Concurrent Processing in Cloud-based AppInventor Development Environment

Carlton McDonald,
School of Computing and Mathematics,
University of Derby.

Abstract

AppInventor (AI) has recently become the language of choice for learning to program. How suitable is it for teaching undergraduates? How efficient are the programs that are developed with AI? Is it likely to replace Andoid development with the Android Development Tools? How can AI Apps be made to execute as efficiently as Possible? AI is an innovative approach to learning to program and has reduced typical development time by 90%. This paper examines the reasons for AI's popularity as a first programming language, the inherent inefficiencies and the way in which the lack of threads can be overcome.

Programming History: From Mathematicians to novices

Programming has traditionally been the domain primarily of computer science graduates and mathematicians. In the late 1990's Visual Basic (VB) attempted to make programming more accessible to non graduates with limited success. The emergence of cloud-based IDEs [5] has presented greater accessibility to programming anywhere, from desktops, laptops and to a lesser extent mobile browsers. Google's AppInventor (AI) for Android developed a loyal following but was taken over by MIT in January 2012, it is still in beta but has provided an opportunity for novices to be more productive in producing Android Apps than traditional Android application development. Why is AppInventor suitable for novice programming? Is it suitable for commercial products? What are the limitations of the product and how efficient are the Apps produced by AI?

Language learning is much easier if one is able to live in the country and converse in the language that one is attempting to acquire. Learning to program is no different. Students that learnt to program in assembler couldn't do much homework, were not very productive due to the volume of code (and its relative complexity). High level Languages, like Algol and Pascal, fared little better. This is because, on a whole students learned to program usually at institutes of learning. By the time students could afford their own computer on which to develop programs in their own time, get immediate feedback on attempts to write code, without going to the place of learning the amount of software being written for that platform in commercial circles was limited to a few multinational organisations. Very little development took place on PCs.

JavaScript changed all of that. JavaScript although not regarded as a structured, strongly typed, or object oriented language has enabled many applications (we stopped calling them programs by the late 1990's) to be written by just about anyone with a browser environment available to them. Ten years ago it would have been an innovative approach to learning to program if the interactive development environment (IDE) that students were introduced to consisted of a browser and a JavaScript development Environment (such as Aptana Studio).

The browser environment is still relevant because it is now the case that more people access the Internet through a mobile device than through desktop computers. The academic view that a good language for learning to program must be Object Oriented may not be the best view of learning to program. Perhaps the best language to learn to program in is the one that allows the learner to express themselves on the device that is available to them and develop required apps.

Why is AppInventor suitable for novice programming?

AI is an environment developed for those learning to program "who have never programmed before", [7]. MacKellar used AI as part of a Health IT undergraduate course. AI requires its users to design storyboards

representing the user visual experience from which blocks of jigsaw puzzle style components are connected together to produce the code that executes on a mobile device.

AI Developers are able to visualise the program instructions, and the inability to connect blocks inappropriately is effectively rudimentary compilation checking of the blocks. The fact that novices are able to program with AI means that expert programmers are also able to develop Apps in the environment albeit with some frustrations with respect to the efficiency of the apps they produce.

AI is quite forgiving in terms of the way in which an app can be made to work in a variety of ways. AI is extremely flexible. This is one of the reasons that novices thrive in the AI development environment. Individual solutions can usually be made to work. However, for mobile apps one of the most important considerations is efficiency [8, 9]. Mobile app developers will have some frustration with AI as there are a number of features lacking which make it difficult to produce applications of reasonable complexity that are anywhere near as efficient as traditional Android development with that Android Development Tools (ADT).

Is AI suitable for commercial products?
Is AI only suitable for novice programmers? For much of AI's early existence it was not possible to upload the app produced by AI due to "technical limitations" according to the AI FAQ page, this has now been updated as it is possible to upload app to the Android Market (Google Play).

There are many apps for which, for the user, the most important thing is the utility of the App, in other words, does it provide the required functionality. Provided there isn't a noticeable impact on battery life users are often happy. They do not have any interest in the underlying technology (either hardware or software, java or C#, OO or C). For this reason there are very many utility Apps that are neither performance nor resource intensive that are perfectly suitable for millions of downloads.

What are the limitations of AI?
The first limitation of AI generated apps is that they have a very large footprint. A simple app with a button for changing the background colour occupied 1.3MB in comparison to a 16KB Eclipse ADT generated app. Consequently, the speed of execution of AI generated apps is noticeably slower than ADT generated apps. The speed of an app is related to the amount of processing performed by the app. An AI generated App has to do far more processing than an ADT generated app for a number of reasons: a lack of encapsulation and a lack of threading.

The inability to encapsulate variables and associated methods makes AppInventor a Visual Basic style environment perfectly suited for novices, but an experienced OO developer familiar with inheritance, polymorphism and reusability will be somewhat frustrated with the duplication required for an app modelling world objects (both real world and game world). Each instance of the real world object must be defined separately, and although procedures may be used to accomplish some of the effects of code reuse, another limitation is the lack of dynamic allocation.

If a developer wishes to create, for example, an app that produced *n* buttons based on user input, a maximum number of buttons would need to be created at design time and hidden. These buttons would therefore occupy unnecessary memory and in the majority of the times when the app is used, this memory would be unused.

How efficient are the Apps generated by AI?
In terms of execution speed, the time taken to execute an app is dependent on the amount of memory locations that have to be searched in order to find program references (variables, procedures and functions). The more memory

that is occupied the longer it takes to find what the processor is looking for. The 'fetch' stage of the fetch-execute cycle is therefore longer in appinventor.

AI apps are actually a visual representation of Scheme programs [1]. Which in turn are compiled via Kawa to Java bytecode. Kawa is both a toolkit for compiling other languages into Java bytecodes, and an implementation of the Scheme language implemented in Java [4].

AppInventor Features

As a programming language AI encourages the developer to think of the user experience first. The author's experience of teaching both novices and experienced programmers AI, is that experienced programmers are more likely to ignore the advice to develop the storyboards first, i.e. the visual representation of the app. AI can be used to generate the screens directly and consequently a separate tool is not needed. The result of this integration of a drag and drop screen designer with the coding process is that once the screens are designed, AI provide immediate feedback of the appearance of the screens either on the built in emulator phone, or on a connected device.

The ability to generate the interface, requires knowledge of layout design, which to some extent is intuitive [6]. The non intuitive concepts required for app interface design are containment, and padding. Containment requires the designer to utilise horizontal and vertical arrangement components to be placed on the designer screen, the may be recursive layouts within layouts The second, non intuitive concept is that of padding: using labels with no text in horizontal and vertical layouts to force other components to have gaps between them.

As a development language, the Java Blocks Editor in which the program is 'assembled' via the visual blocks connecting to each other, captures naturally the basic building blocks of programs: sequence, selection and interation. Additionally, The event handling is entirely intuitive, in that each user interface component and its associated events are automatically available to the develop solely by providing the components on the design screen.

Programming is therefore a system utilising the blocks that are available and connecting them together. For example:

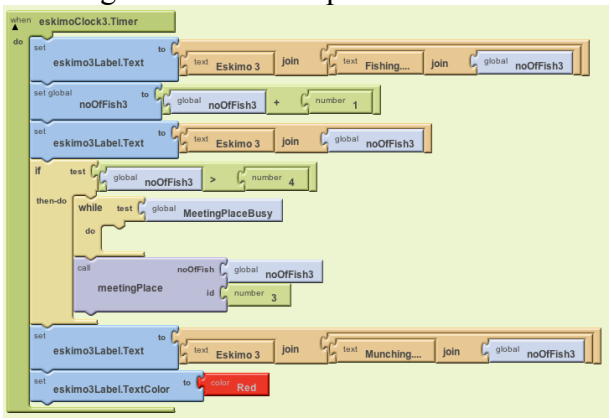

Figure 1 AppInventor Code Blocks

This code snippet, is one of five eskimo threads [2]. Ben-Ari's book was a precursor to Ada's rendezvous and used a scenario of different types of threads interacting in a mutual exclusive zone, where a single transaction could take place securely. AI does not support threads. In 2012, a post at app-inventor-for-android highlighted the absence of threads as the primary issue for which "many people are not using 'app inventor' for development" (wora…, 2010).

Threads provide the ability to perform concurrent processing. Threads also provide a means of optimising even operations such as sorting [2] and allow the processor to not have to wait for one process to complete before another starts. If AI had threads many more developer would use it. The app in Figure 2, is entitled "Thread Simulation" as eight threads of execution are running concurrently on the AI emulator as a means of illustrating the ability to concurrently process in AppInventor.

Threading in AppInventor

Everett (2013) was able to accomplish more efficient processing of animated objects by simulating threading in a classic space invader

game. This was accomplished by utilising a number of timers. The author was able to demonstrate a simulation of Ben-Ari's Ada, Hunter, Baker, Eskimo thread paradigm by means of a timer for the Ada process, a second timer for the Hunter process, a third timer for the Baker process and five timers for each of the five eskimo processes. The timers are independent and provide the inherent behaviour required to control concurrent processing.

Figure 2 Thread Simulation Screenshot

AppInventor Inefficiencies

Development for mobile devices should be as efficient of processor time as possible in order to conserve battery power. AI does not have explicit objects and therefore code reuse, and inheritance are not available. This requires code to be duplicated for what would be an instance of an object. The most notable inefficiency is the inability to create executable components as and when they are required. As demonstrated above, the lack of multithreading can be overcome by means of timers but the synchronisation needs to be handled by the developer, as does the implementation of mutual exclusion zones within the code where only one thread can enter at a time.

Visual Programming

AI has clearly demonstrated that a visual assembly of code is an intuitive mechanism for development. The inherent inefficiencies and limitations of the language and the fact that serious OO developers are unable to write efficient code one can expect that for AI to be the dominant development language of the future there would need to be the facility to define objects and inheritance hierarchies and most importantly the facility to declare objects as required, and destroy objects when they are no longer required. AI is in beta but as more APIs are added to the environment these OO facilities can be sacrificed for the ease of development and the rapidity with which a prototype can be produced.

Conclusion

AI is a novel programming language that has made app development available to novices. The code produced by novices is more likely to be less efficient than that of experienced developers nevertheless there are many apps that the inefficiency of the novice's approach is insignificant as a percentage of the actual processing improvement of an 'expert' solution. Processor intensive apps will be slower when developed using AI even for experts, due to the lack of OO features.

AI's main benefit to an experienced programmer is as a rapid prototyping tool. The ability to assemble an app in a tenth of the time required using the ADT. AI has shown that visual languages provide greater accessibility to beginners but the lack of an underlying Object Oriented system is a serious drawback to adoption by experts or application of processor intensive apps.

References

[1] Abelson, H., Dybvig, R. K., Haynes, C. T., Rozas, G. J., Adams, N. I. IV, Friedman,

D. P., Kohlbecker, E., Steele, G. L. Jr., Bartley, D. H., Halstead, R. Oxley, D., Sussman, D. J., Brooks, G., Hanson, C., Pitman, K. M., Wand, M., Clinger, W. and Rees, J. (1991), SIGPLAN Lisp Pointers, Volume IV Issue 3, ACM

[2] Ben-Ari, M. (1982), Principles of Concurrent Programming, Prentice Hall Series in Computer Science.

[3] Everett, Simon. (2013) A Classic Space Invader Game written in AppInventor, Programming Principles Assignment Submission, BSc. IT, University of Derby.

[4] Kawa, the Java-based Scheme system. http://www.gnu.org/software/kawa.

[5] Krill, Paul. "The rise of cloud-based IDEs; Development tools in the cloud enable programming from anywhere, but they're not suited for all app dev needs." InfoWorld.com 12 Mar. 2013. Computer Database. Web. 10 May 2013.

[6] Adrian Kuhn, David Erni, Oscar Nierstrasz, October 2010 Embedding spatial software visualization in the IDE: an exploratory study
SOFTVIS '10: Proceedings of the 5th international symposium on Software visualization

[7] MacKellar, Bonnie. "App Inventor for Android in a Healthcare IT Course. *SIGITE'12,* October 11–13, 2012, Calgary, Alberta, Canada. ACM.

[8] Marcu, M., Tudor, D., Fuicu, S. Power Efficiency Analysis of Multimedia Secured Mobile Applications. Proceedings of the 6th International Wireless Communications and Mobile Computing Conference, 2010, pp 387-391, ACM.

[9] Marugappan, Anand and Liu, Ling (2010) An Energy Efficient Middleware Architecture for Processing Spatial Alarms on Mobile Clients, Mobile Networks and Applications, Volume 15 Issue 4, August 2010, pp 543 – 561, Kluwer Academic Publishers

[10] Wora…@gmail.com (2010), post at app-inventor-for-android. http://code.google.com/p/app-inventor-for-android/issues/detail?id=352