**Technical Review & Implementation Plan**

**Team 16: Walnut Crusade**

Colin Bradford

Nicholas Smith

Matthew Wilson

**Introduction**

The purpose of our project is to convert the current build system of the Apache Open Office project into a system that builds in Microsoft Visual Studio. Currently, the project can be built in Windows only by running CygWin, a Linux environment that runs in Windows, and building through that as if one were running in Linux. The goal is to allow a user to build easily in Windows through Visual Studio. The majority of the work we will need to do for this project lies in the conversion of makefiles into equivalent Visual Studio .vcxproj files, and building these converted modules in Visual Studio.

**Regular/Default Visual Studio Project**

Visual Studio automatically determines how to build a project/solution by default. This process requires that Visual Studio understands how to build/compile the project/solution. In its current state, the AOO project is unrecognizable by Visual Studio. Converting the AOO project to a project that Visual Studio can recognize would provide a clean solution, but would require more work and there would be more of a disconnect between this build system and the current one.

This system would create an authentic development experience in Visual Studio. It would not require the use of any external build systems. This solution provides the best experience for native Windows and Visual Studio developers which is one of our largest problems. However, there are some shortcomings to this solution.

There are two flaws with this approach: First, there would be a disconnect between the AOO project for Windows and UNIX/LINUX;

jan: There should not be a disconnect, because the unix makefiles will be generated from the visual studio files, in order to secure connectivity. If you remember that was the second goal of the project (if time permits). Its important NOT to have 2 parallel build systems once the result of this project is implemented.

Second, this approach will require more work than other solutions. The disconnect may not be a large problem in the short term, but in the long term there could be more problems than with a solution that allows for the projects to be more connected. The second concern is about the amount of work required to complete this approach. This approach would require understanding how Visual Studio builds projects/solutions with more detail, and then determining how to convert the current project such that Visual Studio can take advantage of its default building process.

**GNU Make as an External Build Tool Within Visual Studio**

Visual Studio allows for different build tools to be used when building the projects contained within it. GNU Make is one such build tool. GNU Make is a free build system which relies on *makefiles* to build its projects. These makefiles consist of a set of *rules*, each of which is used to build a *target*. Each rule consists of a list of *commands*, which tell the target how to build, and a list of *dependencies* which these commands use as input. Though a Linux-based build system, a Windows port exists.

jan: yes one of these ports are “nmake” which you describe as the 3 solution.

Because the current AOO build system utilizes makefiles, it is likely that the GNU Make makefiles needed for most modules would end up looking similar to those already existing for that module,

jan: I really doubt that for several reasons:

a) the current makefiles depend heavily on the dmake macro language which is quite different from gnu make (which only have a very limited language). If this was a goal it would be more natural to use cmake, that has an extensible macro language

b) the current makefiles depend on a linux filesystem, and uses a lot of linux utilities (hence the use of cygwin)

c) Some of the makefiles (solver/400/xxx/inc & bin) are generated by ./configure, which would call for a dynamic solution in vc.

d) the structure of the current makefiles, are hidden in central makefiles that are included, and you often have to go 4-5 layers down to find out what actually happens. The makefiles are by far not as simple as indicated here.

minimizing the amount of effort needed to convert most modules.

Utilizing GNU Make to build modules from within Visual Studio would be a very straightforward way of migrating to a new, Windows-based build system. Most of the existing makefiles could be used as a model for new makefiles, and each module could be built within Windows and then subbed in to the old build system for integration testing.

jan: the same can be done in solution 1, I dont see the difference ?

GNU Make is a stable build system, and because it only compiles where it detects changes, it is efficient. New developers would likely be familiar with the format, as it is a popular and well-known build system. GNU Make has also been shown to work well with large projects with many dependencies.

The primary downside of utilizing GNU Make is that using it would mean relying on an external build tool. This is undesirable partly because it means more setup for Windows developers to build the system, and partly because the Windows port for GNU Make has not seen updates in several years, and if issues were to arise they may never be addressed.

jan: that is because makefile projects today either use nmake (part of vc) or cmake, so there are no real need for the pure gnu make. Btw, gnu make for Linux has also not had any real updates for a very long time.

**Visual Studio Makefile Projects**

Visual Studio allows for the creation of makefile projects which allow for the specification of commands to be executed on the command line from within the editor, similar to using the typical make command within Unix. In this manner we would be able to utilize makefiles similar to those that already exist in the project: this would allow us to use the same programs in some cases, as well as a similar layout within the makefile.

The advantage of this approach is that it mirrors the current build system implementation within the project and allows for us to use a lot of the same logic within the makefiles.

jan: see comments in 2)

Creating a makefile project within Visual Studio is simple, and does not require any outside tools, (but may require replacement tools for building some modules.) Migrating modules to the new build system does not require a complex plan.

On the other hand, this solution is less comprehensive than a general system such as GNU Make or CMake, and as such may not scale quite as well, particularly for any modules that may be added later.

jan: ?? why does it not scale well, its more or less the same, please specify.

Additionally, this solution may require more time than alternative solutions due to not necessarily having a general conversion path per module.

**Decision**

Our choice between these three options is to create Visual Studio projects through the traditional methods.

jan: you mean solution 1, I assume ?

This option is ideal because it provides a unified build system for each platform instead of relying on tools such as nmake, and because it will be the most convenient for the end user to utilize.

jan: since you brilliantly and correctly wrote in solution 1), that it is the most time demanding solution, and give long term problems for the project, I think you need to have a big more explanation, why choose the (in your wording) most bad solution.

jan: I agree with the choice, you simply just need to argue why choose that route, and counter all the negative arguments listed in 1). Once that is done you need to extent the document with 2 parts.

a) how do you solve the problems mentioned in 1)

b) how do you solve the problems that I and steve have pointed out to you (solve == solution in detail).