On the choice of programming languages for developing location-based mobile games

Leif Oppermann

Mixed Reality Lab
University of Nottingham
Jubilee Campus
NG8 1BB, Nottingham, UK
lxo@cs.nott.ac.uk

Abstract: This paper reflects on the choice of programming languages for developing distributed interactive systems such as location-based mobile games. It highlights key strengths and weaknesses of the popular languages and frameworks for the mobile- as well as the server-side. The paper analyses the technological choices made in five different location-based game projects and recommends considering the use of a single language across all different system components to mitigate complexity and strive for productivity.

1 Introduction to the Problem

Developing distributed interactive applications such as mobile location-based games is a complicated task because any distributed system architecture consists of a wealth of technologies that need to be mastered, e.g. HTML, XML, PHP, SQL, Java, C++, widget toolkits, server administration, etc.. Learning the ropes on all sides of this architecture – mobile clients, server and desktop components – can be a daunting experience due to this technological diversity.

This paper focuses on the choice of programming languages which can be used for the development of mobile games in the sense of location-based experiences / pervasive games, which are games that "take the gaming experience out into the real world" [1]. Conventional single-player video-games that are just delivered on a mobile platform are not considered.

Moreover, the argument of this paper is more concerned about the development of research- and event-based mobile games [2] as opposed to professional commercial games. These pre-commercial games are often developed in an academic or research setting, which means that only a small number of people will be involved in the actual development – sometimes the development team can even consist of just a single person! That person or team will have to master programming all components of the distributed system in order to bring the project to a successful finish.

Recommending any particular programming language is often a futile endeavour. This paper is based on the author's practical experience after having worked on a number of different pervasive games over the past 4 years (all of which have used different combinations of programming languages and tools). It also stems from some lecturing experience with undergraduates in Java programming and web application development. Although the paper ultimately describes the author's personal technological preference, it is hoped that it will still be helpful and offer guidance to those readers who wish to implement a distributed location-based mobile game for themselves.

As mentioned above, a typical pervasive game will consist of several software components that are hosted on quite different hardware platforms. Apart from a possible desktop component for orchestration, analysis or visualisation, the most important components are clearly the mobile clients and the server. Development for these profiles is usually constrained in several ways. The following sections try to summarize the most important development aspects from a practitioner's point of view and give a brief overview of the choices made in five different location-based gaming projects.

1.1 Developing for Mobile Clients

How to program for a mobile client, such as a PDA (Personal Digital Assistant) or a mobile phone, is largely dependant on the operating system in use on the desired device. The operating system is usually either Windows Mobile, Symbian OS or a Unix derivate (this includes Mac OS X on the iPhone/iPod touch).

These operating systems all support a range of programming languages with the most commonly used at the moment probably being Java, C/C++, Flash, Python and the .NET framework. They all have certain advantages and disadvantages when seen in the context of developing mobile games (see table 1).

	Java	C/C++	Flash	Python	.Net
Stable Release	Yes	Yes	Yes	Yes	Yes
Low-level Access	Limited	Full	None	Good	Good
Multimedia Support	Fair	High	High	Fair	High
Code Performance	High	Highest	Fair	Fair	High
Programmers Performance	Fair	Slow	High	High	Fair

Table 1: Comparison of programming languages for mobile phones

The selection of programming languages for this comparison was guided by the availability of a stable release. Many other interesting languages like Ruby currently emerge for mobile platforms, but their release states are too early to be considered for productive use and have therefore been omitted from the table.

Java can probably be seen as the de-facto standard for mobile phone development. Java offers a high code performance as well as a fair programmer's performance. It does however only have limited access to device capabilities on mobile phones, e.g. it can access the camera and a Bluetooth GPS but cannot access the cell ID. All device capabilities can be accessed when programming in C/C++. This choice also offers the highest code performance and a very good multimedia support. Basically anything goes, but this is all at the expense of the programmer's performance. This is especially true for Symbian OS C++, which is very different from standard C++. Flash is situated at the opposite end of the productivity spectrum. It offers a well rounded package with very good multimedia support and high programmer productivity. Because it is generally liked by designers, choosing Flash for a project usually also yields very good looking results. The big limitation of using Flash for mobile clients is that it does not support access to low-level device capabilities. Similar to Flash, Python's strength lies in the high programmer's performance. The language also has fair code performance and multimedia support on mobiles. But its major advantage is that the programmer's performance is not bought at the expense of flexibility regarding access to low-level device capabilities. For example with Python S60 most capabilities are supported, incl. GPS, camera, cell ID, Wi-Fi, Bluetooth and accelerometer. If needed, missing capabilities can also be implemented in C++ and provided to Python as modules (note that this is not possible with Java or Flash). The last entry in table 1 is ".Net". This is not a programming language but a framework which comes with a Common Language Runtime (CLR), much like the Java Virtual Machine. Microsoft .Net supports running code generated by a number of programming languages, incl. C#, Visual Basic and Iron Python. It is basically a very good choice if one can accept the significant platform lock-in to Windows (with Mono also Linux). This means that .Net is a good choice for PDA based mobile clients but less so for mobile phones, where the market share of Windows is just too small and .Net is not supported by the majority of the phones.

Access to low-level device capabilities is important for many location-based games. If it is a project requirement, e.g. to obtain a mobile phone network cell ID and use that for coarse grained locations, then Java and Flash would be effectively ruled out. But as some people still wanted to use their preferred language in this context, they came up with interesting workarounds. The influential Placelab project from Intel Research Seattle and the University of Washington based their framework on Java and provide the low-level capabilities via a helper program which is written in Symbian C++ and serves its data via a local socket [3]. Similarly, the Flyer framework for Flash Lite (the mobile phone specific version of Flash) provides access to local capabilities via a helper program which is written in Python and also serves its data via a socket [4].

Another approach to developing mobile games is the web-based approach, where all the program logic resides on the server-side. Here the mobile clients would connect to the server via a mobile web-browser or by other means (e.g. SMS, MMS or Email), change states on the server and deliver the content back to the user. One of the example games presented later in the paper, Love City, took this approach and even incorporated location by utilizing a commercial operator-based positioning service. While this is certainly an interesting alternative for certain cases, it only represents an optimization for very thin clients and is not the solution for all mobile location-based game projects.

1.2 Developing for Servers

Server-side development is usually grounded in existing practices and experiences in the work environment and probably most commonly done in PHP, Java, Perl, Ruby or ASP.Net. Especially PHP is a popular choice with beginners because of its apparent ease of use and wide-spread acceptance which is expressed by its availability at most web-hosting companies.

Scalability and stability are areas where Java, esp. the Enterprise Edition (J2EE) excels. However, it is the author's opinion that the lengthy compile-deploy-run cycle of Java Servlet programming takes too much time. To some extent this effect can be mitigated by incorporating more formalism and applying unit tests to the code so that "compile-deploy-run" turns into "compile-test" and contract violating code fails before being deployed to the server. The more formalistic development workflow of Java is geared towards writing more reliable code from the outset, which is of course a Good Thing. It can however turn into a Bad Thing when the configuration formalism takes over and burns development time which could be otherwise spent on the application prototype itself.

One way to tackle excessive configuration and still maintain a good style is to utilize a web application framework. These frameworks usually inherit the Model-view-controller (MVC) pattern, which clearly separates the data from its presentation and thereby eases the development. One example of a MVC framework is the Zend Framework for PHP. More recent frameworks like Ruby on Rails additionally implemented the "convention over configuration" paradigm which implies that a programmer would only need to explicitly configure certain parts of an application if their behaviour is desired to be different from the convention. If for example the application would have players and they are to be represented by a class "Player", the convention might say that all "Player" objects are going to be persisted into a database table with the name "players". It is only when this is not the intended effect that the programmer would have to touch the configuration files. Otherwise this mapping between objects and the database would just work by default and the programmer wouldn't have to spend much time on it. Originally introduced by Rails, the "convention over configuration" paradigm was quickly adopted by other frameworks such as Grails, Django or Turbogears. A good overview of the many available web frameworks can be found on Wikipedia[1]. Most recently, web giant Google announced the availability of the "Google App Engine"[2] which allows hosting applications on the same infrastructure that Google uses for its other services. The first supported language of the App Engine is Python (others will follow) and the first supported web application framework is Django. An advantage of the Google App Engine is that it completely hides issues related to server administration from the programmer. Another advantage is that it is said to be free of charge for small projects. This would be especially useful for small academic or student projects on a tight budget where getting access to a server is always an issue.

---

[1] http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks
[2] http://code.google.com/appengine/docs/whatisgoogleappengine.html

1.3 Case Studies

This section provides a brief overview of the technological choices for mobile client and server programming languages made in five different projects that the author has been involved in. It omits several desktop components which have been used for visualisation and authoring, some of which have been implemented in C++ with the help of suitable widget toolkits like Qt or wxWidgets.

Epidemic Menace [5, 6] is a location-based cross media game, which means that it has several interfaces to the game. There are several mobile clients. A PDA interface, which accesses a GPS receiver, is programmed in C++ and there is a mobile phone interface which is implemented in J2ME. The server is implemented in C++ and is based on the Morgan Framework supplied by one of the project partners [7].

Tycoon [8] is a location-based game for Series 60 phones. Its location mechanism is based on cell IDs which are provided by the native Placelab helper program mentioned in section 1.1. The mobile application itself is implemented in Java 2 Platform, Micro Edition (J2ME) and the server is programmed in PHP, with the database backend being MySQL.

Heartlands [9] is a location- and heart-rate-based game for PDAs. The mobile client is programmed in Flash and supported by a native C++ application which sends position and heart-rate data to Flash over a local socket. A previous version of the game was called 'Ere be Dragons and used the Mobile Bristol toolkit [10] for this task. The server component of Heartlands is implemented in PHP, with MySQL being the database backend.

Love City [11] is a location-based game for mobile phones which has two different mobile phone clients. The first one is a fully graphical smart-phone version which has been produced as a prototype and used for user-testing. The second mobile client is pure SMS, which means that all of its program logic had to be transferred to the server. Also the location mechanism was transferred to the server-side and made use of an external positioning service, to which the players had to subscribe via SMS. The Love City server is written in Java and uses the Equip2 web application framework [12]. Equip2 uses Hibernate as an object-relational mapper and persists into a MySQL database. Parts of the public website are implemented in Flash, which gave the involved designers some artistic freedom.

Rider Spoke [13] is a location-based game for a console-like internet tablet. It uses Wi-Fi access point scans for determining players' locations. The mobile client user interface is implemented in Flash, supported by a native C++ application which sends data to the Flash and manages the persistent Equip2 dataspace. The Rider Spoke server is implemented in Java, again using the Equip2 web application framework and MySQL.

Table 2 overleaf summarizes the choices made in the five case studies. As it can be seen, none of these games used a single programming language for all parts of the system, which most likely resulted in more complicated designs than necessary to support cross-language integration and communication.

Game	Mobile Client	Communication	Server
Epidemic Menace	Client 1: C++ Client 2: Java	CORBA IDL	C++
Tycoon	Java user interface, backed by C++	HTTP Get with XML response	PHP/MySQL
Heartlands	Flash user interface, backed by C++	HTTP Get with XML response	PHP/MySQL
Love City	Client 1: Python Client 2: SMS (server-side)	Client 1: RPC over HTTP Client 2: SMS	Java/Equip2/MySQL Flash
Rider Spoke	Flash user interface, backed by C++	Hessian (offline, file-based)	Java/Equip2/MySQL

Table 2: Technological choices made in the five case studies

Communication in distributed application, i.e. in these cases between mobile clients and server, always requires some sort of serialization and data transport. In the majority of cases in table 2 the data transport has been done online via HTTP, but Epidemic Menace used TCP/IP and Rider Spoke used an offline file-based approach. Several serialization methods have been implemented, ranging from custom XML responses over RPC wrappers to binary protocols like CORBA and Hessian. This diversity presented in table 2 is typical for distributed applications that incorporate mobile clients and it can become problematic if one does not have a strategy for dealing with it. We therefore recently presented several strategies to address this kind of diversity in another paper [12] which is based on the design of the Equip2 framework[3] and the experiences it supported. The proposed strategies were: *"Prioritise strong server support", "Support very flexible communication"*, *"Use a loosely coupled software approach"* and *"Migrate functionality from the server to more capable handsets"*.

2 A proposed Strategy

In order to further simplify the development of location-based mobile game prototypes I would like to propose an additional strategy: *"Strive for using a single programming language for all parts of the distributed system"*. Achieving this goal will reduce the overhead required for dealing with cross-language integration and thereby increase the productivity of the whole development team.

Any language to be suitable for this task would need to be supported with a stable release on all components of the system, have access to all required capabilities on the mobile component and be supported by a modern web application framework on the server-side.

[3] http://equip.sourceforge.net/equip2/

The popular server-side language PHP is ruled out by this requirement as it is not supported on the mobile component.

Ruby is also currently ruled out as the available releases of Ruby for mobiles are not ready for productive use. But once they matured, Ruby might be an interesting alternative for rapid prototyping of mobile location-based game, not least due to the strong Ruby on Rails framework.

Java is the primary language of the Equip2 framework and a good choice as long as the mobile clients don't need to access certain low-level device capabilities.

C/C++ has access to all capabilities on the mobile side but is comparatively slow to program in (especially on Symbian) and is also far from being first choice when developing the server-side.

Microsoft .Net is available on some mobile devices (like PDAs) as well as on the server-side. It allows for quick development at the expense of a significant platform lock-in and a limited user-base (it is not supported on Nokia phones).

Flash is a popular demand by artists which are usually involved in the making of mobile location-based games. Flash enables them to create good-looking multimedia user interfaces without having to become programmers. Flash also has a server component which can be used to sync multiple clients over the server, e.g. for building a chat application. But Flash doesn't support all capabilities required by mobile location-based games and therefore usually needs to be backed by a local application which has these capabilities.

An interesting option is the interpreted, object-oriented Python programming language [14] which is well supported on all required platforms and leans towards rapid prototyping. Languages like Python have the proven effect of increasing the programmers performance [15], which is a Good Thing, especially for small projects. Python is well supported on the server-side by different web application frameworks and it can be run on Google's App Engine. It is also well supported on the mobile side. There is a version of Python for Windows Mobile and Nokia released Python for their Series 60 mobile phone platform in 2004. Since then they proposed its applicability to research in pervasive computing [16]. Nokia has released a port of the widely used Apache web-server to the Series 60 platform and integrated it with their own Python port [17]. As a result the Series 60 mobile phones are no longer simple clients in a pervasive computing network but can act as servers as well. This will allow for more tasks to be transferred onto the mobiles which will become useful for mobile authoring and synchronisation of content [18].

In conclusion, I believe that deciding on a single cross-platform programming language per project will allow for a more rapid development of mobile location-based games.

References

1. Benford, S., Magerkurth, C., Ljungstrand, P.: Bridging the Physical and Digital in Pervasive Gaming. Communications of the ACM 48 (2005) 54-57
2. Paelke, V., Oppermann, L., Reimann, C.: Mobile Location-Based Gaming. In: L. Meng, A. Zipf, and S. Winter, (eds.): Map-Based Mobile Services – Design, Interaction and Usability. Springer (2007)
3. Placelab. [Online]. Available: http://www.placelab.org/
4. Flyer. Open Source Python Framework for Flash Lite Developers.[Online]. Available: http://code.google.com/p/flyer/
5. Lindt, I., Ohlenburg, J., Pankoke-Babatz, U., Ghellal, S., Oppermann, L., Adams, M.: Designing Cross Media Games. In: Proc. PerGames (2005)
6. Lindt, I., Wenninger, C. IPerG Deliverable D8.4: Crossmedia game prototype, phase one.[Online]. Available: http://iperg.sics.se/Deliverables/D8.4-Crossmedia-Game-Prototype-phase-I.pdf
7. Ohlenburg, J., Herbst, I., Lindt, I., Fröhlich, T., Broll, W.: The MORGAN framework: enabling dynamic multi-user AR and VR projects. In: Proc. ACM Symposium on Virtual Reality Software and Technology (2004)
8. Oppermann, L., Broll, G., Capra, M., Benford, S.: Extending Authoring Tools for Location-Aware Applications with an Infrastructure Visualization Layer. In: Proc. UbiComp (2006)
9. Davis, S. B., Moar, M., Jacobs, R., Watkins, M., Shackford, R., Capra, M., Oppermann, L.: Mapping Inside Out. In: C. Magerkurth and C. Röcker, (eds.): Pervasive Gaming Applications - A Reader for Pervasive Gaming Research. Shaker (2007)
10. Hull, R., Clayton, B., Melamed, T.: Rapid Authoring of Mediascapes. In: Proc. Ubicomp (2004) 125-142
11. Oppermann, L., et al.: Love City: A Text-Driven, Location-Based Mobile Phone Game Played Between 3 Cities. In: C. Magerkurth and C. Röcker, (eds.): Pervasive Gaming Applications - A Reader for Pervasive Gaming Research. Shaker (2007)
12. Greenhalgh, C., et al.: Addressing Mobile Phone Diversity in Ubicomp Experience Development. In: Proc. UbiComp (2007)
13. Adams, M., Flintham, M., Oppermann, L., Benford, S., Ghellal, S. IPerG Deliverable D17.2: Game Design Document - CCG Rider Spoke.[Online]. Available: http://iperg.sics.se/Deliverables/D17.2-Game-design-document-CCG-Rider-Spoke.pdf
14. Rossum, G. v. Python Programming Language.[Online]. Available: http://www.python.org/
15. Prechelt, L.: An Empirical Comparision of Seven Programming Languages. IEEE Computer 33 (2000) 23-29
16. Laurila, J., Tuulos, V., MacLaverty, R.: Scripting Environment for Pervasive Application Exploration on Mobile Phones. In: Proc. Pervasive (2006)
17. Wikman, J., Rácz, F. D. Mobile Web Server.[Online]. Available: http://research.nokia.com/research/projects/mobile-web-server/
18. Rimey, K. Personal Distributed Information Store (PDIS) - Final report.[Online]. Available: http://pdis.hiit.fi/pdis/report-2004/pdis-final-report.pdf