWorks run the routines for detecting and triggering the said event. A statement missing an activity clause is interpreted by EveWorks as being always active, i.e., EveWorks will always be on the lookout for its occurrence and, therefore, will be processing it in every cycle (more details about EveWorks' mechanics can be found in Section 2.2.2).

To illustrate the usefulness of this approach, imagine having a message intended to be delivered at lunchtime, when the user leaves his/her working place (perhaps a digital pamphlet advertising a nearby restaurant's lunch specials). In this situation, it would make sense to restrict the event's activity timespan to, say, 11:00 to 14:00. This way, it will only be triggered if the user leaves his/her workplace within this period of time (i.e. lunch time) and no battery power will be wasted in unnecessary sensors reads out of this timespan.

Examples of two valid activity clauses can be seen in the numbered lines of Snippet 1, in which (1) corresponds to the activity period of the last example – between the 11:00 and 14:00 hours – and (2) defines, in a single clause, two different periods of activity – from 9:00 to 13:00 and from 14:00 to 18:00.

```
(1)   ACTIVE BETWEEN 11:00 AND 14:00
(2)   ACTIVE BETWEEN 9:00 AND 13:00, 14:00 AND 18:00
```

Snippet 1. Two examples of valid activity clauses.

2.3.3.6 Interval Declaration Clause

After the optional active clause, every EveXL statement defines the intervals that are used in the formalization of its event. It does so through a potentially unlimited number of interval declaration clauses, each declaring a single interval. These clauses begin with the identification of the interval being declared through a reference, such as detailed in Section 2.3.3.3; then, after the reserved keyword `AS` and within square brackets, appears the definition of the interval's data invariants or, in case of pure-time intervals, an ellipsis. Like explained in Section 2.3.3.3, square brackets are also used here as a way to establish a visual analogy between EveXL interval references and the timeline, a common visual representation of temporality. Besides, since interval references are themselves enclosed in square brackets, this helps to create the feeling of equivalence between references to an interval and its properties, like data invariants or, in case of pure-time intervals, the passing of time.

The syntactic rule for these clauses is represented in Figure 2-14. The syntax of pure-time and data-invariant intervals, along with examples, can be respectively found in the next two Sections, *Pure-time interval declaration* and *Data-invariant interval* declaration; further details on these two types of intervals have been provided previously, in Section 2.3.1.

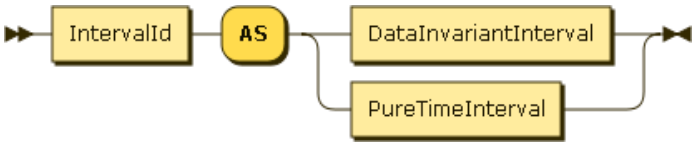


Figure 2-14. The syntax for declaring both data-invariant and pure-time intervals.

Pure-time interval declaration

As previously stated (see Section 2.3.1), pure-time intervals represent the passing of time. In other words, this means that during such an interval context may or may not change, and events may or may not occur – such is irrelevant: the only thing that matters is the passing of time. To emphasize the dichotomy between pure-time and data-invariant intervals, EveXL uses, in the notation of pure-time interval declaration, an ellipsis in the place where data-invariant intervals have their data invariants. Pure-time intervals may be tagged with the "`NOW`" modifier (the topic of interval modifiers is covered in the following Section *Modifiers*).

The notation of this type of interval is represented in Figure 2-15 and all of the three numbered examples of Snippet 2 are correct declarations of pure-time intervals.

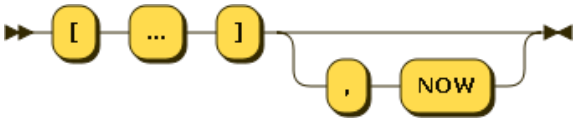


Figure 2-15. Notation of a pure-time interval declaration.

```
(1)    [PureTimeInterval] AS [...]
(2)    [I1] AS [...]
(3)    [TimeIntervalEndingNow] AS [...], NOW
```

Snippet 2. Three examples of the declaration of pure-time intervals.

Data-invariant interval declaration

The other type of time interval, the data-invariant interval (see Section 2.3.1), represents periods of time during which some data-invariants are held across sensor readings. The general syntax of these intervals is represented in Figure 2-16.

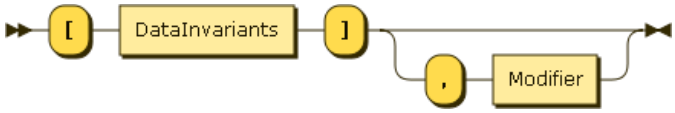


Figure 2-16. The syntax of a data-invariant interval declaration.

Contrasting with pure-time intervals, for which a single ellipsis represents their independence from data other than temporal, data-invariant intervals are defined through combinations of conditions that are written much like the predicates of SQL WHERE clauses. Indeed, like what happens with SQL predicates, data invariants are combined using the logical `AND` and `OR` operators. While the `AND` operator has higher precedence than the `OR`, parenthesis can be used to enclose combinations of data invariants, grouping them and thereby changing the evaluation order. However, instead of imposing conditions on data rows like SQL predicates do, EveXL invariants impose them on sensor data, for the duration of the intervals they define. The syntax of data invariant combinations can be seen in Figure 2-17.

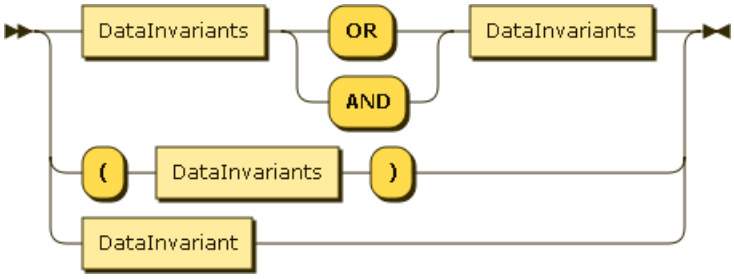


Figure 2-17. The syntax of data invariant combinations.

A single data invariant, represented by the "DataInvariant" nonterminal in the preceding figure, corresponds to a single condition on sensor data. In a data invariant, sensors are referred to through the use of unique identifiers (see Section 2.3.3.1, *Identifiers*), and their readings are compared against values using the comparison operators commonly found in SQL implementations for textual and numerical data, with the same semantics and order of precedence (see Section 2.3.3.1, *Comparators*). On the other hand, the text and numeric values to which sensor data is compared against are written following the rules detailed in Section 2.3.3.1, *Text literals* and *Numeric literals*, respectively. The syntax of data invariants can be seen in Figure 2-18.

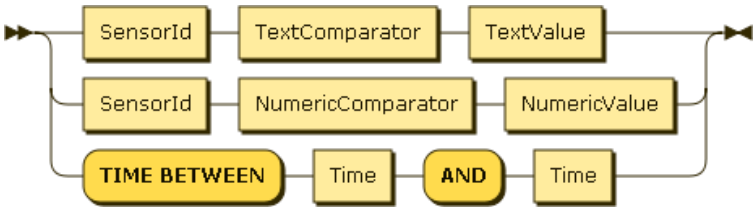


Figure 2-18. The syntax of a data invariant.

Besides sensor data, EveXL also allows time to be used in interval invariants. This may seem somewhat redundant: after all, time intervals are spans of time between two points in time by definition and, therefore, they themselves are invariants in the temporal dimension. However, because they are declared through constraints (invariants) on dimensions other than time, the fact is that chronological time itself assumes a secondary, almost incidental role in EveXL's conceptual framework. In this setting, because some situations are more accurately described if chronological time is involved in their descriptions, EveXL allows it to be conditioned in the scope of time intervals.

For instance, imagine trying to detect if a given user is at home in the morning. The simplest formalization for this situation would be to have an arbitrary interval of time during which the "location" sensor reads the value "home" and with timing between 6:00 and 12:00. This, of course, is just an illustrative example as there are, naturally, alternative approaches to work with **t**his scenario: for instance, to declare an event with an activity clause set to the morning period (see Section 2.3.3.5).

To use time in interval invariants, the reserved keyword *time* must be used, along with the `BETWEEN` operator to compare it against a timespan (see Section 2.3.3.2). Indeed, the `BETWEEN` operator is, in fact, the only comparison operator that EveXL allows for time. Indeed, since the EveWorks' implementation of chronological time follows the 24-hour clock convention the `BETWEEN` operator removes the ambiguity that would otherwise be present if the regular numerical comparators were to be used. For instance, imagine having a hypothetical interval defined with the "`time >= 23:58`" invariant: if the current time is 23:59, the invariant intuitively holds; however, the same is not true it the current time is 00:00 because even though in the 24-hour notation, the 00:00 time is two minutes after 23:58, this time also marks the beginning of a new 24-hour cycle. This, of course, is because the date (day, month and year) is missing from the time specifications. This ambiguity is removed if the invariant is rewritten as "`time BETWEEN 23:58 AND 23:57`". Truly, because the "`23:58 AND 23:57`" is an EveXL timespan, the ending 23:57 time is implicitly assumed to happen after the starting 23:58 time, i.e., it is interpreted as pertaining to the following day.

Data-invariant time intervals can have the "`NOW`", "`LAST`" or no modifier at all (this topic is detailed in the next Section).

Snippet 3 displays three numbered examples of declarations of data-invariant time intervals, each with its own combination of data-invariants. The declarations should be read as "(1) *the interval named 'Home' corresponds to a period of time during which the sensor 'place' reads the value 'home'*, (2) *the 'HomeOrGym' interval expresses an interval during which the sensor 'place' reads the value 'home' or 'gym'*; *and, finally,* (3) *the interval named 'SleepTime' stands for a span*

of time during which the 'place' sensor returned the value 'bedroom' and the value returned by the sensor 'time' was between 20:00 and 8:00".

```
(1)   [Home] AS [place='home']
(2)   [HomeOrGym] AS [place='home' OR place='gym']
(3)   [SleepTime] AS [
          place='bedroom' AND
          time BETWEEN 20:00 AND 08:00
      ]
```

Snippet 3. Three examples of the declaration of data-invariant intervals.

Modifiers

One final element of the interval declaration clause remains to be discussed: the modifiers that may be suffixed to an interval's declaration. Only one modifier may be appended to each interval, if any.

In particular, pure-time intervals may only be suffixed with "`NOW`" and data-invariant intervals may have either "`NOW`" or "`LAST`". The syntax of the interval modifiers is represented in Figure 2-19.

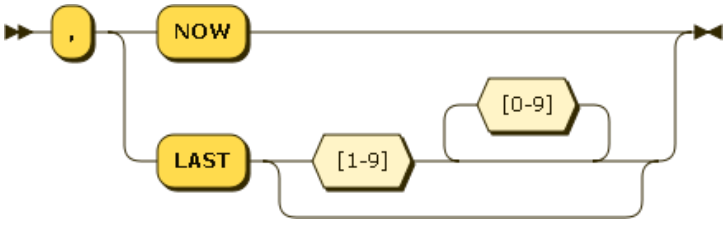


Figure 2-19. The syntax of the interval modifiers.

Like the optional activity clause of an EveXL statement, these modifiers are also explicit ways of optimizing EveWorks' performance. Indeed, to annotate an interval with the `NOW` modifier is to tell EveWorks that the interval is still going on at current time, i.e., its data invariants must hold at the time of the latest sensor reading. On the other hand, the `LAST` modifier, which has an optional `n` parameter, means that the interval being defined corresponds to one of the last n intervals in which the data invariants hold, where n is an integer

greater than 0. The omission of the n value, as well as the total omission of modifiers, equates to `LAST 1`, i.e., it tells EveWorks that the interval corresponds to the last interval of time during which the data invariants held, regardless of being true at current time.

These modifiers are relevant since EveWorks, by design, stores interval data in memory (although, to prevent data loss from eventual memory shortages or mobile phone shutdown, data is also kept in a private database). When EveWorks detects that an event has occurred, the resources that had been allocated to its intervals' data will be made available for garbage collection and their occupied memory freed when necessary. That way, for instance, if an interval has the `NOW` modifier and its invariants do not hold at current time, it will simply be discarded instead of continuing to consume resources. The same applies for intervals having the `LAST n` modifier: only information about the last n intervals will be kept in memory.

For example, imagine trying to detect if the user went twice to the market, one time after the other. To do it, two intervals are necessary, both having the sensor "location" reading the value "market", and one with the `LAST 2` modifier, as in Snippet 4.

```
[Market1] AS [location='market'], LAST 2
[Market2] AS [location='market']
```

Snippet 4. Two intervals of time with the same invariant. EveWorks will have `Market1` referencing either the last or the penultimate period of time in which the location sensor returned the value 'market'; on the other hand, `Market2` will always mean the last.

This way, the interval "Market1" will match either the last or the current period of time during which the sensor "location" read "market", while the "Market2" interval will correspond to the last time that happened. Evidently, since the `LAST n` modifier only tells EveWorks to try to match an interval in the last n times its invariants held, even with the `LAST 2` modifier the "Market1" interval may still be pointing to the very last time such has happened and, thereby, the same as "Market2". To avoid this situation, care

must be taken to clearly express in the statement's predicate clause that the two intervals do not correspond to the same period (these details are covered in the following Sections).

2.3.3.7 *Predicate Clause*

The predicate clause is the mandatory final component of an EveXL statement. As represented in Figure 2-20, this clause starts with the "`WHERE`" keyword, followed by the predicate that defines the relations between the intervals previously declared in the interval declaration clauses, as well as imposing some other conditions on them.

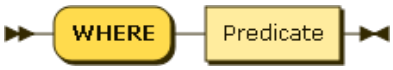


Figure 2-20. The syntax of the predicate clause.

The predicate itself is a combination of other predicates, much like what happens with the data invariants of the interval declaration clause (see Section 2.3.3.6). Its syntax, represented in Figure 2-21, is also similar regarding the use of the logical `AND` and `OR` operators, as well as the use of parenthesis for grouping and changing operator precedence.

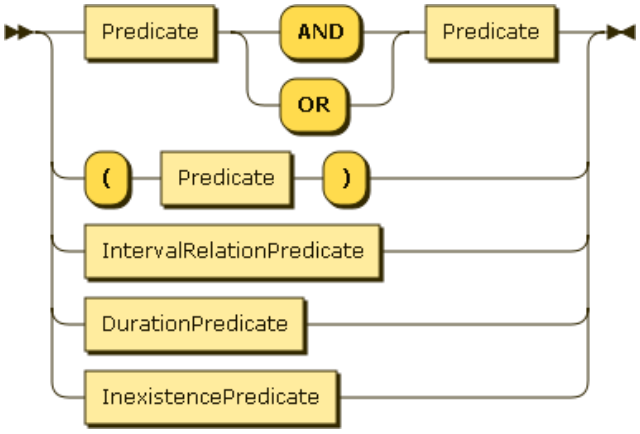


Figure 2-21. The syntax of predicate combinations.

Interval Relation Predicate

The interval relation predicate is used to express a relation between two intervals through the operators of James Allen's Interval Algebra (see Section 2.3.2). The syntax for expressing these relations is represented in Figure 2-22.

Three numbered examples of declarations of relations between intervals can be found in Snippet 5. Because these examples are only intended to illustrate the syntax of relation expressing, the declarations of the intervals have been omitted; likewise, whether they are pure-time or data-invariant is also not relevant here.

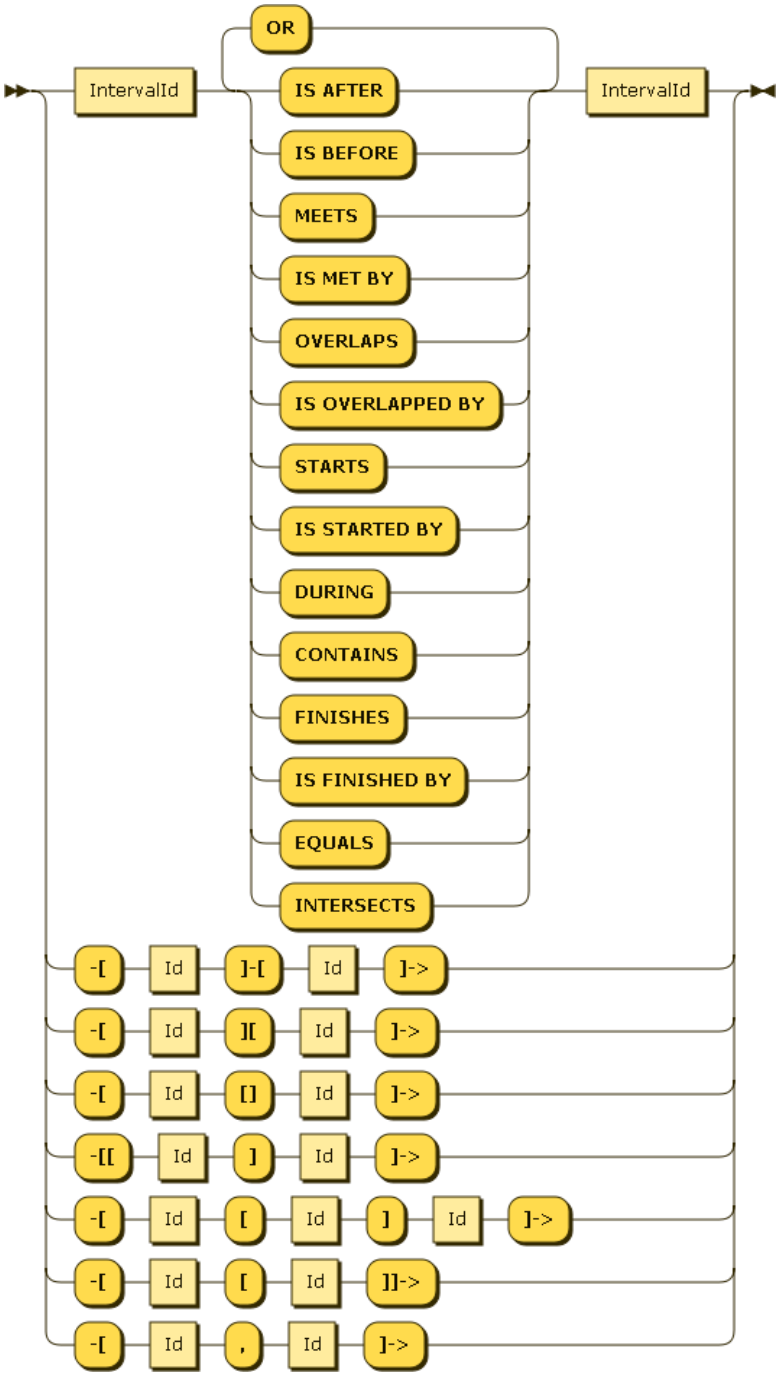


Figure 2-22. The syntax of time interval relations.

```
(1)    [I1] MEETS [I2]
(2)    [I1] MEETS [I2] AND [I2] MEETS [I3]
(3)    [I1] STARTS [I3] OR [I2] FINISHES [I3]
```

Snippet 5. Three examples of the expression of interval relations.

The preceding three examples should be read as "(1) *interval `I1` ends when interval `I2` starts* (the `MEETS` relation); (2) *interval `I1` ends when interval `I2` starts, and interval `I2` ends when interval `I3` starts; and* (3) *interval `I1` starts when interval `I3` starts* (the `STARTS` relation) *and interval `I2` ends when interval `I3` ends* (the `FINISHES` relation)".

Instead of forcing programmers to memorize all of the operators of the Interval Algebra, EveXL offers an alternative notation that resembles visual timeline representations. This topic is further detailed in Section 2.5.1, which describes a study that was designed and conducted to assess the viability of both notations. Nonetheless, for contextualization, a simple example of this notation is depicted in Snippet 6. Each example of the following snippet is equivalent to the one of Snippet 5 having the same number.

```
(1)   -[I1][I2]->
(2)   -[I1][I2]-> AND -[I2][I3]->
(3)   -[I1]I3]-> OR -[I3[I2]->
```

Snippet 6. Alternative notations for the same relations of Snippet 5.

Programmers, however, may prefer to use the textual version of the operators. In this case, since all of the Interval Algebra operators are binary, the expression of a large sequence of alternative relations between a pair of intervals can become quite long. Therefore, EveXL allows programmers to use a condensed notation: just write the first interval followed by a list of relations separated by the `OR` operator and, finally, the second interval. In the condensed notation, the between-relations `OR` operator has a greater precedence than the `AND`, thus the parenthesis are not required. Take the equivalent examples of Snippet 7 and Snippet 8 as illustrations of the simplification brought by this notation.

```
[I1] MEETS [I2] AND (
     [I2] MEETS [I3] OR [I2] OVERLAPS [I3] OR
     [I2] STARTS [I3] OR [I2] FINISHES [I3]
)
```

Snippet 7. The relations between the intervals I1, I2 and I3 expressed as a combination of binary operators.

```
[I1] MEETS [I2] AND
[I2] MEETS OR OVERLAPS OR STARTS OR FINISHES [I3]
```

Snippet 8. The same relations of Snippet 7, expressed in condensed notation.

There is another convenience notation implemented in EveXL that should be mentioned: the `INTERSECTS` operator. In truth, there is no such relation in James Allen's Interval Algebra, but it is just a shorthand format for expressing that two intervals have some time in common. This convenience becomes easy to understand if the equivalent expressions of Snippet 9 and Snippet 10 are contrasted, with the former using only interval algebra's original operators and the latter using the new one.

```
[I1] MEETS OR OVERLAPS OR STARTS OR DURING OR FINISHES OR
EQUALS OR IS MET BY OR IS OVERLAPPED BY OR IS STARTED BY OR
CONTAINS OR IS FINISHED BY [I2]
```

Snippet 9. The notation for expressing that the interval I1 has some time in common with I2.

```
[I1] INTERSECTS [I2]
```

Snippet 10. The INTERSECTS operator as the relation between I1 and I2 is equivalent to the expression in Snippet 9.

Duration Predicate

The duration predicate is used to constrain the duration of an interval. This predicate is useful whenever the duration of an interval is relevant to the description of a given situation, as so often happens in our daily colloquial

speech. For instance, in the scope of public health, research indicates that the time spent watching television is a significant predictor of body mass index (BMI) for young children [46]. In this setting, a hypothetical sensor able to detect television watching could be used in the invariants of an interval with more than 1 hour in duration, to suggest the viewer to take a break (see example 2.3.4.2). On the other hand, it is also convenient to have duration constraints on pure-time intervals. For example, to create a delayed delivery message or an alarm that notifies the user some time after being received (see example 2.3.4.1).

In formal terms, using the notation introduced earlier (see Section 2.3.1) in which an interval `I` is represented by I_α and I_ω – respectively its starting an ending points –, the duration of `I` is simply $I_{dur} = I_\omega - I_\alpha$.

The syntax for using the duration predicate is depicted in Figure 2-23. The interval identifiers are explained in Section 2.3.3.1, *Identifiers*, the numeric comparators are covered in Section 2.3.3.1, *Comparators* and the duration literals in Section 2.3.3.1, *Duration literals*.

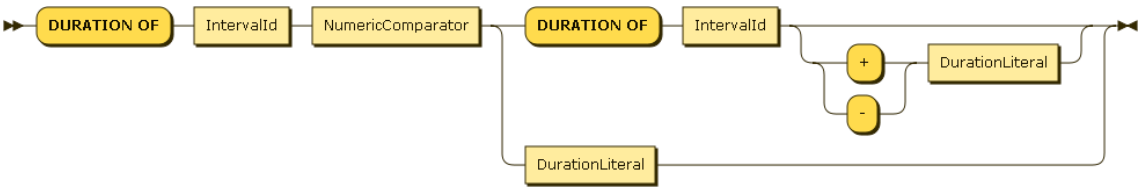


Figure 2-23. The syntax of the duration predicate.

It is possible to compare an interval's duration to a fixed duration time, using a duration literal (see Section 2.3.3.1, *Duration literals*), or to that of another interval, possibly adjusted by some arbitrary duration. There are three examples of these situations in Snippet 11, which should be read as "(1) *the interval `I` has a greater than 2-hour duration,* (2) *the duration of `I1` is the same of `I2`'s and, finally,* (3) *the duration of interval `I1` is greater than or equal to that of `I2`, plus one hour and a half*".

```
(1)  DURATION OF [I] > 2h
(2)  DURATION OF [I1] = DURATION OF [I2]
(3)  DURATION OF [I1] >= DURATION OF [I2] + 1h30m
```

Snippet 11. Examples of duration predicates.

Inexistence Predicate

The inexistence predicate is used to express the inexistence of a given situation in relation to another existing one. From the syntactical point-of-view, it is similar to the SQL "NOT EXISTS" condition, that is, it denies the existence of an EveXL substatement, much in the same way that the NOT EXISTS predicate does with SQL subqueries.

The syntax of the inexistence predicate is represented in Figure 2-24. It is very simple, consisting solely of the `NOT EXISTS` keywords followed by the non-existent substatement enclosed within brackets (for a comprehensive example see Section 2.3.4.7).

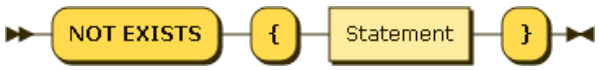


Figure 2-24. The syntax of the inexistence predicate.

2.3.4 Examples of EveXL Statements

This section aims to present some comprehensive examples of EveXL. In each example, an introductory text explains the logic behind the presented solution. But before advancing to the examples, however, some points must be made. First, as the development of sensors is not the goal of this research, in many of the following examples it is assumed that some specific sensors are already implemented in EveWorks and are referenceable through EveXL. Second, other than the solutions presented here, there may be equivalent EveXL expressions that convey the very same reactive behavior and that might, in some way or another, even be superior to the ones presented. However, rather than an exhaustive showcase of EveXL statements, this section is intended to be a introduction to the subtleties of EveXL programming, that is, to the paradigm of time interval programming.

2.3.4.1 *Example 1: trigger when 5 minutes have passed since the time at which the event was received.*

This is a trivial case: since, for an event, time begins counting at the time it is received, only a single pure-time interval is required having a duration greater than or equal to five minutes. The event timeline can be seen in Figure 2-25 and the corresponding EveXL expression in Snippet 12.

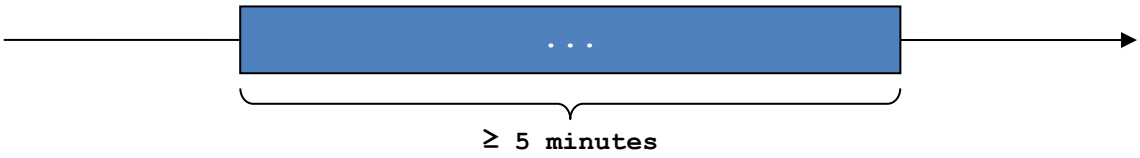


Figure 2-25. A timeline representing a single interval T, in blue, with more than 5 minutes in duration.

```
(1)   [T] AS [...]
(2)   WHERE DURATION OF [T] >= 5m
```

Snippet 12. The EveXL expression defining an interval, T, with more than 5 minutes in duration.

The code should be read as "(1) *T is an interval of time with* (2) *duration greater than or equal to five minutes*".

2.3.4.2 *Example 2: trigger when the user has spent more than 1 hour watching television.*

For this example, assume that there is a sensor "television" that returns either "true" or "false" whether the user is respectively watching television or not. In terms of time intervals, it is necessary to declare an interval whereupon the "television" sensor returns "true", with more than 1 hour in duration. A timeline representing this situation can be found in
Figure 2-26, and a possible EveXL statement in Snippet 13.

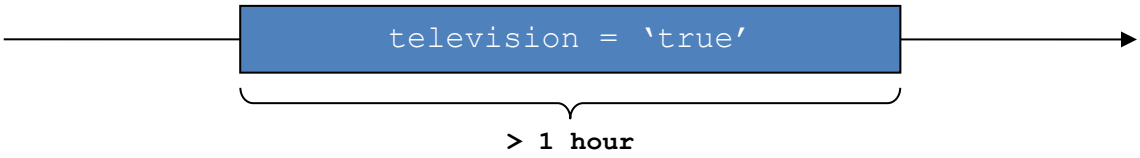


Figure 2-26. A timeline representing a period of more than 1 hour, during which the user is watching television.

```
(1)   [WatchingTV] AS [television = 'true']
(2)   WHERE DURATION OF [WatchingTV] > 1h
```

Snippet 13. Detecting when the user has spent more than 1 hour watching television.

The preceding expression should be read as "(1) *`WatchingTV` is an interval of time during which the sensor 'television' returns the value 'true' and* (2) *the duration of `WatchingTV` is greater than 1 hour*".

2.3.4.3 Example 3: trigger when the user leaves his/her home.

In this example, it is assumed that the "location" sensor exists and that, whenever called, it returns the user's current location, like "home" or "workplace". To formalize this situation, we need two intervals of time such that the first meets the second, i.e., the first ends when the second starts. Additionally, to express that the user has left his/her home, the value returned by the location sensor should be equal to "home" on the first and different on the second. The timeline for this event is represented in Figure 2-27 and its EveXL expression is in Snippet 14.

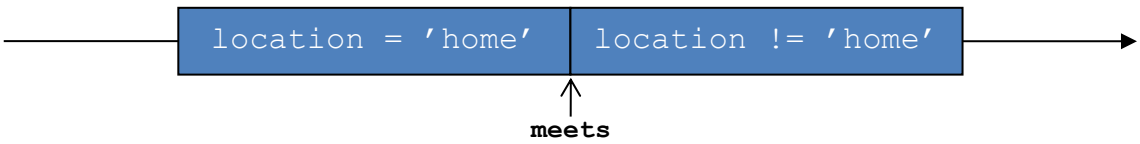


Figure 2-27. A timeline representing the meeting of two intervals.

```
(1)   [IN] AS [location = 'home']
(2)   [OUT] AS [location != 'home']
(3)   WHERE [IN] MEETS [OUT]
```

Snippet 14. Detecting the moment the user leaves home.

The code of Snippet 14 means that "(1) *IN is an interval of time during which the sensor 'location' read the value 'home'*, (2) *OUT is another interval, during which the same sensor read a value different than 'home' and* (3) *the interval IN ends when the interval OUT starts*".

2.3.4.4 *Example 4: trigger when the user leaves his/her home and arrives at his/her workplace.*

Like in the previous example, it is also assumed here that the "location" sensor is already implemented in EveWorks. This situation can be formalized using two intervals of time. In the first, the "location" sensor should read the "home" value whereas in the second it should return "workplace". However, in this example, the two intervals should not be forced to meet. Indeed, there may be a period of time separating the user leaving his/her home and arriving at his/her workplace. The timeline representation for this situation is depicted in Figure 2-28 and the EveXL statement is in Snippet 15

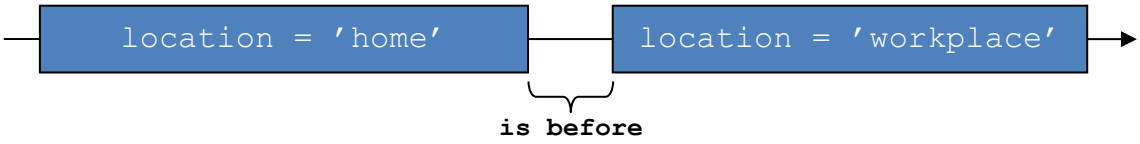


Figure 2-28. A timeline representing two intervals, one occurring before the other.

```
(1)   [atHome] AS [location = 'home']
(2)   [atWork] AS [location = 'workplace']
(3)   WHERE [atHome] IS BEFORE [atWork]
```

Snippet 15. Detecting the moment the user leaves home and arrives at his/her workplace.

The preceding statement should be read as "(1) *atHome is an interval of time during which the sensor 'location' reads the value 'home'*, (2) *atWork is another*

interval, but this time the same sensor reads the value 'workplace' and (3) *the interval* `atHome` *occurs before the interval* `atWork`" .

2.3.4.5 *Example 5: trigger when the user leaves his/her home and arrives at his/her workplace in 30 minutes or less.*

This situation is very similar to the one in the previous example, with the single difference that the user should take no more than 30 minutes to arrive at his/her workplace. It makes sense to use the same intervals of the previous example although, this time, the `IS BEFORE` operator is not enough to express the relation between the two. Indeed, because the operators of the Interval Algebra are, by definition, qualitative (see Section 2.3.2) there is no way to limit the amount of time that separates one interval from the other. This way, the solution is to use a pure-time interval between the two, and limit its duration to 30 minutes or less. The timeline representation of this situation is displayed in Figure 2-29 and its corresponding EveXL expression is in Snippet 16.

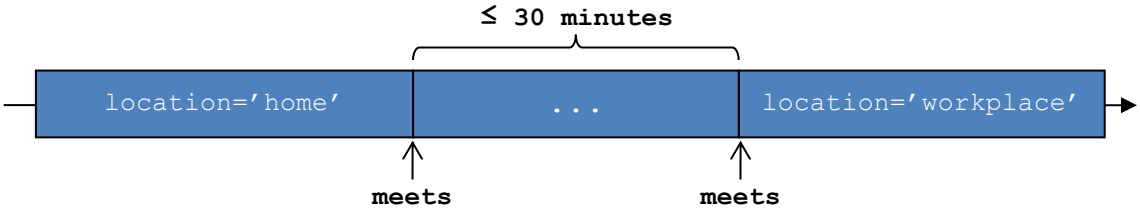


Figure 2-29. A timeline representing two intervals separated by a pure-time interval, with duration equal to 30 minutes or less.

```
(1)  [atHome] AS [location = 'home']
(2)  [onTheWay] AS [...]
(3)  [atWork] AS [location = 'workplace']
(4)  WHERE [atHome] MEETS [onTheWay] AND
(5)        [onTheWay] MEETS [atWork] AND
(6)        DURATION OF [onTheWay] <= 30m
```

Snippet 16. Detecting the moment when the user arrives at his/her workplace, after leaving home.

The preceding code means that "(1) `atHome` *is an interval of time during which the location sensor read the value 'home'*, (2) `onTheWay` *is just an interval of time and* (3) `atWork`*, is another interval whereupon the location sensor read the value 'workplace'*; (4) *the interval* `atHome` *meets* `onTheWay`, (5) `onTheWay` *meets* `atWork` *and, finally,* (6) *the duration of the* `onTheWay` *interval is less than or equal to 30 minutes*".

2.3.4.6 *Example 6: trigger when the user has been at the cafeteria tree times during the last hour.*

Using the same "location" sensor as in the previous examples, three intervals are necessary, with a returned location value of "cafeteria". Also, an interval of pure-time may be used to convey the "last hour" period that should be intercepted by the previous three, in-cafeteria intervals. The timeline representation of this situation is in Figure 2-30.

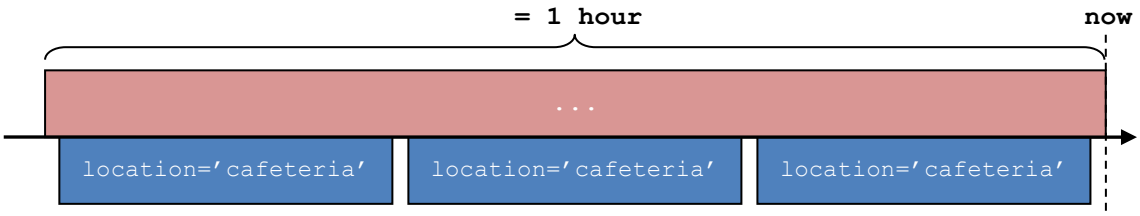


Figure 2-30. A timeline representing two intervals separated by another, pure-time one with duration under 30 minutes or less.

Special care should be taken when formalizing this situation in EveXL since, by default, EveWorks only keeps record of the last interval in which its invariants have held (see Section 2.3.3.6, *Modifiers*). Therefore, it is necessary to make use of the `LAST n` modifier, to tell EveWorks to keep in memory information about the last `n` intervals. Find the corresponding EveXL statement in the following snippet (Snippet 17).

```
(1)  [WithinLastHour] AS [...], NOW
(2)  [AtCafeteria1] AS [location = 'cafeteria'], LAST 3
(3)  [AtCafeteria2] AS [location = 'cafeteria'], LAST 2
(4)  [AtCafeteria3] AS [location = 'cafeteria'], LAST 1
     WHERE
(5)  DURATION OF [WithinLastHour] <= 1h AND
(6)  [AtCafeteria1] STARTS [WithinLastHour] AND
(7)  [AtCafeteria2] DURING [WithinLastHour] AND
(8)  [AtCafeteria3] FINISHES [WithinLastHour]
```

Snippet 17. Detecting if three intervals with location set to 'cafeteria' have, respectively, started, occurred-during and finished another, shorter than 1-hour, interval.

The preceding statement means that "(1) *`WithinLastHour` is a simple interval of time,* (2-4) *`AtCafeteria1`, `AtCafeteria2` and `AtCafeteria3` are three intervals whereupon the location sensor returned the 'cafeteria' value. The* (5) *`WithinLastHour` interval lasts 1 hour or less and* (6) *it starts when the `Cafeteria1` interval does and ends when `AtCafeteria3` ends; also, the `AtCafeteria2` interval takes place within the `WithinLastHour` interval*".

2.3.4.7 *Example 7: trigger when the user leaves the workplace and enters home without having been to the gymnasium.*

For this example, assume we have the same "location" sensor as in the previous examples and that it returns the values '"workplace", "gymnasium" and "home" if the user is in those places. From the situation description, two intervals are necessary: one during which the "location" sensor returns the value "workplace" and another one, whereupon the same sensor returns the value "home". Additionally, between the two, there must not exist any interval having the "location" sensor returning the value "gymnasium". The EveXL statement that conveys as much can be found in Snippet 18.

```
(1)   [work] AS [location = 'workplace']
(2)   [home] AS [location = 'home']
      WHERE
(3)   [work] IS BEFORE [home] AND
(4)   NOT EXISTS {
(5)       [gym] AS [location = 'gymnasium']
          WHERE
(6)       [work] MEETS OR IS BEFORE [gym] AND
(7)       [gym] MEETS OR IS BEFORE [home]
      }
```

Snippet 18. Defining two intervals of time without a third one in between.

The preceding snippet should be read as "(1) *work is an interval of time during which the 'location' sensor reads the value 'workplace'*, (2) *home is another interval whereupon the same sensor read the value 'home' and* (3) *the interval work is before home. Also,* (4) *there is no* (5) *interval (named gym) such that the 'location' sensor reads the value 'gymnasium' and* (6-7) *occurring between work and home*".

2.3.4.8 *Example 8: trigger when the user sleeps less than 8 hours*

This is a deceptively simple example that highlights some of the subtleties of programming through intervals of time. For simplicity, it is assumed that a "sleep" sensor is implemented in EveWorks, that returns either "true" or "false" whether the user is respectively sleeping or awake.

At first sight, to formalize this situation with such a sensor, one would merely have to declare one interval of time during which the sleep sensor returns "true", with less than 8 hours in duration. For additional accuracy, one could also constrain the time of the said interval to normal sleeping hours, say, 22:00 to 9:00. The statement for this approach can be seen in Snippet 19.

```
[asleep] AS [sleep = 'true' AND time BETWEEN 22:00 AND 9:00]
WHERE DURATION OF [asleep] < 8h
```

Snippet 19. Detect a sleeping period of less than 8 hours (first attempt).

However, this approach would fail. Indeed, as explained in Section 2.2.2, EveWorks builds time intervals cyclically, as long as its data-invariants keep holding (or, in the case of pure-time intervals, as time elapses). In other words, for every cycle in which a given interval's invariants hold for that cycle's sensor data reading, it will have its ending time updated to the cycle's time. Consequentially, an interval in which the user is asleep, with less than 8 hours in duration, will be occurring as soon as the user falls asleep, and this event will be triggered by then.

To correct this approach, steps must be taken in order to make sure that the sleeping interval has ended. This can be easily done with a simple pure-time interval that is met by the sleeping interval, with a significant duration. That way, EveWorks is forced to wait for some meaningful time before deeming that the user has stopped sleeping. Snippet 20 shows the statement for this new approach.

```
[asleep] AS [sleep = 'true' AND time BETWEEN 22:00 AND 9:00]
[wait] AS [...]
WHERE
DURATION OF [asleep] < 8h AND
DURATION OF [wait] >= 30m AND
[wait] IS MET BY [asleep]
```

Snippet 20. Detect a sleeping period of less than 8 hours (second attempt).

Once again, this statement is also not correct. Granted, it will trigger upon the end of an interval of sleep with less than 8 hours, but there is nothing in this formalization that will prevent the event from triggering if the user has slept for more than one interval. For instance, imagine that the user sleeps for more than 8 hours, then awakes and, after less than 30 minutes, falls asleep once again for less than 8 hours. The event will still trigger because, as far as the statement of Snippet 20 expresses, EveWorks will be looking for an interval with less than 8 hours, while it should be looking for the inexistence of an 8-hour minimum interval within the normal sleeping hours.

As far as experiments have revealed, these subtleties are a common pitfall for starting level EveXL programmers. Indeed, EveXL requires unambiguous descriptions of temporal situations and, as James Allen [4] has already noticed, there is a lot of ambiguity and incompleteness in people's discourse regarding their temporal knowledge. For instance, when saying something as simple as "*I haven't slept more than 8 hours*", there is a quantity of information that is being omitted – or, even better, that is being transmitted *implicitly*. A less ambiguous statement would perhaps be "*I haven't slept more than 8 hours in a row, between yesterday's 10pm* (22:00) *and today's 9am* (9:00)". Following this logic, the final – and correct – solution can be found in Snippet 21.

```
(1) [SleepHours] AS [time BETWEEN 22:00 AND 9:00]
    WHERE
(2) DURATION OF [SleepHours] = 11h AND
(3) NOT EXISTS {
(4)     [UserSleep] AS [sleep = 'true']
        WHERE
(5)     DURATION OF [UserSleep] >= 8h
(6)     [UserSleep] STARTS OR DURING OR FINISHES [SleepHours]
    }
```

Snippet 21. Detecting a sleep period of less than 8 hours (correct attempt).

The preceding statement should be interpreted as "(1) `SleepHours` *is an interval of time between 22:00 and 9:00 with* (2) *duration equal to 11 hours. Furthermore,* (3-4) *there is no interval (called* `UserSleep`*) during which the sensor 'sleep' sensor returns 'true', with* (5) *duration greater than or equal to 8 hours that* (6) *occurs while the* `SleepHours` *interval is in progress*".

Some final notes on this example: although the invariants of the `SleepHours` interval clearly define it as a period of time happening between the 22:00 hours and the following 9:00 (Snippet 21, line (1)), they do not state that this interval is *that* precise period of time. In fact, its invariants only dictate that the `SleepHours` interval will be a composition of readings that take place between 22:00 and the following 9:00. Hence, the duration predicate is

necessary (Snippet 21, line (2)). Finally, the `DURING` operator is not enough because, due to the *distinct* property of the Interval Algebra operators (see Section 2.3.2), it does not include the cases in which the `UserSleep` interval starts or finishes at the same time the `SleepHours` interval does. Therefore, to take these cases into account, both the `STARTS` and the `FINISHES` operators are required.

2.3.4.9 Participant Suggested Examples

The following three examples result from asking study participants and colleagues for examples of situations in which they feel it would make sense to trigger an interaction, regardless of the nature of that interaction (to receive a notification, a message, an alarm, and so on). Naturally, since the majority of the participants were not focused on the challenges of context-awareness and event detection research, they generally contextualized the events within the scope of short narratives describing real life situations in which specific interactions would be helpful. In this sense, the following examples differ from the ones presented before in that the formalisms must be identified and extracted from those stories.

Although participants were prolific in the amount of suggestions they made, the main distinction between those was related to the narrative and the nature of the triggered interaction, rather than the event itself. For example, although different from the narrative standpoint, the following stories are equivalent in terms of formalism: "*receive a message upon arriving at the mall, to call home and confirm the groceries shopping list*" and "*receive a warning when arriving home from the swimming pool to put wet clothes in the washing machine*". In this sense, in order to provide a broader illustration of potential use cases, the three presented examples were the most distinct from the other suggestions and also from the other examples presented before.

1) For health laboratories with controlled room access, notify the user to perform disinfection after spending some time in non-sterile environments.

The first step is to analyze the situation description in order to understand what is the event to detect, i.e., the *kairos* of the interaction. In this particular case, there is a distinction to make between the interaction's context (*health laboratories with controlled room access*), the form it will take (*a notification*), its contents and semantics (*to perform disinfection*) and the circumstances of its delivery (*after spending some time in non-sterile environments*). Because EveWorks has been designed to operationalize *kairos*, the EveXL expression must model the last of these interaction components, the delivery of the message.

This situation may be modeled using only one interval of time, representing the eventual period during which the user has been in the "non-sterile" environment. The length of this period, likely a parameter in a real life situation, is arbitrarily defined here as "5 minutes". It is assumed here the existence of a "location" and "sterile" sensors that, respectively, return the user's location and the sterility conditions of the place he/she currently is in (see Snippet 22).

```
[NonSterile] AS [location='workplace' AND sterile='false']
WHERE
DURATION OF [NonSterile] >= 5m
```

Snippet 22. Detecting a stay in a "non-sterile" environment at the workplace, for 5 minutes or more.

2) For molecular biology laboratories with controlled room access and unidirectional workflow (Room1 to Room2) to prevent potential contamination, send a warning to prevent entering "Room1" (clean) after being in "Room2" (potential source of contamination), in the same working day.

Once again, the event to formalize must be extracted out of the situation's description. The interaction's context (*molecular biology laboratories with controlled room access and unidirectional workflow*), its nature (*a warning*) and purpose (*to prevent potential contamination*) are of no relevance to EveWorks. Therefore, the

EveXL expression should only be concerned with the expression of the circumstances of delivery (*entering "Room1" after being in "Room2"*).

This situation may be modeled using two intervals of time, one representing the period spent by the user in "Room2", and another for the time spent at the entrance of the secured access "Room1". The "working day" – say, a time period spanning from 9:00 to 18:00 –, may be modeled through the statement's activity clause. It is assumed the existence of a "location" sensor that returns the user's location (see Snippet 23).

```
ACTIVE BETWEEN 9:00 AND 18:00
[AtRoom2] AS [location = 'Room2']
[AtRoom1Entrance] AS [location = 'Room1Entrance']
WHERE
[AtRoom2] IS BEFORE OR MEETS [AtRoom1Entrance]
```

Snippet 23. Detecting the moment the user arrives at the "Room1"'s entrance, after being in "Room2".

3) Deliver a message to a third party receiver (e.g., a family member), whenever the user leaves his/her workplace and is walking (thereby available to talk on the phone).

As in the previous examples, this interaction description contains more than just the event to detect. Truly, it mentions the interaction's context (*the user is walking out of his/her workplace and available to receive a phone call*), its form (*a message*), the receiver of the interaction (*a third party*) and the *kairos* of that interaction (*when the user leaves his/her workplace*). Although EveWorks is only concerned with the last of these components, it is still interesting to note that, in this case the receiver of the interaction is not the user, but a third party interested in the situation's occurrence. The event detection, however, takes place on the user's phone, so who receives the interaction is largely a matter of application design and of no interest to EveWorks.

Therefore, this situation may be modeled with two intervals: one for the time the user spends on his/her workplace and another for when he/she is

walking and no longer there. It is assumed the existence of the "location" sensor, as previously defined, and also the existence of an "activity" sensor that returns the user's current activity (like the Android framework's default activity recognition sensor) (see Snippet 24).

```
[Working] AS [location='workplace']
[OutOfWork] AS [location!='workplace' AND activity='walk']
WHERE
[Working] IS BEFORE OR MEETS [OutOfWork]
```

Snippet 24. Detecting when the user is walking, after leaving his/her workplace.

2.3.5 Interpretation

EveWorks models EveXL event-defining statements as Constraint Satisfaction Problems (CSP), i.e., formalizations composed of a set of variables, the domains of those variables (the range of values they can assume) and a set of constraints that restrict the values that variable subsets can simultaneously assume [80]. To perform the necessary computations involved in solving these problems, EveWorks embeds and interfaces with the JaCoP CSP solver [88].

In more detail, EveWorks allocates two finite domain variables for each interval used in a given expression, one for each of the interval's bounding time points. The domains of these variables are composed of the starting and ending times of the periods during which the data invariants are held. Finally, the model's constraints are direct translations of the relations that were declared in the EveXL expression's WHERE clause (see the 4th column of Table 2-2).

While the set of variables and constraints are static for each event model, the variable domains have a dynamic nature, as they are updated each cycle after time sensors are read.

For instance, imagine that an application, designed to deliver a notification to a user whenever he/she leaves home, registers an event in EveWorks. A possible EveXL expression for this event can be found in Snippet 25 and its respective CSP formalization in Snippet 26.

```
[in] AS [location = 'home']
[out] AS [location != 'home']
WHERE
[in] MEETS [out]
```

Snippet 25. An EveXL expression for the "leaving home" event.

```
Variables:   { IN_α, IN_ω, OUT_α, OUT_ω }
Domains:     { D(IN_α), D(IN_ω), D(OUT_α), D(OUT_ω) }
Constraints: { IN_α < IN_ω & OUT_α < OUT_ω & IN_ω = OUT_α }
```

Snippet 26. The CSP model for the event of Snippet 25.

To keep this example straightforward, assume a simplistic scenario: EveWorks is set to execute its routine every 10 minutes and has begun recording the user's location at 8:40am. At that time, the user is at home and leaves it sometime between 9:00am and 9:10am.

The variable domain updates for this CSP model, for the 8:40-9:10 period, can be seen in Table 2-3.

Table 2-3. The changing domains of 4 different variables, in sequential readings.

Time	Location	$D(IN_\alpha)$	$D(IN_\omega)$	$D(OUT_\alpha)$	$D(OUT_\omega)$
8:40	home	{8:30}	{8:40}	{}	{}
8:50	home	{8:30}	{8:50}	{}	{}
9:00	home	{8:30}	{9:00}	{}	{}
9:10	street	{8:30}	{9:00}	{9:00}	{9:10}

By the time of 9:10am, upon detecting a location different than '*home*', EveWorks updates the domain of the `OUT` interval's starting and ending variables, OUT_α and OUT_ω, setting them to 9:00am and 9:10am, respectively. Therefore, at 9:10am, all of the model's constraints are now satisfied by the new domains, i.e.:

IN_{α}:{8:30} < IN_{ω}:{9:00}

OUT_{α}:{9:00} < OUT_{ω}:{9:10}

IN_{ω}:{9:00} = OUT_{α}:{9:00}

At this time, EveWorks detects the event occurrence and finally notifies its registering application. It should be noted that although EveWorks had only begun running at 8:40, the starting time of interval IN, represented by the IN_{α} variable, had been set to 10 minutes earlier, i.e., 8:30. As explained in Section 2.3.1, this is because of the interval building strategy: when an interval's data invariants are held at the time of a given reading, the same invariants are assumed to be held since the time of its preceding reading. Consequentially, in spite of the fact that, in our example, there are no readings before 8:40am, EveWorks will "artificially" assume that the IN interval spans from 8:30am to 8:40am.

2.4 Related Work

2.4.1 Frameworks for Context-Awareness

Unlike EveWorks, most of the frameworks for context-awareness found in the literature are not specifically targeted to resource-constrained platforms, like smartphones. For instance, Dey and Abowd [84] have introduced the Context Toolkit, a framework designed for rapid development of context-aware applications. The Context Toolkit is composed of five different functional abstractions, each with its own specific purpose: *context widgets* retrieve context information, thereby insulating applications from context acquisition concerns; *interpreters* produce additional levels of abstraction for context information – for instance, location may be expressed at low level of abstraction, through geographical coordinates, or at higher levels, through street names; *aggregators* combine related contextual information into a common repository; *services* execute actions on behalf of applications or, more specifically, they control or change state information on the environment; and, finally, *discoverers*, responsible for maintaining a registry of what capabilities exist in the framework, which is useful for distributed context-aware systems. The Context

Toolkit is mainly focused on the problem of providing functional abstractions to ease the task of context gathering.

Devaraju et al. [31] have proposed the Context Aware Service Platform (CASP), a middleware approach to assist context-aware service providers in building and deploying services. The framework collects raw data from the physical sensors via sensor abstraction programs written by the sensor providers, represents it in an internal format and, therefore, transmits the translated data to the CASP platform for context modelling and reasoning. CASP is designed following a client-server architecture, which centralizes context reasoning development tasks, but forces systems to depend upon network communication and, as consequence, one has to consider network performance, protocols and, their availability, which cannot be taken for granted in mobile device work environments. The framework is mainly targeted for context acquisition and modelling tasks, relying to this end on ontologies to model the gathered information and facilitate reasoning on the server.

Acknowledging the privileged relationship between smartphones and their owners [32], the research community has already proposed some interesting approaches targeting these platforms. For instance, Wissen et al. [92] have developed ContextDroid, an Android-targeted, expression-based framework for context awareness. ContextDroid models context through the following constructs: *context entities*, which are collections of contextual information like the user's location in terms of latitude, longitude and altitude; *context readings*, each composed of one value representing the state of the entity, a timestamp and the time of expiration; *context conditions*, which perform boolean evaluations using context readings and sets of parameters; *evaluators* which are simple interfaces for the evaluation of conditions; and, finally, *context expressions* that allow new context to be produced by combining other contexts. Although ContextDroid is based in expressions, much like EveWorks, its approach to expression building is tightly coupled to the Android's Java-based technological environment. Indeed, ContextDroid interfaces with the embedding applications through an API, with expressions being defined through the instantiation of classes and the invocation of predefined methods

on the resulting objects. Because this logic must be implemented in the application's source code, the expressions will be compiled and, thereby, have to be defined at compile time. As a consequence, it becomes necessary to manipulate the application's sources if its reactive behavior is to be changed.

Kramer et al. [51] have presented a layered, rule-driven, generic context acquisition engine that, similarly to ContextDroid, also targets the Android platform. This engine was developed to provide a single context acquisition mechanism, to potentiate efficient use of resources and avoid monitoring the same context changes from multiple points. Like EveWorks, the engine runs as a standalone instance capable of context capturing and composition. However, while EveWorks has been designed to communicate detected events solely to the application that has submitted the event, Kramer et al.'s engine aims at broadcasting these occurrences to any listening context-aware applications running on the same device. Internally, the engine's architecture is based on the idea of self-contained context components and is hierarchically organized as a tree. In this organization, lower-level, simpler context components can be loosely coupled to form higher level, composite context representations. The engine's concept is composed of three constructs: the *context component*, a self-contained implementation which deals with the tasks of acquiring raw context data from context sources, mapping it into a finite set of predefined values and, in case the new value differs from the last, broadcasting the processed information; the *composite component*, consisting of combinations of loosely coupled context components, whose main role is the runtime handling of aggregated, higher-level context information; and, finally, the *context engine manager*, which deals with the distribution of context information to any listening applications. According to the authors, the engine can be used as an external service by applications through direct calls to its interface specification. Contrasting with EveWorks, which interfaces with applications through expressions interpreted at runtime, the application code that interacts with the engine service will have to be compiled along with the remainder of the application's sources, implying that the code implementing its reactive behavior will be defined at compile time.

Although not directly turned towards context-awareness, Esper [98] is another interesting example of related work that is definitely worth mentioning here. It is an industrial software product of EsperTech Inc and, as stated in the EsperTech's website, Esper is "*a component for complex event processing and event series analysis, available for Java as Esper, and for .NET as NEsper*". Esper has been designed for correlating high-volumes of events, in situations where the occurrence of up to millions of events prevents traditional database architectures from performing well. In a clear demarcation from these, the creators of Esper propose it to be thought of as a "*database turned upside down: instead of storing the data and running queries against stored data, Esper allows applications to store queries and run the data through*". To this end Esper uses the concept of running query: continuous execution rather than upon the submission of queries. Given its orientation, and since all of Esper's computing is in-memory, high-end systems are favoured. This is a clear contrast to EveWorks, which has been designed for running in portable devices with constrained resources, and running periodically to allow memory to be reused for other system needs. Also, instead of focusing on the processing of large quantities of events per second, EveWorks offers an integrated approach, dealing with sensor reading, storing of sensor data and the detection of events.

As a last example, Ferreira's AWARE framework [35] is an instrumentation middleware, aiming to streamline the effort of developing mobile logging tools. AWARE is architecturally distributed, having two main components: the AWARE client and the AWARE server. While the former is mainly a context data gatherer, the latter is a data storage and processing unit, designed to share and reuse context data with other applications and devices. Data collected by the AWARE client can be stored locally, on the device's storage, or be sent to the AWARE server for remote storage and processing. The AWARE client gathers data through AWARE sensors that collect and abstract raw data from the Android platform's sensors or events from other AWARE sensors, thereby creating composite, higher-level contexts. The AWARE client can be extended with new AWARE sensors which, in practical terms, are subclasses of the Android Service class. As previously stated, the AWARE framework is mainly focused on the development of logging tools, whereas EveWorks is focused on the detection of events in people's daily lives.

2.4.2 Languages for Event Detection

The previously mentioned Esper engine (see the previous Section 2.4.1) offers an interface to other applications through its own DSL, the Event Processing Language (EPL). EPL is a declarative programming language for dealing with high frequency time-based event data. In parallel with Esper, which has drawn inspiration from classical RDBMS, so does the EPL have a lot in common with SQL, being syntactically similar to it in the use of the SELECT and the WHERE clauses. However, instead of data stored in data tables, EPL manages event streams and views, with the former corresponding to infinite sequences of events while the latter define the data that is available for querying and filtering. EPL also supports temporal operations through its single "followed-by" operator (->). In a contrast to EveXL, which assumes an interval-based model of time, EPL does not enforce a model of how time flows. Indeed, while EveXL has been designed to express daily life events and, therefore, places time intervals as its language primitives, EPL uses the high-level concept of "event" as a construct devoid of intrinsic temporal dimensions. Additionally, while both the EveXL's conceptualization and syntax have been inspired on an everyday concept of time (that of time interval), those of EPL are data- and object- oriented, as in Object Oriented Programming.

Cohen and Kalleberg's EventScript [22] is another example of a language for the definition of reactive processes. However, instead of being data-oriented, like EPL, EventScript's syntax draws inspiration from regular expressions. Indeed, a stream of incoming events is matched against regular expressions with actions embedded, like the assignment of computed values to variables and the emission of new output events. The heart of the runtime engine is a tight loop that uses the current state and the next input event to index into a Deterministic Finite Automaton transition table to find the actions to be executed and the new current state. Like what happens with Esper's EPL, EventScript does not enforce a model for temporality, instead having events expressed as patterns of other events, rather than temporal changes in context. One of the motivations for the choice of regular expressions as EventScript's main syntax design influence is its familiarity to programmers. However, in spite of offering a concise and effective expression of patterns, regular

expressions are often criticized for being cryptic [39]. Since EventScript's syntax and concept are, in essence, regular expressions enriched with resources for building reactive behaviors, its legibility suffers from the same cryptic syntax problem.

Drools Fusion [96] is the module of the Drools Business Logic Integration Platform [97] dedicated to event processing and temporal reasoning. It has events as first-class citizens, i.e., understands them as special records of changes in state in an application's domain, with some distinguishing characteristics, namely being usually immutable and having strong constraints and relationships. Interestingly enough, Drools Fusion implements the temporal operators of James Allen's Interval Algebra – the same as EveXL –, although it has some operator names changed: *after*, *before*, *coincides*, *during*, *finishes*, *finished by*, *includes*, *meets*, *met by*, *overlaps*, *overlapped by*, *starts* and *started by*. This imparts Drools Fusion with a strong and consistent model of temporality. However, while EveXL uses temporality (time intervals) as its main conceptual structure, in Drools Fusion expressions time appears almost incidentally, as a dimension of events. Some of the operators themselves have been "augmented" with mandatory and optional parameters, to account for temporal distances, thereby imposing constraints on their semantics. Syntactically, Drools Fusion expressions are written in a data-oriented DSL "expansion" of the Drools (which is a rule management system) native rule language.

The issue of temporality has been addressed before in contexts other than languages for reactive behavior programming. For instance, in the design of advanced user interfaces, Guimarães et al. [41] proposed Ttoolkit, an extension of an existing graphical user interface toolkit (Xt toolkit). Ttoolkit complements the traditional two-dimensional coordinate space model used for graphical user interfaces with a model of time. Among the many attributes considered important for its temporal model are the temporal relations among events. In the context of the Ttoolkit, an event is defined as a temporal primitive, having a duration, a start point and an end point – in many regards, it corresponds to EveXL's definition of time interval. The relations between events also correspond to the thirteen operators of James Allen's Interval Algebra that have been implemented in EveXL.

2.5 Studies

Evidently, both the development of context-aware frameworks as well as the creation of DSLs for expressing reactive behaviors are very challenging endeavours. Besides adequate design, and because both are ultimately tools for software development, careful evaluations are required in order to understand if these tools are adequate and efficient resources. In the particular case of this thesis' work, since EveXL is the language that EveWorks "understands", the evaluation of the latter is always intertwined with the evaluation of the former.

Still, the evaluation of programming languages is not a settled matter in the research community. Acknowledging this, Sebesta [87] has proposed three general criteria for language evaluation: *readability*, *writability* and *reliability*. These criteria are affected by some characteristics, with the latter progressively adding new ones to the former's list, i.e., reliable languages contain all of the characteristics of writable ones, and writable languages all those of readable ones (see the following Table 2-4). While these criteria are intended to provide a consistent way of evaluating and comparing general-purpose programming languages, some of them may also be used as a solid base to evaluate languages with restricted domains, like DSLs – and EveXL, consequently.

Table 2-4. Language evaluation criteria and the characteristics that affect them (table in [87]**).**

	CRITERIA		
Characteristic	**READABILITY**	**WRITABILITY**	**RELIABILITY**
Simplicity	•	•	•
Orthogonality	•	•	•
Data types	•	•	•
Syntax design	•	•	•
Support for abstraction		•	•
Expressivity		•	•
Type checking			•
Exception handling			•
Restricted aliasing			•

Therefore, the characteristics that are more relevant to our work are: *simplicity*, which refers to a language's overall simplicity and is afforded by, among other things, reduced numbers of basic constructs, reduced feature multiplicity (different ways of performing the same operation) and no operator overloading (i.e. no more than one meaning for a single operator symbol); *orthogonality*, a concept meaning that a language should have independent features and concepts and allow them to be combined in meaningful ways; *syntax design*, referring to the overall quality of a language's syntax, i.e., how explicitly written statements convey their meaning; *expressivity*, meaning that a language has relatively convenient (rather than cumbersome), ways of specifying computations; and, finally, *type checking*, which refers to the detection of type errors either at run or compile time.

According to the preceding definitions, some of these characteristics require no exhaustive evaluation, as they mostly derive from design decisions. For instance, regarding simplicity, EveXL has only one basic construct – intervals of time – and no operator overloading, as it does not allow programmers to redefine the meaning of operators. Moreover, since EveXL operators are all mutually exclusive, feature multiplicity and orthogonality are, respectively, restricted and reinforced.

Nonetheless, there are a lot of dimensions that can only be assessed through user studies. Indeed, because – once again – programming languages are tools, there is no real reason why concepts usually applied in the domain of HCI user experience evaluations should not be applied here. Indeed, although Sebesta may have chosen adequate names for his set of programming language characteristics, the truth is that concepts like "simplicity" or "expressivity" convey much more than the sum of a number of discrete features that a language may, or may not have. They are, ultimately, the results of the judgement of programmers about their daily work tools. Therefore, three user studies have been conducted, each aiming to evaluate particular aspects of the EveWorks-EveXL dyad.

2.5.1 Study 1, EveXL's Alternative Notation

EveXL features an alternative notation for its operators that was inspired by one of the most ubiquitous representations of temporality, the timeline. Because the Interval Algebra has thirteen different operators, remembering and bearing their meaning in mind when writing expressions may be somewhat problematic. As simplicity is one of EveXL's main development guidelines, an alternative syntax was developed that, given its visual resemblance to the timeline (somewhat reminiscing of ASCII art), may be simpler to remember and use (see Table 2-5). For example, the expressions found in Snippet 27 and Snippet 28 are equivalent.

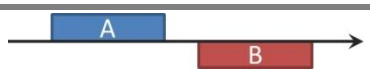
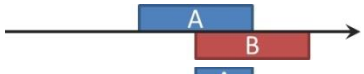
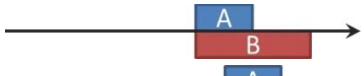
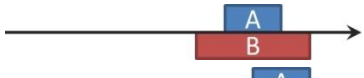
```
[in]  AS [location = 'home']
[out] AS [location != 'home']
WHERE [in] MEETS [out]
```

Snippet 27. Defining two intervals of time without a third one in between, using the operators' textual notation.

```
[in]  AS [location = 'home']
[out] AS [location != 'home']
WHERE -[in][out]->
```

Snippet 28. Defining two intervals of time without a third one in between, using the operators' alternative notation.

Table 2-5. The alternative notation for the operators of James Allen's Interval Algebra.

Operator	Timeline Illustration	Alternative
A IS BEFORE B		-[A]-[B]->
A MEETS B		-[A][B]->
A OVERLAPS B		-[A[]B]->
A STARTS B		-[[A]B]->
B CONTAINS A		-[B[A]B]->
A FINISHES B		-[B[A]]->
A EQUALS B		-[A,B]->

As previously mentioned in Section 2.3, because programming languages are tools, careful evaluations are required in order to understand if they are adequate and efficient resources. This study [15,16] was therefore conducted as a preliminary evaluation of both EveXL as a whole, and its alternative operator notation. Additionally, a field evaluation of the framework's performance was also performed, as detailed below.

2.5.1.1 Methods and Participants

Participants were asked to answer a set of questions on a 9-item online questionnaire:

- Items 1 to 3 were multiple choice questions that, for provided situation descriptions, asked participants to find the correct DSL statement among the presented 4. These item's descriptions were "user left home", "having been at home, and, afterwards, at the supermarket, the user enters work" and "having been at work the user enters a restaurant or home", respectively for items 1, 2 and 3.

- Item 4 presented a more evolved DSL statement and asked participants to explain it in their own words. The statement can be found in Snippet 29.

- Item 5 presented the textual description of an event and asked participants to compose a full statement using the DSL (the test site contained a parser which allowed counting the amount of syntactic errors). The description was "Write the expression for the following event: leaving the room and entering the kitchen".

- Item 6 asked participants to link each timeline textual representation to its meaning (like connecting the shuffled cells of columns "Operator" and "Alternative" Table 2-5);

```
[A] AS [location='home' OR location='work' OR location='mall']
[B] AS [location='pub']
WHERE -[A]-[B]->
```

Snippet 29. The statement that participants were asked to describe using their own words.

Finally, participants were asked to evaluate both the general DSL statements and the textual representation of the operators on multidimensional Semantic Differential Scales (SDS) (see [66] for more information on SDS's).

In order to perform a preliminary field evaluation of the framework's performance, a test application was developed that simultaneously submitted 20 different events to EveWorks, with varying levels of complexity. The smartphone running the test application – a Sony Ericsson Xperia Arc S (LT18i) – was carried by a user across the faculty campus while rehearsing the behaviors that would satisfy the events' triggering conditions. Two of EveWorks default sensors were used in this field test: time and location (more details on Section 2.2.3).

The study had 84 participants (73 male), recruited among undergraduate students of Computer Engineering, with ages ranging from 20 to 59 (averaging 25.6), and the number of years of programming experience ranged from none to 25, averaging 6.9.

2.5.1.2 Results

The results of the 6-item questionnaire suggest that the timeline is a good inspiration for EveXL: item 1 had 76 correct answers (90.5%); item 2 had 84 correct answers (100%); item 3 had 76 correct answers (90.5%); item 4 had 82 correct interpretations (97.6%); item 5 had 71 participants writing the EveXL expression at first try with no syntactic errors (84.5%), with the remaining 13 users averaging 1.7 errors (15.5%); and, finally, in item 6, 53 participants got a fully correct matching (63,1%); 20 mismatched 2 representations (23.8%), 6 participants had 3 mismatches (7.1%) and 4 had more than 3 (4.8%). Regarding the SDS evaluations, results can be seen in Figure 2-31 and Figure 2-32.

Regarding the field evaluation, most of the alarms were raised when expected. Indeed, of the 20 programmed events, 18 triggered consistently at the right moment, while the remaining two occasionally triggered when not supposed to (false positives). Rather than a problem in the concept of EveXL-EveWorks, this behavior turned out to be a consequence of the challenge that is to distinguish between adjacent indoor locations solely through Wi-Fi readings (although out of the scope of this work, it should be said that Wi-Fi signals are subject to noise/strength fluctuations that increase overall inaccuracy).

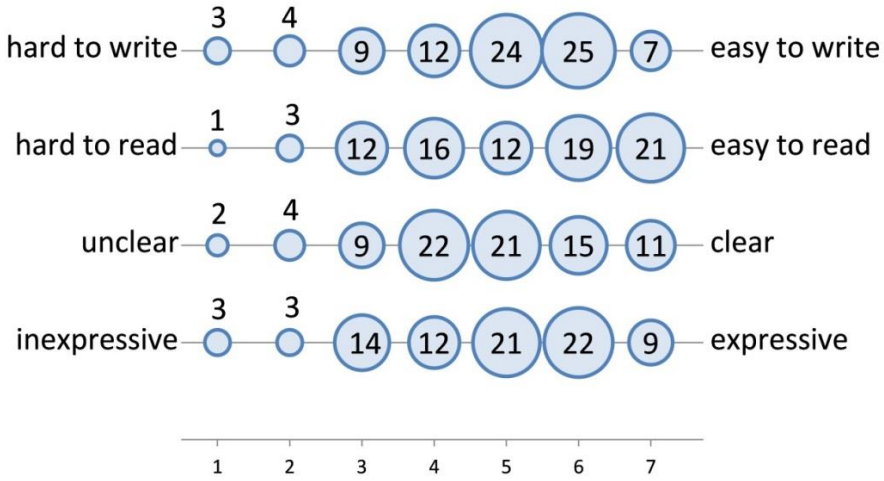


Figure 2-31. Responses to the "*How do you rate the timeline representation of the algebra operators?*" Semantic Differential Scale.

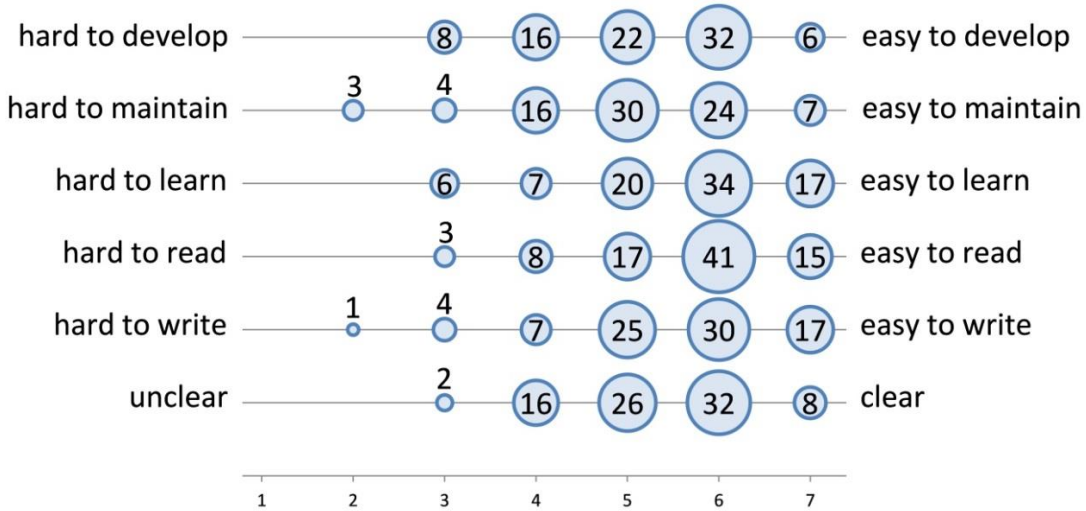


Figure 2-32. Responses to the "*How do you rate the DSL, as a whole?*" Semantic Differential Scale.

2.5.2 Study 2, a Gamified Evaluation of EveXL

As previously stated, EveXL was designed for simplicity. To assess if this objective has been met another study was conducted that primarily targeted participants with little to no programming experience [13,14]. If the results of such an assessment prove that the language is indeed simple, they can be extrapolated to more experienced programmers, as it is expected for them to have increased ease of use due to their familiarity with programming concepts and techniques.

2.5.2.1 Methods and Participants

To assess if EveXL truly is a straightforward event expression language, a browser-based videogame called Dungeon of Colors (DoC) was developed, with a simple setting: the player controls a character that must progress through a sequence of locked rooms, whose doors can only be opened by using a smartphone to perform specific events. In order to understand which specific events must be executed, players are required to read and interpret EveXL expressions. The assumption here is that the correct execution of an event implies the correct understanding of its expression.

From the architectural point of view, the DoC game has three components: first, the *DoCMobileApp*, a mobile application that runs on the smartphone and interfaces with EveWorks; second, the *DoCWebApp*, a web application that runs in a browser and renders the game scenario; and third, EveWorks itself, dealing with the event detection for the *DoCMobileApp*.

Although EveWorks does not have any conceptual restriction on the sensors it can use, to keep the player's attention focused on the interpretation of the expressions, an effort was made to simplify the gameplay. Thus, a new sensor class was implemented that reads the smartphone's Near Field Communication sensor and declared it to EveWorks with the keyword "color". This setup becomes more meaningful since four colored pads were used in the study, each with an embedded NFC tag that transmits a "white", "green", "red" or "black" value, in accordance with the color of its embedding pad (see the DoC's setting in Figure 2-33 and some participants performing the experiment in Figure 2-34).

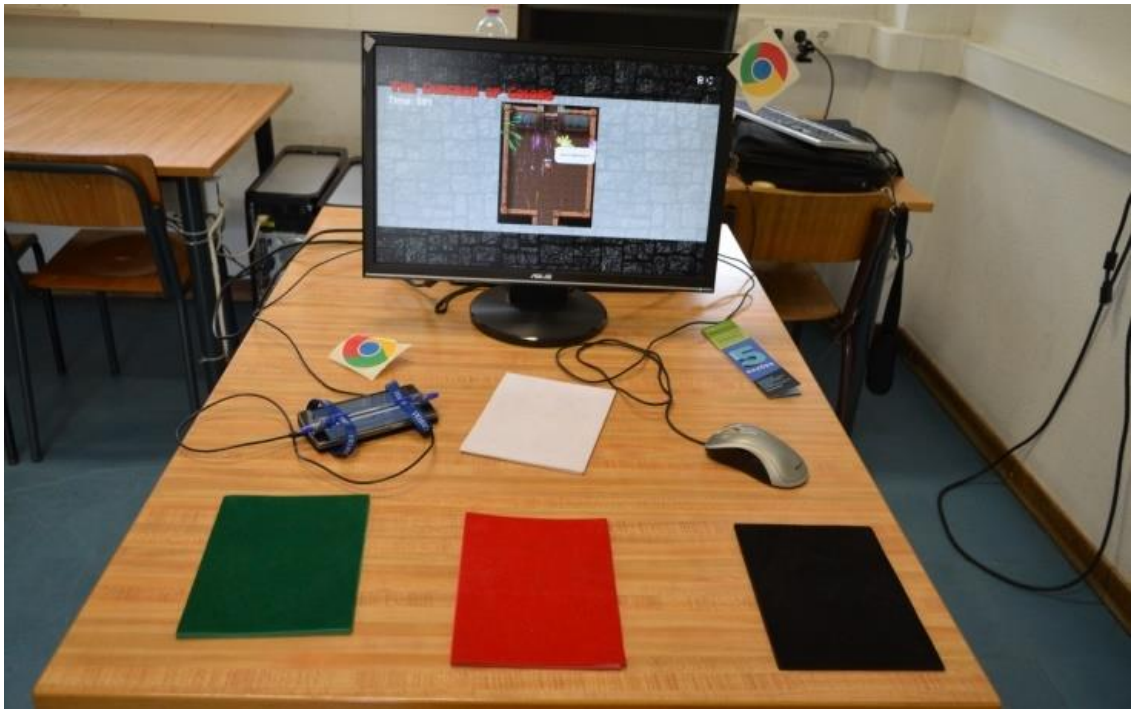


Figure 2-33. The Dungeon of Colors' setting: a browser displaying the DoC scenario (DoCWebApp), a smartphone running both DoCMobileApp and EveWorks (to the left of the white pad) and four colored pads with embedded NFC tags.

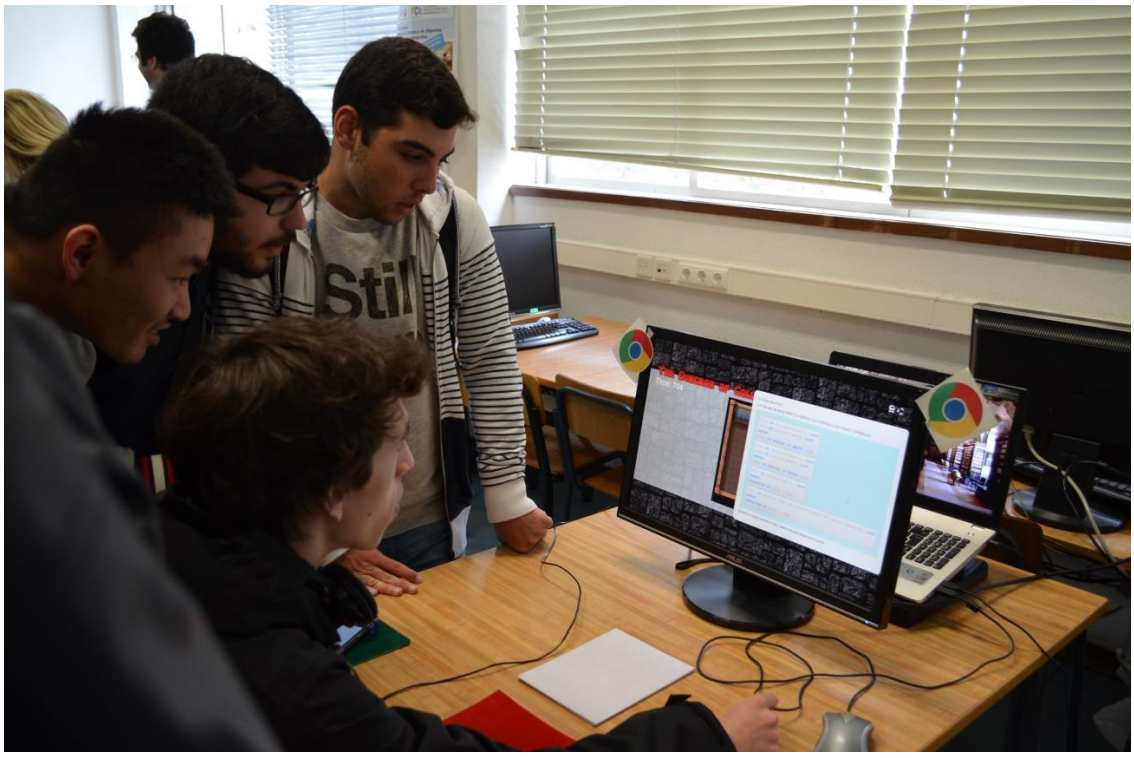


Figure 2-34. Participants had to read and understand EveXL statements and perform the expressed events.

In practical terms, an interval declaration such as the one in Snippet 30 corresponds to an interval of time named "C", during which the smartphone is laid on top of the white or red pad (i.e., the NFC sensor reads a "white" or "red" value from a nearby tag).

```
[C] AS [color = 'white' OR color = 'red']
```

Snippet 30. A data-invariant interval of time declared using the "color" sensor (it should be read as "C is an interval of time during which the 'color' sensor reads a 'white' or 'red' value).

The DoC game begins with a couple of "demonstration" rooms, exemplifying how participants should read and interpret EveXL expressions and, afterwards, how they are supposed to use the smartphone and the colored pads to open the door of each room. After these short examples, the gameplay/experiment is divided in two parts. The first part is composed of 6 rooms, in which users are asked to interpret a single EveXL expression and, afterwards, execute the conveyed event using the smartphone and the colored pads. When the player enters a new room, the *DoCWebApp* sends the corresponding event to the *DoCMobileApp* which, in turn, registers it with EveWorks. When EveWorks detects the registered event has occurred, it notifies the *DoCMobileApp* of this occurrence and the latter sequentially signals the *DoCWebApp*, thereby advancing the character to the next room. During the first part of DoC, the system automatically records the times taken by players to understand and execute the events, the *understanding* (T_{und}) and *execution* (T_{exe}) times, respectively. To be able to effectively measure them, the game requires players to place the smartphone over the white pad before displaying each room's expression. T_{und}, then, is the time counted from when the device is placed over the white pad until it left it. In turn, T_{exe} corresponds to the time counted from when the device leaves the white pad until the event occurrence is detected – i.e., the moment when players execute the event they had previously read. The expressions used on the DoC's first part and their respective T_{und} and T_{exe} times can be seen in Table 2-6.

On the second part of the game (from the 7th level onwards) the game rules change: at each level, players are now presented with the natural language description of an event and four alternative EveXL expressions. The game then asks players to choose, out of the four alternatives, the expression that corresponds to the event description. In the course of the second part, the system records the amount of errors made by participants, i.e., the number of wrong choices.

Finally, the DoC asks participants to evaluate EveXL by means of an SDS featuring 7-point scales between five pairs of bipolar adjectives: *hard to learn-easy to learn*, *hard to read-easy to read*, *hard to understand-easy to understand*, *uninteresting-interesting* and *hard-easy*.

As the main goal of the study was to discover how easily participants with little to no programming experience were able to understand the meaning of EveXL expressions, the DoC game was set up as an attraction, in the context of an annual university event for high school students. The study had 37 participants (73% male), with ages ranging from 14 to 60 years old, averaging 22.1 years old, and programming experience ranging from 0 to 30 years, averaging 2.2 years of programming experience. As it can easily be inferred from the demographics, there were a couple of participants with more experience (hence, the 2.2 average years of experience), though most (73%) were high school students.

2.5.2.2 *Results*

All of the 37 participants have completed the DoC game without reporting major difficulties. The mean understanding (T_{und}) and execution times (T_{exe}) for each of the 6 levels of the DoC's first part can be found in Table 2-6 (for more information on the measurement of these times, see the previous Section 2.5.2.1) and the results of the last three rooms of the second part of the DoC can be found in Table 2-7.

Table 2-6. Average time results (in seconds) for DoC part 1 (levels 1-6).

Expression	T_{und}	T_{exe}
[black] AS [color='black'] WHERE DURATION OF [black] > 5s	8	9
[red] AS [color='red'] WHERE DURATION OF [red] > 5s	7	8
[green] AS [color='black'] [black] AS [color='green'] WHERE [black] MEETS OR IS BEFORE [green]	12	11
[I1] AS [color='black' OR color='green'] [I2] AS [color='red'] WHERE [I1] MEETS OR IS BEFORE [I2]	13	10
[I1] AS [color='red'] [I2] AS [color='green'] [I3] AS [color='black'] WHERE [I1] MEETS OR IS BEFORE [I2] AND [I2] MEETS OR IS BEFORE [I3]	18	21
[I] AS [color='red' OR color='black' OR color='green'] WHERE DURATION OF [I] > 10s	10	14

Table 2-7. Descriptions and alternatives of levels 7-9 (part 2), correct answers signaled with a green left border.

Exercise	Average Number of Errors
The device spends some time over the BLACK pad and, afterwards, over the RED pad.	**(0.16)**

`[I1] AS [color='black']`
`[I2] AS [color='red']`
`WHERE [I1] IS BEFORE OR MEETS [I2]`	

`[I1] AS [color='red' OR color='black' OR color='green']` `WHERE DURATION OF [I1] > 10s`	
`[I1] AS [color='red']` `WHERE DURATION OF [I1] > 10s`	

`[I1] AS [color='black']`
`[I2] AS [color='red']`
`WHERE [I2] IS BEFORE OR MEETS [I1]`	

The device spends more than 5 seconds over the green pad.	**(0.11)**

`[I1] AS [color='green']`
`[I2] AS [color='red']`
`WHERE [I1] IS BEFORE OR MEETS [I2]`	

`[int] AS [color='red' OR color='black' OR color='green']` `WHERE DURATION OF [int] > 10s`	
`[red] AS [color='green']` `WHERE DURATION OF [red] > 5s`	
`[green] AS [color='red']` `WHERE DURATION OF [green] > 5s`	
The device spends some time over the green pad and, afterwards, more than 5 seconds over the red pad.	**(0.08)**

`[I1] AS [color='green']`
`[I2] AS [color='red']`
`WHERE [I1] IS BEFORE OR MEETS [I2] AND`
`DURATION OF [I2] > 5s`	

`[green] AS [color='red']` `WHERE DURATION OF [green] > 5s`	

`[I1] AS [color='black']`
`[I2] AS [color='red']`
`WHERE [I2] IS BEFORE OR MEETS [I1]`	

`[green] AS [color='red' OR color='black' OR color='green']` `WHERE DURATION OF [green] > 10s`	

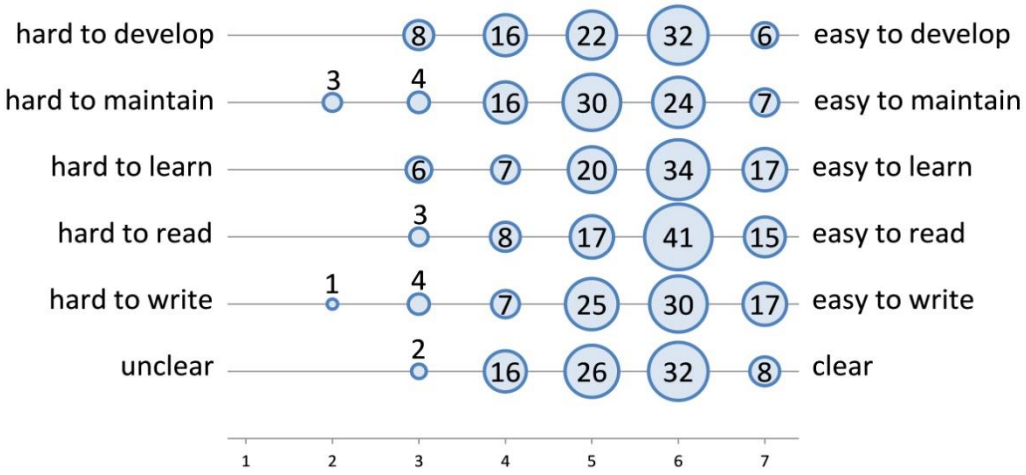


Figure 2-35. Participant feedback to question "*How do you evaluate EveXL, the event expression language*?".

Finally, the results of the Dungeon of Colors' last question, a 5-item Semantic Differential Scale asking participants to qualify the EveXL language, is depicted in Figure 2-35.

2.5.3 Study 3, EveXL Programming

While the results of Study 2 can be extrapolated to experienced programmers – if people with no programming experience can understand EveXL expressions, it is expected that experienced programmers will also do so with increased easiness –, they tell us little about whether programmers can, in fact, use EveXL to express the events they desire to capture. Therefore, another study [13] was conducted with experienced programmers that requested them to write the EveXL expressions that correspond to events described in natural language. This study goes one step further, in the sense that it requires participants not only to understand EveXL expressions, but also to use it to program context reactive behaviors.

2.5.3.1 Methods and Participants

To conduct this experiment, two complementary web applications were set up: the *QuestionnaireWebApp* and the *MonitorWebApp*. The *QuestionnaireWebApp* is, in essence, an online questionnaire composed of 6 items, each featuring a natural language description of an event, and a textbox

on which respondents wrote the EveXL expression corresponding to the description (the used descriptions and possible EveXL solutions can be found in Table 2-8). To facilitate the process, the textbox featured EveXL syntax highlighting. On the other hand, the *MonitorWebApp* is also a web application that allows a human monitor to validate expressions written and submitted by participants through the *QuestionnaireWebApp*.

The experiment's workflow is described next: after an expression is written on the *QuestionnaireWebApp*'s textbox, participants are required to press a "Validate" button which triggers automatic syntax validation. If any syntactic errors are found, the *QuestionnaireWebApp* reports them to participants; otherwise, the *QuestionnaireWebApp* sends the expression to the *MonitorWebApp*, which allows a human monitor to validate the expression's semantics – i.e., to assess if the EveXL expression really corresponds to the event described in natural language. If so, the human monitor issues through the *MonitorWebApp* a "valid" command to the *QuestionnaireWebApp*, causing it to advance to the next exercise. If the expression does not correspond to the described event, the *MonitorWebApp* asks the human monitor to write a short error report and then sends it, along with a "not valid" command, to the *QuestionnaireWebApp*. In turn, the latter displays the error report to the participant and asks him/her to correct the invalid expression and resubmit it.

The system records the time taken by each participant in each of the six exercises, counting from the time when the event description is first displayed to the participant, to the last time that he/she presses the "Validate" button – i.e., the time taken by the participant to produce a syntactically and semantically correct expression. In addition, the system also registers the number of syntactic and semantic errors as well as the sequence of expressions generated by the participants for each exercise, to allow posterior analyses.

Finally, the *QuestionnaireWebApp* asks participants to evaluate both EveXL as a language – *How do you evaluate EveXL?* – and its underlying concept – *How do you evaluate programming through time intervals?* – by means of two 7-point SDSs featuring six and five pairs of bipolar adjectives, respectively.

Table 2-8. Event descriptions and possible solutions.

Exercise	Event Description	Possible EveXL Solution

1	The device spends some time over the red pad and, afterwards, over the green pad.	`[I1] AS [color='red']`
		`[I2] AS [color='green']`
		`WHERE`
		`[I1] MEETS OR IS BEFORE [I2]`

2	The device spends some time over the red then the green pads and, finally, over the white pad.	`[I1] AS [color='red']`
		`[I2] AS [color='green']`
		`[I3] AS [color='white']`
		`WHERE`
		`[I1] MEETS OR IS BEFORE [I2]`
		`AND`
		`[I2] MEETS OR IS BEFORE [I3]`

3	The device spends some time over the red pad and then moves to the green pad without, in the meantime, having spent time over the black pad.	`[I1] AS [color='red']`
		`[I2] AS [color='green']`
		`WHERE`
		`[I1] MEETS OR IS BEFORE [I2]`
		`AND`
		`NOT EXISTS {`
		`  [I3] AS [color='black']`
		`  WHERE`
		`  [I1] MEETS OR IS BEFORE [I3]`
		`  AND`
		`  [I3] MEETS OR IS BEFORE [I2]`
		`}`

4	The device spends some time over the green pad and, afterwards, moves to the red or white pad.	`[I1] AS [color='green']`
		`[I2] AS [color='red' OR`
		`  color='white']`
		`WHERE`
		`[I1] MEETS OR IS BEFORE [I2]`

5	The device spends some time over the green pad and, after more than 5 seconds out of it, moves to the black pad.	`[I1] AS [color='green']`
		`[I2] AS [...]`
		`[I3] AS [color='black']`
		`WHERE`
		`[I1] MEETS [I2] AND`
		`[I2] MEETS [I3] AND`
		`DURATION OF [I2] > 5s`

6	The device has left the black pad more than 5 seconds ago.	`[I1] AS [color='black']`
		`[I2] AS [...]`
		`WHERE`
		`[I1] MEETS [I2] AND`
		`DURATION OF [I2] > 5s`

As a sample with significant programming experience was required, the participants were recruited by email and direct invitation to students and Professors at the university's Computer Science Department. This procedure yielded 23 participants (73% male), with ages ranging from 20 to 51 years old, averaging 34.9, and programming experience ranging from 0 to 31 years, averaging 15.7 years. Regarding education, almost all (91%) participants had university graduate or post-graduate degrees.

2.5.3.2 Results

As previously stated (see Section 2.5.3.1), a web-based questionnaire (*QuestionnaireWebApp*) was used that featured 6 natural language event descriptions (see Table 2-8) and asked participants to write the corresponding EveXL expressions. The number of syntactic and semantic errors and the time that took each participant to write a syntactically and semantically correct expression were recorded. These results are in Table 2-9.

Table 2-9. Results of the experiment's six exercises.

	Syntactic Errors			Semantic Errors			Time (seconds)		
Exercise	**Avg**	**Min**	**Max**	**Avg**	**Min**	**Max**	**Avg**	**Min**	**Max**
1	0.83	0	6	0.83	0	3	224	40	512
2	0.26	0	2	0.17	0	1	189	69	413
3	1.39	0	4	0.96	0	5	414	112	887
4	0.35	0	2	0.30	0	2	211	51	571
5	0.86	0	5	0.82	0	3	322	102	657
6	0.74	0	6	0.30	0	2	211	63	605

The questionnaire had two final questions, which asked participants to evaluate EveXL as a language for event programming, as well as the concept of programming events through the articulation of time intervals. These questions were answered through 6- and 5-item SDSs, respectively, which have been derived from Sebesta's [87] criteria for language evaluation (see introductory

text of Section 2.5). The responses to these two SDSs are in Figure 2-36 and Figure 2-37, respectively.

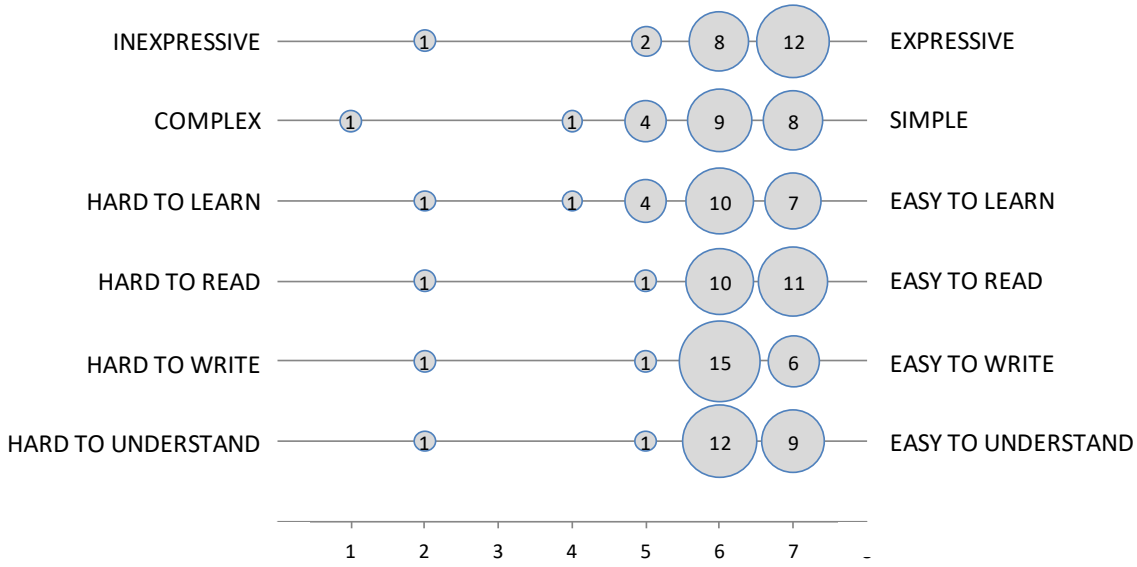


Figure 2-36. Participant responses to question "How do you evaluate EveXL?".

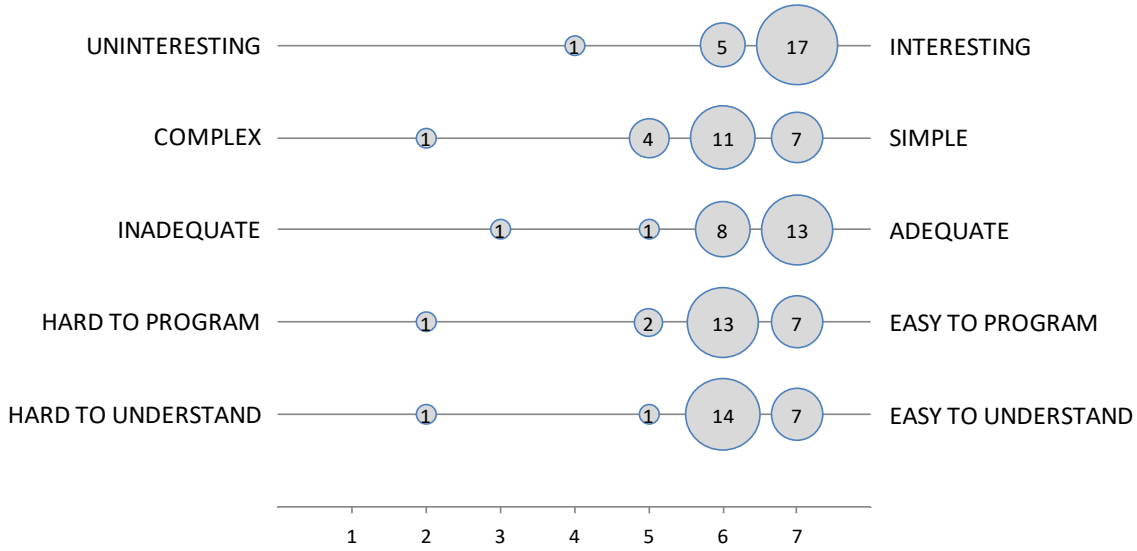


Figure 2-37. Participant responses to question "*How do you evaluate programming through time intervals?*".

2.6 Discussion

The results of the three conducted studies support that the EveWorks-EveXL dyad is a valid approach to context event detection in mobile platforms.

Indeed, adaptability is a desirable feature for context-aware applications to support the deployment of meaningful interactions on mobile devices. As such, flexible enough architectures are necessary to provide direct answers to applications requiring changes in their reactive behavior at runtime.

Study 1 (see Section 2.5.1) has explored the viability of EveXL's alternative notation for the operators of the Interval Algebra and has produced evidence indicating that the timeline visual representation is a good source of inspiration for the DSL. Indeed, participants were required to answer an online, 9-item questionnaire, and no participant manifested relevant difficulties while answering it. This is all the more encouraging since almost no participants declared to have had any previous contact with similar problems.

The visual analogy between the textual timeline representation of EveXL's alternative notation and the familiar visual diagrams of the timelines did not go unnoticed, and was particularly appreciated by those participants who had declared having little to no programming experience.

The results of Study 1's field evaluation were also encouraging, with the dominant majority of the alarms triggering consistently when expected. This is evidence of the adequateness of EveWorks' concept, as well as, of course, of a solid implementation.

Study 2 (see Section 2.5.2), on the other hand, was conducted to test if simplicity – one of EveXL's main design guidelines – has been achieved and EveXL is consequentially simple and expressive. To that end, another study was conducted that primarily targeted participants with little to no programming experience. The underlying hypothesis here is that if inexperienced programmers can effectively interpret EveXL expressions, more experienced ones will also be able to do so, with increased easiness. To that end, a videogame was developed, that required players to read and interpret EveXL expressions, the Dungeon of Colors (DoC).

The results of this study allow conclusions regarding EveXL's accessibility. Truly, in the first part of the DoC game, the reduced times taken by participants to understand EveXL expressions (see column "T_{und}" of Table 2-6) indicate that the DSL is easily readable and – what is more important – that

it conveys its meaning adequately. This claim is further supported by the also reduced execution times (see column "T_{exe}" of Table 2-6), which would likely be longer if participants didn't fully understand what they were supposed to do, and attempted to execute the requested events by trial-and-error.

Though still reduced in all cases, the understanding and execution times increase slightly along with the number of time intervals that are declared in each expression. The correlation between the amount of intervals in an event and its time of understanding may be explained by the role of functional unit that time intervals play in EveXL's conceptual framework. Indeed, as the number of interrelated time intervals increases in an expression, the more information the expression conveys and, consequentially, the more complex it potentially becomes. On the other hand, the correlation between the amount of intervals in an expression and the execution time of its event is self-explanatory: indeed, an interval of time in EveXL means exactly that – literally, an interval of time. Therefore, the more intervals are sequenced in an expression, the longer will take its event to occur.

The results of the last three rooms of the second part of the Dungeon of Colors are also revealing (see Table 2-7). Indeed, the low average number of errors made by participants when selecting the EveXL expression that matches a given event's natural language description confirms that largely inexperienced participants are able to interpret and relate the expressions to real-life actions (even if limited to placing a smartphone over some colored pads).

The results of this study's final SDS are also very encouraging (see Figure 2-35), as they reveal that the majority of our participants found EveXL to be interesting and easy to learn, read and understand. Additionally, as an interactive videogame, the Dungeon of Colors required reliable and responsive event detection, a task that was solely performed by EveWorks. Along with the similar results of the field evaluation of Study 1 (see Section 2.5.1), the consistent and problem-free performance is proof of EveWorks' reliability.

To conclude the discussion of Study 2, it is perhaps worth emphasizing the support that these results provide to that study's main hypothesis (i.e. EveXL is simple to read and interpret). Indeed, because EveWorks and EveXL

are tools intended to be used by programmers, the positive results – largely coming from people with negligible programming experience – can be extrapolated to experienced professionals, as they are likely to have better working knowledge of programming languages and technology and, therefore, increased easiness in understanding formal expressions.

Finally, Study 3 was conducted to find if, besides being readable, EveXL is also writable. To that end, and contrasting to Study 2, this study was targeted at evaluating EveXL with experienced programmers, who were asked to respond to a web-based questionnaire (*QuestionnaireWebApp*). The *QuestionnaireWebApp* featured 6 exercises, i.e., items displaying natural language event descriptions (see Table 2-8) that asked participants to write the corresponding EveXL expressions.

With the exception of exercise 3, all exercises have systematically registered a low mean number of syntactic errors (see Table 2-9, column "Syntactic Errors") – i.e., less than 1 syntactic error per participant, on average. The relatively higher mean number of syntactic errors of exercise 3 may be explained in the light of the higher complexity of that particular exercise. Indeed, the event requested users to write an expression which, in its most direct form, would be composed using a `NOT EXISTS` operator, thereby requiring participants to write a subexpression nested in the outer expression (see the exercises, in Table 2-8; read about the operator in 2.3.3.7, *Inexistence Predicate*).

Where semantics are concerned, participants have also not incurred in many errors – i.e., expressions that, in spite of being syntactically correct, would not detect the precise event that was described in the exercise's natural language description. Once again, the mean number of semantic errors per participant is below 1. On a closer observation of these errors, it is noteworthy that on the first exercise, upon the initial contact with EveXL, roughly half of participants wrote expressions that incompletely defined the event required by the exercise, i.e., expressions that defined events that would not be detected when and every time they were supposed to. This mainly happened because participants often used a colloquial interpretation of the meaning of the relations between intervals. The most representative case is the confusion that

seemed to exist between the `IS BEFORE` and the `MEETS` operators, assuming that the former included the latter – or, in other words, that an interval that meets another is also before it. This is not true, due to the *distinct* property of the interval algebra's relations (see Section 2.3.2). Another likely explanation may be found on the inherent ambiguity of natural language. As each exercise's event description was provided in natural language, it follows that it may be subject to misinterpretations and, therefore, to deviant EveXL event specifications.

This kind of error became much less frequent or nonexistent in the following exercises, proof of a learning curve for understanding the EveXL's (the Interval Algebra's) operators. On the other hand, also worthy of comment, are erroneous expressions that would mean that the event specified by the exercise would not be detected, or a different one would. This type of error, although rare, was detected more often in exercises 3 and 5. On the former, this may be due to its relatively increased complexity, entailing the use of a subexpression to convey a nonexistent time interval. On the latter, exercise 5 was the first time that participants could use a pure time interval to describe the event – the most direct solution to the problem, even if not the only one.

The time taken by participants to produce fully correct expressions – both semantically and syntactically valid – is also indicative that EveXL has an accessible syntax and its underlying concept – programming through the articulation of time intervals – is simple enough to be mastered quickly. These conclusions are further supported by the fact that, in spite of their programming experience, this was the first contact these programmers ever had with EveXL; moreover, the answers to Study 2's final SDSs, also corroborate this conclusion, as they strongly suggest that both EveXL and its underlying concepts are adequate tools for programming the reactive behavior of mobile applications.

It was definitely interesting to see how each programmer wrote EveXL code using their own style and indentations, some applying "CamelCase" notation for interval identifiers, while others adapted and applied other conventions. But perhaps the most interesting case was when programmers attempted to develop their own programming patterns or – what perhaps is

more accurate -, temporal patterns. For instance, more than once programmers tried to simplify expressions by using sequences of pure-time intervals as a temporal replacement for the `IS AFTER` relation (i.e., instead of two intervals related by the `IS AFTER` operator, programmers would queue three intervals using the `MEETS` operator, with the middle one corresponding to a pure-time interval). This reveals that these participants attained a deeper understanding of EveXL's concepts, as they not only wrote the correct expressions, but were actually testing the language's conceptual limits by playing with temporal models.

Incidentally – regardless of being or not the correct answers to the exercises of Study 3 –, the accurate detection of the events programmed by the participants implies the coherence of the connection between the logic of EveXL and the EveWorks event detection mechanism, i.e., the sound temporal models that EveWorks creates out of EveXL expression interpretation.

3

CAAT, Emotions in Interaction Design

3.1 Introduction

More than being a part of our daily experiences, emotions are an intrinsic part of our condition as human beings. They permeate our existence, influencing our behavior and judgments so directly that some neurologists, like António Damásio, advocate for the inexistence of "pure reason" [28], i.e., reason without the interplay of emotions. Indeed, emotions play a lead role in very important and innate heuristics that guide us when we take risk-involving decisions [36] and decisions strongly depend on habits, which are implicit memories whose storage and retrieval are also mediated by emotions [52]. Additionally, as previously mentioned in Chapter 1, research indicates that people's affective states have a strong influence on how susceptible they are to persuasion, with positive states generally translating to increased receptiveness.

Aside from the situational, external aspect of *kairos*, there are also relevant variables that are intrinsic to the end-receiver of a persuasive effort, namely affective states. Indeed, research suggests that people are more likely to be susceptible to persuasion when they are in a good mood, and this claim appears to find support on research both on the fields of Psychology [10,55] and Human-Computer Interaction [48,50]. Additionally, recent research suggests that boredom is also a positive factor for persuasion. Indeed, acknowledging that boredom is a common human emotion which may lead to an active search for stimulation, Pielot et al. [59,69] have designed a system that recommends multimedia content to its users upon detecting they are likely to be bored – i.e., looking for stimulation on their smartphones. Their results suggest that people are more likely to engage with recommended content when they are bored, as inferred by their machine learning model of user boredom.

The assessment of affective states, however, is a very challenging task. While the field of Psychology has proposed a number of assessment tools, there is not yet a universal standard method for allowing computer applications to assess people's affective states reliably.

Given the importance of their role in our behavior, it is not surprising to note the growing interest on the topic of emotions by several fields of research, some of them branches of Computer Science. For instance, we can often find emotions figuring prominently on Computer Entertainment's body of literature, where the user's affective experiences are central to the description of the gaming experience [58] or even an actual element of the gameplay [78]. Even other fields without such an obvious link, like Information Retrieval, are also beginning to explore this facet of human nature [63]. As emotions play an important role in the way we experience arts [34] and classify multimedia – for instance, think of the many movie genres with names borrowed from the realm of emotions, like "horror" or "comedy" –, the assignment of emotional tags to multimedia content has been an active research topic in the last decade [89]. But perhaps the strongest evidence of the interest that Computer Science has in this topic was the inception of Affective Computing, a field of research that, as stated by its founders, studies "*computing that relates to, arises from, or influences emotions*" [68]. One of Affective Computing's research aims is to enable machines to understand people's emotional states through the analysis of the physiological responses associated with particular emotions. Of course, this is a very ambitious goal; even aside from the technical difficulties, the variation of physiological responses across individuals and cultures further adds to the task's complexity [40]. In fact, the reliable assessment of affective experiences is not a settled matter. People's affective states are traditionally assessed by psychologists through retrospective self-reporting, either in the context of clinical interviewing, or with paper-and-pencil questionnaires, generally administered at the beginning and/or end of clinical or experimental settings [74]. Some researchers, however, argue that our memory of emotional states varies greatly over time, with a notable tendency towards oversimplification [40,79]. Hence, immediate, *in situ* (*opportuno tempore*) measurements of emotion are potentially more accurate than retrospective assessments. Moreover, for what concerns Human-Computer Interaction (HCI), these relatively lengthy

and time consuming questionnaires with detailed instructions (more details in Section 3.4) are not easily integrated into functional user interfaces (UI).

The need for simple tools that enable quick assessments of emotional reactions while being amenable to seamless integration in user interfaces is well manifested in the ad-hoc-developed tools that the HCI community has been using in research projects. While most of these approaches are undoubtedly interesting and creative (e.g., [78]), thoroughly validated tools of this sort are still rare.

Some of the approaches found in the literature are based on automatic classification of physiological responses, while others follow the self-assessment approach. While the former are unquestionably interesting, the latter have the potential to provide insight into the cognitive and subjective emotional components that can only be assessed with subjective self-reports [86]. In addition, self-reporting of emotions promotes self-awareness, a fundamental goal in clinical and experimental interventions.

3.1.1 Models of Human Emotion

The topic of emotional experience is not a simple one. In fact, research has revealed emotions to be complex constructs, with fuzzy boundaries and substantial individual variations, both in experience and expression [64]. That being said, the problem of emotion conceptualization is one of the fundamental issues of the subject, and the literature presents two prevalent perspectives: the discrete and dimensional theories.

The discrete theory of emotions posits that emotions are caused by a discrete number of biologically basic reactions, or urges to act, that humans share with all other mammals [8]. One of the most distinct aspects of this approach is that emotions are considered as recurrent or, in other words, they are conceptualized as physiological states repeatedly experienced during lifetime. This theory has a recognizable appeal to common sense and finds reflections in our everyday colloquial conversations, where emotional experiences are designated by common words like “joy” or “anger”. Perhaps due to its appeal to common sense, we can trace this conceptualization back to classical antiquity, where lists of emotions were compiled by philosophers and

scholars – Aristotle, for instance, listed 13 emotions in his work *Rethoric* (book II).

On the other hand, the dimensional theory of emotions proposes a conceptualization where emotional experiences are defined by a number of component dimensions and can be represented as points in dimensional spaces. The first dimensional model of emotions was proposed by Wundt [94], who defined three dimensions for affective experience: *pleasurable* vs. *unpleasurable*, *arousing* vs. *subduing* and *straining* vs. *relaxing*. Although in later works different researchers have proposed different component dimensions, most of them agree that valence (hedonic pleasure vs. displeasure) and arousal (high activation vs. low activation) are fundamental qualities of affect. For example, the experience of "terror" in the valence-arousal space would be composed of very low valence and very high arousal, while "boredom" would likely have low valence and low arousal values. Russell's circumplex model is a good example of one such dimensional model, where emotions are represented as points in a two-dimensional, valence-arousal space [81,82] (see Figure 3-1). Another dimension that also figures recurrently in the literature, though to a lesser extent, is dominance (dominant vs. submissive) [60].

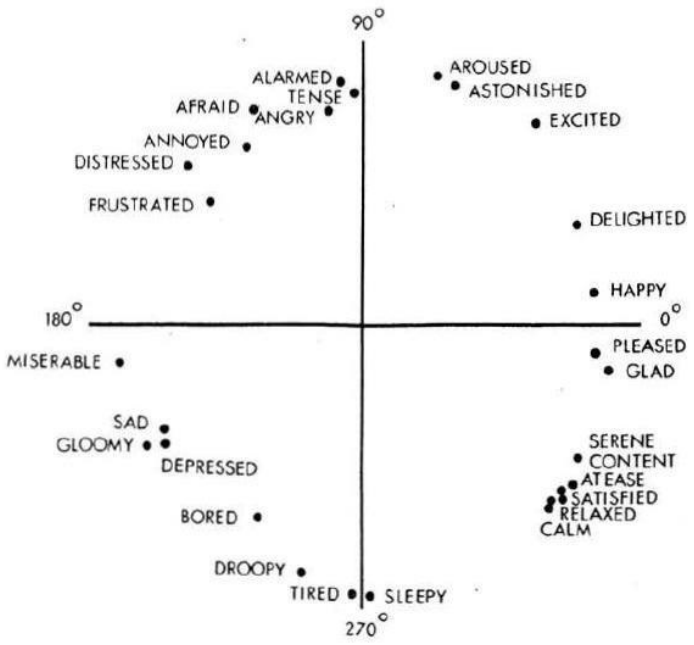


Figure 3-1. Russell's Circumplex model of emotion-related categories (in [82]).

Although the discrete and dimensional perspectives may, at first sight, appear to mutually exclude one another, recent research works offer a more conciliatory perspective on the subject. For instance, Barrett's conceptual act model of emotions [7] proposes to conceptualize emotions as instantiations of affective feelings, an act of cognitive categorization that allocates common (discrete) names to the momentary evaluation of one's core affect. Core affect is the "*ongoing, ever changing state that is available to be categorized during emotion conceptualization*", and may be evaluated along the common dimensions of valence and arousal. Moreover, there is empirical evidence that, when modeling affective reactions to musical stimuli, discrete and dimensional models of emotions produce highly compatible ratings [34]. This suggests that emotions may be best modeled using approaches that combine the essential characteristics of both theories, i.e., to consider that they can be abstracted beyond their colloquial names, without disregarding the practical value of the latter. Indeed, the use of common labels, like "joy" or "terror", should not be overlooked, as they have a natural presence in people's everyday lives. Indeed, as Russell [82] mentioned, "*people have an informal and implicit 'naive theory' of emotion, which they use when they anticipate, identify, communicate about, and try to influence the emotional states of others*".

Along the same line of thought, Robert Plutchik proposes an integrative perspective between these theories in his psychoevolutionary theory of emotions [70–72] . According to him, emotions are feedback processes whose function is to "*restore the individual to a state of equilibrium when unexpected or unusual events create disequilibrium*". His theory assumes the existence of 8 basic emotion dimensions (each with a number of synonyms or related terms), arranged in four opposing pairs: joy vs. sadness, trust vs. disgust, fear vs. anger and surprise vs. anticipation. The model allocates 3 intensity variants for each of these dimensions, thereby featuring a total of 24 emotion words. The visual representation of the model was designed in an analogy to a color wheel (more details are in the following Section 3.2). This metaphor is also maintained for those emotions that do not figure explicitly in any of the eight dimensions: they are obtained by "mixing" two or more basic emotions (e.g., "love" is modeled as a combination of "joy" and "trust").

3.2 The CAAT

The Circumplex Affect Assessment Tool (CAAT) (see Figure 3-3) is a tool for performing quick assessments of emotional experiences. It is a widget dedicated to the choice of emotional states, inspired by discrete item selection controls (e.g., drop-down lists). As far as revealed by literature searches, it is the first and only emotion assessment tool to have been conceived and designed as a widget. Analogous to a drop-down list, it has two states: *closed* and *open*. While closed, the CAAT has a text area that displays the currently selected emotions (if any) and a lateral button to open it; once open, the tool displays 25 selectable emotion nodes, each with its own emotion word and color, arranged in a layout that is a close resemblance to Plutchik's circumplex model (see Figure 3-2).

However, because the CAAT is not an illustration of a model of affect but rather a tool for the assessment of emotional responses, the emotion ordering on its radial axes has been reversed. Indeed, Plutchik's circumplex model has its most basic and simple emotions side-by-side at the model's center and the more complex ones positioned on its outer circles [73], and there is not an explicit representation of emotional neutrality – rather an implicit, zero-intensity periphery (see Figure 3-2). Since we intended to create an assessment tool, a way was needed to allow for emotional indifference to be expressed as easily as any other emotion. To address this requirement, the axes have been individually reversed and an explicit, selectable "nothing" center has been added (see Figure 3-3).

The tool's selection mechanism was also inspired by Plutchik's work. Indeed, much in the same way that Plutchik has proposed fundamental emotions to be "combined" in order to obtain new ones, so does the tool: it asks users to select the one or two emotion words that best describe their emotional experience. It is here, in the simplicity of use, that lies one of the CAAT's strongest points. It only has one simple instruction "*Select one or two emotions*" and resorts to easily recognizable, mostly colloquial emotion words to ask users how they feel. This approach contrasts with more traditional tools for affect assessment, with their lengthy instructions and often requiring more than a superficial understanding of the concepts involved.

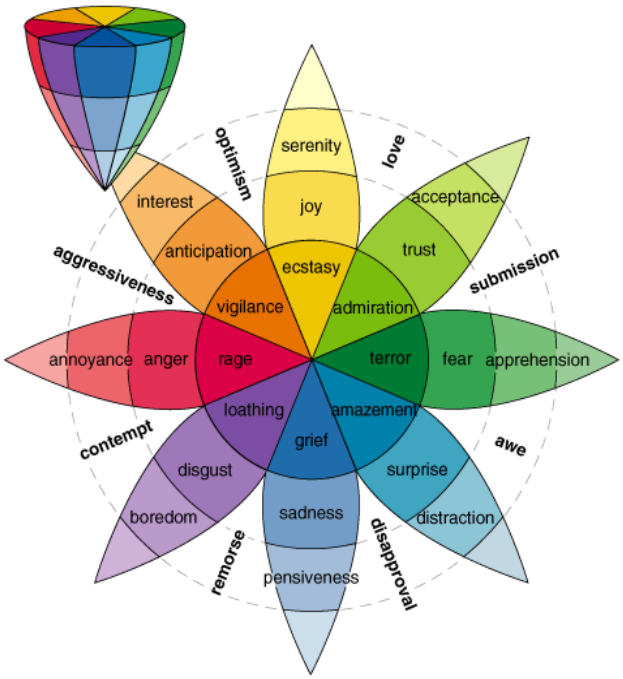


Figure 3-2. Robert Plutchik's model of emotions, analogous to a color wheel.

Figure 3-3. The Circumplex Affect Assessment Tool (CAAT) in its two forms: closed as a select box (top) and open (bottom).

3.3 CAAT's Output

The CAAT output is twofold: it returns the selected emotion word(s) along with two derived numerical scores, the CAAT valence and CAAT arousal. Although the user's selection of emotion words is, by itself, a valid and useful product of the CAAT, it is mostly interesting in cases where only categorical emotion expressions are required. Given the possibility of mapping discrete emotions to dimensional spaces [38,81,85], a literature survey was undertaken and found Warriner et al.'s [90] broad scale survey, presenting the affective norms for nearly 14 thousand English lemmas. Therefore, each of the CAAT's emotion words has been assigned with a particular pair of valence and arousal values, as they were found in Warriner et al.'s study.

However, this survey does not present the affective norms for all of the CAAT's emotion words, as the word "pensiveness" is not featured. To address this situation, a search was conducted for closely related emotion words whose affective ratings could be used instead of the missing pensiveness'. Those of "melancholy" were chosen since this word is commonly listed as a synonym of "pensiveness" in thesauri across the web[1]. Further supporting this replacement, we can find both words to be close to one another in the WordNet[2] lexical database, using several measures of similarity. For example, the Wu and Palmer similarity [93] between "pensiveness" and "melancholy" is 0.9412 (in this particular measure, 1 means that the concepts are the same and 0 means total dissimilarity). The final affective scores for each emotion word are in Table 3-1.

Higher values of valence indicate pleasurable and controlled emotions (e.g. admiration, ecstasy), whereas lower valence implies unpleasurable reactions (e.g. terror, grief). On the other hand, higher arousal values indicate highly aroused states (e.g. rage, amazement), while low arousal reflects inactive states (e.g. boredom, serenity).

[1] https://www.powerthesaurus.org/pensiveness

[2] http://marimba.d.umn.edu/cgi/bin/similarity/similarity.cgi?word1=pensiveness&senses1=all& word2=melancholy& senses2=all& measure=all&rootnode=yes

Table 3-1. The CAAT score system [*Emotion (Valence, Arousal)*].

Terror (2.8, 6.4)	**Fear** (2.9, 6.1)	**Apprehension** (4.2, 4.2)
Rage (2.3, 5.2)	**Anger** (2.5, 5.9)	**Annoyance** (3, 4.1)
Admiration (7.6, 5.5)	**Trust** (7.2, 4.3)	**Acceptance** (6.8, 4.3)
Loathing (2.4, 4.5)	**Disgust** (3.3, 5)	**Boredom** (2.8, 2.6)
Ecstasy (7.2, 6.1)	**Joy** (8.2, 5.6)	**Serenity** (7.8, 3)
Grief (2.3, 5)	**Sadness** (2.4, 2.8)	**Pensiveness** (3.7, 4.1)
Vigilance (5.7, 3.6)	**Anticipation** (5.3, 5.4)	**Interest** (6.7, 4.4)
Amazement (7.3, 6.3)	**Surprise** (7.4, 6.6)	**Distraction** (4.1, 3.9)

3.4 Public Availability

The CAAT is available for download, ready to be used in web-based user interfaces as a jQuery extension that turns a given html input tag into a CAAT instance, at http://img.di.fct.unl.pt/bmcardoso/caat.

3.5 Related Work

As previously stated, typical emotional assessments in psychology are conducted by means of relatively time-consuming pen-and-paper questionnaires, often with extensive and detailed instructions. One of the most widely accepted of such questionnaires is PANAS (Positive and Negative Affect Schedule) [91], a 20-item questionnaire consisting of a list of emotion- and feeling-related terms; it requests subjects to indicate the extent to which they have felt in that specific way, either at the time they are filling in the questionnaire or in the course of a specific time span (for instance, last week), and produces twofold results: a PA score (Positive Affect) and an NA score (Negative Affect). The first score, PA, reflects the extent to which a person feels positive and energetic – whereas a high PA indicates enthusiasm, optimism, alertness or pleasurable engagement, a low PA is characterized by sadness,

pessimism or lethargic states. On the other hand, the NA score is a general dimension of subjective distress, with high NA co-occurring with anger, contempt or fear and low NA indicating a feeling of calmness and serenity (see Figure 3-4).

PANAS Questionnaire

This scale consists of a number of words that describe different feelings and emotions. Read each item and then list the number from the scale below next to each word. **Indicate to what extent you feel this way right now, that is, at the present moment *OR* indicate the extent you have felt this way over the past week (circle the instructions you followed when taking this measure)**

1	2	3	4	5
Very Slightly or Not at All	A Little	Moderately	Quite a Bit	Extremely

_____ 1. Interested	_____ 11. Irritable
_____ 2. Distressed	_____ 12. Alert
_____ 3. Excited	_____ 13. Ashamed
_____ 4. Upset	_____ 14. Inspired
_____ 5. Strong	_____ 15. Nervous
_____ 6. Guilty	_____ 16. Determined
_____ 7. Scared	_____ 17. Attentive
_____ 8. Hostile	_____ 18. Jittery
_____ 9. Enthusiastic	_____ 19. Active
_____ 10. Proud	_____ 20. Afraid

Figure 3-4. The Positive Affect Negative Affect Schedule (PANAS) (figure adapted from [57]**).**

Another widely used tool for emotion assessment is Russell et al.'s Affect Grid [83] (see Figure 3-5), which consists of a 9x9 table representing Russell's circumplex model's valence-arousal affect space. Respondents are required to check the grid position that best describes their current disposition (the x-axis stands for valence and y-axis for arousal). Although primarily designed for pen-and-paper use, its simple design makes its translation to interactive

environments very straightforward – indeed, it has already been used in Computer Science research [58]. However, given the length of its use instructions, it may be unwieldy to include directly in user interfaces [74].

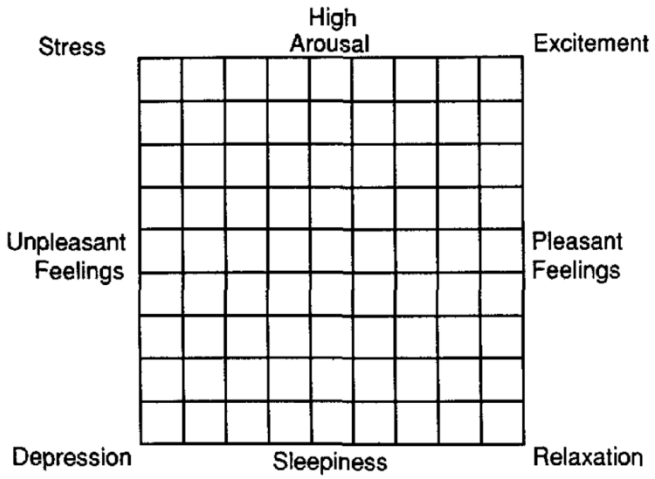


Figure 3-5. The Affect Grid, a single-item affect assessment method (in [83]**).**

Also based on the same valence-arousal space, Bradley and Lang [11] propose the Self-Assessment Manikin (SAM), a single item measure of affect that presents respondents with three strips of progressively changing drawings of a simplistic manikin. Each strip represents variations along a single affective dimension - valence, arousal and dominance - and the number of figures varies in accordance with the desired result range. In order to use the SAM, subjects are required to select the three figures (one per strip) that they think best represents their overall emotional state. However, they are also required to have more than a superficial understanding of the dimensional concepts involved (valence, arousal and dominance) something that is reflected on the SAM's detailed, lengthy instructions.

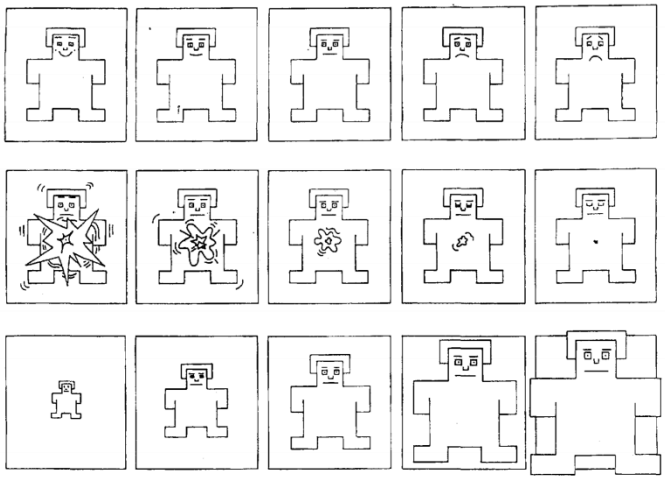


Figure 3-6. The Self-Assessment Manikin (SAM), with its three affective dimensions: valence (top), arousal (middle) and dominance (panel) (in [11]).

Pollak et al. [74] have developed the Photographic Affect Meter (PAM), an interesting tool that performs single-item assessments of user's affective states by presenting them with a variety of photos and having them select the one that best describes their current disposition (see Figure 3-7). It was designed for quick, *in situ* evaluations of emotional reactions and, hence, it has been targeted to operate on mobile platforms. The assessment procedure is very simple: it starts by presenting 16 randomly selected photos to a user and ends when he/she chooses one of the images (i.e., the one that evokes on him/her the same type of emotion that she/him is currently experiencing). In case the user does not consider that any of the photos accurately describes his/her current emotional state, the PAM features a "More Photos" button which will display a new selection of 16 photos to him/her. Each set of 16 photos is arranged in a 4x4 grid according to their relationship to affect (based on empirical evidence) and along the variation in the axes of Russell's Affect Grid [83]: valence for the x-axis and arousal in the y-axis.

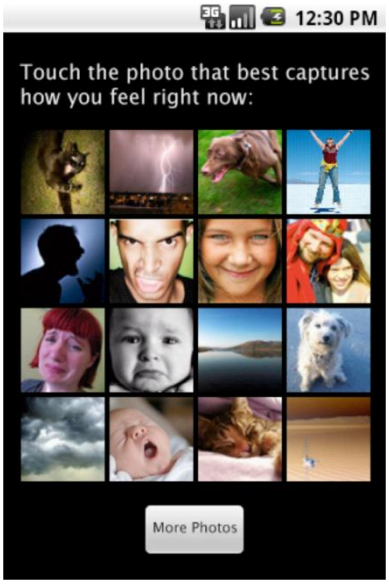


Figure 3-7. The Photographic Affect Meter (PAM) (in [74]**).**

While the PAM has been validated against the PANAS [91], thereby demonstrating its construct validity, as far it was ascertained from the literature, its test-retest (temporal) reliability is yet to be assessed. Moreover, because the assessment is performed by selecting a single photo among a set of 16 randomly selected photos, it is likely that the user makes his/her decision by comparison. Thereby, in order to recall why he/she had made a particular selection in the past, he/she would probably have to view the whole set. While this may not be a problem in itself, it may become a hindrance for longitudinal studies requiring reapplication of the tool for assessment of emotional reactions to different stimuli. Finally, as the authors have acknowledged, an important bias is raised: photo analysis requires interpretation, and interpretation is culturally biased [33].

Cowie and Sawey' work explore emotions from a different perspective, having developed both the FEELTRACE [23] and the GTrace [24] tools. This discussion, however, will just cover the latter, as it is a successor to the former. GTrace is, then, an application whose primary function "*is to let observers watch and/or listen to a designated person in a recording, and to indicate how particular features of the person's state appear to fluctuate with time*". The result is a trace, a stream of values representing the changes in particular emotion-related dimensions over time. This tool contrasts with the CAAT in a few defining

aspects: GTrace is a full-fledged application, while the CAAT is a widget designed to integrate emotion assessments in other applications' user interfaces; instead of recording temporal traces, the CAAT performs momentary evaluations and, finally, rather than assessing the emotions perceived in videos of others, the CAAT is a self-assessment tool aiming at the capture of its users' own emotional states.

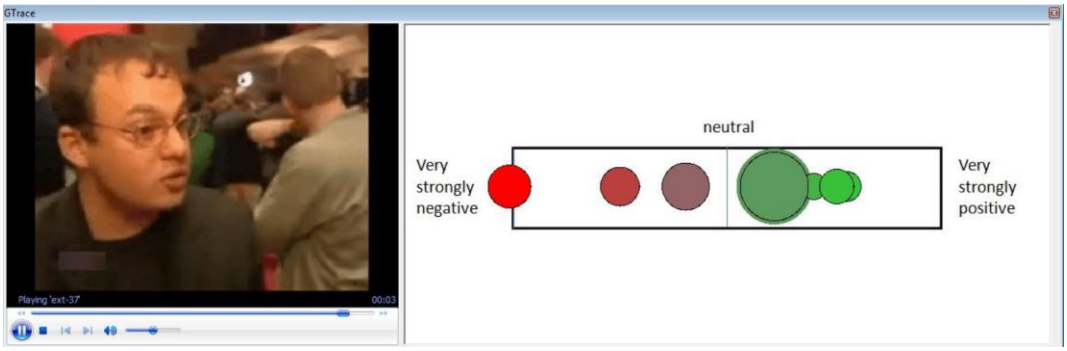


Figure 3-8. GTrace, the General Trace Program from Queen's, Belfast (in [24]**).**

Reflecting the lack of easy-to-integrate, methods for intuitive and quick assessment of emotions, other solutions for affect assessment have been developed, mostly in an ad-hoc fashion. The assessment methods implemented in Reis and Correia's Imaginary Friend [78] (Figure 3-9) and Morris et al.'s Mood Map [62] (Figure 3-10) are mentioned as examples.

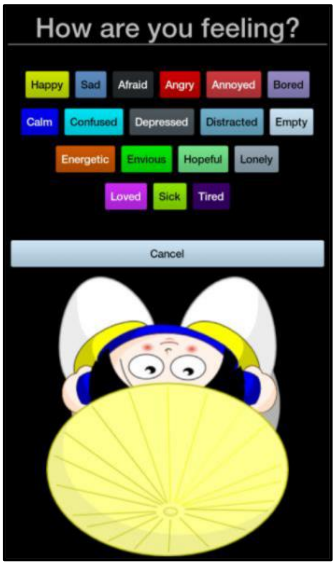


Figure 3-9. The Imaginary Friend (in [78]**).**

Figure 3-10. The Mood Map (in [62]**.**

The Imaginary Friend is a game that asks players about their emotional states upon detecting changes in their galvanic skin response (GSR). Since changes in GSR indicate changes in physical arousal, the game is potentially able to react to emerging emotional experiences. However, in order to assess which specific emotion is being experienced, the game asks players to select an emotion word out of a previously compiled list. To enhance user-friendliness, the list items' backgrounds are colored in accordance with some studies that established relations between emotions and colors (among them, Plutchik's [73]). The Mood Map, on the other hand, is a touch-screen translation of Russell's circumplex model of emotion, in which users indicate their current affective state by pointing its location on a two-dimensional, valence-arousal space. Morris et al. have used it, along with other single-dimension mood scales, to study and improve user emotional awareness in moments of stress.

Although both approaches may be viable, a search on the literature has not yielded studies that report formal validations of those tools.

3.6 Studies

Given the CAAT's novelty, two careful validation studies have been conducted to assess if it truly is reliable, and an overall valid tool for the assessment of emotional reactions: study 1 (see Section 3.6.1) evaluated the CAAT's usability along several relevant dimensions, and study 2 (see Section 3.6.2) gathered evidence regarding some important aspects of psychometric tools.

3.6.1 Study 1, CAAT Usability

The CAAT can be perceived as a list of emotion words or – what may be more accurate – an organized arrangement of emotions. Indeed, one of the CAAT's most distinctive properties is its layout, the way that its 25 emotion words are arranged and colored. To understand if this arrangement is, indeed, an effective ordering scheme for the selection of emotion words, a study [17] was conducted comparing the CAAT's usability against that of an alphabetically ordered list featuring the same 25 emotion words as the CAAT, the Ordered Emotion List (OEL) (see Figure 3-11). Both tools allow the user to select the one or two emotion words that better describe his/her current emotional state. This comparison allows to understand if the CAAT's arrangement is more adequate for the selection of emotion words than the traditional alphabetical ordering. Additionally, to provide sounder grounding to our study's conclusions, participants were also asked to report their emotional state on a 9-point SAM (i.e., given its wide acceptance by the research community, the SAM has been considered a standard against which to compare the results of the CAAT and the OEL).

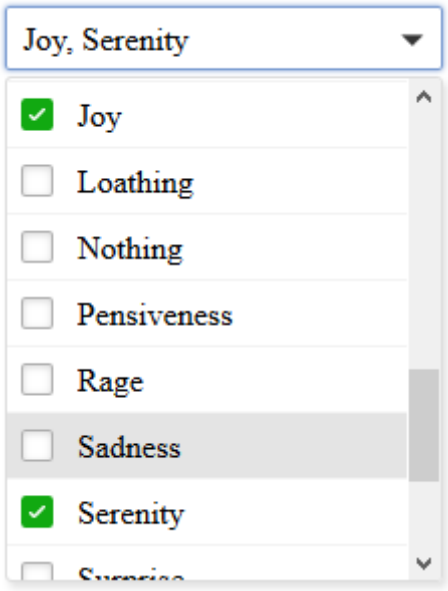


Figure 3-11. The Ordered Emotion List (OEL). A selectable, alphabetically ordered list of emotions, featuring the same 25 emotion words as the CAAT.

3.6.1.1 A Preliminary Score System

A tentative CAAT score system (see Table 3-2) was used in this study, to enable to understand if the responses with this tool correlate more strongly with the SAM's than do those of its alphabetically ordered counterpart, the OEL. This score is not related with the CAAT's current score system (see Table 3-1, in Section 3.3). Indeed, as it happens, by the time this study was conducted, Warriner et al.'s [90] survey was not yet published.

To design this first score system, a small scale survey was conducted, in which 12 users were asked to rate each of the CAAT emotions in terms of valence, arousal and dominance, using a nine-point SAM [11]. The ratings were then averaged and binned into seven equally spaced "containers" (as much as the emotions on each opposing pair of axes, plus the central "nothing"). Afterwards, the container index (1 to 7) was used as the score for each emotion's valence, arousal and dominance dimensions (Table 3-2, 2nd to 4th columns, respectively). Finally, since valence and arousal are the prevalent dimensions in most of the literature and valence is the one that accounts for most of the variance, both valence and dominance scores have been merged

into a single score, S_1. Finally, the S_2 score is the obtained SAM arousal score directly. These simple equations can be found in Figure 3-12.

Table 3-2. The Study 1 CAAT's preliminary score system.

Emotion	SAM Valence	SAM Arousal	SAM Dominance	CAAT S_1	CAAT S_2
Terror	1	7	1	**1**	**7**
Fear	2	6	2	**2**	**6**
Apprehension	3	5	3	**3**	**5**
Annoyance	3	5	5	**3.7**	**5**
Anger	2	6	6	**3.3**	**6**
Rage	1	7	7	**3**	**7**
Admiration	7	5	5	**6.3**	**5**
Trust	6	4	5	**5.7**	**4**
Acceptance	5	4	4	**4.7**	**4**
Boredom	3	3	5	**3.7**	**3**
Disgust	2	5	3	**2.3**	**5**
Loathing	1	6	2	**1.3**	**6**
Ecstasy	7	7	7	**7**	**7**
Joy	6	6	6	**6**	**6**
Serenity	5	3	5	**5**	**3**
Pensiveness	3	3	3	**3**	**3**
Sadness	2	2	2	**2**	**2**
Grief	1	1	1	**1**	**1**
Vigilance	4	7	7	**5**	**7**
Anticipation	4	6	6	**4.7**	**6**
Interest	4	5	6	**4.7**	**5**
Distraction	4	3	3	**3.7**	**3**
Surprise	4	6	2	**3.3**	**6**
Amazement	4	7	1	**3**	**7**

$$S_1 = \frac{2V+D}{3} \; ; \; S_2 = A$$

Figure 3-12. Equations used for Study 1's preliminary S1 and S2 scores

It should be emphasized that, for the purposes of this study, the S_1 and S_2 scores are *properties of the emotion words* featured on both the CAAT and the OEL, rather than properties of any of those particular tools. This means that a

given selection of emotion words has the same S_1 and S_2 values, regardless of the tool through which the selection was made.

3.6.1.2 Methods

This study was conducted on a quiet University classroom, in which participants were asked to provide information about their emotional reactions to stimuli – six photos taken from Lang et al.'s [53] International Affective Picture System (IAPS) collection. The IAPS consists of a large database of photos that have been validated to evoke emotional responses on viewers. Therefore, after a stimulus was presented to each user (for a minimum of 7 seconds), he/she was required to express about his/her emotional reactions on the CAAT, the OEL and the SAM (across participants, the tools and stimuli order was random). The tests were implemented as a computer application and the results and response times were automatically recorded on a database. To be able to measure and compare the CAAT's and the OEL's response times, and since the CAAT features a click-to-activate mechanism by design, we have implemented the OEL as a drop-down list. Therefore, whenever a user wants to make a selection on any of those tools, he/she is required to activate it beforehand by clicking on its respective "open" button. This action triggers an event that, once detected, will be used to start counting the response time, i.e., the time that takes a user to select one or two emotions among the possible choices.

Finally, participants were also requested to answer a short questionnaire for each of both the CAAT and the OEL. This questionnaire was composed of three seven-point Likert scales and a Semantic Differential Scale (SDS) composed of six bipolar pairs of adjectives extracted from Benedek and Miner's [9] Microsoft Reaction Card methodology, that account for six general dimensions of satisfaction. The three Likert scales were as follows: (1) "*this tool enables me to accurately express my emotions*" (1-Totally disagree to 7-Totally agree); (2) "*It is easy to find the emotions I'm looking for*" (1-Totally disagree to 7-Totally agree); and (3) "*the emotions on this tool were ordered*" (1-Totally disagree to 7-Totally agree). Finally, the questionnaire's usability SDS asked "*how would you describe this tool*", and its six pairs of bipolar adjectives were: *boring-fun*,

intimidating-friendly, *confusing-clear*, *amateur-professional*, *slow-fast* and *dated-cutting edge*.

3.6.1.3 Participants

Participants were recruited through direct invitation or snowball sampling, yielding a total of 62 participants (47 male). Ages ranged from 18 to 58 years, averaging 24.6. All were Portuguese or had Portuguese as their main language and all reported advanced English understanding and fluency. Roughly half of the participants (49.2%) were undergraduate students while the rest had university graduate (30.2%) or post-graduate degrees (20.6%).

3.6.1.4 Results

Response Time results

As previously stated, the response times were recorded automatically for both the CAAT and the OEL. Since the online questionnaire presented 6 stimuli, each user interacted 6 times with each tool. Therefore, in order to understand how easily users familiarize themselves with the tools – i.e., how learnable the tools are – the across-participants average values have been calculated, for each of the 6 times participants interacted with both tools. These mean values are graphically represented in Figure 3-13. It can easily be verified that mean response times are tendentially lower for the CAAT than for the OEL, except at the time of the first interaction, when the latter registered inferior mean response time.

Besides these automatic response time measurements, participants were also explicitly asked how fast they believed their responses with each tool were. As it can easily be verified in the fifth scale of Figure 3-17 and Figure 3-18 (the slow-fast adjective pair), the differences between these two tools are much more pronounced where this subjective time is concerned.

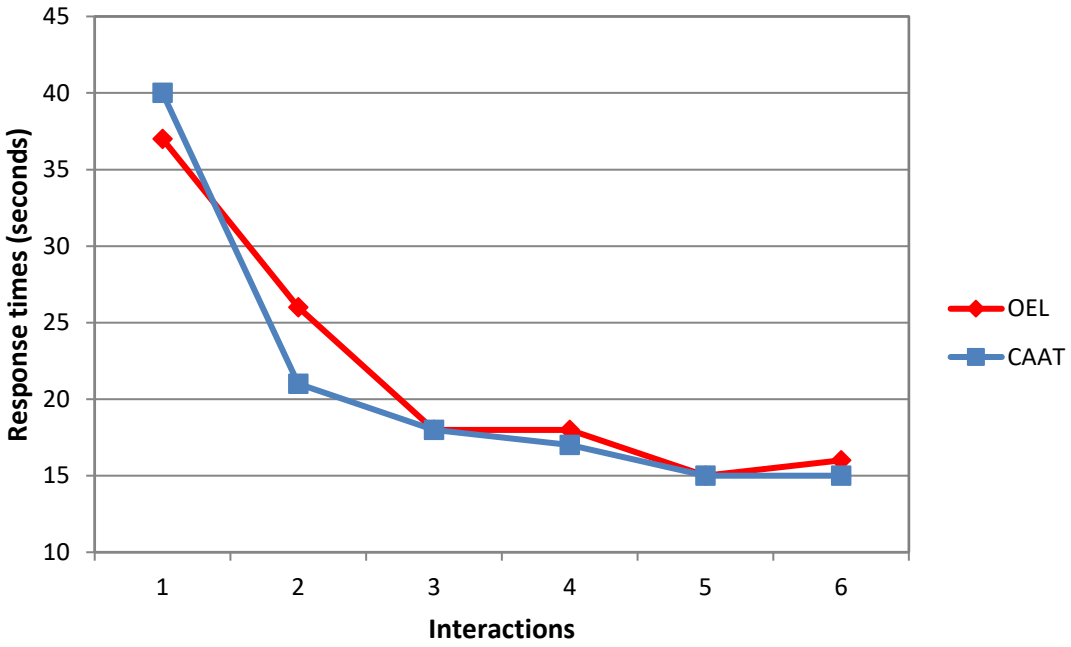


Figure 3-13. Across-participants average values of the automatically measured response times, for each of the six times the CAAT and the OEL were used.

User Experience Evaluation

The answers to the three Likert scales of the final user experience questionnaire also generally favored the CAAT over the OEL. Indeed, they judged the former to be more adequate than the latter for expressing emotions (Figure 3-14). Additionally, participants also tend to think it is easier to find the emotion words they are looking for on the CAAT than on the OEL (Figure 3-15), and they also perceive emotion words to be more ordered when they are laid out following the CAAT's arrangement than under the OEL's traditional alphabetical ordering (Figure 3-16).

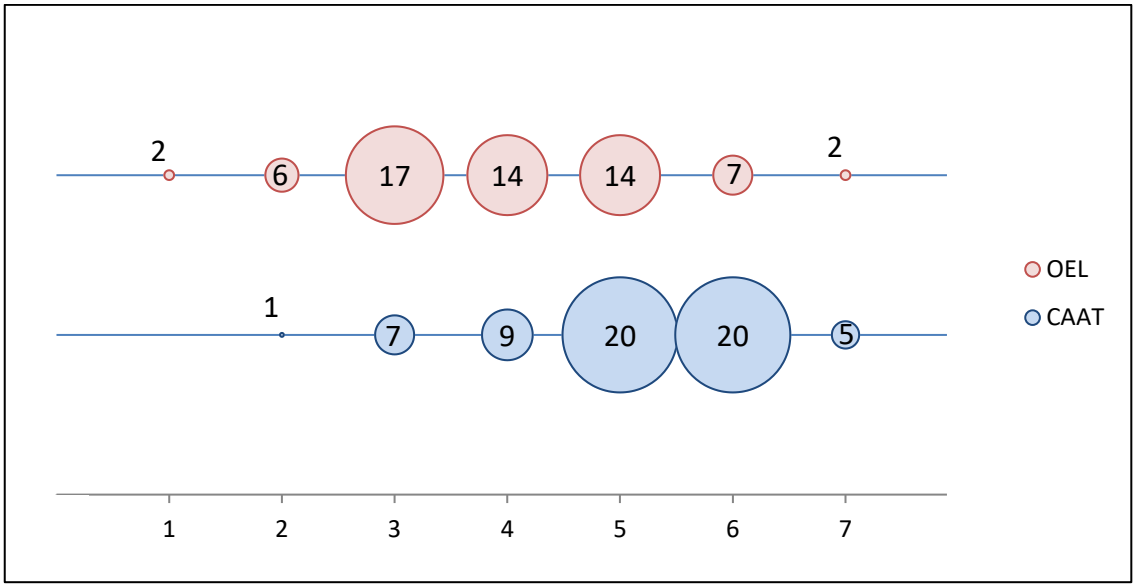


Figure 3-14. Answers to question "*This tool enables me to accurately express my emotions*" (1 - Totally disagree, 7 - Totally agree).

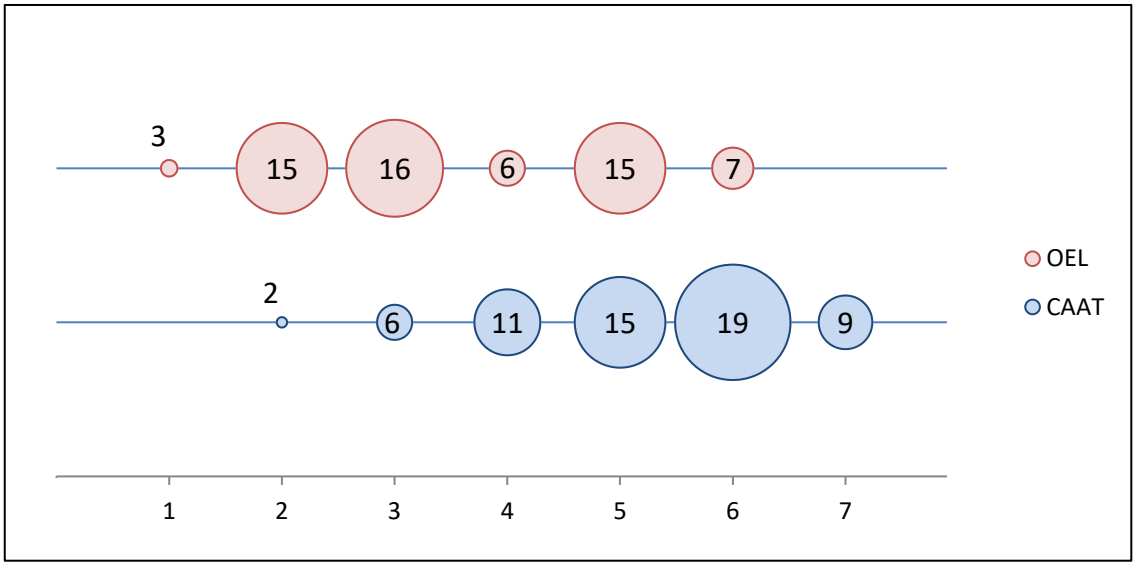


Figure 3-15. Answers to question "*It is easy to find the emotions I'm looking for*" (1 - Totally disagree to 7 - Totally agree).

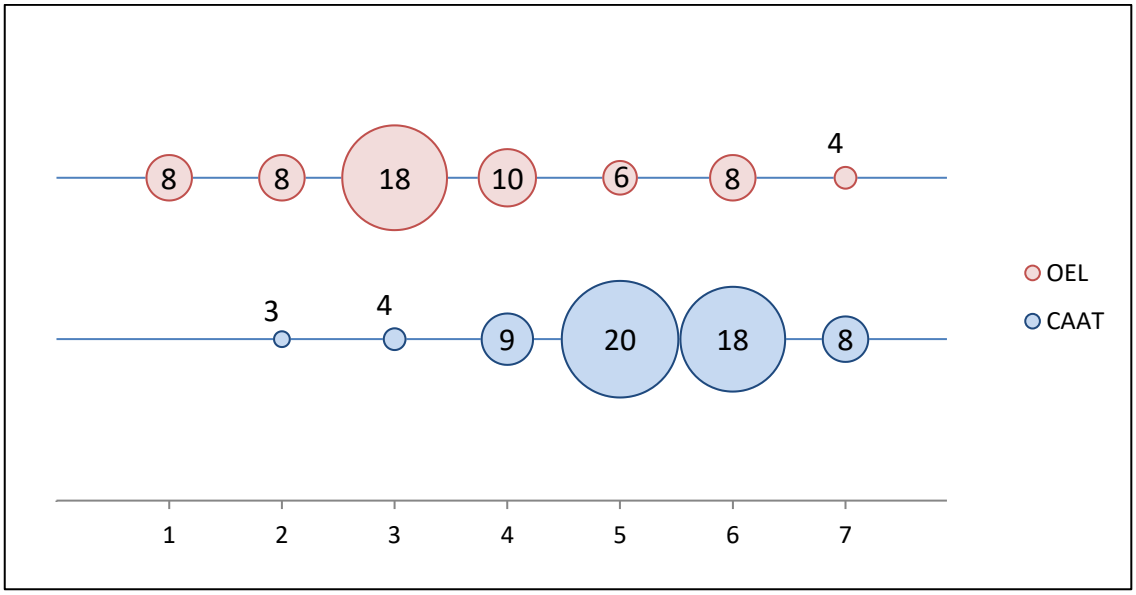


Figure 3-16. Answers to question "*The emotions on this tool are ordered*" (1 - Totally disagree to 7 - Totally agree).

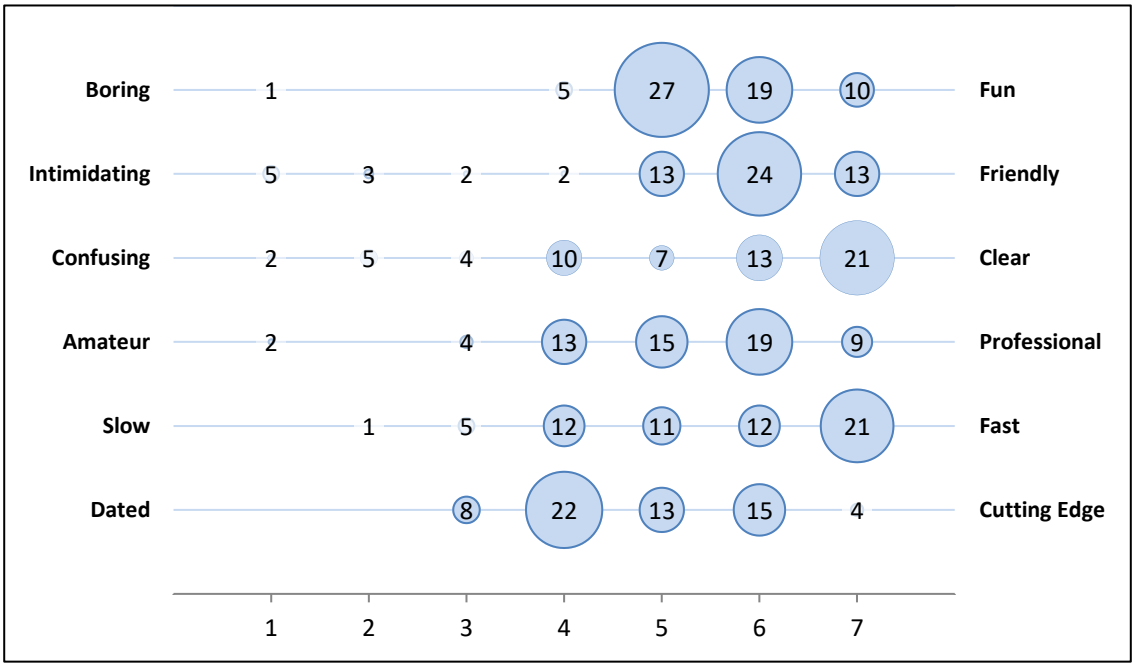


Figure 3-17. Answers to question "*How would you describe this tool?*" regarding the CAAT.

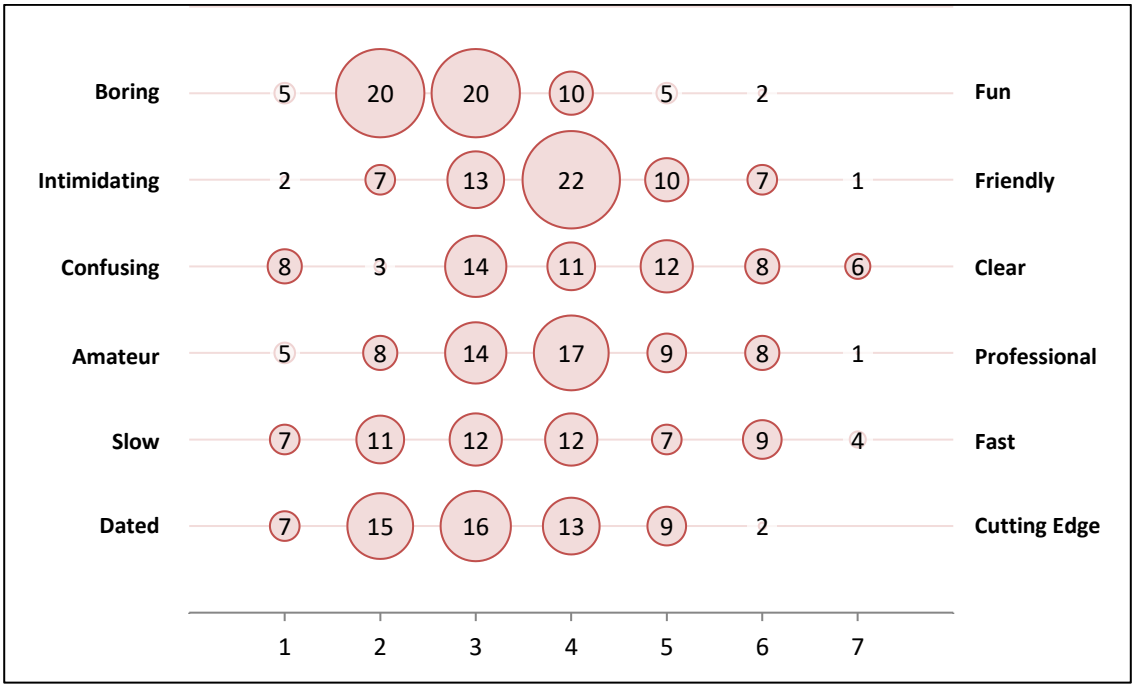


Figure 3-18. Answers to question "*How would you describe this tool?*"regarding the OEL.

Associations with the 9-point SAM

Perhaps the most significant results of the data analysis are the Pearson correlation coefficients between the CAAT and the OEL results against those of the 9-point SAM (see Table 3-3). Since S_1 is derived from the emotion words' valence and dominance scores (see Section 3.6.1.1), this score was correlated to the SAM valence and dominance scores, while the S_2 was correlated against the SAM arousal.

Table 3-3. Correlations between SAM valence, arousal and dominance scores and CAAT/OEL S_1 and S_2 scores

		CAAT		OEL	
		S_1	**S_2**	**S_1**	**S_2**
SAM	**Valence**	0.857		0.828	
	Arousal		0.543		0.434
	Dominance	0.695		0.649	

On their own, the CAAT S_1 score is very strongly correlated with SAM's valence and dominance scores (0.857 and 0.695 respectively) while its S_2 score correlates strongly with SAM's arousal (0.543). Regarding the OEL, its S_1 score is also very strongly correlated with the SAM's valence and dominance scores (0.828 and 0.649, respectively) and its S_2 score has a strong correlation with SAM's arousal (0.434). All correlations had their p-values below 0.001.

It is noteworthy that the CAAT scores manifest systematically higher coefficients, even though the very same emotion words were available for selection in both tools.

3.6.2 Study 2, CAAT's Viability

While Study 1 (see Section 3.6.1) was mainly concerned with evaluating the CAAT along some general dimensions of usability, many essential questions still remained to be answered and more conclusive testing was required before the CAAT could be considered a reliable and valid tool. Therefore, a broader scale study [18] was designed and conducted, under the general hypothesis that the CAAT is a viable option to perform quick and consistent assessments of emotional reactions. To verify it, the study gathered evidence regarding the CAAT's construct validity (*does a tool measure what it is intended to?* – in this case, emotional reactions), internal and temporal reliability (*how accurate and consistent are its measurements?*) and response times (*how quick is the tool to be used?*).

3.6.2.1 Methods

As previously stated, in order to better understand the CAAT's viability as an adequate tool for emotional reaction assessment, evidence for its construct validity and test-retest reliability is required. To gather it, a web application was developed, serving an online questionnaire which displayed 12 stimuli to participants, one at a time, extracted from a database of normatively rated affective stimuli (more details in Section 3.6.2.2).

The experiment was structured in three sessions, separated by intervals of time of varying duration: the first two sessions were separated by an interval with a minimum of 30 seconds, while sessions 1 and 3 had a 7-day minimum

interval in between (Figure 3-19). The correlation of the results across different time frames, allows to conclude about the tool's inherent stability.

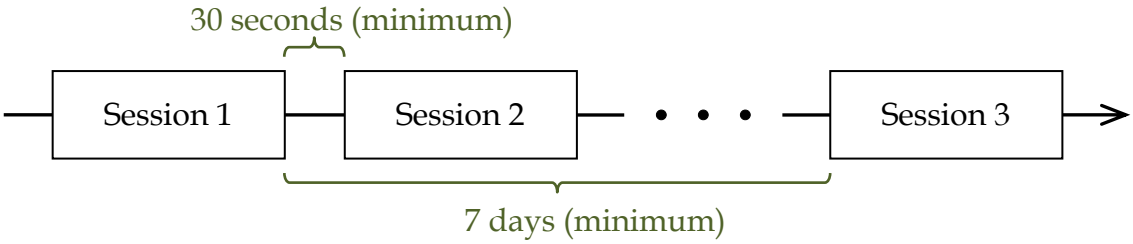


Figure 3-19. The temporal plan of the experiment: three sessions separated by intervals of time of different duration. Sessions 1 and 2 were separated by a 30-second (minimum) interval and sessions 1 and 3 had a 7-day (minimum) interval in between.

In each session, the online questionnaire presented each participant with the same, randomly-reordered 12 stimuli and asked them to express their emotional reactions using the CAAT (downloaded from [12]), while registering the time spent on the evaluation of each stimulus (every time the CAAT was opened and afterwards closed with a selection made, the system registered the number of seconds it had remained open). The last page of the questionnaire featured a text box, inviting participants to provide feedback.

With the goal of assessing the CAAT's temporal reliability, special care was taken to choose the durations of the intervals that separated the experiment's sessions. The 30-second interval was chosen because a period of time was needed, short enough so that participants would not give up and to minimize the likelihood of occurring life events that could impact their affective states, and yet long enough so as to minimize the potential interference from short-term memory in the users' responses. Indeed, Atkinson and Shiffrin [6] have suggested the 30-second interval to be the upper limit of short-term memory for some simple tasks, after which its performance degrades. On the other hand, the 7-day interval is long enough to allow us to extrapolate and conclude about the CAAT's long-term reliability, while minimizing the chance of eventual interferences from "cold" mnesic processes (memory not mediated by emotions). In addition, the stimuli were always randomly reordered

between sessions, so as to prevent participants from "chunking" [61] their responses to stimuli sequences.

Participants were reminded via e-mail after the 7-day interval to return to the website and complete their participation by answering to the questionnaire's third session. Both 30 seconds and 7 days were the minimum durations for the intervals between sessions, implying that responses were accepted from sessions separated by more than 30 seconds (1st and 2nd sessions) and more than 7 days (1st and 3rd sessions). Indeed, the amount of elapsed time between sessions 1 and 2 ranged from 34 seconds to over 1 hour, averaging 102 seconds, and the period between sessions 1 and 3 ranged from 7 to 54 days, averaging 13 days.

Although, for the purposes of assessing a tool's test-retest reliability, the conditions under which the measurements are performed should be the same, enforcing this requirement in this study would have made it impossible to conduct the tests with remote users. Nonetheless, in order to minimize the impact of uncontrolled variability in the measurement conditions, the web application's homepage displayed an introductory message that, besides greeting and informing participants about the study, asked them to answer the questionnaire in places or situations where they would not be easily interrupted or distracted, and also without performing parallel activities with inherent and possibly interfering emotional appeal (like hearing music or chatting with others).

3.6.2.2 The Affective Stimuli

Since this study was designed to understand and validate the CAAT as an emotional reaction assessment tool, it was desirable to have a measure of control (criterion validity) over the elicited emotional responses. Ideally, one would be able to provoke a specific emotion on a subject and then use the CAAT to see if the same emotion is reported. However, it is very difficult to accurately predict how someone will react to a given stimulus, without having some intimate knowledge about that someone else beforehand – an impossibility, given the scale and web-based nature of this experiment. Acknowledging this fact, the research community has proposed a number of

databases of normatively rated affective stimuli, with the explicit aims of allowing better control in the selection of emotional stimuli and allowing the comparison and replication of experiments across different studies and research labs [29].

The affective stimuli used in this experiment came from the Geneva Affective Picture Database (GAPED) [29], a database of normatively rated, emotion-eliciting pictures. The GAPED contains 730 pictures chosen according to their specific contents, and distributed between three distinct categories: *negative*, *neutral* and *positive*. The negative category has four topics: snakes, spiders, explicit violation of human rights and animal mistreatment. The neutral category is composed of pictures of inanimate objects, buildings and furniture. Finally, the positive category contains pictures of human activities and animal babies. The GAPED was created to increase the availability of visual emotion stimuli to the research community. Since viewer habituation can compromise the value of affective stimuli, very careful approach was followed to prevent the images from being downloaded from the system (for instance, images were finely tiled). The 12 stimuli used in the study are briefly characterized in Table 3-4.

Table 3-4. Stimuli used in the experiment, and their respective GAPED valence and arousal scores.

Description	Reference (GAPED)	Mean Valence (0-100)	Mean Arousal (0-100)	Category (GAPED)
Snake	Sn049	18.035	69.814	Negative
Spider	Sp136	9.521	78.436	Negative
Prisoners of war	H032	1.929	77.861	Negative
Animal Cruelty	A055	9.603	78.258	Negative
Child poverty	H059	29.62	36.969	Negative
Lab mouse	A124	47.95	48.518	Neutral
Bookcase	N002	55.04	41.503	Neutral
Mother and baby	P033	96.202	15.369	Positive
Resting tiger	P100	89.699	39.643	Positive
Kitten	P101	97.818	5.851	Positive
Canoe ride	P105	81.696	66.009	Positive
Skier	P124	77.764	51.203	Positive

3.6.2.3 Participants

Participants were recruited through snowball sampling, with individuals being initially sampled from student and departmental mailing lists of Nova University of Lisbon. This procedure yielded 133 participants (52.6% male) answering to sessions 1 and 2; of these, 106 (49.0% male) also answered to session 3.

Although the majority are Portuguese (accounting for 69.2% of the total and 67.3% of those who have completed session 3), other nationalities are also represented, mainly the US (15.0% of the total count and 18.3% of the total who answered the 3rd session) and Spain (8.3% of the total count and 5.8% of the total who answered to session 3), but also, to a lesser degree (1 or 2 participants), Germany, India, Latvia, Russia, Guatemala, Namibia and Brazil. Regarding education, most of the participants have university graduate or post-graduate degrees (78.9%), and ages ranged from 20 to 77 years old, averaging 34.9 years old.

3.6.2.4 Results

Temporal Reliability

To learn about the CAAT's temporal reliability, the Pearson's correlation coefficient was computed between the CAAT valence and arousal scores (see Table 3-1) of each subject's answers to sessions 1 and 2 (133 participants; 30'' of minimum interval between stimuli presentation, for short term temporal reliability) and to sessions 1 and 3 (106 participants; 7 days of minimum interval between stimuli presentation, longer-term temporal reliability). That is, for each participant and each stimulus, the CAAT valence score obtained in session 1 was correlated with the ones obtained in sessions 2 and 3 (likewise for the CAAT arousal scores). The obtained coefficients can be found in Table 3-5.

Table 3-5. Inter-session correlations for CAAT Valence and Arousal scores, for sessions separated by time intervals of different duration.

	Sessions 1 & 2	Sessions 1 & 3	Sessions 1 & 3 (7-16 days in-between)	Sessions 1 & 3 (17-26 days in-between)	Sessions 1 & 3 (> 26 days in-between)
n[(a)]	1596	1272	1020	84	168
Valence	0.935**	0.849**	0.870**	0.889**	0.753**
Arousal	0.828**	0.654**	0.677**	0.596**	0.643**

** *p*-values < .001

[(a)] This sample size corresponds to the number of pairs of answers for the same stimulus between sessions, per participant (12 pairs per participant).

The very strong correlation between the scores of sessions 1 and 2 is very elucidating about the tool's test-retest reliability within very short timeframes. Regarding the test-retest reliability across a longer timeframe, the scores obtained in sessions 1 and 3 are also strongly correlated, though to a slightly lesser degree (Table 3-5, third column).

Because the 7-day duration was a minimum value for the interval of time separating sessions 1 and 3, there was considerable heterogeneity in the time that had elapsed between these sessions, across participants. This allowed to explore the tool's temporal reliability in different timeframes by creating three bins of "elapsed time between sessions 1 and 3" (7-16, 17-26 and over 26 days) and, afterwards, compute the correlations between the scores for each participant (Table 3-5, columns 4 to 6).

Construct Validity: Association between GAPED and CAAT Scores

The GAPED database includes information about each stimulus, namely, its mean valence and arousal ratings. Therefore, the stimuli can be sorted in terms of their valence and arousal scores. For each stimulus, the mean of the CAAT valence and arousal scores obtained in the first session were computed across participants (n = 133), yielding 12 pairs of mean CAAT valence and arousal scores (one for each stimulus). Afterwards, a Spearman's rank-order correlation was run to determine the relationship between the resulting mean CAAT scores and the mean GAPED valence and arousal. A remarkably strong

positive correlation was found to exist between the mean CAAT valence and the mean GAPED valence (n = 12, $r_s(10) = .949$, $p < .001$) and a lower, but still strong positive correlation between the mean CAAT arousal and the mean GAPED arousal (n = 12, $r_s(10) = .696$, $p < .001$).

Construct Validity: GAPED Category Differentiation

As previously stated (Section 3.6.2.2), the GAPED has its stimuli classified into three, valence-based categories: *negative*, *neutral* and *positive* [29]. Therefore, the CAAT can be expected to produce overall different valence scores across these categories. To verify this hypothesis, a one-way between subjects ANOVA was conducted, using the answers gathered during the first session, averaged per user and each of the three categories (133 participants, 3 mean scores per participant, n = 399).

A significant difference was found between the valence scores produced for stimuli across the three categories [$F(2,396) = 339.39$, $p < .001$]. After conducting a post hoc comparison using the Tukey Honest Significant Difference (HSD) test, it was found that the mean CAAT valence scores are significantly different across all of the GAPED categories [negative (M=3.74,SD = 0.61), neutral (M = 5.39, SD = 1.23) and positive (M = 6.49, SD = 0.59)]. These results are illustrated in Figure 3-20.

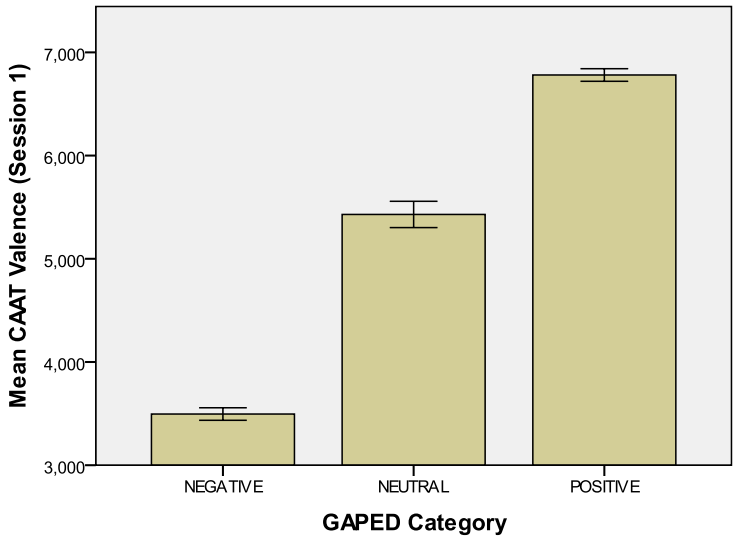


Figure 3-20. Mean CAAT Valence score by GAPED categories (error bars: +/- 1 standard error).

While arousal may not be the most discriminating dimension across GAPED categories [29], valence ratings are seldom independent from arousal levels [85]. This means that one can expect to find differences in the produced CAAT arousal scores for the three GAPED categories, even if not so pronounced. To test this hypothesis a similar analysis was conducted, now using the CAAT arousal scores.

The ANOVA test has indicated a significant difference [$F(2,396) = 39.11$, $p < .001$] and Tukey HSD has detected significant differences between stimuli from the neutral category and the ones from both positive and negative; however, no significant difference was found between the arousal scores for stimuli belonging to the GAPED positive and negative categories [negative ($M = 4.75$, $SD = 0.58$), neutral ($M = 4.14$, $SD = 0.67$) and positive ($M = 4.63$, $SD = 0.52$)]. These results are illustrated in Figure 3-21.

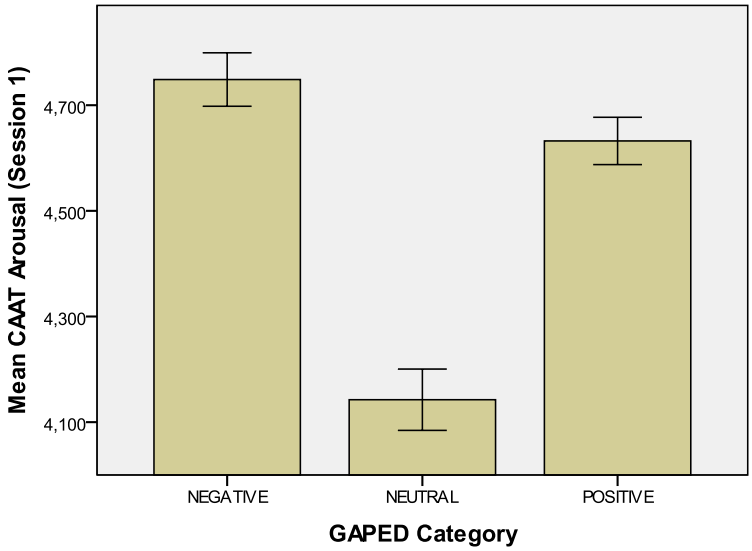


Figure 3-21. Mean CAAT arousal score by GAPED categories (error bars: +/- 1 standard error).

Burden of the CAAT: Response Time Results

The first CAAT experiment, Study 1 (see Section 3.6.1), reported response times that decreased each time participants interacted with the CAAT. Indeed, the mean response time was around 40 seconds for first contacts, tending to 15 seconds as users continued to use the tool. The findings of this study confirmed

the existence of this process/interaction-learning curve, although the response times were lower. The mean response time was 28 seconds for the first use, tending to 7 seconds with continued use along the study sessions (Figure 3-22).

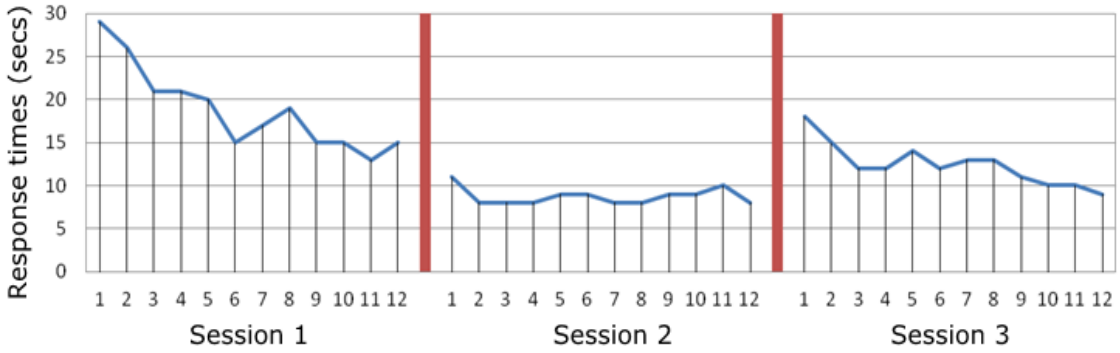


Figure 3-22. Mean response times, each time participants interacted with the CAAT (vertical black lines). The red bars represent the intervals of time that separated our study's three sessions (the first represents the 30-second minimum interval and the second represents the 7-day minimum).

After the 7-day minimum interval between sessions, the mean of the first-time use was significantly lower (18 seconds), tending to a 7-second intra-session asymptote. The most important aspect to note here is the interaction learning that seems to have taken place between sessions 1 and 3, lowering the first interaction time from 28 to 18 seconds, before tending to the same, 7-second asymptote.

Burden of the test: Participants' Feedback

As previously said, the questionnaire contained a text box for participants to enter feedback. In general, participants were very positive about taking part in this study, with the CAAT receiving positive appraisals. Interestingly, a few participants stated that they would have liked to be able to explain the rationale behind their emotion selection for some of the stimuli. Other participants expressed the need for more emotion gradients, while others suggested the addition of new emotion words to the featured set.

3.7 Use Cases

Illustrating the range of applicability of the CAAT, there are other research projects on which the tool was used for assessing people's emotions.

For instance, it was used in the scope of a research project intended to enhance the experience of remotely viewing sports events [20]. To that end, an emotion-oriented chat system was implemented, that made use of both mobile phone and television to allow viewers to exchange emotions and commentaries about the event being watched. Needless to say, emotions must be gathered prior to being exchanged, and this is where the CAAT plays its part: a sports-adapted version of the CAAT was implemented in the system and used to ask users what emotion they would like to share with their interlocutor. This adaptation did not make use of any score system, as only the emotion words themselves are exchanged between users, along with some optional text messages.

The CAAT was also used by Madeira et al. [56], who have developed a smartphone application geared towards users with stuttering speech disorder. It allows its users to register contextual information regarding stuttering episodes, in order to improve self-awareness and control. Among other relevant data, the system used the CAAT to collect the emotional state that the user was experiencing at the time of the stuttering episode.

As a last example, noticing that the availability of progressively more advanced software and less expensive hardware allows museums to preserve and share artifacts digitally, Alelis et al. [2] have conducted a study on the emotional connectedness that people belonging to two target populations - young adults and elderly, 18-21 years and 65 years or older, respectively – have with interactive, tridimensional digital models of artifacts. Concretely, the authors collected and analyzed the time spent in interacting with the virtual object, as well as their enjoyment and emotional responses, concluding that digital modalities were enjoyable and encouraged emotional responses, regardless of age; moreover, seeing the physical artifacts after the digital ones did not reportedly lessen their enjoyment or emotions felt. These results suggest clearly that digital artifacts are effective in stimulating emotional responses,

both in younger and older people. The emotional assessments – or, more explicitly, the association of valence and arousal values to participants' affective reactions to digital artifacts –, were accomplished using the CAAT.

3.8 Discussion

The two presented studies have provided evidence towards the CAAT being adequately reliable as a tool for quick assessment of emotional reactions.

The results of Study 1 (Section 3.6.1) support the validity of the tool's overall design. Indeed, as previously stated, the CAAT, the OEL and the SAM were presented six times to each user for emotion assessment. The averages of the CAAT's and the OEL's automatically recorded response times, for each of these six contacts, reveal the learning curves of both tools (see Figure 3-13). Bar the first contact, when OEL had an inferior mean response time, the results generally favor the CAAT. This first contact exception may be explained by the ubiquity of drop-down lists and traditional alphabetical ordering schemes in user interfaces – as previously stated, the OEL is a drop-down list of alphabetically ordered emotion words. In contrast, the CAAT has a less common arrangement which may explain its longer first contact.

However, one of the items of Study 1's usability SDS required users to describe both tools using a seven-point scale between the slow-fast adjective pair. Here, the differences between the CAAT and the OEL become more evident, as participants judged the CAAT to be significantly faster (see the 5th item of Figure 3-17 and Figure 3-18). This may be explained in light of Czerwinski et al.'s [27] work, who have discovered an association between subjective duration estimates and the success of a user interface design. Indeed, Czerwinski et al state that, "*as a task becomes more difficult (perhaps due to an interface design), participants will likely overestimate how long that task takes. In contrast, if participants can complete the task, either with minor assistance or on their own, they are more likely to underestimate how long that task took in comparison to the actual task time*". Applied to the results of Study 1, these conclusions support the claim that the CAAT's design is more successful for the task of emotion word selection than OEL's alphabetical ordering.

However, more than an effective design for the selection of emotion words, the CAAT aims to be a viable emotion assessment tool. Therefore, if the SAM is considered as a widely accepted standard for emotional assessment, a comparison between the correlations between the answers obtained from the CAAT/OEL and those of the SAM reveals that, since the CAAT's scores are more strongly correlated with the SAM's, then the CAAT is more adequate to the task of emotion assessment than is the OEL, with its alphabetically ordered emotion words (see Table 3-3).

Finally, the participants' answers to the three Likert scales and the usability SDS of Study 1 also favor the CAAT over the OEL, revealing that users thought it to be an overall better tool to accurately express their emotional experiences (see Figure 3-14), being more ordered (see Figure 3-16) and easier to find the wanted emotions (see Figure 3-15), while also finding the CAAT to be more fun, friendly, clear, professional, fast and cutting-edge (see Figure 3-17 and Figure 3-18).

Complementarily, Study 2 provided a deeper insight into some essential aspects of the CAAT as a psychometric tool. Indeed, it explored its construct validity (*does a tool measure what it is intended to?*), internal and temporal reliability (*how accurate and consistent are its measurements?*) and response times (*how quick is the tool to be used?*).

As a testament to its overall stability, both short-term and longer-term correlation coefficients are significantly strong for both CAAT scores (valence and arousal) (see Section *Temporal Reliability*). This appears to be consistent across timeframes of varying durations, since the correlations remained strong even after more than 26 days had passed in-between answers (see Table 3-5). This supports the hypothesis that the CAAT has adequate test-retest reliability; thereby the HCI community (or indeed other communities in need of reliable quick emotional assessment) can expect the tool to produce consistent results across experiments and uses.

The strong but not perfect (i.e., not equal to 1) inter-session correlations for the CAAT scores suggest that users did not tend to choose the same emotion words for the same stimuli across sessions, as it would likely happen if they had memorized their previous answers (a "cold" mnesic task). Although both

scores show strong correlation, the coefficients are systematically higher for the valence score. This may be explained in light of the findings of Study 1, which suggested that the process of emotion word selection on the CAAT is mainly driven by valence. In other words, when pondering about which emotion words to choose, users tend to select like-valenced emotions without so much consideration for their arousal component. This scenario, in terms of the CAAT scoring system, would result in highly correlated valence scores and slightly more variation on the arousal scores – something that is in line with the observations of Study 2.

For each individual stimulus, both the mean CAAT valence and arousal scores produced across participants also did correlate strongly with the mean valence and arousal ratings included in the GAPED database, respectively (see Section 3.6.2.4, *Construct Validity: Association between GAPED and CAAT Scores*). This implies that, for each stimulus, the CAAT produces scores that agree significantly with the normative ratings of the GAPED database, for both scores.

In addition, an analysis of variance (ANOVA and Tukey HSD) has indicated that the tool had also produced significantly different valence scores for stimuli belonging to different GAPED categories, discriminating between *positive*, *neutral* and *negative* stimuli – i.e., the CAAT valence score is consistent with the valence-based categories of the GAPED database (see Figure 3-20). This convergence is an indicator of the tool's psychometric capabilities and, consequentially, of its construct validity. On the other hand, an analysis of variance on the CAAT arousal scores did not identify significant differences between images belonging to the *negative* and *positive* GAPED categories; it has only managed to differentiate stimuli belonging to the neutral category from the other two (see Figure 3-21). This may be explained in light of the fact that emotions on the extremes of the valence scale (negative and positive) tend to be more arousing, i.e., there is no literal independence between the valence and arousal dimensions of affective spaces (these dimensions are not truly orthogonal) [86]. Put simply, pleasurable (e.g., a kitten) or unpleasurable (e.g., prisoners of war) stimuli tend to be more arousing than stimuli of a more neutral nature (e.g., a bookshelf) – and the CAAT is sensible to this. Another

likely reason is the noticeable overlap between categories that seems to exist in the GAPED database, where the arousal dimension is concerned [29].

Study 2 also revealed that the CAAT response times are also considerably short, especially considering that the tool asks users to select two emotion words out of a list of 24 – a task not likely to be common for the majority of people. Moreover, the response times tend to lower quickly towards a 7-second asymptote with use, thereby hinting at the adequacy of the tool's design and confirming its quickness of use (see Figure 3-22).

Participant feedback is another interesting result of Study 2, as it reveals that the CAAT had the users thinking about their emotional experiences. Substantiating this claim, some feedback was received suggesting to add more emotions, and other messages in which participants tried to justify particular selections of emotions to a given stimulus. In addition, a couple of emails were received from participants, in which they justified the disparity that – they assumed – existed between their answers to particular stimuli that – they also assumed – would be very consensual. More than evidencing some eventual pressure for conformity, these episodes further confirm the emotional awareness promoted by the CAAT.

Finally, regarding the CAAT's ease-of-use, it should be highlighted that since the sessions of Study 2 were not supervised, the only instructions that participants received regarding interaction with the CAAT were provided by the questionnaire page and the CAAT's one-line instruction ("*Select one or two emotions*"). No reports of usability problems were received – an indication that the tool's simple instruction is sufficient and that its unique widget design facilitates use.

4

General Discussion

This thesis describes work responding to the research questions enumerated in Section 1.1. Inspiration was drawn from *kairos*, one of the fundamental concepts of rhetoric and – what is closer to the field of HCI – persuasive technologies. In parallel with rhetoric, where *kairos* stands for the opportune moment to deliver a message, in the realm of HCI it conveys the ideal moment to trigger an interaction in order to maximize its potential for effectiveness. Starting from this concept, the current work has followed two different threads that, in the end, lead to the same outcome: the deepening of the understanding that machines may have of their users, and the consequential leveraging of their potential for delivering the right interaction at the right time.

The first research thread of the present work responds to research question 1 (see Section 1.1) and entailed the development of a framework and a DSL, respectively EveWorks and EveXL, for detecting daily life events. The second thread, on the other hand, addresses research question 2 (see Section 1.1) and consisted in the development of the CAAT, a tool designed for quick assessment of affective reactions through the selection of emotion words. To play with words a little, while the development of the EveWorks-EveXL dyad is closer to the "C" side of HCI, that of the computer and technology, the CAAT research is directly related to the "H" side of HCI, the human side.

The development of EveWorks and EveXL – a framework and a DSL for the detection of daily life events – had very positive outcomes. Three studies were conducted for evaluation of the framework and its DSL: the first one was intended to evaluate EveXL's alternate notation for the operators of the Interval Algebra and also perform a preliminary field test on EveWorks (see Section 2.5.1); the second experiment was designed to assess the DSL's overall simplicity, through a gamification experiment in which a videogame requested

players with little to no programming experience to read and interpret EveXL statements (see Section 2.5.2); and, finally, the third study was targeted at testing the language as a programming tool and, to that end, experienced programmers have been asked to write the EveXL expressions corresponding to pre-defined natural language event descriptions (see Section 2.5.3).

The first study produced evidence that supports the timeline as an interesting and adequate inspiration for EveXL's syntax. Indeed, the timeline is one of the most ubiquitous and familiar representations of temporality, and temporality is the essential core of EveXL. As such, having a reflection of the former into the latter's syntax may lend EveXL some of the timeline's sense of familiarity. These conclusions might be generalized to the domain of programming language design, especially for the particular case of DSL creation. Indeed, when planning new DSLs, perhaps language designers should consider what, if any, are the prevalent visual representations in use in the domain of the new DSL and then – if appropriate – consider adapting and including some characteristics of those representations into the language's syntax. Indeed, as far as the results of this study suggest, there is value to be gained in terms of language usability and programmer satisfaction.

The outcomes of the second study indicate that participants found EveXL to be straightforward and easy to understand. Indeed, none manifested any particular difficulty while performing the game actions which, once again, requested them to read and execute events written in EveXL. This claim finds further support on the low mean number of errors per player, as well as the low average time taken to execute those events – i.e., placing the smartphone on the pads in the correct sequence –, as both would likely be higher had participants not understood the EveXL expressions, and merely played the game in a trial-and-error fashion. These results become even more conclusive bearing in mind that the participants of this study were markedly inexperienced programmers. Due to this very same fact, it can likely be extrapolated that the fundamental concepts of EveWorks – that of time intervals and the temporal relations between them – are potentially simple and consistent enough to be mastered by more experienced programmers. Additionally, because the design of the videogame required reliable and responsive event detection, the consistent and

problem-free performance of EveWorks is proof of the tool's reliability for event detection and reaction.

Finally, the third study has positively attested with experienced programmers that EveXL is an adequate tool for the expression of events. The obtained results confirm those of the second study, in that EveXL is a language whose concepts and syntax are easy to understand, and also that programmers with some experience can easily read and write accurate EveXL expressions. Furthermore, the customizations that programmers applied to their EveXL code were also noteworthy, encompassing both "cosmetic" applications of naming conventions to interval identifiers, and also the tentative development of programming patterns. This implicitly reveals that programmers felt that EveXL's syntax was flexible enough to create personalized code, and that its underlying temporal model was sound enough to be explored and intuitively played with.

To conclude the discussion of EveWorks and EveXL, the results of these three studies strongly suggest that DSLs based on high-level, daily-life constructs – such as intervals of time – do have a place among programming tools. Additionally, the heterogeneity and the architectural simplicity of the tools developed for these studies confirm the plasticity brought by having an external entity dedicated to event detection instead of having the codes for context gathering and event reaction mixed with the application's logic.

Because, as previously stated, the best moment for delivering an interaction – its *kairos* – may be found by analyzing a conjunction of variables that encompass not only the receiver's context but also his/her own internal state, the assessment of emotional experiences is of paramount interest for the deployment of meaningful interactions.

Therefore, the second research thread of the present work was the development of the CAAT, a tool designed for quick assessment of affective reactions through the selection of emotion words. The results obtained with the CAAT experiments are conclusive enough about the tool's viability. Two studies have been conducted, one intended to assess the CAAT's overall usability (see Section 3.6.1) and the other aiming towards a formal validation of its psychometric properties (see Section 3.6.2). The results of the first study,

conducted in laboratory settings, strongly indicate that the CAAT is a quick and pleasant-to-use way to assess affective states, while also suggesting that the tool is reliable as well. Indeed, since the CAAT can be considered an arranged and styled list of discrete emotion words, it has been compared against the OEL, a more traditional, alphabetically-ordered list containing the same emotion words as the CAAT.

The CAAT scores have a stronger correlation with the SAM's, than do the OEL results. Additionally, the CAAT has also scored higher than the OEL in some common dimensions of usability, and responses with the former tool took participants less time than with the latter. Overall, these results suggest that the CAAT is a superior tool than ordered lists of emotions for affective assessments based on discrete models of emotions, all the while being more reliable than ad-hoc compilations due to the robustness of its underlying theory and model of emotions, Robert Plutchik's Psychoevolutionary Theory of Emotions [73].

On the other hand, the second study entailed a more thorough evaluation of the CAAT along several dimensions that characterize viable tools for emotion assessment. To this end, an online questionnaire was implemented and then used to perform the evaluations, on the field and without supervision. The study results positively assessed the CAAT's temporal reliability in periods of time of different duration; provided evidence towards its construct validity against the ratings of stimuli from a normatively rated database; and confirmed the tool's quickness of use. While, of course, ensuring the validity of any measure involves the thorough assessment of various forms of validity – being, as noted by Cronbach [25], an iterative and potentially unending task – this study has nonetheless produced solid evidence towards the CAAT's reliability.

Indeed, whether gathering affective reactions to multimedia (similar to what was done in this study with the stimuli/pictures) or using emotions as input to applications with innovative ends, the CAAT should be considered an option when simple and quick emotional reaction assessments are required. This may become particularly true in cases where it is necessary (or convenient) to integrate emotional assessments in graphical user interfaces. To be sure, the design of the CAAT as a widget made the process of deploying it in the study's online questionnaire (a web page) very straightforward.

Additionally, in a showcase of the tool's applicability in different domains, Madeira et al. [56] have used it in the development of an interactive mobile application for people with stuttering disorders; Alelis et al. [2] have conducted a study on the emotional connectedness that people belonging to two target populations - young adults and elderly, 18-21 years and 65 years or older, respectively – have with interactive, tridimensional digital models of artifacts; and, finally, Centieiro et al. [20] have used the CAAT as the basis for the development of a simplified, sports-oriented tool that allowed remote sports viewers to share their emotions (these use cases are further detailed in Section 3.7).

Of course, several other interesting use scenarios can be imagined for the CAAT. For instance, media could be tagged with emotion words. A database of such media could be queried through "emotional searches" and the retrieved results sorted by "emotional relevance", according to some hypothetical similarity measure adequate for emotional spaces (e.g., "photos that leave me calm and positive" would retrieve photos with tags similar to acceptance, trust or serenity). On a different domain, as personalization is about the accommodation of individual differences in systems and interfaces, and being emotions an essential part of individuality, personalization models may result "closer" to people if emotional dimensions are considered. Incidentally, because the tool also appears to promote emotional awareness, researchers and practitioners in clinical-related areas may also find a useful ally in it.

Now that the conclusions of the five studies have been discussed here, there is still an important question that should be addressed: how do these two tools, the EveWorks and the CAAT, integrate into a coherent whole?

Indeed, given the relevance of the role that emotions play in delivering significant interactions (see Chapter 1), it makes sense to have the CAAT incorporated into the EveWorks framework. Be that as it may, there is a design feature of the CAAT that, at first sight, may conflict with its seamless integration as one of EveWorks' sensors: because the CAAT evaluates affective states following a self-assessment approach, users are explicitly asked to report how they are currently feeling; and since EveWorks performs periodic sensor readings, the inclusion of the CAAT as a sensor in data invariants would imply

that users are periodically asked about how they are feeling. While, at first sight, this may appear to be a problem, the truth is that it may be a fitting approach depending on the specificities and purposes of the application being developed. For instance, studies that follow the Experience Sampling Method [54] do require that participants make in-situ notes of their experience. In such settings, a strategy of periodic assessments of affective states might not be misplaced.

On the other hand, a more flexible approach that may be more appropriate to other study/application designs, is to have applications make direct calls to the EveWorks CAAT sensor directly, whenever necessary (more details on this approach can be found in Section 2.2.3). For instance, picture the following situation: an application has a message to deliver to its user, when detecting that he/she has just entered home, but it can be presented in either a supportive or cheering tone. A likely strategy for it would be to have the application submit an "arrive at home" event to EveWorks and, whenever its occurrence is reported back, to make a direct call to the EveWorks affective sensor (the CAAT) and, in accordance with the response, have the message displayed to the user in whatever form is more adequate.

It is then put forward how the two research threads of the presented work may be intertwined into a consolidated contribution to the field of HCI.

5

Future Work

Both the EveWorks-EveXL dyad and the CAAT, as the research projects that they are, have important work to be done in the future in order to further develop and study them.

Indeed, with respect to the EveWorks-EveXL dyad, a useful work effort would be to develop and integrate more – and more reliable – sensors in EveWork's default set. While the included set of sensors was more than enough to build a functional prototype enabling conclusive research to be conducted, EveWorks requires more potential in order to truly be an option worth considering by mobile application developers. For instance, useful sensors would likely include one for external location and another for detecting nearby devices. Possible enabling technologies for these sensors are the Global Positioning System (GPS) commonly built-in in current smartphone devices or the more recent Bluetooth Low Energy (BLE) technology and, for nearby device scanning, the (now classic) Bluetooth protocol.

The version of EveXL that was presented here is the first one, and even though it is a functional prototype – as proven in the conducted studies that are presented in Sections 2.5.1, 2.5.2 and 2.5.3 –, there are still some missing implementations to be included in future versions.

The simplest of these missing features is the inclusion of escape characters, as the lack of this feature prevents certain characters, like double quotes, to be used in the body of text literals.

Also, in this first version of EveXL, there is no way for a programmer to write comments. To be sure, even though EveXL may not be an overly complex tool and its statements are atomic and compact descriptions of real life

situations, there should still be a way to add annotations in order to leverage code flexibility, reuse, and facilitate understanding by programmers.

Regarding the fundamentals of EveXL, alongside the conjunction (the `AND` operator) and the disjunction (the `OR` operator) operations, the negation is one of the three elemental operations of relation algebras [44]. Since James Allen's Interval Algebra is a relation algebra, the lack of this operator makes EveXL's implementation incomplete. Therefore, it makes sense to add the negation operator (`NOT` operator) to the Boolean algebras of both EveXL's interval declaration and predicate clauses (see Section 2.3.3.6 and 2.3.3.7, respectively).

Furthermore, there is expressive power for EveXL to gain, should a way to use aggregate functions in invariants be implemented. The set of aggregate functions commonly found in SQL implementations are likely useful additions to EveXL, e.g., the average, minimum, maximum, sum or median. For instance, for an application designed to measure the temperature of the Central Processing Unit (CPU), instead of solely performing periodic evaluations, it would perhaps make more sense to monitor interval-averaged values of temperature.

Due to the way EveWorks builds intervals of time (explained in Section 2.3.5) the aggregate functions invariants should be kept separated from the data invariants. Indeed, while building intervals, EveWorks expects its data invariants to hold for each of the individual sensor readings that compose it. Conversely, the aggregate functions, by definition, compute a single value out of a set of values – in this case, the readings of an interval. This approach is similar to the one found in SQL, which isolates the conditions that use aggregate functions in a clause other than the WHERE clause – namely, the HAVING clause.

For clarity, the separation between the aggregate functions should also have a reflexion on EveXL's syntax. For simplicity and to maintain the analogy with the well-known SQL, we propose this separation to be implemented through the keyword `HAVING`. Given the future-looking purposes of this Section and to help detailing this idea, there is an example in Snippet 31 illustrating a possible look of an EveXL statement using the aggregate function for the average (`AVG`).

```
(1)  [I] AS [temperature > 0 AND temperature < 50]
(2)  WHERE DURATION OF [I] = 5m
(3)  HAVING [I].AVG(temperature) < 23
```

Snippet 31. Hypothetical EveXL statement using an aggregate function.

The snippet above means that "`I` *is an interval of time during which the sensor 'temperature' read values between 0 and 50,* (2) `I` *has a duration of 5 minutes and* (3) *during* `I`*, the average of the values read by the 'temperature' sensor is less than 23*". In different words, this event will be triggered whenever EveWorks detects an interval of time with 5 minutes in duration and having a mean temperature below 23.

An additional advantage brought along by having aggregate functions is to be able to perform data-wise comparisons between different intervals. Indeed, so far, the only possible relations between intervals of time are the temporal ones articulated through the operators of the Interval Algebra. An example of such a situation can be seen in Snippet 32, which presents an event that will be triggered whenever an interval of 5 minutes is found (`I2`) that has a lower average temperature than a preceding interval of the same duration.

```
(1) [I1] AS [temperature > 0 AND temperature < 50], LAST 2
(2) [I2] AS [temperature > 0 AND temperature < 50]
    WHERE
(3) DURATION OF [I1] = 5m AND
(4) DURATION OF [I2] = 5m AND
(5) [I1] IS BEFORE [I2]
    HAVING
(6) [I1].AVG(temperature) < [I2].AVG(temperature)
```

Snippet 32. Hypothetical EveXL statement using an aggregate function.

The last snippet should be read like "(1-2) `I1` *and* `I2` *are intervals of time whereupon the temperature is between 0 and 50 and* (1) `I1` *refers either to the last*

interval or to the one before. (3-4) *Both intervals have 5 minutes in duration.* (5) `I1` *takes place before* `I2` *and* (6) *the average temperature of the former is less than that of the latter*".

Finally, a modifier of a different nature will be added to EveXL's set. Currently, EveXL uses the `NOW` and the `LAST n` modifiers (see Section 2.3.3.6, Modifiers). Both, however, only restrict the number of intervals that will be stored in memory for EveWorks to process, saying nothing about their chronological range. For instance, one could be interested only in readings that took place in the last 5 minutes. Syntactically, the simplest way to do this is to allow duration literals as arguments for the `LAST` modifier.

On the other hand there is also important future work concerning the CAAT. Specifically, although the CAAT has been validated in different ways, there is still more tests that should be conducted in order to truly understand this tool, and to assess what its strong and weak points are. Although studies have indicated solid test-retest reliability and construct validity, the tool should be tested against different affect assessment methods like, for instance, the PANAS [91]. The sheer number of citations and the number of times that this tool was used in studies reveals the trust that the research community deposits in its bidimensional, Positive Affect-Negative Affect model of human emotion.

Other than this convergent validation, it also would be important to devise new field studies that use the CAAT in different contexts and demographics, in order to understand the conditions that may impact its performance.

Although this might have gone unsaid, as it happens with most research projects, it would be highly important to have the tests repeated by other researchers. Indeed, although the CAAT test-retest study has already provided evidence towards the tool's reliability, an assessment of reproducibility would be the better way to underpin the validity of our tests (and further support that of the tool itself).

The previous discussion has highlighted some aspects of both the EveWorks-EveXL dyad and the CAAT that would benefit from more work or further research. While some of them are simple additions and changes that

would likely require little development effort, others open new directions for research that should be carefully considered.

Naturally, there are many other features that could also be included in this list of future work but, in spite of this, both the EveWorks-EveXL and the CAAT are solid and promising contributions to the field of HCI even as they are herein presented.

6

Final Remarks

As a final note to this thesis, it would be appropriate to consider if the two main contributions of the research presented here provide a complete and final solution to the problem of finding *kairos* in the lives of users. The short answer is *no*. The long – and more correct – answer is more positive and nuanced like *kairos*.

Indeed, there are many subtleties that must be addressed before – if ever – reaching a final answer to the challenge of creating an artifact capable of grasping *kairos*, or opportunity, in whatever form and whenever it may "appear". The truth is that *kairos* is much too circumstantial, too broad – ultimately, too elusive – to be completely bounded within any framework, technological or not. Indeed, reflect on the number of times that we humans miss opportunities in our daily lives, and we are much more adaptive to scenarios of overall uncertainty and unexpectedness than any computer algorithm developed so far.

Nonetheless, collected evidence supports that the EveWorks is an adequate tool to detect changes in the context of mobile devices and that the CAAT is able to perform valid assessments of peoples' affective states, thereby respectively providing valuable insights into the daily life of smartphone users and into an essential part of human beings.

Since opportunity combines aspects from both the subjective-inner and the objective-outer worlds, from ourselves and our surrounding context – with all their complexity and multilayered intricacies –, the contributions of EveWorks and the CAAT must be taken for what they were designed to be: not a general and complete solution to the challenge of finding *kairos*, but two significant steps towards the future of building truly *kairos*-aware applications.

References

1. Alberti, M., Dell'Acqua, P., and Pereira, L.M. 2011. Observation strategies for event detection with incidence on runtime verification: Theory, algorithms, experimentation. In *Annals of Mathematics and Artificial Intelligence* 62, 3–4: 161–186. DOI: 10.1007/s10472-011-9259-5
2. Alelis, G., Bobrowicz, A., and Ang, C.S. 2015. Comparison of engagement and emotional responses of older and younger adults interacting with 3D cultural heritage artefacts on personal devices. In *Behaviour & Information Technology* 34, 11: 1064–1078. DOI: 10.1080/0144929X.2015.1056548
3. Allen, J.F. 1981. An Interval-Based Representation of Temporal Knowledge. In *Proceedings of the 7th international joint conference on Artificial intelligence - Volume 1 (IJCAI'81), Vol. 1*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 221–226.
4. Allen, J.F. 1983. Maintaining knowledge about temporal intervals. In *Communications of the ACM* 26, 11: 832–843. DOI: 10.1145/182.358434
5. Alspaugh, T.A. 2005. *Software Support for Calculations in Allen's Interval Algebra*. Irvine.
6. Atkinson, R.C., and Shiffrin, R.M. 1968. Human Memory: A Proposed System and its Control Processes. In *Psychology of Learning and Motivation - Advances in Research and Theory* 2, C: 89–195. DOI: 10.1016/S0079-7421(08)60422-3
7. Barrett, L.F. 2006. Solving the emotion paradox: categorization and the experience of emotion. In *Personality and social psychology review : an official journal of the Society for Personality and Social Psychology, Inc* 10, 1: 20–46. DOI: 10.1207/s15327957pspr1001_2
8. Barrett, L.F., Gendron, M., and Huang, Y.-M. 2009. Do discrete emotions exist? In *Philosophical Psychology* 22, 4: 427–437. DOI: 10.1080/09515080903153634
9. Benedek, J., and Miner, T. 2002. Measuring Desirability: New Methods for Evaluating Desirability in a Usability Lab Setting. In *Proceedings of the Usability Professionals' Association Conference*.
10. Bless, H., Bohner, G., Schwarz, N., and Strack, F. 1990. Mood and Persuasion: a Cognitive Response Analysis. In *Personality and Social Psychology Bulletin* 16, 2: 331–345. DOI: 10.1177/0146167290162013
11. Bradley, M., and Lang, P. 1994. Measuring Emotion: the Self Assessment Manikin and the Semantic Differential Scale. In *Journal of Behavior Therapy and Experimental Psychiatry* 25, 1: 49–59. DOI: 10.1016/0005-7916(94)90063-9

12. Cardoso, B. 2013. CAAT - The Circumplex Affect Assessment Tool. Retrieved July 18, 2016 from http://img.di.fct.unl.pt/bmcardoso/caat

13. Cardoso, B., and Romão, T. 2015. Avoiding "…too late!" - Expressing and Detecting Opportunity with EveWorks and EveXL. In *Proceedings of the 13th International Conference on Advances in Mobile Computing and Multimedia (MoMM 2015)*, ACM, New York, NY, USA, 293–302. DOI: 10.1145/2837126.2837139

14. Cardoso, B., and Romão, T. 2015. Making Sense of EveXL , a DSL for Context Awareness. In *Proceedings of the 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services on 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 291–292. DOI: 10.4108/eai.22-7-2015.2260303

15. Cardoso, B., and Romão, T. 2014. Presenting EveWorks, a Framework for Daily Life Event Detection. In *Proceedings of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems (EICS '14)*, ACM, New York, NY, USA, 289–294. DOI: 10.1145/2607023.2610279

16. Cardoso, B., and Romão, T. 2014. The Timeline as a Programming Interface. In *CHI '14 Extended Abstracts on Human Factors in Computing Systems (CHI EA '14)*, ACM, New York, NY, USA, 1423–1428. DOI: 10.1145/2559206.2581303

17. Cardoso, B., Romão, T., and Correia, N. 2013. CAAT - A Discrete Approach to Emotion Assessment. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems (CHI EA '13)*, ACM, New York, NY, USA, 1047–1052. DOI: 10.1145/2468356.2468543

18. Cardoso, B., Santos, O., and Romão, T. 2015. On Sounder Ground: CAAT, a Viable Widget for Affective Reaction Assessment. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15)*, ACM, New York, NY, USA, 501–510. DOI: 10.1145/2807442.2807465

19. Carlson, J., and Lisper, B. 2004. An Event Detection Algebra for Reactive Systems. In *Proceedings of the Fourth ACM International Conference on Embedded Software (EMSOFT '04)*, ACM, New York, NY, USA, 147–154. DOI: 10.1145/1017753.1017779

20. Centieiro, P., Cardoso, B., Romão, T., and Dias, A.E. 2014. If You Can Feel It, You Can Share It!: A System for Sharing Emotions During Live Sports Broadcasts. In *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology (ACE '14)*, ACM, New York, NY, USA, Article 15. DOI: 10.1145/2663806.2663846

21. Codd, E.F. 1970. A Relational Model of Data for Large Shared Data Banks. In *Communications of the ACM* 13, 6: 377–387. DOI: 10.1145/362384.362685

22. Cohen, N.H., and Kalleberg, K.T. 2008. EventScript. In *Proceedings of the*

2008 ACM SIGPLAN-SIGBED conference on Languages, compilers, and tools for embedded systems - LCTES '08, May: 111. DOI: 10.1145/1375657.1375673

23. Cowie, R., Douglas-Cowie, E., Savvidou, S., McMahon, E., Sawey, M., and Schröder, M. 2000. "FEELTRACE": An Instrument for Recording Perceived Emotion in Real Time. In *Proceedings of the ISCA Workshop on Speech and Emotion: A Conceptual Framework for Research*, 19–24.

24. Cowie, R., and Sawey, M. 2011. GTrace - General trace program from Queen's, Belfast. 23. Retrieved from http://www.psych.qub.ac.uk/GTrace/GtraceNotesWeb.pdf

25. Cronbach, L.J. 1988. Five perspectives on the validity argument. In *Test Validity*, H Wainer and H. I. Braun (eds.). Lawrene Erlbaum Associates, 3–17.

26. Crowley, S. 2004. *Ancient Rhetorics for Contemporary Students*. DOI: 10.2307/358304

27. Czerwinski, M., Horvitz, E., and Cutrell, E. 2001. Subjective Duration Assessment: an Implicit Probe for Software Usability. In *Proceedings of IHM-HCI 2001 Conference* 2: 160–170.

28. Damásio, A. 1994. *Descartes' Error: Emotion, Reason, and the Human Brain*. Putnam Publishing, New York, New York, USA.

29. Dan-Glauser, E.S., and Scherer, K.R. 2011. The Geneva affective picture database (GAPED): a new 730-picture database focusing on valence and normative significance. In *Behavior research methods* 43, 2: 468–477. DOI: 10.3758/s13428-011-0064-1

30. de' Rossi, F. Il Tempo come Opportunità (Kairòs) [Painting]. Retrieved from https://commons.wikimedia.org/wiki/File:Francesco_Salviati_-_Il_Tempo_Opportuno.jpg

31. Devaraju, A., Hoh, S., and Hartley, M. 2007. A context gathering framework for context-aware mobile solutions. In *Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology (Mobility '07)* 7: 39–46. DOI: 10.1145/1378063.1378070

32. Dey, A.K., Wac, K., Ferreira, D., Tassini, K., Hong, J.-H., and Ramos, J. 2011. Getting closer: an empirical investigation of the proximity of user to their smart phones. In *Proceedings of the 13th international conference on Ubiquitous computing - UbiComp '11*: 163–172. DOI: 10.1145/2030112.2030135

33. Douglas, M. 1982. Cultural Bias. In *In The Active Voice*. Routledge & Kegan Paul, London, 183–254.

34. Eerola, T., and Vuoskoski, J.K. 2011. A comparison of the discrete and dimensional models of emotion in music. In *Psychology of Music* 39, 1: 18–

49. DOI: 10.1177/0305735610362821

35. Ferreira, D. 2013. AWARE: A mobile context instrumentation middleware to collaboratively understand human behavior

36. Finucane, M.L., Alhakami, A., Slovic, P., and Johnson, S.M. 2000. The affect heuristic in judgments of risks and benefits. In *Journal of Behavioral Decision Making* 13, 1: 1. DOI: 10.1002/(SICI)1099-0771(200001/03)13:1<1::AID-BDM333>3.0.CO;2-S

37. Fogg, B.J. 2003. *Persuasive Technology: Using Computers to Change What We Think and Do*. DOI: 10.4017/gt.2006.05.01.009.00

38. Fontaine, J., Scherer, K., and Roesch, E. 2007. The world of emotions is not two-dimensional. In *Psychological Science* 18, 12: 1050–1057. DOI: 10.1111/j.1467-9280.2007.02024.x

39. Fowler, M. 2010. *Domain Specific Languages*. Addison-Wesley Professional.

40. George, J. 1996. Trait and state affect. In *Individual Differences and Behavior in Organizations*, Kevin. R. Murphy (ed.). Wiley, San Francisco, CA, USA, 145–171.

41. Guimarães, N., Correia, N., and Carmo, T. 1992. Programming Time in Multimedia User Interfaces. In *Proceedings of the 5th annual ACM symposium on User interface software and technology (UIST '92)*, ACM, New York, NY, USA, 125–134. DOI: 10.1145/142621.142637

42. Hakkyun, K., Akshay, R.R., and Angela, Y.L. 2008. It's Time to Vote: the Effect of Matching Message Orientation and Temporal Frame on Political Persuasion. In *Journal of Consumer Research* 35, 6: 877–889. DOI: 10.1086/593700

43. Hewett, Baecker, Card, et al. ACM SIGCHI Curricula for Human-Computer Interaction. Retrieved April 7, 2016 from http://old.sigchi.org/cdg/cdg2.html#2_1

44. Hirsch, R., and Hodkinson, I. 2002. *Relation Algebras by Games*. Gulf Professional Publishing.

45. Intille, S.S. 2004. Ubiquitous computing technology for just-in-time motivation of behavior change. In *Studies in Health Technology and Informatics* 107, Pt 2: 1434–1437.

46. Jago, R., Baranowski, T., Baranowski, J.C., Thompson, D., and Greaves, K.A. 2005. BMI from 3-6 y of age is predicted by TV viewing and physical activity, not diet. In *International journal of obesity (2005)* 29, 6: 557–564. DOI: 10.1038/sj.ijo.0802969

47. Kay, N.M. 2001. *Ausonius: Epigrams: Text with Introduction and Commentary*. Duckworth.

48. Khooshabeh, P., McCall, C., Gandhe, S., Gratch, J., and Blascovich, J. 2011.

Does it matter if a computer jokes. In *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems*, 77–86. DOI: 10.1145/1979742.1979604

49. Kinneavy, J.L., and Eski, C.R. 1994. Kairos in Aristotle's Rhetoric. In *Written Communication* 11, 1: 131–142. DOI: 10.1177/0741088394011001006

50. Krahmer, E., van Dorst, J., and Ummelen, N. 2004. Mood, Persuasion and Information Presentation. In *Information Design Journal* 12, 3: 219–232. DOI: 10.1075/idjdd.12.3.10kra

51. Kramer, D., Kocurova, A., Oussena, S., Clark, T., and Komisarczuk, P. 2011. An extensible, self contained, layered approach to context acquisition. In *Proceedings of the Third International Workshop on Middleware for Pervasive Mobile and Embedded Computing (M-MPAC '11)*, ACM New York, NY, USA, Article 6. DOI: 10.1145/2090316.2090322

52. Lally, P., and Gardner, B. 2013. Promoting Habit Formation. In *Health Psychology Review* 7, Supplement 1: S137–S158. DOI: 10.1080/17437199.2011.603640

53. Lang, P.J., Bradley, M.M., and Cuthbert, B.N. 2008. *International Affective Picture System (IAPS): Affective Ratings of Pictures and Instruction Manual*

54. Larson, R., and Csikszentmihalyi, M. 2014. The Experience Sampling Method. In *Flow and the Foundations of Positive Psychology*. 21–34. DOI: 10.1007/978-94-017-9088-8_2

55. Mackie, D.M., and Worth, L.T. 1991. Feeling Good, But Not Thinking Straight: the Impact of Positive Mood on Persuasion. In *Emotion and Social Judgments* (1st ed.), Joseph P. Forgas (ed.). Pergamon Press, Elmsford, NY, US, 201–219.

56. Madeira, R.N., Macedo, P., Pita, P., Bonança, Í., and Germano, H. 2013. Building on Mobile towards Better Stuttering Awareness to Improve Speech Therapy. In *Proceedings of International Conference on Advances in Mobile Computing & Multimedia (MoMM '13)*, ACM, New York, NY, USA, 551–554. DOI: 10.1145/2536853.2536911

57. Magyar-Moe, J.L. 2009. Worksheet 3.1: The Positive and Negative Affect Schedule (PANAS; Watson et al., 1988). In *Therapist's Guide to Positive Psychological Interventions* (Illustrate). Academic, 2009, 52. Retrieved July 18, 2016 from http://booksite.elsevier.com/9780123745170/Chapter 3/Chapter_3_Worksheet_3.1.pdf

58. Mandryk, R.L., Atkins, M.S., and Inkpen, K.M. 2006. A Continuous and Objective Evaluation of Emotional Experience with Interactive Play Environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*, ACM, New York, NY, USA, 1027–1036. DOI: 10.1145/1124772.1124926

59. Matic, A., Pielot, M., and Oliver, N. 2015. Boredom-Computer Interaction: Boredom Proneness and The Use of Smartphone. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2015)*, ACM, New York, NY, USA, 837–841. DOI: 10.1145/2750858.2807530

60. Mehrabian, A. 1996. Pleasure-Arousal-Dominance: a General Framework for Describing and Measuring Individual Differences in Temperament. In *Current Psychology* 14, 4: 261–292. DOI: 10.1007/BF02686918

61. Miller, G.A. 1956. The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information. In *Psychological Review* 63, 2: 81–97. DOI: 10.1037/h0043158

62. Morris, M.E., Kathawala, Q., Leen, T.K., et al. 2010. Mobile Therapy: Case Study Evaluations of a Cell Phone Application for Emotional Self-Awareness. In *Journal of Medical Internet Research* 12, 2. DOI: 10.2196/jmir.1371

63. Moshfegi, Y. 2012. The role of emotion in information retrieval. In *Glasgow Theses Service* 2: 300.

64. Murphy, K.R. 1996. *Individual Differences and Behavior in Organizations*. Jossey-Bass Publishers.

65. Núñez, R., and Cooperrider, K. 2013. The Tangle of Space and Time in Human Cognition. In *Trends in Cognitive Sciences* 17, 5: 220–229. DOI: 10.1016/j.tics.2013.03.008

66. Osgood, C.E., May, W.H., and Miron, M.S. 1975. *Cross-Cultural Universals of Affective Meaning*. University of Illinois Press.

67. Patel, S.N., Kientz, J.A., Hayes, G.R., Bhat, S., and Abowd, G.D. 2006. Farther Than You May Think: an Empirical Investigation of the Proximity of Users to Their Mobile Phones. In *Proceedings of the 8th International Conference on Ubiquitous Computing (UbiComp 2016)*, Springer Berlin Heidelberg, 123–140. DOI: 10.1007/11853565_8

68. Picard, R.W. 1998. Affective Computing. In *Pattern Analysis and Applications* 1, 1: 71–73. DOI: 10.1007/BF01238028

69. Pielot, M., Dingler, T., Pedro, J.S., and Oliver, N. 2015. When Attention is not Scarce - Detecting Boredom from Mobile Phone Usage. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '15)*, ACM, New York, NY, USA, 825–836. DOI: 10.1145/2750858.2804252

70. Plutchik, R. 1982. A Psychoevolutionary Theory of Emotions. In *Social Science Information* 21, 4–5: 529–553.

71. Plutchik, R. 1990. Emotions and Psychotherapy:a Psychoevolutionary Perspective. In *Emotion, Psychopathology, and Psychotherapy*, Robert

Plutchik and Henry Kellerman (eds.). Elsevier Inc., 3–41. DOI: 10.1016/B978-0-12-558705-1.50007-5

72. Plutchik, R. 1983. Emotions in Early Development: a Psychoevolutionary Approach. In *Emotions in Early Development*, Robert Plutchik and Henry Kellerman (eds.). Elsevier Inc., 221–257. DOI: 10.1016/B978-0-12-558702-0.50014-5

73. Plutchik, R. 2001. The Nature of Emotions. In *American Scientist* 89, 4: 344–350. DOI: 10.1511/2001.4.344

74. Pollak, J.P., Adams, P., and Gay, G. 2011. PAM: a Photographic Affect Meter for Frequent, In Situ Measurement of Affect. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*, ACM, New York, NY, USA, 725–734. DOI: 10.1145/1978942.1979047

75. Rademacher, G. Railroad Diagram Generator. Retrieved April 1, 2016 from http://www.bottlecaps.de/rr/ui

76. Räisänen, T., Oinas-Kukkonen, H., and Pahnila, S. 2008. Finding Kairos in Quitting Smoking: Smokers' Perceptions of Warning Pictures. In *Persuasive Technology (Third International Conference, PERSUASIVE 2008, Oulu, Finland, June 4-6, 2008. Proceedings)*, Harri Oinas-Kukkonen, Per Hasle, Marja Harjumaa, Katarina Segerståhl and Peter Øhrstrøm (eds.). Springer Berlin Heidelberg, 254–257. DOI: 10.1007/978-3-540-68504-3_25

77. Reardon, K.K. 1981. *Persuasion, Theory and Context*. Sage Publications, 1981.

78. Reis, S., and Correia, N. 2011. An Imaginary Friend that Connects with the User's Emotions. In *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology (ACE '11)*, ACM, New York, NY, USA, Article 1, 8 pages. DOI: 10.1145/2071423.2071425

79. Robinson, M.D., and Clore, G.L. 2002. Belief and Feeling: Evidence for an Accessibility Model of Emotional Self-Report. In *Psychological Bulletin* 128, 6: 934–960. DOI: 10.1037/0033-2909.128.6.934

80. Rossi, F., van Beek, P., and Walsh, T. 2006. *Handbook of Constraint Programming (Volume 35)*. Elsevier, 2006.

81. Russell, J. 1980. A Circumplex Model of Affect. In *Journal of Personality and Social Psychology* 39, 6: 1161–1178. DOI: 10.1037/h0077714

82. Russell, J. 1989. Measures of emotion. In *The Measurement of Emotions*, Robert Plutchik and Henry Kellerman (eds.). Elsevier Inc., 83–111. DOI: 10.1016/B978-0-12-558704-4.50010-4

83. Russell, J., Weiss, A., and Mendelsohn, G. 1989. Affect Grid: A Single-Item Scale of Pleasure and Arousal. In *Journal of Personality and Social Psychology* 57, 3: 493–502. DOI: 10.1037/0022-3514.57.3.493

84. Salber, D., Dey, A.K., and Abowd, G.D. 1999. The Context Toolkit: Aiding

the Development of Context-Enabled Applications. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI '99)*, ACM, New York, NY, USA, 434–441. DOI: 10.1145/302979.303126

85. Scherer, K., Dan, E., and Flykt, A. 2006. What Determines a Feeling's Position in Affective Space? A Case for Appraisal. In *Cognition and Emotion* 20, 1: 92–113. DOI: 10.1080/02699930500305016

86. Scherer, K.R. 2005. What are Emotions? And How Can They be Measured? In *Social Science Information* 44, 4: 695–729. DOI: 10.1177/0539018405058216

87. Sebesta, R.W. 2012. *Concepts of Programming Languages*. Pearson Education Limited, 2012.

88. Szymanek, R., and Kuchcinski, K. JaCoP - Java Constraint Programming Solver. Retrieved from http://jacop.osolpro.com

89. Wang, S., Wang, Z., and Ji, Q. 2015. Multiple Emotional Tagging of Multimedia Data by Exploiting Dependencies Among Emotions. In *Multimedia Tools and Applications* 74, 6: 1863–1883. DOI: 10.1007/s11042-013-1722-3

90. Warriner, A.B., Kuperman, V., and Brysbaert, M. 2013. Norms of Valence, Arousal, and Dominance for 13,915 English Lemmas. In *Behavior Research Methods* 45, 4: 1191–1207. DOI: 10.3758/s13428-012-0314-x

91. Watson, D., Clark, L.A., and Tellegen, A. 1988. Development and Validation of Brief Measures of Positive and Negative Affect: the PANAS Scales. In *Journal of Personality and Social Psychology* 54, 6: 1063–1070. DOI: 10.1037/0022-3514.54.6.1063

92. van Wissen, B., Palmer, N., Kemp, R., Kielmann, T., and Bal, H. 2010. ContextDroid: an Expression-Based Context Framework for Android. In *Proceedings of International Workshop on Sensing for App Phones (PhoneSense 2010)*, 1–5. Retrieved from http://sensorlab.cs.dartmouth.edu/phonesense/proceeding.pdf

93. Wu, Z., and Palmer, M. 1994. Verb Semantics and Lexical Selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics (ACL '94)*, Association for Computational Linguistics, Stroudsburg, PA, USA, 133–138. DOI: 10.3115/981732.981751

94. Wundt, W. 1910. *Principles of Physiological Psychology*. Swan Sonnenschein & Co, New York, NY, US. DOI: 10.1037/12381-000

95. Zimbardo, P.G., and Leippe, M.R. 1991. *The Psychology of Attitude Change and Social Influence*. McGraw-Hill, 1991. DOI: 10.5860/choice.29-3609

96. Drools - Drools Fusion. Retrieved January 23, 2014 from http://drools.jboss.org/drools-fusion.html

97. Drools - The Business Logic Integration Platform. Retrieved May 23, 2014

from http://drools.jboss.org/

98. EsperTech - Event Series Intelligence. Retrieved February 20, 2016 from http://www.espertech.com/