Massive Mobile Games Porting: Meantime Study Case

Tarcísio P. Câmara Rodrigo B. V. Lima Rangner F. Guimarães

Alexandre L.G. Damasceno Vander R. Alves* Pedro H. Macedo Geber L. Ramalho*

Meantime Mobile Creations, Brazil *University Federal of Pernambuco, Brazil

Abstract

Game development for mobile devices is usually regarded as a simpler task when compared to games developed for desktop platforms. Indeed, the resources provided by the latter do support more complex applications, therefore, increasing the final product value, and also making the development cycle longer. Although mobile games (and mobile applications, in general) do not have the same amount of resources to be explored, they must adhere to a very strong portability requirement and, since the whole development cycle is rather short, this porting phase must be as efficient and cheap as possible, so that it does not have a huge impact on the final product. In this present work, we will discuss Meantime's experience in developing and porting J2ME games to a large amount of different devices, elucidating how we have evolved from an immature porting process up to a new process that has proved to be more scalable, efficient, cheaper and easier to maintain.

Keywords: porting process, mobile build system, J2ME mobile games

Authors' contact:
{tarcisio.camara, pedro.macedo}@meantime.com.br
{rbvl, rfg, algd}@cesar.org.br
*vander@acm.org, *glr@cin.ufpe.br

1. Introduction

Nowadays, in order to reach a significant share of the market, mobile game developers shall make their games available to as much platforms as possible and also to a great number of wireless carries, all this in a time spam of only three to four months (typical duration of the whole development cycle). Therefore, mobile games shall be developed, from the very beginning, focusing on portability, otherwise, this phase of the development cycle, which tends to be extremely time consuming, will easily become expensive and difficult to manage, endangering the project as a whole.

There are various problems that impair portability [Sampaio et al 2004]: different API optional packages; different carriers' requirements; different implementation of the KVM; different screen size and resolution; sound capabilities; processor power; just to name a few. All this myriad of resources the developer must take into account in order to build competitive games, associated with the fact that the development cycle for a mobile game should not be too long, and that a game must be available for dozens of platforms and wireless carriers, usually in various languages, makes porting a very expensive and complex task.

Other previous researches have already focused on how to make the porting process easier for the mobile domain. The Unified Mobile Framework (UMAK) [UMACK 2006], for instance, provides reusable components and a set of tools combined in an application that intends to ease the porting process of J2ME applications, while J2ME Polish [J2ME Polish 2006] provides a pre-processing feature by which guidelines define a conditional compilation of the source code according to a given device. However, as it will be shown in Section 3, these solutions have some limitations. In particular, we seek approaches that provide a high level of scalability. An industrial effort is also done by mobile game companies to improve porting, however they normally do not publish their methods, considering them industrial secret.

In a previous work, we have already presented an ad hoc solution to the portability problem [Sampaio et al 2004; Alves et al 2005]. In this present paper, we will present an evolution of that solution: the MG2P (Meantime Game Porting Platform). MG2P includes a set of techniques, tools and artifacts to provide generality and, above all, scalability to the process. This current approach was conceived and validated in an industrial scale by Meantime, a leader mobile game studio and publisher sited in Brazil, which develops and publishes games since 2003 for some of the most important wireless carriers in the world.

In the remainder of this paper, we will address the porting problem based on Meantime's experience in developing J2ME games. In Section 2, we present some of the difficulties associated to porting during game development. In Section 3, we will start by describing some related work in the area. Afterwards, we will present how Meantime used to conduct the porting process in the past (Section 4.1), and how it is done now (Section 4.2). Lastly, we will draw some conclusions about our work.

2. Mobile Porting Difficulties

There is a significant amount of different mobile devices out in the market, each one with different capacities, functionalities and retail prices. These different devices coexist especially due to the fact that there are different segments of the market with distinct needs and financial resources. Besides, operators and publishers need that the developed games be delivered to the greatest possible number of users, forcing the developer to provide multiple versions of the application, each optimized to a specific device. The demand of porting mobile games is so critical in the industry that there are currently specialized companies in providing such service [Tira 2004].

Even ignoring BREW [Qualcomm 2004] and Symbian [Symbian 2004], and focusing on the J2ME [Sun 2004] universe (currently the most used platform for developing mobile games), porting demands significant efforts from the development team due to several challenges. The main challenges, according to our experience, are as follows:

- Different features of the devices regarding user interface, such as screen size, number of colors, screen resolution, sounds, and keyboard layout;
- Total heap capacity and maximum application size;
- Different profiles (MIDP 1.0 and MIDP 2.0);
- Different implementations of a same profile in J2ME (different JSRs);
- Proprietary APIs and optional packages;
- Device-specific bugs;
- Carrier specific requirements;
- Internationalization;

Despite the manufacturer's efforts to make their devices totally compatible with the J2ME standard specification, some devices have known bugs, requiring a number of device-specific workaround when a programmer has to use these defective libraries. Once again, porting is compromised.

There is also the natural language issue: developers and publishers, which operate globally, inexorably need to translate their games to a great variety of other languages. In some cases, several languages can be included in a single version; however, most of the times, it is more convenient and efficient, in terms of final size of the application, to have several versions, one for each language.

Generally, wireless carries demand that a game be available to a minimum amount of distinct devices (about twenty five families), adding to that some language variations, as well as other factors, the number of SKUs (Stock Keeping Units) for a single game can easily get close too more than a thousand. Suppose a developer wants to sell its game all over Europe, USA and Latin America. In order to cover these territories it must release its games in, at least, six languages, for example, English, Portuguese, Spanish, French, German and Italian, assuming seventeen wireless carriers, each with their own naming conventions and demanding the usage of specific APIs and features, and a total of twenty-five different families of devices (at the lowest), we would end up with 2,550 SKUs. From these numbers, it should be obvious to notice that meeting the portability requirement is a critical issue in this business.

Therefore, providing consistent maintenance of these game versions or variations becomes a more expensive and error-prone task, as the functional common core of the application is normally dispersed across such variations. Despite this, market pressure demands that the time span for game development be around three or four months, after which a game must be available in dozens of platforms, wireless carries and in several different languages. As discussed previously, meeting the portability requirement is a critical issue in this business.

A well-defined porting process is vital for the development of mobile games. Ideally, one efficient porting process must fulfill the following requirements:

- It should be scalable: once the core game play is fully developed, creating a new version shall not be an extremely complicated and time consuming task;
- It should provide a good level of maintainability: new features or bug fixes shall be easily replicated through all different versions of the game;
- Total development effort shall be reduced regarding both time and cost issues, as well as, number of people involved;
- It should be adaptive, so developers can extract the most out of the particular features of each device.
- It should provide correctness and reproducibility, so that, if needed, developer can create the same SKU many times without including additional errors.

In this paper we are going to present the porting process defined and currently used by Meantime: The MG2P (Meantime Game Porting Platform). As it will be shown in Section 4, this porting process adheres to the requirements stated before.

3. Related Work

Current approaches to porting can be classified in the following categories: device-independent frameworks, pre-processing tools, general guidelines, specific guidelines, semi-automatic services, and formal approaches.

The Unified Mobile Application Framework (UMAK) [UMAK 2006] is a framework that provides reusable components and a set of tools combined in an application that intends to ease the porting process of J2ME applications. The reusable components are used as a framework, the developer writes his application targeting the framework's API instead of J2ME's API and any native APIs that a particular vendor may provide. UMAK also provides several facilities for seamlessly supporting build-time transformations of images, use of different images based on screen size, grouping of images, multi-language text, and selection of the most optimized audio format. It also uses a code pre-processor for insertion or removal of variation points in the code. However, developer cannot add support for new devices. Moreover, UMAK-defined directives used in the code prevent it from compiling during development, which makes code editing more difficult because of error warnings on IDEs.

Similar to the approach presented in this paper, J2ME Polish [J2ME Polish 2006] provides a pre-processing feature by which guidelines define a conditional compilation of the source code (written to comprise several platforms) according to the device in question. Besides that, J2ME Polish contains a device database (described with their peculiarities), which is used in the process of instantiating a specific variation. However, such database is not open and contains recurring bugs.

Some approaches are specific to source and target devices, and consist of a descriptive document of the characteristics of these [Motorola 2004]. They specify the direction (source/target devices) of portability, but are more descriptive in terms of device features than prescriptive in terms of actually carrying out the porting.

Other approaches offer broader guidelines [Facon 2004], involving a research of the target device, an architecture reorganization and source code transformation, but underestimate the effort necessary for this last task.

A more recent approach [Tira 2004] consists of specifying reference devices and specific guidelines for programming for these devices, and then generating the code for the target device with tool support. The tool carries its tasks by using a transformation system following principles similar to those of Apects-Oriented Programming (AOP) [Kiczales et al. 1997]. Such approach is described as automatic, but demands that the game be coded according to the guidelines, which may itself be a resource-demanding task.

Some recent formal approaches [Gajos et al 2004; Cardone et al. 2002; Hua Chu et al. 2004] propose an abstract specification of the elements of Graphical User Interface (GUI), devices characteristics, and user interface usage scenarios. Based on these, they generate code for different types of GUI. Unfortunately, such approaches depend on hypotheses which restrain the GUI's organization, have a considerable specification effort and address only GUI, not taking into consideration issues like heap memory and maximum application size constraints.

In another previous work, a language-independent way to represent porting-related variability is provided, and it is shown how it can be used to port J2SE applications to a J2ME product line [Zhang 2003]. This is similar to the program transformation approach we describe, but differs in that ours relies on language-specific constructs and variation points are identified in the program transformation language, whereas the latter is language independent, but requires the developer to explicitly specify the variation points in the base code.

In previous research [Alves 2005], we have also used conditional compilation (pre-processing approach) to manage porting issues in mobile games. However, as discussed in next section, this paper introduces a significant evolution of the previous work. The new process addresses more variability issues, such as language, service carriers, and more device features, providing a strong improvement of porting scalability. Accordingly, the build system has been significantly improved to generate a number of SKUs that is an order of magnitude higher than what had been previously accomplished.

4. Porting Process

In the remainder of this section we will describe two porting approaches, one ad hoc method, no longer used by Meantime (section 4.1); and a more elaborated and efficient one (section 4.2), which is based on a mobile domain database, a base architecture and a robust build system.

4.1 Previous Ad Hoc Process

The previous process adopted by Meantime was based on an incremental approach [Sampaio et al 2004; Alves et al 2005]. Initially, one version of the game was developed for a specific mobile device and, afterwards, the source code produced was replicated for other devices, until all device families were attended. The whole process was based on copying and pasting the original developed code, making the necessary changes like, for instance, user interface simplifications, animations and removing images that were not strictly necessary in order to adapt it for the porting device.

By using this method, one source code was created/maintained for each ported device. One of the main problems with this approach is that, once a common bug (a bug related to all versions) is found or any new feature has to be included, a lot of time is necessary in order to replicate these alterations

throughout all versions of the game, generating, hence, more costs and demanding more time.

In order to clarify how painful this process can be, bellow follows the description of some of the problems faced by us during the development of a game based on this approach.

The game was initially developed for a Nokia Series 60 device, with the purpose of making the most usage of its capabilities. By the time this game was developed, Nokia Series 60 devices were amongst the most powerful devices available, therefore, this version of the game became an upper limit regarding heap memory usage, processing power, and application size. This version of the game was about 180KB and used more than 1,5MB of heap memory.

After completing this first version of the game, we have decided to port it to Nokia's Series 40 devices, especially because these two device families have similar specifications and APIs in common. Besides that, Series 40 represents a fair portion of the devices with J2ME in markets like Europe, Asia and Latin America. However, this series presents at least two significant constraints: maximum application size of 64KB and heap memory limit of 200KB. Regarding theses constraints, we have decided to decrease the number of levels in the game, without, however, making it too short. Besides that, since the levels were shortened because of the smaller screen size of the target devices, the level files became smaller as well. Another choice made was to reduce image size, for the same reason stated before. With all this work, the game size was reduced from 186KB to 63KB.

Another problem found is that, series 60 devices have a serious bug preventing the garbage collector from completely freeing the memory used by an image object. This way, there is a memory leak every time an image resource is allocated and freed more than once during the execution of the application. Therefore, the approach used on the Series 60 version was to load all images during game start up and leave them in memory as long as the application is running. Since the heap memory of this series' devices is large enough, adopting such technique was not an issue. Series 40 devices, however, have the 200 KB heap size constraint; hence, this technique is not possible. One suitable image allocation policy for this platform is to keep in memory only those images that will be used immediately, and make them eligible to be collected as soon as they had been used, since garbage collection on Series 40 does not cause memory leak. This way a specific code was developed to support this variation of approach.

From the example shown before, we can notice that porting a game from one device family to another involves a high amount of changes, and that these changes are not only related to the source code, but to the resources files as well. Having all this in mind, we can conclude that this previous porting approach is not applicable to large scale porting. It generates an extra coding effort when it is necessary to develop a version to for a new device, or when there is a change request that affects all versions. The process to be described in the next section solves most of these management problems and it is easier to use as a background process.

4.2 The MG2P (Meantime Game Porting Platform)

Nowadays, we address the porting problem using a new approach, the Meantime Game Porting Platform (MG2P). As the company gained more experience in this area, we were able to come up with a solution that, for our particular case, has made the porting process more efficient (as it will be shown in section 5).

As stated before, this approach is grounded on three major pillars: a mobile domain database, a base architecture (MBA – Meantime Base Architecture) and a robust build system (MBS – Meantime Build System). One of the main advantages of the MG2P is that there is only one source code, shared by all versions. As a result, the code is easier to maintain, reducing overall costs.

In the following subsections we will detail further each one of the modules that are part of the MG2P.

4.2.1 Mobile Domain Database

Based on our past experiences we have conducted an analysis of the mobile domain in order to verify the underlying variability and also similarity between its elements. Our first step was to abstract the hundreds of devices into families. By defining families of devices, we were able to group together those devices that have similar characteristics and known issues. Some of these characteristics are very important for the porting job, such as: the real size of the screen; the version of MIDP/CLDC; the size of the heap memory; the maximum size of the final JAR file and the presence of the Multimedia API for sound playback. We have identified the most relevant features and described their variability, categorizing them as follows:

- ***Device specific variations***: differences regarding the device itself, like screen sizes, key codes, sound playback approach, etc; presence of vibration API; image transformation API, etc;
- ***Game feature variations***: presence of specific game APIs;
- ***Known issues***: general issues encountered in more than one device;
- ***General variations***: support of multi-language and graphical font;
- ***Feature variations***: presence or not of features like game ranking upload.

Once these characteristics have been mapped out, we were able to aggregate the most significant devices of each manufacturer into families, being each family a combination of similar characteristics.

For each family, we have elected a device as being its representative. Usually, this family representative is the less powerful device of the family (in terms of processor speed, memory capabilities and overall performance), and also the one that has the highest amount of known issues for that family. As the porting job to a specific family is done by targeting its representative, we can assume that, if the game runs fine on this device, then we will probably have the job done to all other devices of the same family, without hassles. However, we do test the version generated on some other devices of the same family.

It is important to notice that, although some manufacturers, like Nokia, have created the concept of device families, the concept of families defined by us is not necessarily equal to the ones specified by the manufacturers. Our categorization is based on criteria that are relevant for the porting job. In our case, families are defined based on devices' characteristics and past experiences. This information is compiled in a spreadsheet and must be used throughout the porting job.

Since our goal with this new approach was to have a single code base shared between all different versions of the game, we have mapped all these variations and sub-variations into preprocessing tokens, which are heavily used during development.

Such variability often affects both the source code and the resource files. For example, to implement a sound API for a specific family, we need to change the source code itself, as well as selecting the sound files in the appropriate format (compatible with that API). In order to map such variability into the source code, we rely on mapping specific features under preprocessing tokens; to map such variability regarding the resource files, we rely on the build system to select the proper resources for each family (section 4.2.3).

Table 1 shows an example of preprocessing tokens related to screen size variations and also to the use of a game specific API. Table 2 lists some of the preprocessing tokens used to define a particular family of Nokia handsets.

The NOK1 family represents Nokia phones with MIDP 1.0 and screen size of 128x128 pixels. The token device_screen_128x128 is used specify the size of the device's screen. The key codes for all Nokia handsets are defined by the token device_keys_nokia. The canvas class of these phones must extend a proprietary class of Nokia UI package. This variation is handled by the device_graphics_canvas_nokiaui token. As the family is MIDP 1.0 compliant, it does not have built-in game specific classes, image transformation API and also do not have sound capabilities. These features are implemented by using Nokia's proprietary classes as well as Meantime's internal classes that are under the tokens device_graphics_transform_nokiaui, game_sprite_api_meantime and device_sound_api_nokia.

Category	Sub-Category	Variation	Token
Device specific	Screen Size	128x117	device_screen_128x117
		128x128	device_screen_128x118
		130x130	device_screen_130x130
		128x142	device_screen_128x142
		128x149	device_screen_128x149
		…	…
Game Features	Usage of Tiled Layer API	Meantime API	tiledlayer_api_meantime
		MIDP 2.0 API	tiledlayer_api_midp2
		Siemens Game API	tiledlayer_api_siemens

Table 1. Example of preprocessing tokens.

Family ID	Tokens Used
NOK1	device_screen_128x128
	device_keys_nokia
	device_graphics_canvas_nokiaui
	device_graphics_transform_nokiaui
	game_sprite_api_meantime
	device_sound_api_nokia

Table 2. Preprocessing tokens for the NOK1 family.

As it will be shown in section 4.2.3, the build system uses the preprocessing tokens defined in the property file to preprocess the base code. After preprocessed, the code is then compiled. In order to correctly package the code for deployment, the build system uses the resource information available in a property file, so that it can obtain the right resources to be used by a specific family.

4.2.2 Meantime Basic Architecture (MBA)

The result of domain implementation is the reference architecture, the MBA. This architecture embeds porting-related variability, which is identified by the preprocessing tokens and by a suggestion of directory structure to organize all resource files.

The basic idea of this architecture is to be a guide for developers, helping to produce code that follows the standards adopted by the company. All design patterns, classes and recommendations came up from past experiences. The architecture was designed to make the porting job easier and avoid common mistakes during the design and development phases.

All basic variations are handled by the MBA. This approach speeds up the development of a new game as

it allows developers to focus only on the game features, passing to the architecture the responsibility of handling major code variations between the devices.

The development team begins the coding phase from a stable version of the architecture, creating the base game core. This unique base code will evolve up to the full game itself, and for all required families. Game specific features should be developed taking into account the fact that the source code must be portable across all the families defined by the mobile domain database. Some coding standards and guidelines are adopted in order to accomplish that. If a new general variation (not mapped by the architecture) appears during game development, it should be analyzed and, if it is the case, incorporated in the mobile domain database as one or more new preprocessing tokens and the MBA will be updated properly.

4.2.3 Meantime Build System (MBS)

We have developed a build system based on Ant [Ant 2006] and Antenna [Antenna 2004] in order to allow the compilation and resource packaging during the deployment phase. For each family described in the past sections, we have a property file that lists all the preprocessing tokens used for a particular device, as well as, paths for resource files that should be used to pack the SKU. The property file also includes other things the MBS needs to know in order to compile and package the application for a specific device. The final result of the build system is always the executable SKU, ready to be installed in the device.

Figure 1 represents a scheme of the MBS functionality. It shows the core game code and also the resources repository. In this example, a total of four SKUs are created, two for the MOT1 family and two for the NOK1 family, one in English (en) and another in Portuguese (pt).

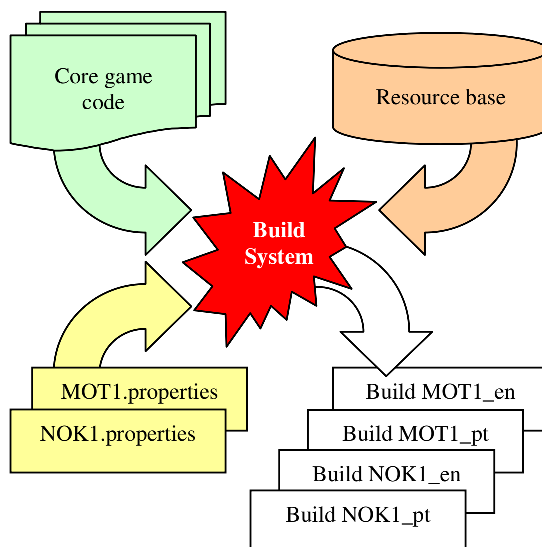

Figure 1. Meantime Build System

The build system uses preprocessing tokens defined in the property files to preprocess the base code. This preprocessing step selects the appropriate pieces of code for the particular family and comments all other pieces of code. After preprocessed, the code is compiled. Thus, the base code contains code for all families in a unique set of source code files.

To package the application, the build system uses the resource information also available in the property file to select the right resources for a given family. For example, if a device has a small screen, its property file should refer to appropriate directory with smaller image files.

Property files are designed so that they contain minor game specific information. Thus, property files provided by the architecture can often be reused by many games, with almost no changes. The list of preprocessing tokens is predefined for each device family, as described in previous section. It uses tokens already coded in the architecture code. The set of resource paths for each device is based on resource directory structure suggested by the architecture.

4.2.4 Porting Activities

According to Meantime's experience in developing mobile games, we have observed that it is extremely important to pay close attention to the portability problem from the very beginning of the development cycle. The earlier the development team focuses on portability issues, the easier the porting process will be. Therefore, we have included some porting activities in the formal Meantime's development process. All phases have been affected. During the analysis and design phase, for instance, it is important to define which key features will be part of the game core, that is, the functionalities that will be present in all versions of the game to be deployed. In the same way, it is also necessary to identify what additional features (if any) will be made available exclusively to a certain family of devices. During the coding phase, it is important to focus on code quality, maintainability and legibility. This can be achieved by, for instance, defining a set of good coding practices and making them available to all programmers, and also by defining a basic architecture from where all starting projects should inherit.

By having all these pieces of information regarding variability amongst versions of the same game beforehand, and all the device specific information mapped into properties files (provided by the MBA), the porting process occurs without major difficulties. In order to generate a version for a specific family, we only need to submit the appropriate property file to the MBS. Once this first version is created, minor game specific adjustments might be necessary to get the game running perfectly in this family. Such adjustments are often related to the position of

graphical elements in the screen and can be generally handled in less than a day.

As the knowledge base grows bigger and more devices families are being mapped, the process of porting to a new family, not previously provided by the MBA, becomes increasingly easier. The probability that a new family has all its characteristics already mapped in the code base becomes very high, transforming the porting process in a mere creation of a property file, therefore, reducing the porting time to less than one hour. In this case, no coding is needed.

For carrier variations, we just need to copy and update the property files to match carrier specific requirements. Variations by carriers often refer just to application naming conventions or some minor information that requires no coding. Even when a carrier requires some specific feature, such as an additional logo screen, the code variation is implemented in the core game code under a certain preprocessing tag. Then, all we need to do is enable that tag when generating the SKUs for the respective carrier. Generally, this coding effort can be concluded in a few days.

5. Results

In this section, we will illustrate how we were able to improve the porting process using the MG2P, as described in this paper. We are going to analyze the results obtained in five different games: Zaap, Big Brother Brazil, Zaak, Madagascar's Jungle Mix and Ronaldinho Juggling [Meantime 2006]. For this analysis, we have considered the following variables:

- Project start up date;
- Total project duration;
- Number of families;
- Total number of devices;
- Number of available languages;
- Number of Stock Keeping Units (SKUs).

The development of the game Zaap started on April 1st, 2004. It is non connected platform/puzzle game, created by using an ad hoc process with no base architecture or advanced build system. The same SKUs were used by all carriers since it was not needed to change the SKU nomenclature, nor any of the application's attributes

The game Big Brother Brazil had its development cycle started on October 1st, 2004. It is a connected Tamagochi like game and it was created using the first version of the build system, base architecture and porting process. The same SKUs were used by all carriers. As Zaap, it was not necessary to change the SKU nomenclature, nor any of the application's attributes.

For the game Zaak, the development cycle started on May 25th, 2005. This game is a refactor of the game Zaap, using an initial draft version of MBS and MBA. It is a connected game (players can publish their score in an online rank) and it was distributed by many wireless carries throughout the globe.

The development of Madagascar Jungle Mix started on May 26th, 2005. It is a puzzle game, also created using a preliminary version of the MBS, MBA and porting process. It is not a connected game and the same SKUs were used by all carriers since it was not needed to change the SKU nomenclature, nor any of the application's attributes.

Ronaldinho Juggling is the newest game amongst the ones mentioned here. Its development started on December 22nd, 2005. Differently from the other ones, it was created based on the new technologies presented in this paper. It is a connected, casual, one button game, where players can publish their scores on an integrated global ranking system: the Meantime Arena [Meantime Arena 2006]. It was published by many carriers around the world, but unlike Zaak, most of these carries have requested specific changes on the game's attributes, languages and SKU nomenclature.

Table 3 presents an analysis of these five games in order to show how the base architecture and the build system have improved the development process.

Game	Dev. Time	Ported Families	Ported Devices	Languages	SKUs
Zaap	4 Months	6	103	1	6
Big Brother Brazil	3,5 Months	10	123	1	10
Zaak	3 Months	9	122	2	18
Jungle Mix	2,5 Months	9	132	1	9
Ronaldinho Juggling	3,5 Months	24	276	5	2316

Table 3. Games analysis

As it can be observed, the average development time for all games is about 3 months. However, the use of MG2P the build system and the base architecture increased the number of ported device families and SKUs substantially. The difference between Ronaldinho's game and the others is extremely significant.

Ronaldinho's game has demanded such high number of SKUs because most of the carriers where the game was published have specific requests about nomenclature, attributes related to connection and languages. Without this new porting approach and the underlying technology, it would not be possible to generate and manage this large amount of SKUs. Other games have also been created using this technology, for instance, Senna's Best Lap and Senninha's Race games, where the number of SKUs and the total development time are almost the same as Ronaldinho's game.

It is clear to see that this new approach has allowed us to publish our games in more wireless carriers than before (even considering a great number of restrictions) keeping a reasonable low budget and increasing the number of deployed versions.

6. Conclusion

Porting is an essential activity in the development of mobile applications, particularly in games. The great variety of devices together with business requirements demands that the same application be available in lots of different devices in a short period of time. This adds complexity to the development process, and also to software maintenance. In order to compete globally in this market, all these issues need to be properly handled. In our case, the support of dedicated in-house toolchains and porting oriented process were essential to accomplish our objectives.

This paper presents the MG2P, a novel and successful porting process for easily creating thousands of different versions of mobile games. MG2P includes a set of Meantime developed techniques, tools and artifacts, supported by some industry standard technologies. MG2P also defines all activities required to develop massively ported games, as well as practices to continuously improve the porting process.

It is known that some big companies have the same number of ported devices, maybe using different technologies, however, no one has ever published their methods, keeping their technologies in secret.

The results of applying MG2P show that it was possible to increase the number of SKUs developed from 6 to 2316 and increase the number of ported devices from 103 to 276 with almost the same budget. Hence, these results show a significant success of applying MG2P to scale the number of ported devices.

The proposed process is well suited to a small company like ours because, by adopting it, we are able to develop a game with a team of 3 engineers working in the game core (following the guidelines to let the code ready for porting), and, at the same time, only 2 more engineers are necessary to handle the porting job. The level of maturity of our process allows the development of more than two thousand versions of a single game with a very small team; this would be impossible using ad hoc processes, as the one used previously by Meantime. As a comparison, companies like Jamdat and Gameloft, leaders in mobile gaming market have teams from 50 to 100 engineers and QA team just to port and localize the game in order to fulfill carriers' requirements. We could have opted to hire another company to port our games to other devices, but if we consider that a porting company charges about U$ 1,500 for each single ported version, we are saving a lot of time and money by using this new process.

Besides that, this porting process allows us to compete in a global marketing, where devices coverage is one of the most important requirements, which determines whether or not a given game will accepted by wireless carriers.

Acknowledgements

The authors would like to thank FINEP and FACEPE, two Brazilian foundation organizations, which have supported process documentation and final evolution of MG2P until current mature state. Additionally, the authors would like to thank Meantime for allowing us to publish the results and details of the process.

References

ALVES, V., CARDIM, I., VITAL, H., SAMPAIO, P., DAMASCENO, A., BORBA, P., AND RAMALHO, G., 2005. Comparative *Analysis* of Porting Strategies in J2ME Games. In *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05)*, Budapest, Hungary, pages 123-132, September 2005. IEEE Computer Society

ANTENNA, 2004. Available from: http://antenna.sourceforge.net [Accessed 17 August 2006].

ANT, APACHE, 2006. Available from: http://ant.apache.org/ [Accessed 17 August 2006].

CARDONE, R., BROWN, A., SMCDIRMID, AND LIN, C.. 2002. Using mixins to build flexible widgets. In *AOSD '02: Proceedings of the 1st international conference on Aspect-oriented software development*, pages 76–85. ACM Press, 2002.

FACON, X., 2004. Porting Your MIDlets to New Devices. Available from: http://www.microjava.com/articles/techtalk/ [Accessed 17 August 2006].

GAJOS, K. AND WELD, D. S., 2004. Supple: automatically generating user interfaces. In *IUI '04: Proceedings of the 9th international conference on Intelligent user interface*, pages 93– 100. ACM Press, 2004.

HUA CHU, H., SONG, H., WONG, C., KURAKAKE, S., AND KATAGIRI, M., ROAM, 2004, a seamless application *framework*. *Journal of Systems and Software*, 69(3):209–226.

J2ME POLISH, 2006. Available from: http://www.j2mepolish.org/ [Accessed 17 August 2006].

KICZALES, G., LAMPING, J., MENDHEKAR, A., MAEDA, C., LOPES, C. V., LOINGTIER, J. M., AND IRWIN, J., 1997. Aspect–Oriented Programming. In *European Conference on Object–Oriented Programming, ECOOP'97*, LNCS

1241, pages 220–242, Finland, June 1997. Springer–Verlag.

KNUDSEN, J., 2003. Understanding JSR 185. Available from: http://developers.sun.com/techtopics/mobility/midp/articles/jtwi/ [Accessed 17 August 2006].

MEANTIME, 2006. Available from: http://www.meantime.com.br/ [Accessed 17 August 2006].

MEANTIME ARENA, 2006. Available from: http://www.meantimearena.com/ [Accessed 17 August 2006].

MIDP, MOBILE INFORMATION DEVICE PROFILE, 2006. Available from: http://java.sun.com/products/midp/ [Accessed 17 August 2006].

MOTOROLA, 2004. Porting guide: Motorola i95cl to T720. Available from: www.microjava.com/articles/MJN_Porting.Guide_i95cl-T720.pdf [Accessed 17 August 2006].

QUALCOMM, 2004. *Qualcomm Brew Home*. Available from: http://brew.qualcomm.com/brew/en/ [Accessed 17 August 2006].

SAMPAIO, P. DAMASCENO, A., SAMPAIO, I. ALVES, V. RAMALHO, G. & BORBA, P. (2004). Portando Jogos em J2ME: Desafios, Estudo de Caso, e Diretrizes. III Workshop de Jogos e Entretenimento Digital. (pp. 82-88). Curitiba: Sociedade Brasileira de Computação

SUN., 2004. *Java 2 Platform, Micro Edition (J2ME)*. Available from: http://java.sun.com/j2me/ [Accessed 17 August 2006].

SYMBIAN, 2004. *Symbian OS*. Available from: http://www.symbian.com [Accessed 17 August 2006].

TIRA WIRELESS, 2004. *TiraJump*. Available from: http://www.tirawireless.com/jump/, [Accessed 17 August 2006].

UMAK, 2006. Available from: http://www.unifiedmobiles.com/. [Accessed 17 August 2006].

ZHANG, W., JARZABEK, S., LOUGHRAN, N., and RASHID, A., 2003. Reengineering a PC-based system into the mobile device product line. In *Proceedings of the Sixth International Workshop on Principles of Software Evolution (IWPSE'03)*, 2003.