KiCAD Work Packages Definition

Draft Copy

CERN, 2011

Signal Software S.L.
Tlf. 985 30 89 08
Fax 985 30 82 28
Parque Científico y Tecnológico de Gijón
C/ Los Prados, 166, 33203 Gijón (Asturias)
CIF B33936170

	Name	Signature	Date
Authors	Luis F. Ruiz Gago Martín Bosque		
Revision	Martín Bosque		

Changelog			
Date	**Name**	**Pages**	**Changes**

1 ABSTRACT

This document belongs to the *OHR Meta Project* conducted by the CERN. The aim is to develop a tool for designing *OpenSource PCB's* with a functionality similar to commercial tools.

Since there are several *Open Source* tools, we start from one of them, *KiCAD*, to give it the features specified by CERN and could be find at the following link:

http://www.ohwr.org/projects/ohr-meta/wiki/FeaturePriorities

2 GOALS

The main goal of this document is **to describe the work packages (WP)** to give at *KiCAD* suite the features listed in the above link.

In order to reach this goal, we will perform the following tasks:

- *KiCAD* Analysis: we will take an analysis of the software architecture of *KiCAD* and the modules contained.
- Features list analysis: we will analyze each feature in order to get the tasks and operations that will be performed to reach the objective.
- Operations grouping. Where possible, the features will be grouped in order to get a coherent work package and to avoid tasks duplication.

Despite of Signal vast experience in software development, we are not currently participating in *KiCAD* development. Thus there is room for improving this work package definition with suggestions from *CERN* team and *KiCAD* developers.

KiCAD has got a guide for coding standards that must be followed. Developers must take in account that non compliant code will be rejected. The guide is distributed with KiCAD source code.

3 KICAD

The *KiCad EDA Suite* project aims at creating a portable, cross-platform, Free/ Libre/Open-Source EDA Suite. That is capable of schematic and printed circuit board design. The code is licensed under the terms of the GNU GPL. [From KiCAD wiki]. It consists of the following modules:

- Project Manager
- Schematic Editor
- NetList converter
- PCB Designer
- Gerber Viewer

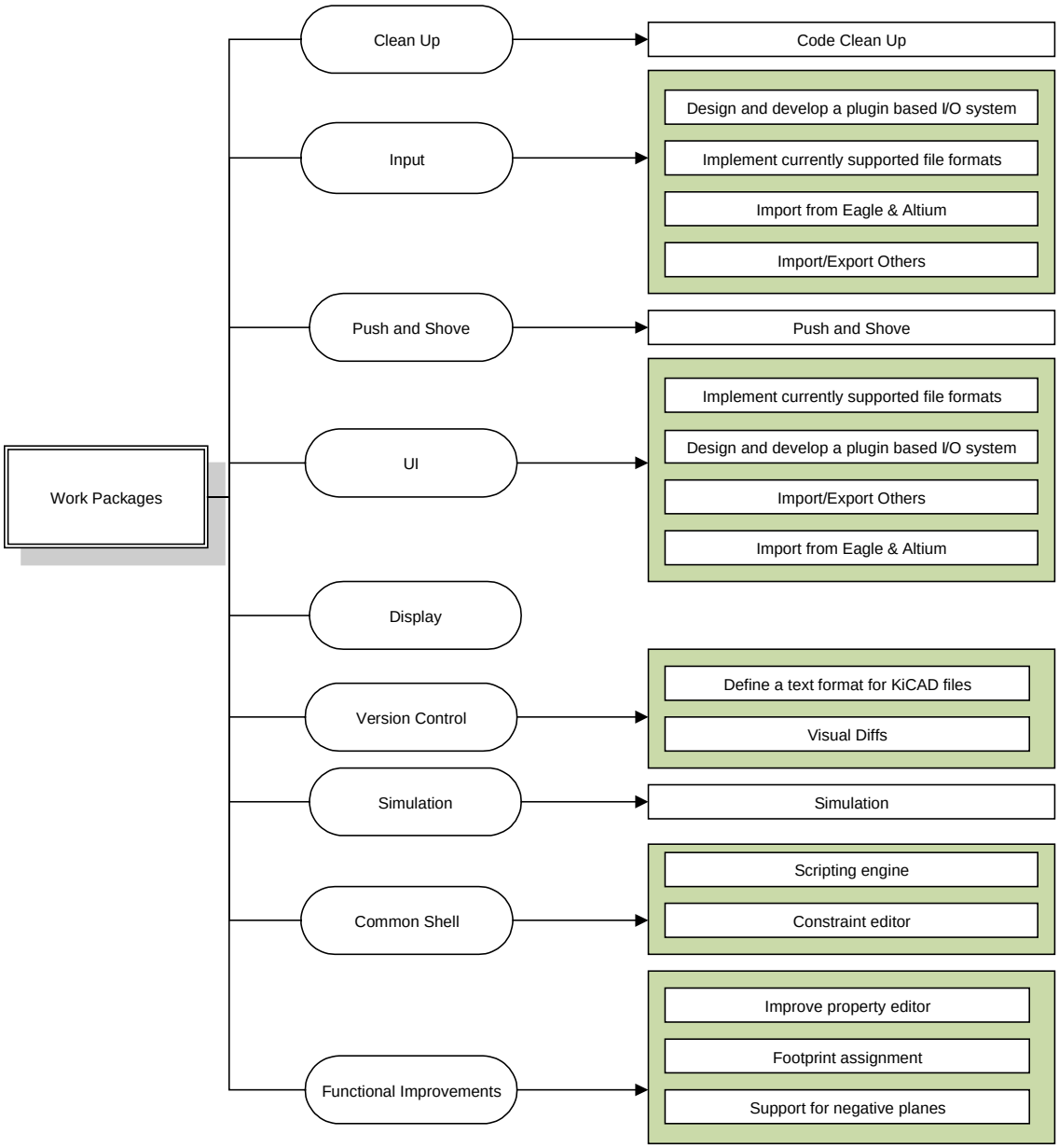
KiCAD strongly depends to w*xWidgets* library and despite having an object oriented language as C++, it has not got either a clear software architecture nor a functional encapsulation.

This fact will directly affect to this write up. Even though some parts should be refactored, we will part from current architecture due the lack of time for WP specification (and maybe implementation). In our opinion some parts (specially display related ones) should be redesigned for getting a more up-to-date application.

4 WORK PACKAGES DEFINITION

This epigraph describes proposed work packages. Some of them which are concerned to similar task have been grouped by scopes:

- Cleanup: Defines code clean up tasks.
- Input: Defines every work package related to input and outputs.
- Push and Shove: Defines push and shove related packages.
- UI: Work packages related to user interface.
- Display: Defines every work packages related to drawing operations.
- Version control: Work packages related with version control of KiCAD's projects are found in this section.
- Simulation: Defines work packages related with functional simulations on KiCAD.
- Common Shell: Work packages directed toward giving KiCAD a common shell.
- Functional improvements Some assorted work packages oriented to improve KiCAD functionality and user experience.



4.1 Work Package: Code cleanup

Objectives

Some parts of current KiCAD code are (or could be) deprecated. For instance, *trigo.cpp* uses a look up table for trigonometric functions. But other side *sin()* and *cos()* functions are already being used in quite a few places in the PCBNew code instead of the look up tables.

Features Covered

n/a

Tasks

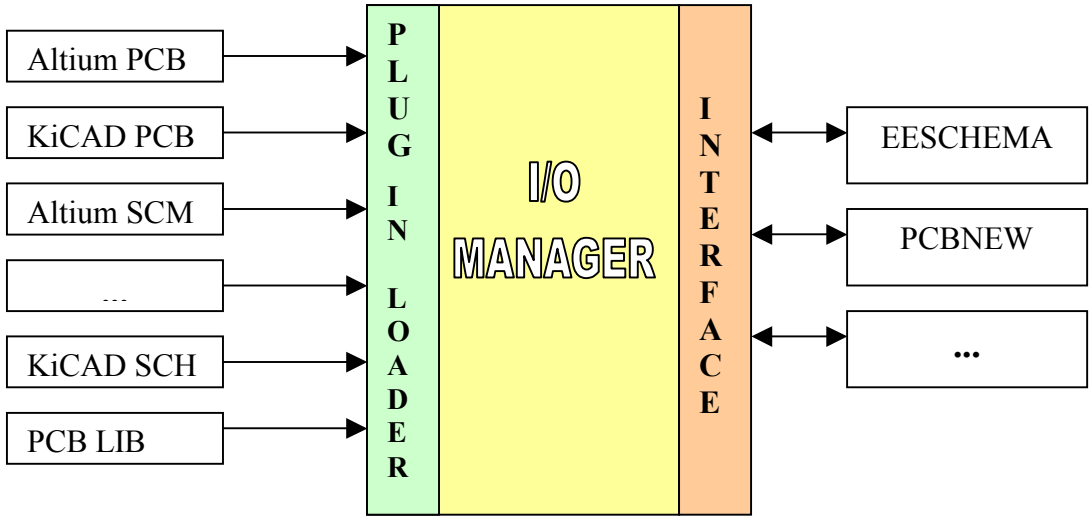
1. **Current source analysis looking deprecated or duplicated code blocks**.
2. **Clean unused code and standardize some calculations (like *sin()* and *cos()* described above).**

4.2 Work Package: I/O

4.2.1 Description

The aim of this work package is to implement an input module in KiCAD for centralizing file format management. Objectives are both manage native KiCAD's file formats and other software (or standard) formats.

In order to achieve this, a development of a plugin-based solution is proposed. This kind of solution is used by already existing software that manage an enormous variety of input/output formats. For instance The Gimp, OpenSceneGraph (OSG) or GDAL library use this approach.



Besides the main goal of getting centralized module for input/output operations in all *KiCAD* applications, these kind of solution might be a hook to attract external developers from other EDA tools interested on having some kind of interoperatibility.

Note that input and output operations include import and export from/to other EDA o mechanical software, and plot file formats.

4.2.2 Features Covered

By accomplishing all of the work packages specified in this field, the following features will be covered:

- **[4]** Import from Eagle & Altium - the two most popular commercial tools in OH community.
- **[3]** Import/export from/to other FOSS tools.
- **[1]** Export to commercial tools.
- **[3]** Export/import to a 3D mechanical CAD application.
- **[3]** Move import/export/plot functions into shared library plugins.

4.2.3 Work Package : Design and develop a plugin based I/O system

Objectives

Develop plugin based I/O system's base architecture. That's necessary to develop new KiCAD's I/O plugins.

Features Covered

n/a

Tasks

1. **Define an interface to load dynamically the libraries (plugins) from disk**

   Design a multiplatform plugin loader, capable of loading dynamic libraries.

2. **Establish an interface to perform input/output operations to/from KiCAD**

   Identify KiCAD requirements for file loading and design method signatures that allow plugins creation of KiCAD's memory structures. These signatures must be as simple as possible, falling all complexity on plugins inner implementation. Thinking on load() methods with a string parameter (file name) and returning a list of EDA_ITEM objects may be a good start point.

3. **Implement a functional I/O module prototype**

   Program an I/O module prototype following previous interfaces.

4. **Implement common tools to help file parsing and managing**

   There will be some common plugin operations, like file system management or text parsing. The goal of this point is to identify these regular operations and provide a set of helpers (classes) to assist plugin development.

Dependencies

n/a

4.2.4 Work Package: Implement currently supported file formats

Objectives

Once the plugin based I/O architecture has been defined, it's necessary to develop every plugin related to the I/O file formats currently used in KiCAD. It's also included in this work package the integration of the plugins in KiCAD

Features Covered

- **[3]** Move import/export/plot functions into shared library plugins.

Tasks

1. **File formats definition**

   It's necessary to specify all working file formats (not obsolete) used in KiCAD. To achieve this task, this information should be asked in the developers/users mailing list.

2. **Develop plugins compliant with 4.2.3 interface for the currently supported file formats**

   Once the input system architecture is defined, all input and output plugins must be developed in order to achieve current file format KiCAD functionality.

3. **Integrate plugin based I/O system with main development branch**

   When plugin loader and main plugin files are developed and tested is time to integrate it with current KiCAD branch, replacing all in line file management with calls to new input module.

Dependencies

WP 4.2.3

4.2.5 Work Package: Import from Eagle & Altium

Objectives

Given the fact that *Eagle* and *Altium* are the most popular applications in *PCB* design, it seems that importing these file formats into *KiCAD* is required. This will be achieved by developing the appropriate I/O plugins.

Features Covered

- **[4]** Import from Eagle & Altium - the two most popular commercial tools in OH community.
- **[1]** Export to commercial tools.

Tasks

1. **Define *Eagle's* file format specifications**

   In this task it's needed to know if the Eagle's file format structure can be obtained. In case this structure cannot be obtained, other file formats that Eagle exports will be evaluated to see which one suites best to develop in KiCAD.

2. **Implement *Eagle's* file format**

   Once the previous point is done. The selected file format's Importer and Exporter plugins will be implemented in *KiCAD's* plugin system.

3. **Define *Altium's* file format specifications**

   In this task it's needed to know if the Altium's file format structure can be obtained. In case this structure cannot be obtained, other file formats that Altium exports will be evaluated to see which one suites best to develop in KiCAD.

4. **Implement *Altium's* file format**

   Once the previous point is done. The selected file format's Importer and Exporter plugins will be implemented in KiCAD's plugin system.

Dependencies

WP 4.2.3 y WP 4.2.4

4.2.6 Work Package: Import/Export Others

Objectives

Once the I/O plugin-based system is complete, new file formats can be added according to users' needs. This work package expects to compile these needs in order to implement most priority I/O formats.

Covered Features

- **[3]** Import/export from/to other FOSS tools.
- **[3]** Export/import to a 3D mechanical CAD application.

Tasks

1. **Compile file formats**

   A list with the most used formats in common CAD applications and other FOSS tools will be enumerated. This list has to be validated with *KiCAD's* users and developers in their mailing list.

   File format list overview (unsorted):

   - IPC-356 netlist
   - WRML (with full parser) not only Wings3D WRML
   - STEP (not free, license issues)
   - IGES

2. **Implement import/export Others**

   Once the file format list from the previous point is completed, the plugins will be implemented by order of priority using *KiCAD's* plugin system's interfaces.

Dependencies

4.2.3 Work Package : Design and develop a plugin based I/O system
4.2.4 Work Package: Implement currently supported file formats

4.3 Work Package: Push and Shove

Objectives

This work package objective is to provide an assisted wire routing to *KiCAD* by using some kind of *push and shove* algorithm.

In order to provide auto routing features to *KiCAD* two approaches are considered:

- A program inside *KiCAD* suite (as the same way that other programs included, like *PCBNew* or *Gerbiew*) which allows importing a working layout for route tracing. This import would be done transparently since common shell for program communication will be used.

  At present most users work with *FreeRouting* program using *Specctra DSN* interface for data sharing. This first approach follows that stand-alone routing application idea but giving and integrated solution on *KiCAD* suite.

  As main advantage, this solution opens up the possibility of having a simplified user interface oriented to wire routing. Besides it brings facilities

to develop a *push and shove* solution simultaneously with other work packages execution, minimizing conflicts.

- *Push and shove* routing as an option in PCBNew program.

Covered Features

- **[5]** Push and walkaround mode (router doesn't try to find alternative paths for conflicting traces).
- **[3]** Full push and shove.

Tasks

1. Design a *push and shove* algorithm. Maybe a research about *FreeRouting* (http://www.freerouting.net/) algorithm is a good start point.
2. Implement algorithm and demo in C++. The aim is to develop a push and shove demonstration. Operation and work streams from *FreeRouting*, MUCS-PCB, or similar software could be taken as reference. This demo program would provide a way to choose an operational behavior for *KiCAD's* "push and shove" through user revisions.
3. Integrate algorithm in *KiCAD*. Once demonstration program is approved it should be fitted into *KiCAD* suite. This task includes to find a mechanism to modify *EDA_ITEM's* positions according to push and shove algorithm results.

Dependencies

n/a

4.4 Work Package: UI Improvements

The purpose of these WPs is to improve KiCAD's user experience through giving new features to the edition window and widget interface.

4.4.1 Work Package: Widget toolkit upgrade

Objectives

KiCAD implementation has a high coupling with wxWidgets library. It is used not only for user interface construction and management, but in data types (as wxString o wxPoint). For that reason an wxWidgets library update is as recommended as tricky.

At this moment wxWidgets latest version is 2.92 and provides important changes for KiCAD:

1. Merge of the old ANSI and Unicode build modes in a single build.
2. Changes in some control behavior.
3. Advanced toolbars.

4. New AUI (Advanced User Interface). Library for docking windows.

Advanced user interface module is the key feature to achieve movable and dockable windows. Also a tab based version of KiCAD could be obtained by using wxAUI.

Thorny problems of moving to wxWidgets 2.9.2 are related to string management changes (Unicode and ANSI integration) and new behavior in some controls (widgets). Briefly:

1. Unicode/ANSI The biggest changes in 2.9.2 version are the changes due to the merge of the old ANSI and Unicode build modes in a single build. Many wxWidgets functions taking or returning "const wxChar *" have been changed to take or return "const wxString&", consequently KiCAD methods should be changed too.
2. Changes in behavior. Latest wxWidgets version defines two kind of behavior related changes:
    a. First set of modifications includes changes which may result in compilation errors, so can be fixed easily.
    b. Other group is for changes in behavior not resulting in compilation errors. High KiCAD architecture knowledge is required to check how current code is affected. An extensive behavior change list is provided with wxWidget 2.9.2.

Covered Features

- **[4]** Schematics/PCBs in tabs
- **[3]** Make all toolbars and property windows movable and dockable.

Tasks

- Upgrade to latest *wxWidgets* version (2.9.2 or higher).
    - Check *wxWidgets* 2.9.2 changelog (changes.txt), where relevant changes - function signatures and behavioral modifications - are showed.
    - Assess widgets improvements, especially the event related ones, with the aim to get better (and faster) UI response.
- Port current widgets (toolbars, window, etc.) to new wxAUI library (Advanced User Interface) to make the most of its features to attain a modern interface with features like:
    - Dockable and rearrangable windows.
    - Window tabs.
    - Floating toolbars

4.4.2 Work Package: Global shortcut system

Objectives

Provide an user custom method to assign keyboard shortcuts to as many functions as possible.

Covered Features

[3] Possibility of assigning a keyboard shortcut for every function in the program

Tasks

- Make a list of the functions that are susceptible (or desirable) of being controlled by keyboard entries.
- Design a way in KiCAD to get a global shortcut manager. This point is related to getting a common shell on KiCAD for schematic editor and PCB layout applications.
- Design an user interface to make possible the assignment of the hotkeys.
- Implement solution.

Dependencies

4.8 Work Package: Common Shell

4.4.3 Work Package: Improve object selection

Objectives

In current KiCAD version you can't just mark an object as selected, but only select a rectangular region and immediately perform an operation such as move/delete/etc.

The objective of this work package is to develop a way to select arbitrary groups of objects.

Covered Features

[5] Object selection and possibility to select arbitrary groups of objects

Tasks

- Implement a class to manage selected objects. Class could maintain a list of selected EDA_ITEMS in order to apply global actions like unselect, delete, property changing, etc.
- Define an input method to arbitrary adding objects (GUI behavior).

Dependencies

n/a

4.4.4 Work Package: Align and distribute tool

Objectives

Align objects and elements (like labels) in layout to fit user preferences in a semi automatic way. User interface should be able to select a group of objects and arrange them according diverse criteria: align centers, tops, distribute uniformly, equal distances, etc.

Covered Features

[3] Align & distribute tool.

Tasks

- Identify use cases where automatic alignment and arrangement are required.
- Add contextual menu and toolbar options for the new functions gathered on the previous task.
- Implement algorithm for alignment and distribution. That should not be a complex task because it all comes down to moving operations.

Dependencies

Probably is a good idea waiting for the accomplishment of "4.4.3Work Package: Improve object selection"

4.5 Work Package: Display

Display related tasks are main points on graphical applications. Having an isolated render module is helpful when a change in drawing is ready to be done. Unfortunately, as previously seen in section 0, KiCAD suffers from lack of clear architecture in this subject, which derives in low procedural cohesion and high functional coupling.

In our opinion at this point there are two courses to follow:

- Total refactoring of drawing function in order to isolate drawing function in one module which manages *Drawables*.

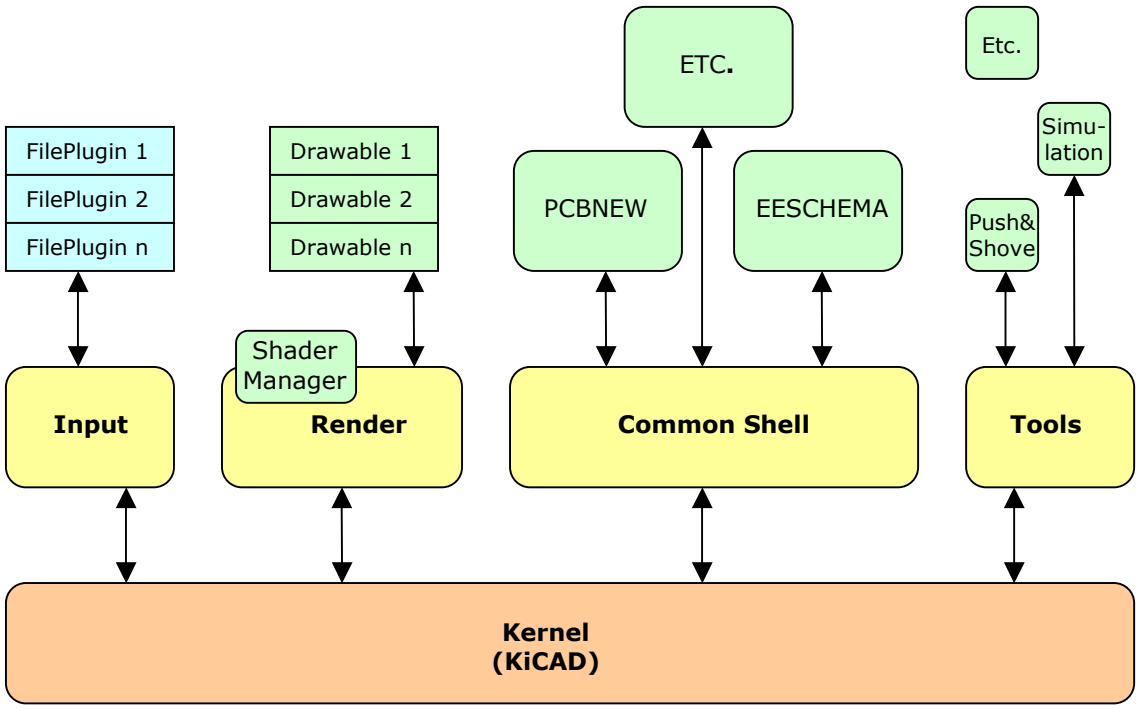
  A Drawable is a general abstraction for "something that can be drawn."

  In current code EDA_ITEM class and derived ones have a similar concept. But in some classes like SCH_COMPONENT drawing methods coexists with file management or component fields storage. As refactoring example, we might have abstract objects representing each component where each of them would contain a Drawable reference. This Drawable is registered in *render module* during component initialization.

  Different drawing methods (direct, hardware accelerated) result in different render modules, and switching between render modes would only require reimplement Drawables. Furthermore on-the-fly changes are possible in this kind of approaches.

- Modify current code to replace drawing functions to get hardware accelerated rendering. This solution involve add new draw functions to those items whom need to be painted. Also screen items should be changed to relay on wxGLCanvas or similar. Maintaining a dual display mode with advanced rendering functions (like GL) and old one would be difficult.
  This option is fast in short term but could result in problems due the aforementioned coupling.

Since other WP needs a quite deep code reorganization, such as in plugin based input module, out proposal is to take first course with the aim to reach an architecture similar to this one:



- **Kernel (KiCAD):** This is the core of the system. As in the current architecture, this module works like a coordinator between all modules.
- **Input:** It is the I/O file formats manager. It defines the common plugins' interfaces, and is responsible for loading and manage file plugins.
- **Render:** The render module encapsulates all the drawing functions. It manages the *drawables* in the system, that are the objects that only performs drawing functions. Also will manage the drawing device contexts.
- **Shader Manager:** It belongs to the render module and has the functions to manage the shaders (small programs used primarily to calculate rendering effects on GPU with a high degree of flexibility). It loads and compiles the shaders programs and will provide and interface to access them.
- **Common Shell:** Is the common system that manages the PCB's modules. It will be responsible for providing a communications pipe to the PCB's modules.
- **Tools:** It is a heterogeneous set of utilities and classes used to compute push and shove, simulations and so on.

4.5.1 WP Definition

Since architecture described above must be evaluated by the development team at CERN, the definition of work packages depends on the decision taken in this subject.

If current KiCAD architecture is chosen this section will be completed with WP for necessary changes. The definition of these WP would be done in general terms, because Signal Software have not get the knowledge of KiCAD's internals needed for a detailed task definition. With the available time and resources we can't make an in deep analysis of current drawing system, so the help of current development team is needed.

But if the route of the proposed solution is followed, all parties involved should agreed in the new system definition. This process of defining a new module architecture is a workpackage itself. That task is beyond of this document scope.

4.6 Work Package: Version Control

When project grows, a way to control file changes is needed, specially in collaborative environments. The objective of this WP is to provide methods for quick file version comparison. This goal must be achieved by defining a manageable text file format and by the implementation of graphic diffs viewers.

4.6.1 Work Package: Define a text format for KiCAD files

Objectives

Define a text file format.

An human readable format could help to advanced users to track changes with third party applications for text comparison. Since KiCAD files store graphical information is not easy to be entirely descriptive using just text. For this reason a full human-readable format is not prioritary (we would dare to say that is not possible). So the file format designer must to balance the need of having an human readable description and the need of a quick parsing and efficient storing.

Covered Features

[2] Designs in human-readable (1) text format so text diffs can be used to efficiently track changes using version control systems (2)

[4] Extra points for a format which has no significant parsing overhead (s-expressions look nice) (4)

[1] Preserving file content order so the diffs are minimal and easier to interpret (1)

Tasks

- Define text format for KiCAD files, as far as possible choosing a descriptive, easy to parse and human-readable language for its definition.
- Write input and output plugins for that format.

Dependencies

n/a

4.6.2 Work Package: Visual Diffs

Objectives

Give to the user a method for tracking project changes in a visual way.

Covered Features

[4] Visual diff-ing involving reasonable development effort (4)

Tasks

- Study a way to make visual diffs. Take in account the existent works in this area, like Qi hardware's visual diffs for KiCAD files[1] or evilmadscientist.com solution[2] based on DiffPDF .

[1] http://projects.qi-hardware.com/schhist/atusb/

- Once a visual diff method is chosen, implement an internal viewer or interfaces to external applications (it depends on what solution is chosen).

Dependencies

WP 4.6.1 Work Package: Define a text format for KiCAD files.

4.7 Work Package: Simulation

Objectives

The aim of this WP is provide an integrated functional simulation. There are several open source projects related with circuit simulation, among which the most important are GNUCap, NGSpice and Qucs.

Provide an interface from KiCAD to one of these applications may be a good solution.

Covered Features

[3] Integrated functional simulation (interface to ngspice/Qucs). *A simulation run should be as simple as with ltspice.*

[4] Automatic pin assignment file generation for ISE/Quartus/Lattice from the schematic

Tasks

- Study capabilities and features of existent FOSS circuit simulation software: GNUCap, NGSpice, Qucs and others, if any.
- Choose one of them, keeping in mind the needs for KiCAD integration.
- Design an interface for KiCAD – simulator interoperation.
- Implement solution with file generation for automatic pin assignment.

Dependencies

n/a

4.8 Work Package: Common Shell

At the moment KiCAD is a suite of applications launched through a main window. Although apparently they are connected because the window management is done from a single starting point, they are independent applications in practice. Because of this an inter application communication is required.

The aim of this WP is to provide a common shell for KiCAD suite, which allows the intercommunication between applications. For instance, this will be possible having synchronized changes in EESCHEMA and PCBNEW, or highlight elements in one window/application where are selected in another one.

It would also be interesting to provide a scripting language (like LUA or Python) to the shell, for using it with a command line interpreter. Through it could be possible made both simple and complex actions: object selection, set restrictions or design rules, etc.

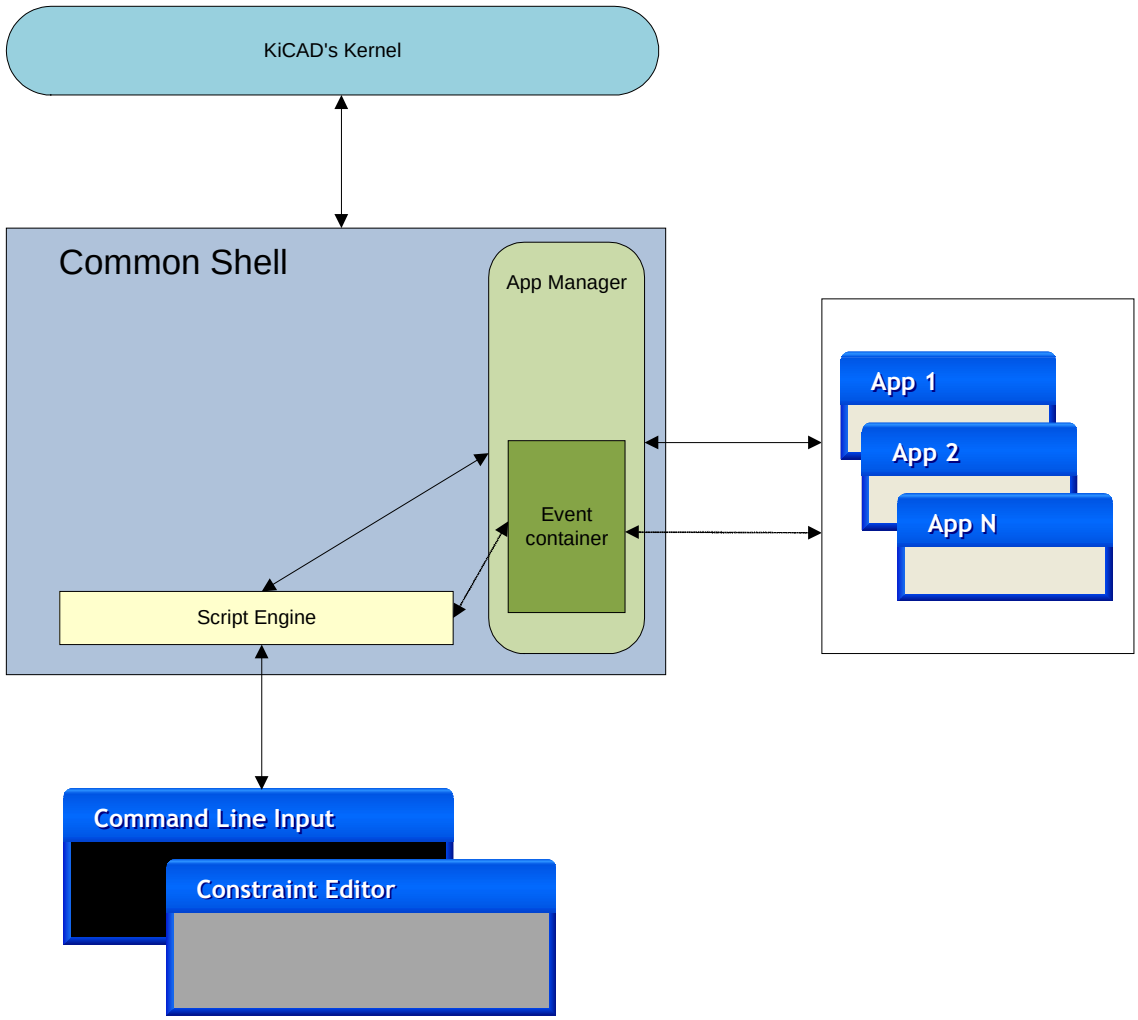
[2] http://www.evilmadscientist.com/article.php/visdiff

One first approach to the common shell architecture is seen at 4.5 Work Package: Display. Even that architecture solution is not approved, this common shell idea can be developed anyway.

The idea is to provide a common entry point for all KiCAD applications. In practice it would be a module that would control the execution of applications and its communication. That communication would be done through callbacks and delegates (events) registered and managed in this module.

One example. One application might link itself to events (like *OnNetlistChanged* o *OnComponentSelected*) which would be triggered by the common shell module. Communication is achieved by message passing, avoiding other overloading mechanisms like sockets or RPC.

Other advantage of this approach is having a scripting module. With the use of a script language, a text command interaction with applications is possible. It would be a strong tool for advanced users.



4.8.1 Work Package: Scripting engine

Objectives

The objective of this work package is to provide to KiCAD a design system based on command line functions. This will be accomplished through a scripting language (actually, a subset of it).

Covered Features

[3] Command line scripting shell. At the beginning it shall at least allow for filtering PCB objects - for example, *typing obj.type==Track && obj.layer==Front && obj.net=="MyNet"* would select all tracks belonging to net "MyNet" on the front layer.

[3] Scripting: Python/Lua.

[2] Scripted evaluation of some object properties

Tasks

- Requirements analysis for scripting language: Define operations to be done through scripting.
- Depend on requirements and other factors (like simplicity, performance) choose a base scripting language (Phyton, Lua, etc.).
- Define a way for scripting engine and C++ interaction. In our experience SWIG[3] is a good solution.
- Program the engine and the needed tools for testing.
- Program the C++ wrapper (SWIG can turn this point in a really simple task).
- Program a command line window integrated in KiCAD suite that uses the scripting.

Dependencies

Although the engine scripting WP is included in the WP: Common Shell, its development can be done in parallel with other tasks. In fact it is necessary to do it in this way to define interfaces for applications and events to be implemented.

4.8.2 Work Package: Constraint editor

Objectives

In addition to using the scripting engine directly through a command interface, the creation of a constraints editor is required. The objective of this WP is the implementation of that editor, which makes use of scripting to define constraints, but through an user friendly interface (at least, more friendly than a command line).

Following the style of the formula editors in spreadsheets, the user can make scripting operations in an assisted way.

Covered Features

[3] http://www.swig.org/

[3] Scriptable constraint editor. There are constraint categories, like Length, Width, Clearance where the user can specify the acceptable values and also the conditions (using the scripting language) under which the constraint is checked.

Tasks

- Requirements analysis.
- Choose an operational way of working with the editor: User interface.
- Implement application.

Dependencies

4.8.1

Work Package: Scripting engine

4.9 Work Package: Functional improvements

This WP includes two improvements to speed up some KiCAD tasks. Although both are related to the user interface have not been included under section 4.4 Work Package: UI Improvements since they have more to do with the way of work is done than to interface behavior

4.9.1 Work Package: Improve property edition

Objectives

Create a property view window similar to Altium Inspector for multiple properties edition (e.g. changing the net of a selection of tracks/pads/vias).

Covered Features

[4] Group property editor (a grid view similar to Altium's Inspector) capable of editing different kinds of objects.

Tasks

- Analyze Altium Inspector behavior.
- Implement a grid view similar to Altium’s one.

Dependencies

n/a

4.9.2 Work Package: Footprint assignment

Objectives

The purpose of this package is to provide a method to assign footprints to the schematic. For user convenience, it has to provide a display component library in a docker window. This window must have the ability to visualize footprints of the objects or symbols.

Covered Features

[5] Assigning footprints directly on the schematic. Library browser in a docker window with footprint/symbol viewer.

Tasks

- Requirements analysis for the library browser.
- Implement browser and footprint assignment.

Dependencies

n/a

4.9.3 Work Package: Support for negative planes

PCB Editors have two ways of representing solid copper planes - positive and negative. A positive plane is one in which you place the copper you want with pads, fills, and poured polygons. A negative plane is one in which you place marks where you want to remove copper.

The advantage of the negative plane is much smaller Gerber files for board fabrication. For a positive plane, there has to be a Gerber entry for each region of copper, and Gerber photo plotters draw that area with tens of thousands of line segments. For a negative plane, the photo plotter only has to draw the smaller areas where copper is to be removed.

Objectives

Implement support for split planes. Such planes are defined by cutting a contiguous copper surface and assigning the resulting slices to appropriate nets.

Covered Features

[5] Support for negative (a.k.a. split) planes.

Tasks

- Analyze and design a solution.
- Implementation.

Dependencies

n/a