Computer Languages, Systems & Structures 40 (2014) 112–136

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

AmbientTalk: programming responsive mobile peer-to-peer applications with actors

Tom Van Cutsem*, Elisa Gonzalez Boix[1], Christophe Scholliers[2], Andoni Lombide Carreton, Dries Harnie[1], Kevin Pinte, Wolfgang De Meuter

Software Languages Lab, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium

ARTICLE INFO

Article history:
Received 9 August 2013
Received in revised form
12 May 2014
Accepted 20 May 2014
Available online 2 June 2014

Keywords:
Actors
Event loops
Futures
Mobile networks
Peer-to-peer
Service discovery
Leasing

ABSTRACT

The rise of mobile computing platforms has given rise to a new class of applications: mobile applications that interact with peer applications running on neighbouring phones. Developing such applications is challenging because of problems inherent to concurrent and distributed programming, and because of problems inherent to mobile networks, such as the fact that wireless network connectivity is often intermittent, and the lack of centralized infrastructure to coordinate the peers.

We present AmbientTalk, a distributed programming language designed specifically to develop mobile peer-to-peer applications. AmbientTalk aims to make it easy to develop mobile applications that are resilient to network failures by design. We describe the language's concurrency and distribution model in detail, as it lies at the heart of AmbientTalk's support for responsive, resilient application development. The model is based on communicating event loops, itself a descendant of the actor model. We contribute a small-step operational semantics for this model and use it to establish data race and deadlock freedom.

1. Introduction

Throughout the past decade, we have seen the rise of mobile platforms such as J2ME, iOS and Android. These platforms, in turn, enable a new class of applications: *mobile peer-to-peer* (*P2P*) *applications*. What is characteristic of such applications is that they are often used on the move, and that they sporadically interact with peer applications running on neighbouring phones (often communicating via a wireless ad hoc network [1]).

Developing such applications is a challenge not just because of the inherent difficulty of developing distributed applications. Connectivity between phones is often intermittent (connections drop and are restored as people move about) and applications may not always rely on fixed infrastructure or a reachable central server to support the coordination.

In this paper we present AmbientTalk, a distributed programming language designed specifically to develop mobile P2P applications. AmbientTalk is the first distributed object-oriented language that specifically targets applications deployed on mobile phones interconnected via an ad hoc wireless network. On the surface, the language is similar to other OO scripting

* Corresponding author.
E-mail address: tvcutsem@vub.ac.be (T. Van Cutsem).
[1] Funded by the Prospective Research for Brussels (PRFB) program of the Brussels Institute for Research and Innovation (Innoviris).
[2] Funded by a doctoral scholarship of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

http://dx.doi.org/10.1016/j.cl.2014.05.002
1477-8424/© 2014 Elsevier Ltd. All rights reserved.

languages such as JavaScript, Ruby or Python. However, contrary to these languages, it offers built-in support for concurrent and distributed programming. Its concurrency model is founded on actors [2]: loosely coupled, asynchronously communicating components.

We show how AmbientTalk facilitates the development of mobile P2P applications that are resilient to intermittent network failures by default, and how this compares to mainstream distributed object-oriented middleware such as Java RMI (Section 7).

Novelty: This paper complements previous expositions of AmbientTalk [3–5] with a precise operational semantics (Section 8). To the best of our knowledge, this is the first formal account of an actor-based language based on communicating event loops with non-blocking futures. We use the operational semantics to establish data race and deadlock freedom.

Availability: AmbientTalk currently runs as an interpreter on top of the JVM and specifically targets Android-powered smartphones. It is open sourced under an MIT license and available at ambienttalk.googlecode.com. The Android version is published on the Google Play Store (http://bit.ly/HM7Kzv).

2. Mobile ad hoc networks

AmbientTalk's concurrency and distribution features are tailored specifically to mobile ad hoc networks. We briefly describe the features characteristic of mobile ad hoc networks and why they present a challenge.

There are two discriminating properties of mobile networks, which clearly set them apart from traditional, fixed computer networks: applications are deployed on *mobile* devices connected by *wireless* communication links with a limited communication range. Such networks exhibit two phenomena which are rare in their fixed counterparts:

- *Volatile connections*: Mobile phones equipped with wireless media possess only a limited communication range, such that two communicating phones may move out of earshot unannounced. The resulting disconnections are not always permanent: the phones may meet again, requiring their connection to be re-established. Often, such *transient* network partitions should not affect an application, allowing it to continue its collaboration transparently upon reconnection. Partial failure handling is not a new ingredient of distributed systems, but these more frequent transient disconnections do expose applications to a much higher rate of partial failure than that which most distributed languages or middleware have been designed for. In mobile networks, disconnections become so omnipresent that they should be considered the rule, rather than an exceptional case.
- *Zero infrastructure*: In a mobile network, phones (and thus the applications they host) may spontaneously join or leave the network. Moreover, a mobile ad hoc network is often not administered by a single party. As a result, in contrast to stationary networks where applications usually know where to find collaborating services via URLs or similar designators, applications in mobile networks have to *discover* partner applications while roaming. Services must be discovered on proximate phones, possibly without the help of shared infrastructure. This lack of infrastructure requires a *peer-to-peer* communication model, where services can be directly advertised to and discovered on proximate phones.

Any application designed for mobile networks has to deal with these phenomena. It is therefore worth investigating models or frameworks that ease the development of mobile P2P applications. Because the effects engendered by partial failures or the absence of remote services often pervade the entire application, it is difficult to apply traditional library or framework abstractions. Therefore, support for distributed programming is often dealt with in dedicated middleware (e.g. Java RMI [6], Jini [7]) or programming languages (e.g. Erlang [8], Emerald [9], Argus [10]). In the spirit of such systems, we designed AmbientTalk as a new distributed programming language.

3. Standing on the shoulders of giants

We briefly describe the foundations of AmbientTalk: Where did its features originate?

Object model: AmbientTalk is a dynamically typed, object-oriented language. It was heavily inspired by Self [11] and Smalltalk [12]. Like Ruby, however, AmbientTalk is text-based (not image-based). Inspired by Scheme [13] and E [14], it places an additional emphasis on lexical nesting of objects and lexical scoping.

Concurrency: AmbientTalk embraces actor-based concurrency [2]. In particular, it embraces a particular flavor of actor-based concurrency known as *communicating event loops*, pioneered by the E programming language [14], whose distinguishing features are (a) the treatment of an actor as a coarse-grained component that contains potentially many regular objects, and (b) the complete absence of blocking synchronization primitives. All interaction among actors is purely asynchronous.

The event loop model maps well onto the inherently event-driven nature of mobile P2P applications. Phones may join or leave the network and messages can be received from remote applications at any point in time. All of these events are represented as messages sent to objects, orderly processed by actors. The use of event loops avoids low-level data races that are inherent in the shared-memory multithreading paradigm [15,16].