

Parametric and distributed computer aided design (CAD) functionality integration into Blender 3D modelling and animation platform

(Technical feasibility study)

JSC „Atviras kodas verslui“
Naugarduko 91, Vilnius
Company code: 302253973
Phone.: 8 5 213 6604

Pranas Butkus
Algirdas Noreika
Justinas Jaronis
Justas Ingelevičius

2011 Kaunas

Table of Contents

1. INTRODUCTION.....	4
1.1. Why this study.....	4
1.2. Main objectives.....	4
1.3. Tasks.....	4
2. SUMMARY.....	6
2.1. Common requirements.....	6
2.2. Current situation and existing solutions.....	6
2.3. Alternative solutions	7
2.4. Recommended solutions.....	7
2.5. Work hours estimation.....	7
3. ABOUT COMPANY.....	9
4. SITUATION.....	9
4.1. CAD systems.....	9
4.2. CAD systems market and demand.....	9
4.3. Competitors analysis.....	10
4.4. Imperfections of existing solutions.....	15
4.5. Blender platform overview.....	15
4.6. Uniqueness of product and services.....	16
5. TECHNICAL ANALYSIS OF CAD FUNCTIONALITY INTEGRATION INTO BLENDER PLATFORM.....	17
5.1. Extension and modulation of Blender.....	17
5.1.1. Ways to extend Blender.....	17
5.1.2. Performance bottlenecks and risks of Blender real-time CAD extension.....	19
5.1.3. Cloud and Distributed computing application in Blender.....	22
5.1.4. OpenCL integration into Blender.....	23
5.1.5. Integrating external rendering platforms in Blender.....	25
5.1.6. Import/export ability for popular formats.....	29
5.1.7. Usage of digital signature for Blender files.....	31
5.2. Creating and editing parametrized objects (PO).....	33
5.2.1. User interface (UI) elements for PO creation and editing.....	33
5.2.2. Setting the detail level of a parametrized object (PO).....	49
5.2.2. PO Tools.....	50
5.2.3. Tools for architectural designing.....	54
5.3. Parametric objects distribution and synchronization	66
5.3.1. Server platform and architecture.....	66
5.3.2. Model exchange protocol requirements.....	71
5.3.3. Model exchange portal requirements.....	72
5.3.4. External models integration in Blender.....	73
5.3.5. Model exchange in P2P method.....	74
5.3.6. Replication-synchronization with other distribution servers.....	75
5.3.7. Integration of Jabber protocol.....	76
5.3.8. Work-flow formation, tracking and confirming changes. Version Control. Teamwork.....	77
6. CONCLUSIONS.....	80
REFERENCES.....	82

1. INTRODUCTION

1.1. Why this study

Although architecture CAD tools are heavily used by the industry in these days CAD software market offers only a few commercial and practically no commercial solutions for both – business and non business users. One of the main reasons is scale, complexity and work time required to develop such applications. Till now only well established commercial companies could provide design, development and support of advanced architecture CAD based solutions. The advantage of this is out-of-the box CAD products for consumer and business. However commercial nature comes with closed source code and licensing limitations which hinder companies more effectively adapt software to their needs. One of the possible ways in this situation is to use open source alternatives where software code is freely available for developers. Unfortunately, currently there are very few CAD architecture dedicated, free, open source alternatives to commercial products.

Architecture designers could benefit a lot from open source architecture CAD solutions. First – they could implement new features themselves, adapt it to their needs, share it with the community and get a number of new features from other contributors with minimal investment in this way. Open source, collaborative, community based software development model already proved itself as effective and flexible, especially on large scale projects so the main aim of this study is to research possibilities to develop such an open source based solution using well established and freely available open source 3D modeling and animation software Blender.

1.2. Main objectives

The main objectives of this study is:

- To research Blender platform features and possibilities;
- Find the best ways to extend Blender to meet the needs of an architecture designer;
- To research architecture models exchange service development and establishment possibilities;
- Recommend existing technologies, define best ways and solutions;
- Estimate projects design and development times

1.3. Tasks

The objectives, structure and tasks of this study:

1. The abilities for extension and modulation of Blender:
 1. Find out the best way to extend Blender platform, including creation of new functions and user interface, while preserving consistency with current code-base. Currently, the Blender Foundation does not intend to include code for CAD functionality into mainstream version. However, it is necessary to maintain compatibility with the existing Blender code-base and provide an easy way of its easy modification.
 2. Evaluate performance bottlenecks and risks of developing Blender as a real-time architectural design software.
 3. Evaluate the ability to set level of detail for PO (*parametric objects*).
 4. Evaluate the ability of using cloud computing and distributed computing to perform

complex calculations.

5. Evaluate OpenCL technology integration into Blender.
6. Evaluate the integration of Blender with external rendering platforms, such as Octane or Lux Renderer.
7. Evaluate the ability to import and export files of popular formats.
8. Evaluate the ability to sign Blender files with PGP or other digital signature, allowing to identify the author.
2. Creation and modification of Parametric Objects (hereafter it is abbreviated PO):
 1. Evaluate the ability to create elements of UI for PO creation and modification:
 1. An UI element representing PO parameter values – a table showing parameter values and allowing to change them.
 2. A tool for management of connections between PO.
 3. Project tree – an UI element that shows views, their hierarchy, POs and their types. Enables one to group POs by parameter values, perform component search in a project.
 4. Ability to program PO behavior by its parameter values.
 2. Evaluate the ability to create tools for working with POs:
 1. Reference planes (*refplanes*) – objects to which geometry is aligned.
 2. Ability to align refplanes of the same category, which are located in separate POs.
 3. Controls – graphic elements that enable to change PO parameters and its geometry, without entering 'edit component' mode.
 3. Evaluate the ability to create tools for architectural designing:
 1. 2D views generation from a 3D model – cuts and plans. A view should also be a PO of a certain category.
 2. Generating quantity reports - the ability to generate tables that present parameter values and quantities of POs in the project and to select POs, constrained by other POs, e.g. finding how many objects are inside another object.
 3. Arrangement management of 2D objects (images, reports, bitmap views) in sheets.
 4. Exporting sheets to other formats, such as PDF, DWG.
 5. Graphic marker tool – to represent PO parameter values. Graphic marker tool is also a PO.
 6. A tool to set dimension values in views.
3. Parametric objects synchronization to distribution servers. It is required to analyze abilities and the most effective ways for implementation:
 1. optimal server platform and architecture;
 2. requirements for model exchange protocol;
 3. requirements for model exchange portal;
 4. ability to integrate external models into Blender;
 5. ability for model exchange in P2P method;
 6. ability for replication-synchronization (exchanging data) with other distribution server installations.
 7. revision control mechanism;
 8. teamwork ability (to allow editing a file for several people simultaneously);
 9. integration of Jabber protocol (for real-time communication);
 10. abilities for work-flow formation, tracking and confirming changes;
 11. evaluate implementation, technologies used and user practice of other participants in the market.

2. SUMMARY

2.1. Common requirements

The first major part (5.1) requires to analyze Blender modulation and extension abilities, including research of the optimal choice for Blender platform extension, identifying potential performance bottlenecks and exploring possible external technologies that could speed up the execution of Blender computations. Non-native format file import/export support and extension and file signing also have been researched in the same part of this study. The results, findings and recommendations for these issues are presented consequently in section's 5.1. chapters 5.1.1. through 5.1.7.

Parametric Objects functionality is another major section. This section divided into three smaller sections. The first one analyzes the ability to create elements of user interface for PO manipulation. It includes a table to show parameter values, a node-based editor for connection between PO management, project tree UI element, programming PO functional behavior according to its parameter values and other closely related features. All these tasks' results are presented in section 5.2.1.

Second section related to PO functionality were tools for working and manipulation of POs. It includes reference planes (*refplanes*), their usage in separate objects, graphic elements to change PO parameters and geometry as well as other related functionality. These points are presented in section 5.2.2.

In the last part a research of abilities to create tools for architectural designing, including cuts and plans generation from 3D model, generating quantity reports, arrangement of 2D objects in sheets functionality, sheet exportation into non-native formats, as well as tools for dimension and other parameter values representation. These topics are discussed in section 5.2.3.

The last part of this study includes requirements for parametric object synchronization to distribution servers. The requirements focus on larger scale issues than single PO creation. More precisely, it is needed to find optimal server platform and architecture, requirements for model exchange protocol and for model exchange portal, external models integration into Blender, P2P method usability for model exchange, replication-synchronization with other distribution server installations, revision control mechanism and teamwork ability, which also includes communication via Jabber. This part is presented in part 5.3.

2.2. Current situation and existing solutions

Most of the features and functionality required by tasks of this study are not yet implemented. However, already there are similar features for much of the desired functionality. New editors required for the functionality implementation can also be adapted from existing editors. For example, the project tree, mentioned in chapter 5.2.1. requires an editor which is very similar to the outliner in Blender. Thus, the editor should be created using the latter. Some features can be implemented using already created python libraries, such as Jabber functionality, mentioned in chapter 5.3.7. can be implemented by using Jabber python library. In general case, Blender provides a well established API for data manipulation and new features development. Thus, it is possible to create new add-ons for non-native format file import/export or for complex scene object manipulation, using the Blenders data API. All these python scripts that are needed to create already have analogous files, thus analyzing them or even copying some of their code (thanks for the GPL license) can reduce the development times. Other functions, such as cloud rendering or distributed computing also have solutions at the very moment. For example, OpenCL possibilities discussed in

chapter 5.1.5. are already tested by LuxRender developers. Some plugins are developed at the moment as well. Thus, it is needed to consider whether it is required to create the same functionality from scratch when it is possible to get existing features.

2.3. Alternative solutions

Even when there is no product for direct functionality or technology desired, there are also many alternative solutions, which are presented in mostly every issue required to research. For example, in chapter 5.1.6. Import/export ability for popular formats there is an alternative for import scripts in Blender presented: a platform-independent C++ library Open Asset Import Library. To ease the process, there is a python script to connect the library functionality to Blender. Thus, again, it is needed to consider whether the functionality is needed to implement as a new system with analogous functions or to use existing solutions.

Alternative solutions also include decisions in analogous design programs, such as revision controlling in Revit with revision clouds (chapter 5.3.8. *Work-flow formation, tracking and confirming changes. Version Control. Teamwork*) or teamwork with BIM servers. However, it is generally recommended not to focus on simplicity of use of other architectural design applications, but to keep newly implemented functions consistent with Blender functions and their usage.

2.4. Recommended solutions

As the study shows (chapter 5.1.1. *Ways to extend Blender*), putting all the new and non-native functionality with current Blender mainstream is not a good idea and is hard to accomplish in consistent way. For this reason it is advised to extend existing code base, adding new modules or files where needed to preserve consistency with Blender code. Modifying existing mainstream files should be as minimal as possible and every modification should be discussed by Blender developers community.

Generally there are two recommended ways to add new features. The first one is to use existing open-source or free software solutions for the Blender Architect. Thus, it would be easy to introduce new changes whenever the client wants and whatever way he wants it to be done. If there are no solutions available for integration or code for use, the functionality should have to be created from scratch. In this case, the first recommendation is to program resource and time critical operations in C, inside Blender source. Other, not critical tasks, such as importers/exporters, custom panels or new object creators should be created as Python scripts (add-ons). This will be consistent way with current Blender architecture using the convenient Blender type (DNA) and Data API (RNA) systems, which can be used from Blender Python scripts [16, 17, 65, 70]. Many of the desired features also require extending DNA and RNA, as well as modifying blendloader module to read and write new structures.

2.5. Work hours estimation

It is estimated, that initial (without bug fixing, etc.) project development time could take about 2170 days (men time, MT). That could take about 6 years for one developer. The well managed developers team and contributors community could significantly reduce this time. The estimated Blender Architect project and server solutions development times are presented in table 2.1.

Table 2.1. Project total estimated hours

Tasks group	Estimated MT
Blender API research (table 5.1)	120
Blender bottleneck reduction solutions design (table 5.2)	42
GPG signature integration into Blender (table 5.5)	18
PO components implementation (table 5.6)	720
PO detail level implementation (table 5.7)	30
PO tools implementation (table 5.8)	150
Tools for architectural design implementation (table 5.9)	420
Server platform and architecture design (table 5.11)	60
Models exchange portal (table 5.12)	300
External models integration (table 5.13)	90
Model exchange via P2P (table 5.14)	60
Jabber protocol integration (table 5.15)	40
Teamwork and version control (table 5.16)	120
Total:	2170

3. ABOUT COMPANY

Company “Atviras kodas verslui” Ltd. specializes in products, services and IT solutions based on open source technologies by applying the open-source software for individual IT solutions, thus saving the client money and increasing the efficiency of the decision. One of the missions of “Atviras kodas verslui” is to promote open source software in all areas of IT. Company has a team of professionals who are familiar with open source software, open-source technologies.

4. SITUATION

4.1. CAD systems

Computer-Aided Design (CAD) is a system based on computer technology and used for the process of design and design-documentation. CAD is also known as Computer-Aided Design & Drafting (CADD).

Such systems makes designing more efficient, compared to traditional drafting by hand. The lines and other figures inside systems data structures are stored as vectors and complex equations, which are modified (recomputed) when a certain system's tool is used. CAD software enables a designer, engineer or architect to prepare the model using a plethora of modern and effective tools, including:

- adding individual or certain groups of primitive presets or copies of previously created figures to the model;
- modifying the whole model or its distinct parts: moving, scaling, stretching them, etc;
- zooming in and out for close-up and distant views;
- hiding unnecessary parts for easier creation of modification of other parts of the model;
- rotating the model to view it from different perspectives;
- generating reports of created models, including information about construction's physical parameters, amounts of particular materials required, estimated costs.
- rendering realistic images of the model;
- More sophisticated CAD software also provides a feature to make presentations of created models.

Models created with CAD systems are frequently used in the further process – manufacturing, which is driven by another – computer-aided manufacturing (CAM) software. Currently, CAD systems are virtually used in all industries, including construction, automotive and computer industries.

4.2. CAD systems market and demand

The market of CAD systems is wide and still growing. CAD systems are extensively used in many applications, including automotive, shipbuilding and aerospace industries, industrial and architectural design, prosthetics, and many more. CAD is also widely used to produce computer animation for special effects in movies, advertising and technical manuals. Modern computer-aided design tools are used to reduce time needed for the design process, perform complex operations and

calculations, which otherwise would require additional employees in the project teams or would be even impossible. As a consequence, it becomes possible to minimize design as well as the whole project costs.

A CAD system that suits a company's needs, i.e. provides the necessary design and documentation features, is essential to gain advantage against competitors in the market. Using free and open-source CAD software would make it possible to reduce financial risks and costs when choosing a CAD system with the required functionality. The latter software guarantees an easy way to modernize, optimize and extend design process in the company both because of the functions not yet utilized and a way to easily adapt a system to a company's needs by itself because of availability to reach and extend the software's source code. Nevertheless, many companies tend to purchase a system with all the required functionality and are only willing to extend features that improve compatibility with other software products used in the process rather than build all the required functionality themselves. The main drawback of commercial, closed source software systems is that companies faces large scale customization problems if they are willing to adapt these systems to their needs. Support and feature requests development typically takes long run cycles while ability to modify system code or create required functionality could take significantly less time.

Because of the reasons mentioned, a free open-source CAD software which delivers powerful functionality and flexibility has a great potential in the target market and should be considered creating.

4.3. Competitors analysis

Today the market of CAD tools consists of more than a hundred software products, although only some of them are widely used. All these products differ in both – the range of provided features and in price.

The first group in the market is lower-end 2D design software, which in many cases is free and open source. This kind of software offers essential set of design features, like placing primitive figures on a sheet, moving and scaling them. Open-source nature of the software enables easy modification, adaptation and extension opportunities – to better meet the company's needs. Extension of 2D drafting is 3D wire-frame, which enables to create a 3D model by manually inserting lines into the drawing. The systems mentioned are often just drawing editors and does not provide rendering or documentation functionality.

Companies working with complex projects and creating detailed models often prefer CAD systems that have sophisticated 3D solids manipulation features, which also enable generating 2D images and drawings of a 3D model, realistic images rendering systems, distributed computing (*e.g. ray-tracing*) over a network of computers and complex report generators. These systems are commercial in many cases, although there are some alternatives in development that are expected to become competitors for commercial CAD software in the near future.

Today, the most popular architectural CAD systems among professional designers are AutoCAD Architecture, Revit, ArchiCAD, Google SketchUp Pro, which are all commercial products. The most popular free architectural CAD systems are Google SketchUp, DraftSight, progeCAD 2009 Smart! The following (table 4.1.) provides information and feature comparison of some of these CAD systems used for architectural design [5-7, 9, 22, 24, 33, 38].

Table 4.1. Comparison of architectural CAD systems

Product	AutoCAD Architecture	Revit	ArchiCAD	Google SketchUp Pro	Google SketchUp	DraftSight
Operating systems	Windows, Mac OS X, iOS, Android,	Windows	Windows, Mac OS X	Windows (XP and later), Mac OS X Leopard (or later)	Windows (XP and later), Mac OS X Leopard (or later)	Windows, Mac OS X (beta version), Linux
Price	\$4995	\$5495 Subscription \$725	\$4250 (version 11) Subscription \$695	\$495	Free	Free
Field	Any. Used by engineers, designers, architects	Architectural, structural, mechanical	Architectural	Architectural, civil, mechanical, film and game making	Architectural, civil, mechanical, film and game making	All (DWG editor)
Commercial application	+	+	+	+	+	+
Dimensions	2D, 3D	2D, 3D	2D, 3D	2D, 3D	2D, 3D	2D
Native format	DWG	RVT	PLN	SKP	SKP	DWG
Import/ Export formats	FBX, DWT, DWF, DXF, PDF, 3DS, BMP, JPG, TIF, PNG.	DXF, DWG, 3DS (export), VIZ (export), JPG, PNG, TIF, GIF.	DXF, DWG, DWF, DGN, JPG, PNG, TIF, BMP, PDF, 3DS.	JPG, TIFF, PNG, PDF, DXF, DWG, EPS, 3DS. Export only: OBJ, XSI, FBX, VRML and DAE.	3DS (import), JPG, TIFF, PNG, PDF (import).	PDF, DWT, PLT, JPG, PNG, SVG, TIF, STL, SLD, BMP, EMF, DXF
Command line input	+	+ (Programmer friendly interface)	-	-	-	+
Dimension tool	+	+	+	+	+	+
Materials	+	+ a plethora of premade materials, each of which can be modified to the user's desires. The user can also begin with a "Generic" material, which can be customized to a level of detail not offered by many 3D modeling programs.	+	+	+	
Rendering	+	+ (better quality with	+ many complains about	- (requires external	- (requires external	-

		external rendering software)	poor quality. The offset is to to export to other formats (e.g. *.3DS) and use external renderers (Cinema4D, 3D Studio Max)	rendering software)	rendering software)	
Distributed computing	- (requires external rendering software)	- (requires external rendering software)	- (requires external rendering software)	- (requires external rendering software)	- (requires external rendering software)	- (requires external rendering software)
Documentation and presentation tool	+	+	+	+	-	-
Workgroup capability	Available only with additional Autodesk Vault data management software	Local file (on the workstation) and central file (on the server) concept.	Local file (on the workstation) and central file (on the server) concept.	-	-	-

4.4. Imperfections of existing solutions

Although current architectural CAD products provide rich functionality, these systems have some serious flaws that often need to be modified in order to improve design and documentation process. Some of them can be detected from the previous table (table 4.1.).

The first issue is that the most popular CAD software is proprietary. It means that possible software extensions and improvements are in its developers hands only. It also infers that adapting a system to the company's business might be problematic, often requiring additional negotiation with the system's developer.

Next, often there is incompatibility issues between different file formats – if an application doesn't support a certain file format. Special converters or 3rd party importers/exporters have to be used, which sometimes happen to have flaws as well, causing data, precision or other losses. In order to avoid format incompatibility, certain universal file format standards have been created (e.g. DXF format), which is integrated by mostly all popular CAD products through so far reliable importers/exporters. Some incompatibility problems exist even between different versions of the same format, DWG, for example. The DWG format of AutoCAD drawings is regularly updated after a couple of releases. New AutoCAD release always supports reading of all older DWG file formats and writing of some older DWG formats. Unfortunately, older products can't read newer DWG format files, which makes older software less suitable for work.

There is also a problem common to all architectural CAD systems – lack of distributed computations feature. Sending data to other workstations through a network is an expensive process, so distributed computations mostly suits the tasks that require less data sending and a lot of CPU processing. Today distributed computing in CAD systems is almost all about rendering 3d scenes and animations. Currently none of the architectural CAD systems has a native distributed computing feature. Autodesk suggests using commercial 3D Studio Max package to perform distributed rendering of scenes made with other Autodesk products, such as Revit or AutoCAD. Other architectural CAD systems are using external rendering software that has distributed computations feature.

One more drawback that might have impact on company's business is that most of the proprietary CAD systems can't be used in different operating systems. Typically there are versions for most popular platforms only and companies are additionally forced to buy a commercial operating systems just to fulfill their business needs. Most commercial CAD products are made for Microsoft Windows and Mac operating systems, while very few for Linux and Unix. Although this could be explained by desktop OS market share, but from different perspective of view companies are forced to use Microsoft Windows and Mac OS, losing the benefits other operating systems might offer.

4.5. Blender platform overview

Blender [22] is a free open source cross-platform suite of tools dedicated primarily for 3D models and animation development. There are versions available for Linux, Mac OS X, FreeBSD, OpenBSD and Windows operating systems. As 3D development package Blender is mostly focused in 3D modeling, UV unwrapping, texturing, rigging, water, smoke, particle and other simulations, skinning, animation, rendering, video editing, compositing, and the ability to create interactive 3D applications, video games, animated film, or visual effects. More advanced tools include rigid, realistic body, fluid, cloth, soft body dynamics simulation, modifier-based modeling, character animation, a node-based material and compositing system, and embedded scripting via Python.

Blender is written in three programming languages. Its main part (kernel, essential functions

and operators) is coded in C language. The user interface layout (like menus, panels) is created with Python scripting language, which is also used as main tool to create additional functionality and tools through extensible add-on system. Blender Game Engine (BGE) is written in C++ and can also be extended and controlled by Python scripts. Blender and Blender Game Engine use OpenGL, a cross-platform graphics layer, to communicate with graphics hardware.

One of the most powerful Blender features due to its open-source nature is that it can be easily and thoroughly modified and extended. First, it's possible because of its add-on system, which allows creating and adding new system functionality units, called operators. The add-ons system abilities are also enhanced by the Data API called RNA [13], which was developed in Blender version 2.5. The API allows to access Blender data structures, read and modify their values in a uniform manner. Moreover, being an open-source software, Blender provides the ability to access its source code. Thus, it's allowed to create new and modify existing internal data structures, editors, menus and other features – it's even permitted to change Blender's core functions to meet everyone's needs.

All Blender projects data is stored in a single .blend format file, which structure and management is based on database-like concepts. 2D images and 3D scenes can be imported and exported to different currently available formats: 3DS, AC, DXF, OBJ, X3D, DAE and others. The new add-ons system allows, this list to be easily extended.

Recent products made with Blender include a computer animated short films “Sintel”, “Big Buck Bunny” and computer game based on the latter movie environment and characters - “Yo Frankie!” These open source projects were created by the Blender Institute and supported by Blender Foundation, with the main aim in mind – to promote the open source modeling and animation tool Blender. Due to highly extensible and open-source nature Blender is used in science, art medical, robotics and other fields. As an example a robot called “Robo Thespian” has been programmed to move precisely as an animated model in Blender software. Blender is widely used as free tool for 3D open-projects and educational purposes.

4.6. Uniqueness of product and services

Blender, as a potential architectural CAD system, might offer new features and, most importantly, solve problems that other products in the market are confronted with. Blender is a free open-source product and will stay so due to its GPL license. Furthermore, as mentioned before, it supports long list of operating systems – longer than any other popular architectural CAD systems. These characteristics would let every organization better adapt the CAD software to its individual needs, extend the software features and be able to stay independent from a certain operating system.

5. TECHNICAL ANALYSIS OF CAD FUNCTIONALITY INTEGRATION INTO BLENDER PLATFORM

5.1. Extension and modulation of Blender

5.1.1. Ways to extend Blender

Requirements

Find out the best way to extend Blender platform, including creation of new functions and user interface, while preserving consistency with current code-base. Currently, the Blender Foundation does not intend to include code for CAD functionality into mainstream version. However, it is necessary to maintain compatibility with the existing Blender code-base and provide an easy way of its easy modification.

Current situation and solutions

Currently, Blender provides more than one way for its modification and extension.

The first and probably the easiest way to extend Blender's features is by creating add-ons – Python scripts, powered by Python API [19]. There already exist scripts for operators (they perform actions with certain data in the project, e.g. file exporter or mesh modifier), generating GUI layouts (menus, panels and headers) for each of the spaces provided in Blender, setting keying sets on start-up and other system-specific operations. Such features as operators, menus, panels and headers are easy to add or modify in Blender. One just needs to follow the required declaration, i.e. create a Python class that inherits from a certain base class, e.g. *bpy.types.Operator* or *bpy.types.Menu*, and define certain methods, which are called when the operators or user interface elements are used or registered into Blender [15]. Despite being easy to specify a script, some scripts are still rather complex and contains many subsidiary functions due to the operations they are intended to perform. When a script is prepared for use, in most cases it just has to be placed in the specified add-on scripts folder. Scripts are interpreted in real-time – no recompilation is required.

Although menus or panels for Blender spaces are very easy to develop, it is not true about spaces themselves. They are written in C language and compiled into binary machine code. The code for spaces and editors can be found in `source/blender/editors` folder. For each space-type there is a folder of files that contains code for keying sets, necessary operators registration, each of its area's regions initialization and drawing rules definitions, events management. Operators source code typically resides outside the space-type folder.

All complex and execution speed sensitive features of Blender is written in C (or C++ for Blender Game Engine). As open-source software, Blender provides its source code, which can be modified, compiled and the result product obtained can be used to fulfill company needs. Since Blender version 2.5 the code base has been highly modified and improved – fully rewritten older, (2.4x) versions code. 2.5 version code is better structured, contains less bad level calls and thus the system architecture is simpler and takes less time to grasp. The figure 5.1 illustrates current Blender code layout [21].

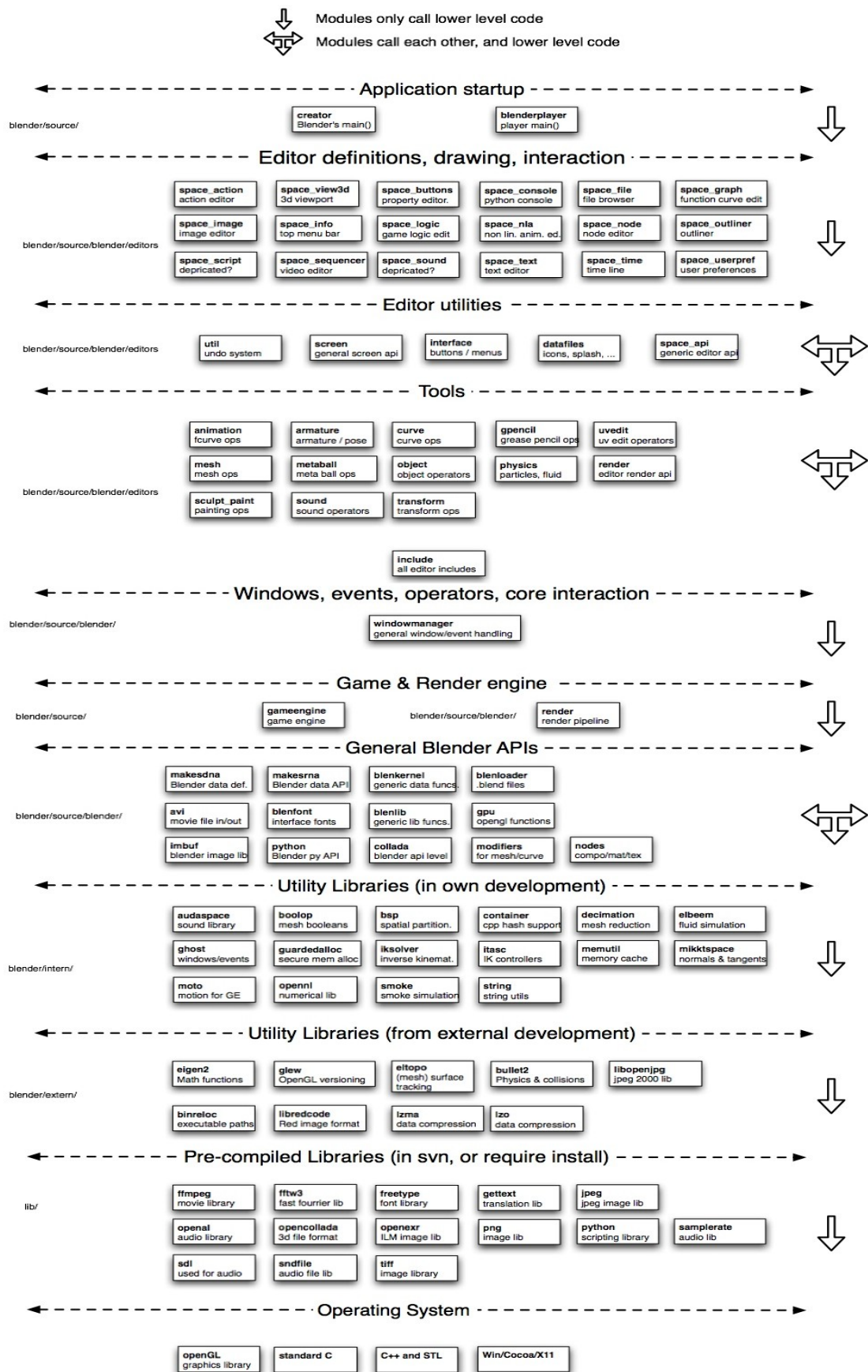


Fig. 5.1. Blender system code structure layout

Alternative solutions

One of the possible solutions to extend Blender is to create a separate module or entire module set oriented to architectural CAD functionality. All the custom spaces, file exporters/importers and other features not related to original Blender would be kept outside Blender's source and would communicate with it via a certain interface, which would be needed to implement and then integrate into Blender. The new functions would be called remotely from Blender through the interface.

This solution is not recommended though, since the new interface would probably be too complex and mess the Blender's code base. Furthermore, it would not be resistant to code modifications in future Blender versions and the interface would still require changing. Finally, calling remote functions would cost more time than calling local ones.

Summarizing, although this way to extend Blender is technically feasible, it has many drawbacks and there are no significant benefits expected, if excluding the additional functionality. Thus, it is not recommended as a solution.

Recommended solutions

In order not to digress from the current Blender code base, it is recommended to use currently available ways to extend application – Python scripts for creating new operators, menus and similar features (for example, creating new file importers/exporters or synchronizing files with the ones in a certain distributed server). For more complex tasks (e.g. layout editor, node-based editor for parametric objects links) it could be suitable to extend Blender code base by adding necessary files in the same manner it is currently done or modify (probably most often append to) existing files.

Despite the convenience of the supported ways for extensions mentioned, its important to notice that Blender related target is to keep improving APIs and modules, which means that scripts and modifications in Blender internal code might not be compatible with future versions of Blender and may require some modifications. On the other hand, currently no significant changes in Blender code base are expected, which means that custom modifications to the Blender software core functionality should be also adaptable to its versions in the next several years.

Solutions development estimation and planning

Table 5.1. Blender API research estimation

Task description	Estimated MT
Blender API research and project design	120
Total:	120

5.1.2. Performance bottlenecks and risks of Blender real-time CAD extension

Requirements

Evaluate performance bottlenecks and risks of developing Blender as a realtime architectural design software.

Current situation and solutions

Current Blender version provides smooth 3D content development workflow in nearly all cases, including working with complex and detailed objects. The only final operation that takes

significant amount of time and also is relevant to 3D architectural designing is high quality visualization – final scene rendering. Fortunately, there are already a number of solutions developed to make this step less time consuming. One of the latest is to use external rendering software, capable to apply distributed computing to perform rendering task. More information on this topic can be found in chapter 5.1.4. *Cloud and Distributed computing application in Blender*.

There are also several possible performance threats in Blender as architectural design application. These can be identified and better explained by Blender MVC (model-view-controller) model [16]. In short, the model is a UI development paradigm which proposes to strictly separate application structure into three parts – “Model” (data), “View” (interface) and “Controller” (operations on data). According to it, a certain “Controller” directly changes the “Model” and not the “View”. The “View”, however, changes and updates itself according to the data provided by “Model”. The general scheme for the MVC model can be seen in figure 5.2.



Fig 5.2. *General MVC model*

Blender 2.5 MVC architecture:

- The “View” is the only place where event handling is performed and this is the new WindowManager in Blender v2.5.
- The “Controller” is split in two parts. There is a “View” related component (*Event Handler*) and a data related component called “*Operator*”. Separating these two is important for re-use of Operators, like for macros, history, redo or python scripting.
- Events are strictly separated from “Notifiers”. The first means user input events (timers, external events) have to be handled each, and in order of occurrence. The latter, Notifiers, are merely suggestions, and are only targeted at the UI (“View”) to function properly.

Blender 2.5 MVC model is presented in figure 5.3.

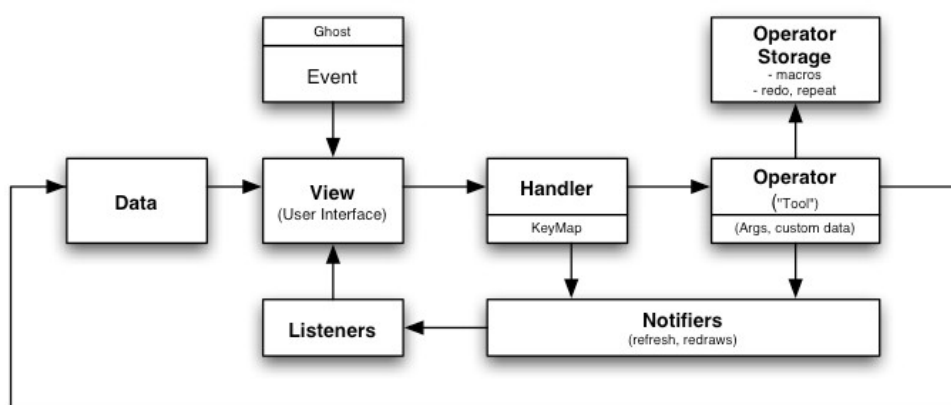


Fig 5.3. *MVC model of Blender 2.5*

One of the pitfalls where decrease in performance and efficiency may occur is in the “View” part where a very complex object or a scene modifications are performed. Due to the need to update elements in the screen when “Data” is being changed because of the use of a certain “Operator”, e.g. rotating or scaling the elements, a lot of computing has to be done. This might result in screen flickering, continuous pauses or even application freezes for a minute or more. The easiest and least time-consuming way to overcome this situation is to use more capable hardware, especially core processor and graphics card. Another solution is to utilize parallel computing using OpenCL (see alternative solutions and chapter 5.1.5) The final option when modifying complex objects or a scenes would be to optimize objects visual clipping and level of detail algorithms. This would allow reducing the elements displayed in the screen and provide a smoother work-flow. More on this can be found in chapter 5.1.3. *Setting the detail level of a parametrized object (PO)*.

The second problem might occur in “Operator” block while recalculating and setting parameter values for parametric objects groups. If the group is fairly large, these computations on group objects would require a lot of time and may become a work-flow bottleneck, causing constant program freezes. One of the possible solutions to implement system that would calculate and update values for elements visible on the screen while not visible objects modifications are cached and executed later, in the background. This may allow to perform other various tasks during calculations.

Alternative solutions

One of the possible technologies that could increase performance is OpenCL. OpenCL is an open API framework designed to develop applications requiring highly intense parallel calculations which are executed across heterogeneous platforms consisting of CPUs and GPUs. More on this subject can be found in chapter 5.1.5. *OpenCL integration into Blender*.

Recommended solutions

The following figure shows what operations might appear to be bottlenecks (colored yellow and red, depending on existence of circumvents and improvements) and which should execute without lag (shown in green). The biggest threat at the moment is python add-ons. When they are used for panel creation or import/export features, they do not cause a problem for performance. However, when they are used for intensive tasks, such as updating parameters of a complex object, it becomes rather slow, due to the interpreted language. This could and should be solved translating necessary Python scripts into C language. The C code is compiled and should cause much less performance problems.

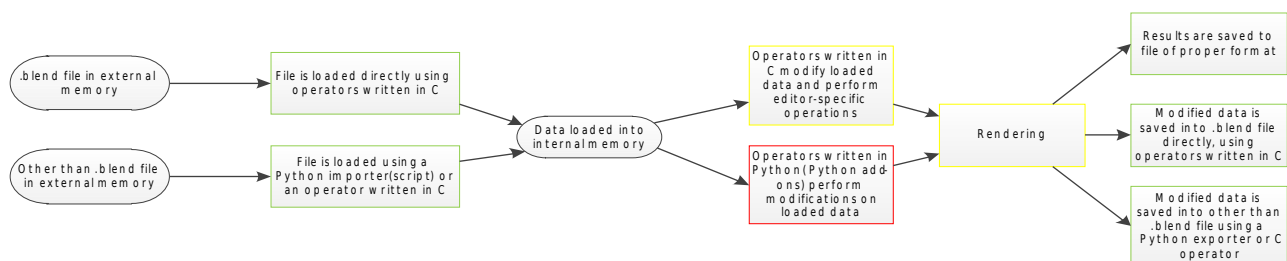


Fig 5.4. Possible bottlenecks in Blender 2.5

The other bottleneck might be current Blender renderer. For rendering acceleration external renderers utilizing parallel computing technologies should be used. Most of them are able to import Blender native *.blend format or 3ds format, which Blender is able to export to. Therefore, no

additional work is required, although a plug-in that performs export/import, render initiation and render result integration into Blender window would be rather handy. Other bottlenecks should be bypassed by using parallel calculation technologies like OpenCL. Without a network, OpenCL would distribute the computations among the processing units in a single workstation. Using distributed computing technologies, such as MPI, together with OpenCL allows utilizing OpenCL in a network, further increasing rendering speed, which is recommended if hardware resources are available.

Solutions development estimation and planning

Table 5.2. Blender bottleneck reduction estimation

Task description	Estimated MT
Bottleneck reduction solutions design	42
Total:	42

5.1.3. Cloud and Distributed computing application in Blender

Requirements

Evaluate the ability of using cloud computing and distributed computing to perform complex calculations.

Current situation and solutions

Current version of Blender does not natively support cloud and distributed computing and it is not expected to be implemented in the near future. However, currently there are quite a few products that can reduce rendering times using cloud or distributed computing technologies.

As for distributed computing there is quite a few options both for split and frame rendering. Arion rendering platform for example allows to perform distributed computing in a network. It is a commercial hybrid-accelerated and physically-based light simulator developed by RandomControl [55]. Arion uses all CPUs and GPUs in a workstation and even in the network simultaneously and guarantees an increase in performance by plugging in additional GPUs. It currently can be used with Blender only through OBJ file export (Blender) - import (Arion) process.

Exceptionally for Mac OS X users, there is yet another distributed rendering solution available – Blender with Xgrid technology [3]. Xgrid is a proprietary software that enables parallel graphics rendering in a network of computers with Mac OS X.

There are also many scripts created for distributed rendering in a network.

For cloud computing there are also several solutions. The first one comes from Greenbutton [36]. It is a commercial service that provides in-cloud rendering service. To use it with Blender a special plug-in is required to be preinstalled into workstation [35]. It is worth mentioning that Greenbutton service is also compatible with other software products, including specialized rendering engines like LuxRender, YafaRay or a bioinformatics software Geneious, used to search, organize and analyze genome and protein information. The alternative solution is ResPower service. It allows to perform rendering tasks for Blender, 3ds Max, Maya or any other programs in a cloud.

However all these solutions are suitable for final graphics visualization rendering only. There are no widely known analogous solutions for other computation intense tasks in Blender.

Alternative solutions

Blender competitors in architectural design market have a better situation of the distributed and cloud computing implementation in their packages. While there exists no native distributed or cloud rendering functionality for any of them there are highly integrated external renderers that support distributed computing and can be used by these architectural design products. For example, Mental Ray by Mental Images runs as a standalone software and is also integrated into several packages (native of them -NVIDIA ARC GmbH [48]) such as Autodesk 3DS MAX, VIZ, Inventor, AutoCAD, and Revit, as well as SolidWorks, Matucad DomuS3D and others. AccuRender is a renderer for AutoCAD 2002-2010 that also supports distributed rendering.

As for cloud computing, AutoCAD provides a cloud service AutoCAD WS – to view and edit DWG files. There are tests being performed with an environment for Revit, AutoCAD and Inventor to be run as a cloud service. Furthermore, there is a cloud server for Revit called Stratus. The user accesses the application using the browser. Revit also provides functionality to assign certain computations for a remote server as a feature of the Subscription Advantage Pack.

Recommended solutions

The most important part in distributed computing is the sort of computations and their range. It is quite obvious that currently and in the future the biggest resource-draining task will be graphic scenes rendering (single frames or animations) – the following will focus on them.

Using Arion for this task could provide distributed computing and high quality [55]. On the other hand, it would be expensive and the top quality might not always be what is required. Thus, chances for free alternative to succeed are relatively large. The easiest way in Blender could be a scripted application (a plug-in). There are already a few distributed rendering scripts on the internet that perform this operation for Blender but they are poorly documented and not always well integrated into Blender itself. Of course, it could also be created from scratch, possibly making the script better fit designers needs and easier network configuration. This way the renderer could be either Blenders native, or a free, open source specialized software, like YafaRay or LuxRender.

Other recommended solution that could provide great results and require practically no time or resources to implement functionality is external, free, high quality renderer that supports distributed computing, e.g. LuxRender [41]. LuxRender has integrated tools to run in a network and provides high quality results. It also highly integrates with Blender through a plug-in.

For other tasks to be performed in a network it is worth to decide whether they really runs efficiently in a single workstation. For example, updating and recalculating some parameter values might be more efficient after updating and reorganizing the way the calculations are performed.

For other tasks using cloud services is not recommended, because such solutions will be highly inefficient. Each time sending, registering tasks in a queue and retrieving results would take a significant amount of time, sometimes even larger than doing task locally, especially for real-time tasks. Analogously to distributed computing, it would be probably wise to optimize the way the operations are performed and only then considering transferring the computations to the cloud.

5.1.4. OpenCL integration into Blender

Requirements

Evaluate OpenCL technology integration into Blender.

Current situation and solutions

OpenCL (Open Computing Language) is an API standard for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, and other processors [40]. It includes a language for writing kernels (functions that execute on OpenCL devices), plus APIs that are used to define and then control the platforms. OpenCL provides parallel computing using task-based and data-based parallelism. It has been adopted into graphics card drivers by both AMD/ATI and Nvidia, which offers OpenCL as equal choice to its CUDA in its drivers.

OpenCL gives any application access to the Graphics Processing Unit for non-graphical computing, extending the power of the GPU beyond graphics. OpenCL is analogous to the open industry standards OpenGL and OpenAL, for 3D graphics and computer audio, respectively. OpenCL is managed by the non-profit technology consortium Khronos Group.

At the moment, there is already a few projects developed as Blender branches where OpenCL is utilized - nodes compositor and new rendering engine called - "Cycles" When talking about computer graphics tasks such as 3d scene or animation rendering, OpenCL looks very promising. Till now all the basic rendering was done by the processor (or several of them, if more than one are available), while highly parallel calculation abilities of GPUs were not used at all. As it was said, OpenCL enables combining CPUs and GPUs and other processors for rendering, highly increasing performance of intense calculations tasks. LuxRender is currently being developed and tested for OpenCL integration into the system [43]. It should include not only CPU and GPU, but also network rendering, meaning that resources of other computers in the network will also be available for use.

Alternative solutions

There are several alternatives that provide analogous functionality to OpenCL.

The first and the most similar alternative to OpenCL is CUDA (Compute Unified Device Architecture), - a parallel computing architecture developed by Nvidia [49]. It is Nvidia graphics processing units (GPUs) based computing engine that is accessible to software developers through variants of industry standard programming languages. CUDA architecture shares a range of computational interfaces with competitors - the Khronos Group OpenCL and Microsoft DirectCompute. Drivers for Nvidia graphics cards now also provide the ability to use either CUDA or OpenCL technology for processing. CUDA, just like OpenCL, enables performing general purpose computations on the GPU (GPGPU). CUDA has been used to accelerate non-graphical applications in computational biology, cryptography and other fields. Although CUDA is considered to be more mature than its competitor OpenCL, the former is only available on Nvidia GPUs, whereas the latter is an open technology and can be used with Nvidia, AMD/ATI and even other graphics cards, furthermore, it is available to use on CPUs as well.

Another alternative, as already mentioned is Microsoft's DirectCompute. It is an application programming interface that supports GPU on Microsoft Windows Vista and Windows 7. DirectCompute is part of the Microsoft DirectX API collection and was initially released with the DirectX 11 but currently runs on both - DirectX 10 and DirectX 11 compatible GPUs. Since it is a proprietary technology, requires DirectX and runs only on the latest Windows operating systems, it is not exactly suitable for use in Blender which is based on OpenGL API.

The third alternative is OpenMP (Open Multi-Processing) [50]. It is an API that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most processor

architectures and operating systems, including Unix, AIX, Solaris, Mac OS X, and Microsoft Windows platforms. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior. Unfortunately, OpenMP itself can only run on CPUs, however, it is possible to make OpenMP work together with CUDA, utilizing both CPUs and GPUs. OpenCL in comparison to OpenMP natively support CPUs and GPUs.

Recommended solutions

CPUs and GPUs use different approaches to perform a certain task. While CPU in general uses a single thread and performs it quickly, the GPU deploys a large number of threads relatively slowly (because of the clock frequency) but executes them in parallel. Overhead occurs by creating and managing these multiple threads, moving data from/to GPU is required. In this way – a handy combination of both – CPU and GPU shows the way where OpenCL technology could be used. To use OpenCL first of all the computations must fit for segmentation and parallelization. Secondly, the amount of computations must be sufficiently large to surpass the tasks management overhead and perform faster than on CPU. Furthermore, it is still important to note that results of deploying OpenCL greatly depends on the level of code optimization, which might appear to be different for different hardware architectures. This shows, that, creating a high-performance OpenCL code is a rather difficult task, requiring both - knowledge of different types of hardware architecture and time to implement and test the code.

As for OpenCL application, there are at least several cases where it can be used. First of all, as mentioned before, it can be used for rendering photo realistic scenes or animation images. Currently this task could be performed by a free open-source engine LuxRender, which currently works on OpenCL functionality utilization for rendering purposes.

The second subject where OpenCL could speed-up is modifying data structure. Here it is necessary to have a suitable data structure, which should have the ability to be processed by several threads at the same time, e.g. ability to have independent iterators. As an alternative, different parts of the structure could be copied to new, independent data structures and then passed to the devices. But this most probably take too much time and the overhead would outweigh the benefits of decreased modification time. For the same reason, it is worth considering if there would possibly be sufficiently large amount of data to process and how often this could happen.

The current bottleneck, not only for OpenCL, but for CUDA as well, is rather slow data transfers between Host memory and Device memory. This can be solved by using OpenGL to access functionality of its vertex buffer objects and index buffer objects. This way the memory in the graphics card can be accessed directly, without a need to copy data.

When utilizing OpenCL it is important to remember, that some, especially older computers with older graphics cards might not support it. For this reason the ability to perform the functionality on CPU must be preserved. It should be enough to check on start-up if the environment is configured for OpenCL and load libraries/files accordingly.

5.1.5. Integrating external rendering platforms in Blender

Requirements

Evaluate the integration of Blender with external rendering platforms.

Current situation and solutions

Currently, the market is rich with rendering software, which can be used with Blender. Some of the most popular external renderers are presented and detailed in the table 5.3. [28, 29, 30, 41, 42, 44, 55, 56, 58, 60, 61, 70]

A renderer is considered to be sufficiently integral with Blender if it provides a certain plug-in (sometimes called exporter) for Blender. These plug-ins usually are available with a set of tools which can be displayed and can be managed in a panels, located in a Blender UI. These settings are then passed to the external rendering software and the results are usually presented in a separate renderer window, which itself can have tools to change rendering settings, save or modify the result.

Another, but not so convenient way to make Blender work with external renderers is to export scenes in some common 3D files formats. One of the most common formats is OBJ. However, it is not an ideal solution, because OBJ files does not preserve camera settings and number of other scene parameters, making the exported scene a mere object, which then still has to be customized in the external rendering software.

Table 5.3. External renderers integrated with Blender

Platform	Arion	Fryrenderer	Indigo Renderer	LuxRender	Octane	Maxwell Render	Thea Render	YafaRay
Operating systems	Windows, (Mac OS X, Linux to be released)	Windows, Mac OS X, Linux	Windows, Mac OS X, Linux	Windows, Mac OS X, Linux	Windows, Mac OS X, Linux	Windows, Mac OS X, Linux	Windows, Mac OS X, Linux	Windows, Mac OS X, Linux (Ubuntu)
Price	1 lic: Single-GPU 795€ Multi-GPU 995€	Master Package (1 Master, 2 Slaves) 795€	Master Package (1 Master, 2 Slaves) 595€	Free (GPL)	99€ (v1.0 Beta Lic)	895€ (v2 Standard)	€295 (Application lic + 2 node lic)	Free (LGPL 2.1)
Devices	CPU+GPU+Network	CPU+Network	CPU+Network	CPU+Network	CPU+GPU (Nvidia)	CPU+Network	GPU+Network	CPU
Plugin for Blender	-	-	+	+	+	-	+	+
File Format Compatibility	OBJ	OBJ	-	-	OBJ, DAE	OBJ, 3DS, LWO	-	-

Recommended solutions

For rendering purposes it is recommended to use one of the alternatives presented in the table 5.3, mainly focusing on those providing plug-ins for Blender. According to reviews and opinions of Autodesk Revit users, a possible pleasing solution would be to use Octane, which provides unbiased methods and GPU computing to render the images or animations. Although GPU computing is currently provided only for Nvidia cards (CUDA technology), there are promises by octane developers that it will provide OpenCL functionality in the future. As an free alternative to Octane is a physically-based raytracer - LuxRender. Another very promising feature is as new Blender internal rendering engine developed - “Cycles”.

5.1.6. Import/export ability for popular formats

Requirements

Evaluate the ability to import and export files of popular formats.

Current situation and solutions

Currently Blender supports a number of file formats for both – import and export (see table 5.4.) [14]. Importers and exporters in Blender are actually plug-ins, written in Python. Currently, for Blender v2.58 they are located in *Blender/2.58/scripts/addons* folder. Although it is easy to add new importers/exporters in Blender, creating them might be sometimes complicated.

Table 5.4. Supported import/export formats in Blender

File Format	Importable	Exportable
Collada (.dae)	+	+
Motion Capture (.bvh)	+	+
Scalable Vector Graphics (.svg)	+	
Stanford (.ply)	+	+
STL (.stl)	+	+
3D Studio (.3ds)	+	+
Wavefront (.obj)	+	+
X3D Extensible 3D (.x3d/.wrl)	+	+ (.x3d)
Autodesk FBX and XNA FBX animations / model (.fbx)		+
Autodesk DXF (.dxf)	+	
AC3D (.ac)	+	+
md5		+
DirectX (.x)		+
M3G		+
OpenScenegraph (.osg)		+
HiRISE DTM from PDS IMG (.IMG)	+	
Blizzard M3 (.m3)	+	

GIMP Image to Scene (.xcf, .xjt)	+	
LightWave Object (.lwo)	+	
Skeleton Mesh (.psk)	+	+
MakeHuman (.mkx)	+	
LightWave Point Cache (.mdd)	+	+
Raw Faces (.raw)	+	+
Images as Planes (.jpeg, .jpg, .png, .tga, .tiff, .tif, .exr, .hdr, .avi, .mov, .mp4, .ogg, .bmp, .cin, .dpx, .psd)	+	
Cameras & Markers (.py)		+
Pointcache (.pc2)		+
Animation Data (.psa)		+
Quake MAP (.map)		+
Acclaim Motion Capture Files (.asf, .amc)	+	
Lipsync: Moho (Papagayo, Jlipsync)	+	

Alternative solutions

There is a platform-independent library written in C++ called Open Asset Import Library aka Assimp. It provides the functionality to easily import 3D scenes and models from more than 25 file formats, including Collada .dae, 3DSMax .3ds, Wavefront .obj, Lightwave .lwo, Milkshape .ms3d, Stanford .ply, .md5, .md2, .md3, .mdl, and DirectX .x. Assimp is geared mainly towards 3D engines and other GPU-accelerated renderers, providing data in a straight-forward, in-memory format and offering a multitude of post-processing steps to further enhance the imported data. It supports meshes including multiple UVs, vertex colors and tangent space, bones, scene hierarchies, materials, node/bone animations, lights and cameras. The library can be used in Blender by using the script “import_scene_assimp.py” (not fully functional yet). There are also specialized libraries importing files containing image, scene and model information that are used by specific software, e.g. 3DS import/export for Solid Edge.

Recommended solutions

The popular architectural formats compatibility with Blender is currently rather poor and importing files of these formats could be at least inconvenient.

The popular DWG format currently has no importer in Blender. Since it is a native format of AutoCAD, problem can be solved by exporting the modeled 3D content data to AutoCAD not as a DWG, but as DXF file. However, sometimes there might be no ability to ask the client to provide a file of suitable format or to use AutoCAD to convert the file by oneself. Thus, to avoid the situation, there are several options. First is to use an open-source program like Lx-Viewer (Linux/Unix only) or QCAD [59] that can convert files from DWG to DXF. To avoid an inconvenient way to invoke these programs a script calling these programs in the command line mode could be created. For import/export purposes a Python library (scripts) could be used, requiring little effort to make them work. A common problem to all the previous solutions is that when importing architectural design drawings, essential architectural details, e.g. dimensions, will be missing in Blender, since they are currently considered unnecessary and are not imported. Eventually, probably best and recommended

way is to create a script to import/export DWG files, thus preserving the consistency of Blender system and being able to choose the details which are imported and the way they are imported. It is possible, since the format specifications are available. When developing such a scripts, DXF import/export scripts could be used as a base. One more detail to mention is that because almost every new version of AutoCAD presents a new DWG version, it will require constantly update import and export script for every to match the every new AutoCAD version.

As for Google SketchUp format (.skp), there are similar options. Firstly, different formats can be used to export drawing from SketchUp Pro [33] like 3DS, FBX, DAE. Unfortunately, the free version of SketchUp does not provide any alternative formats for files that could be already imported into Blender. It probably can only be done with workarounds for KMZ file format. Since there is currently no specification presented for SKP format, probably the best way is to use Google SDK: SkpReader C++ API and SkpWriter C++ API [31]. The first one is for reading information out of a .skp file and transforming it to some other format or object model. The other one is for constructing a .skp file from another format or object model data. There are specialized libraries that support interoperability of Python and C++, thus it should be easy to create such Python scripts to import/export .skp files that would be consistent with current Blender structure.

As for Autodesk's Revit formats (.rvt, .rfa), there are currently no format specifications or SDK provided, thus creating importers/exporters for it is impossible. The only possible solution to preserve compatibility (although it would most probably not meet the needs) between Revit and Blender, would be use Revit importers and exporters to export to other formats, supported by Blender as well, e.g. DXF or DWG in the future if proper importer for the latter is created. Anyway, for all the details to load in Blender, both DXF and DWG format importers/exporters should be modified and adopted to load more information, since today there are no scripts suitable for architectural design parameters visualization in Blender.

The file format .ifc (Industry Foundation Classes) contains data model which is intended to describe building and construction industry data. The format is important to the client. It is a neutral and open specification that is not controlled by a single vendor or group of vendors. It is a well-known standard in architectural design field and has importers/exporters in many architectural design applications, including Revit, ArchiCAD and others. However this format is criticized for causing data fidelity losses when importing/exporting data from/to a file of this format. There is an add-on created to import .ifc files to Blender, called IfcBlender, although it is not widely used and not proved to be accurate. Still again it is recommended to create the importer/exporter in Python, because of the open specification and the need to read/write data that provides technically sensitive and important information, like dimensions.

As for .pdf, there is an importer for Blender, which imports vector files as mesh polygons. In the same way curve objects can also be added from Adobe Illustrator and SVG files. Unfortunately, there is currently there are no exporter for .pdf.

The only text file format currently supported by Blender is a plain text file, i.e. TXT. There are specifications provided for MS-DOC binary file, OpenDocument text (.odt) and others, which could be used to save or load data if proper plug-ins are created. However, viewing and editing the information of these files would require new editors in Blender.

5.1.7. Usage of digital signature for Blender files

Requirements

Evaluate the ability to sign Blender files with PGP or other digital signature, allowing to

identify the author.

Current situation and solutions

According to Blender Foundation possibilities to encrypt and/or digitally sign the .blend file format [22] are available, although particular technologies are not widely mentioned. There is also a separate encrypted Blender file format (.block) which only Blender can read. However, according to the publicly available information Blender files aren't usually signed

Although PGP and later products, implementing the OpenPGP standard, are initially created to sign and encrypt messages and e-mails, it also allows signing and encrypting virtually every file, including .blend files. This functionality is not integrated into Blender and works as an external software.

Alternative solutions

Over the past decade (since 1991), PGP, and later OpenPGP, has become the standard for nearly all of the world's encrypted email [51]. By becoming an Internet Engineering Task Force (IETF) standard (RFC 4880), OpenPGP may be implemented by any company without paying any licensing fees to anyone. However, there are many commercial products, implementing the standard. For this reason there is a rather popular alternative that implements OpenPGP – The GNU Privacy Guard, also known as The GnuPG or GPG. There are GPG versions available for Windows, Linux and Mac OS X.

Despite the widespread OpenPGP, there are some other encryption and signing solutions. One of them – Safester. It provides security via a third party (server). However, since it is not widely used and is intended basically only for emails, it does not create a challenge to OpenPGP.

An alternative to OpenPGP could be Secure Shell (SSH). It works a bit differently, although the purpose remains the same, i.e. secure and authenticated data transfers. In effect SSH is a network protocol that allows data to be exchanged using a secure channel between two networked devices. The encryption used by SSH is intended to provide confidentiality and integrity of data over an unsecured network, such as the Internet. The authentication, integrity and security here is kept during the connection, i.e. when files are being sent. As a comparison, in OpenPGP it is checked whether the files or messages are from the one that is declared and the information has not been tampered with. Security (forbidding someone to intercept the data) is achieved the same way as in OpenPGP – with encryption.

Recommended solutions

It is recommended to use a cross-platform and possibly free implementation of OpenPGP. If the system is not going to be used commercially, it might be best to use GPG, which is free and uses GNU GPL license. Since it is currently not part of Blender, GPG should be integrated into Blender as a Python script (plug-in). This possibility is available because of the Python wrapper for GPG, which is partially created by the GPG developers themselves [69]. The easiest way to integrate this functionality would be to call the operator `wm.save_as_mainfile` first and then, if file is saved, create a signed and/or encrypted version of it using a wrapper. However, the best solution would be to provide an options to encrypt and sign the file in the file save dialog. This would be more complex, since it will require modifying Blender source files – probably modifying the operator `WM_OT_save_as_mainfile` in `wm_operators.c` and dependent functions. Also it may require, eventually adding code for GPG actions in `wm_save_as_mainfile_exec` function. Adding

functionality to open encrypted files should be also considered. However implementation will be similar to the “save a file” implementation. All the modifications should be focused on *WM_OT_open_mainfile* and dependent functions in *wm_operators.c*.

Solutions development estimation and planning

Table 5.5. Estimation of GPG signature integration into Blender implementation

Task description	Estimated MT
GPG signature integration into Blender	18
Total:	18

5.2. Creating and editing parametrized objects (PO)

5.2.1. User interface (UI) elements for PO creation and editing

Parametrized object (PO) is an object that binds together geometry, information about parameter values and additional visual data – bitmap images, textual descriptions and other material. The main parameters of a PO are the following:

- PO title
- PO type name – defines certain parameter values.
- Dimensions (length parameters) – height, width, length and other.
- Material parameter – references the material that is set to the object.

PO are basically .blend files, supplemented with certain data, consisting at least of template title, category and object's name. A single PO can be treated as a link which can be either embedded or related to a PO library in a local network or a computer. The latter means a local version should dynamically update when the version in the library is modified.

PO to the project could be added in one of the following ways:

1. With a certain key press combination the activated geometry is stored as a new PO, to whom the template can be changed at a later time.
2. An already created PO is added to the project from the project tree using a drag-drop operation.
3. The most frequently used POs can be stored in a panel, from which they can be easily added to the project.

Requirements

Examine the possibility to add the functionality of creating and managing parametrized objects (PO) in Blender.

Examine the features and implementation possibilities of PO specific UI elements.

1. *An UI element: table of PO parameters.* It's an element representing PO parameter values.

This tool also allows modification of these values.

Parameter is an object used to describe a certain object feature, e.g. size or color. It consists of unique title and additional meta-information, e.g. title translation into a certain language or an abbreviation.

Parameter data types:

- Length (float)
- Yes/No (boolean)
- Textual (string)
- Area (float)
- Volume (float)

To implement the functionality of PO's and their parameters in Blender, the first thing to do is to adapt PO data types requirements to Blender's DNA and RNA types [16, 27]. These modules are located in */blender/source/blender/makesdna* and */blender/source/blender/makesrna*. Then certain new operators will be needed. They are normally located in two places: in editor modules or in Python scripts. The editor modules are used to provide a certain set of related operations, such as for video editing, animation, etc. Since all the menus and panels are being built from existing tools using Python scripts, located in */blender/release/scripts/startup/bl_ui/*, there probably will be no need to implement new editors for creation and modification of PO – python scripts for panel creation should be sufficient (an example of table for PO parameter values in a panel is presented fig 5.4). However, other PO related functionality will require new editors, since the functionality proposed is new in Blender. If existing tools for ui creation are not sufficient, new ones should be created and placed in */blender/source/blender/editors/interface* module. Accessing Blender data (through RNA) is accessible from python scripts and internally from C functions.

The creation of the PO template, defining parameters, describing their features may require new editors for parameter definition. These editors should be located in */blender/source/blender/editors* and creating it would require supplementing several other files in the editors module and below, especially the file */blender/source/blender/editors/space_api/spacetypes.c*. The structure of parameter definition in the figure 5.4 allows to suggest that the parameters could be defined in an XML file. Thus, a certain parser would be needed. Since there could be a special importer/exporter for these parameter files, an add-on in Python would be the best solution. It should be located in */blender/release/scripts/addons/*. Other exporters/importers in that folder could be used for reference. The parser created in C and located in */blender/source/blender* would possibly be more extensible and suitable for future needs, thus only panels are recommended to create in python scripts in this case while everything else should be created in C. The coloring of fields in panels (fig 5.4) currently is not implemented, thus appropriate changes will have to be performed in the *Interface* module.

The parameters for a PO and template should be stored in Blender data structures, which means modifications of DNA and RNA structures will be required. A necessity of changes in the *Blenloader* module is also expected.

The title and possibly some other parameters of the PO template should be customizable on creation in the way it is shown in the fig 5.4. This should be allowed only when the required parameters are added to the PO template. Managing the creation of the name dynamically, i.e. letting to choose the parameters involved in another parameter value for a PO from a list with a mouse or in some other way, would require a new editor and would be more user-friendly, whereas

creating a title when merely writing something like “Cube {Length}x{Width}x{Height}” in a text box would be error-prone and less friendly, although easier to implement.

The direct use of the functionality considered here is certainly beneficial in the mainstream version. However, it could be adopted to the latter for specific purposes other than BIM. The parameters might be useful for storing additional information for certain types of objects. Applying templates to the objects could be useful for categorizing objects, e.g. for distinction of moving objects, buildings and other.

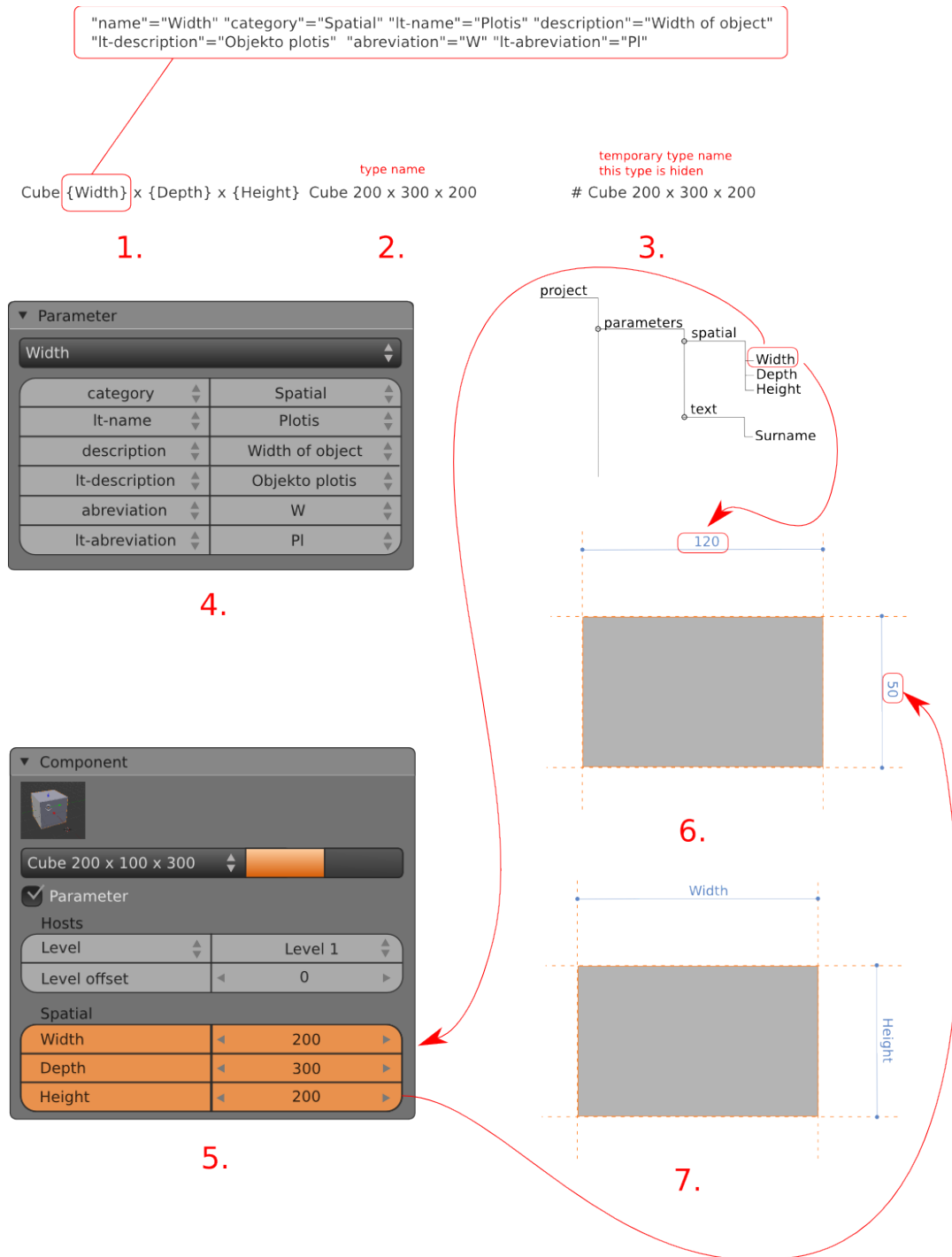


Fig 5.5. Parameter concept

2. UI element: node based editor for management of connections between PO's.

The first thing to mention for the PO node based editor is that although such floating panels (see figure 5.5) are technically allowed in Blender and possible to create it is unrecommended to do so, because it would violate the current Blender design which declares that all elements should have fixed positions (despite dimensions) and must belong to a certain parent window or editor. However, if no other options are available and desirable design should be used the floating elements of UI should belong to a certain dedicated space and moving them outside of it should be forbidden.

Anyway, building this functionality has very few basic points to start with. It might require change the *Ghost* system (*/blender/intern/ghost*) as well as interface module (*/blender/source/blender/editors/interface*). Using python scripting system might also appear to be complex, but appropriate changes for the interface module should allow it.

Any reimplementations of the interface will require changes to the DNA and RNA for storing and processing information related to PO nesting, parent fields and other. A module for drawing curves joining one floating element to another will be required. The module should be placed in */blender/source/blender/*.

Benefits of the node-based editor functionality for better visualization in the mainstream version of software are quite obvious. It could also be used to express other relations, such as showing parent objects and their children, although it would probably be good to consider to implement it as a strict tree, not an editor with freely floating elements.

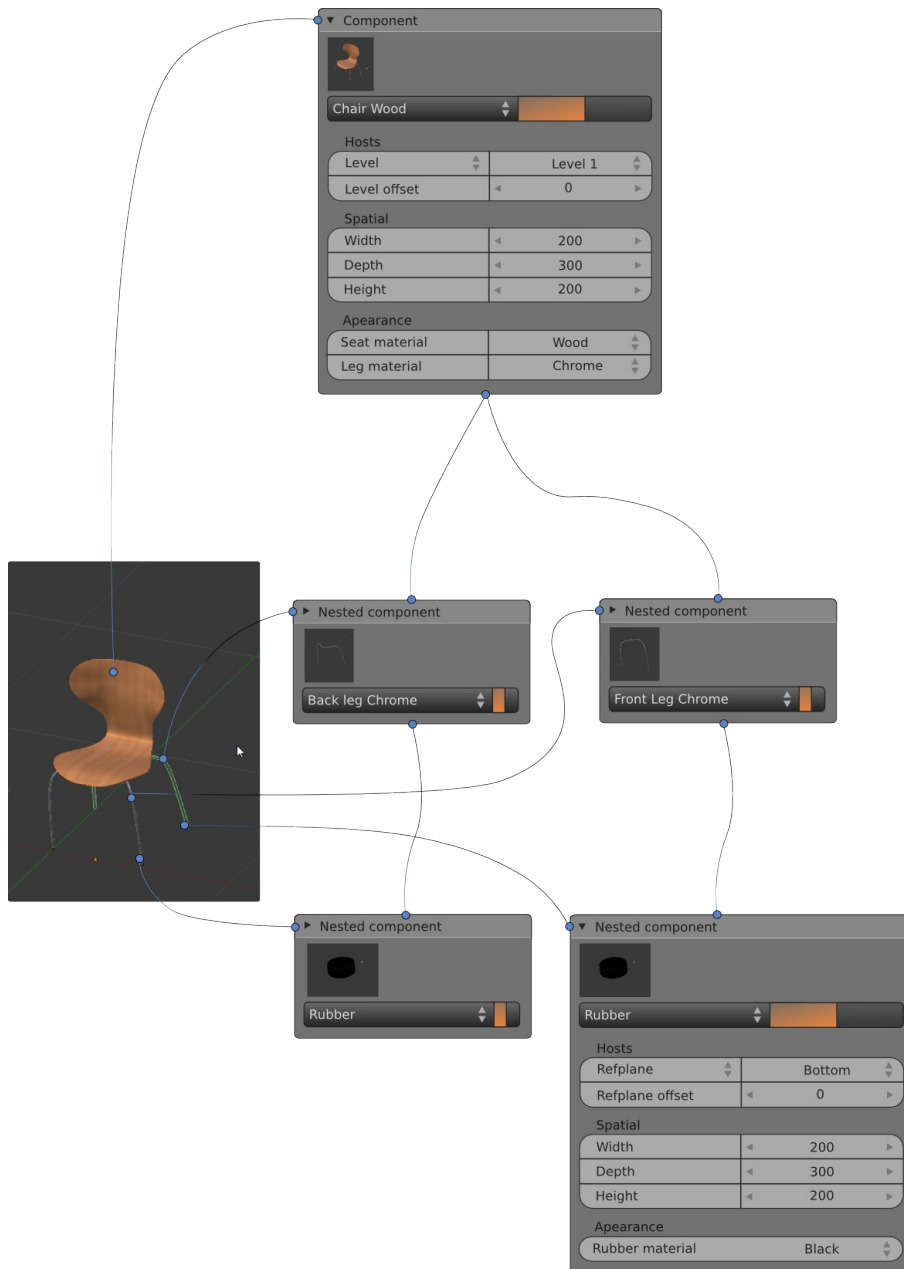


Fig 5.6. *Parametric component concept*

2.1 Modifier components. Modifier integration into PO is a rather difficult task, because implementing this functionality would require changing plenty of modules, including *makesdna*, *makesrna*, *blenloader*, *editors*, */blender/source/blender/modifiers*, and even more. Technically modifiers are equivalent to objects in Blender and are available in objects hierarchy of current Blender data structure. Thus, creating a modifier as a child to a PO should be possible to implement.

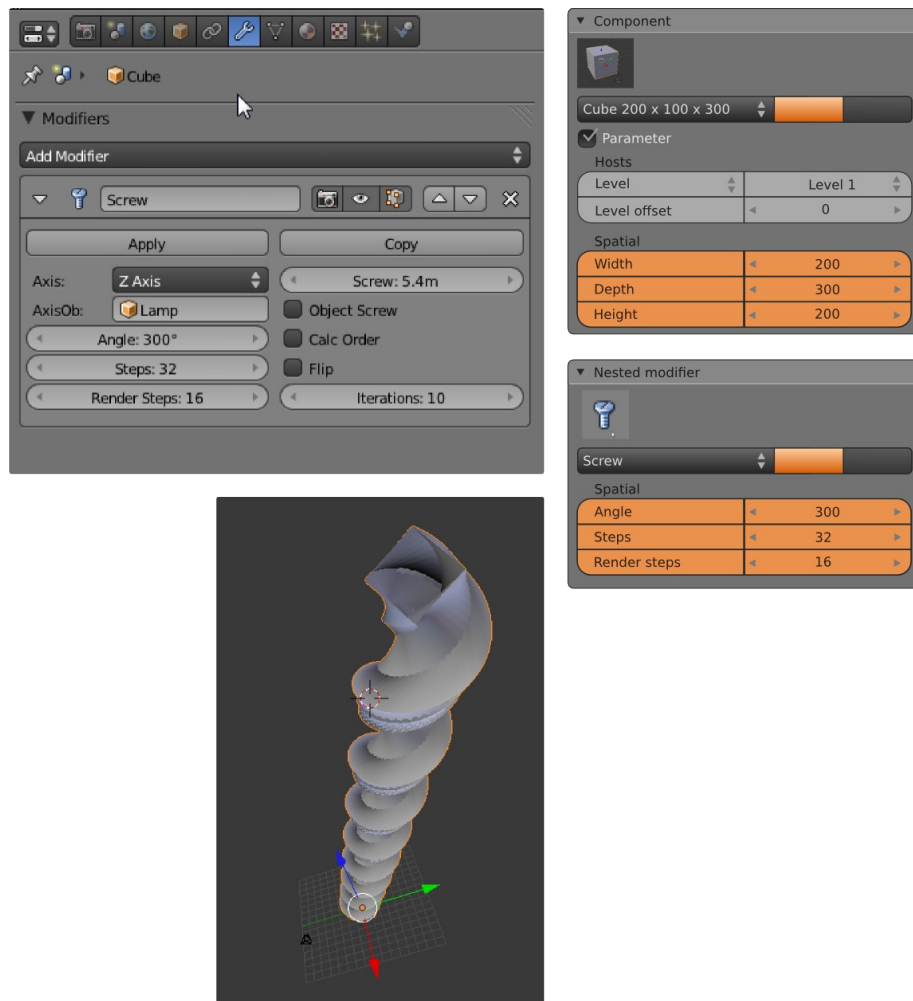
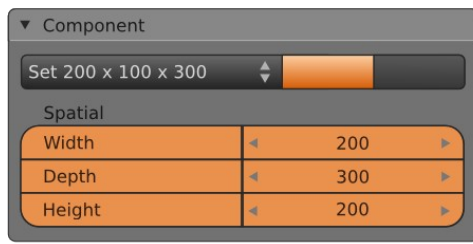
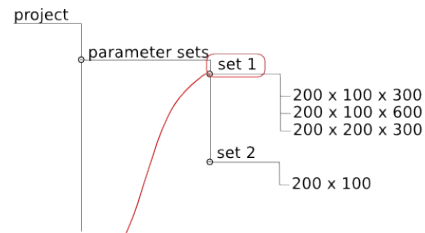


Fig 5.7. *Modifiers integration into parameters concept*

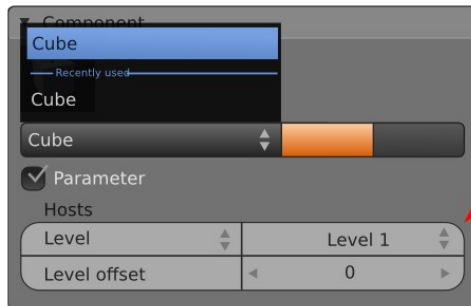
2.2. *0D Parametric objects* – sets of parameters and their values. Used as a template or standard size tables. The features of 0D PO is expected to be easy to implement since most of the functionality would be implemented while programming PO parameter tables and parameter mechanism. The changes to Blender source should be made in the same modules as described in PO parameter tables tool description. 0D element concept is displayed in the figure 5.7. The project tree element presented in the pictures is discussed in *UI element: project tree*.



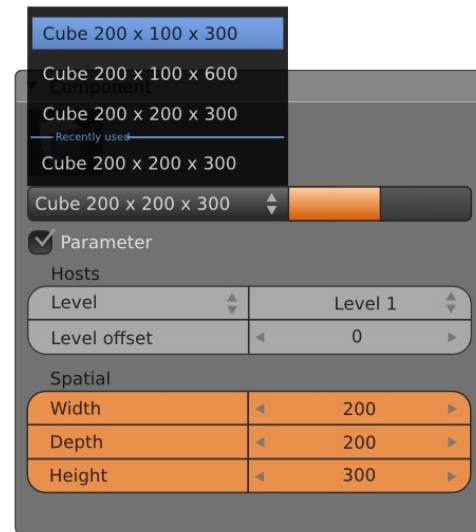
1.



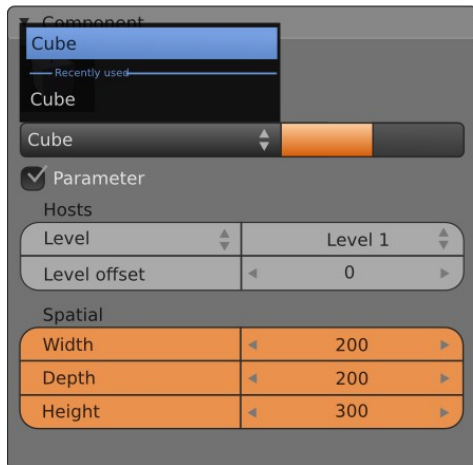
2.



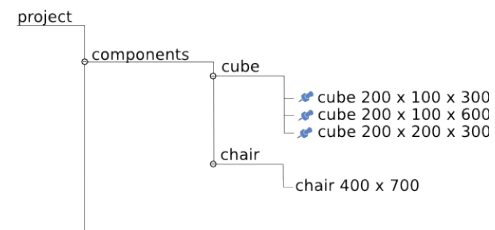
3.



5.



4.



6.

Fig 5.8. 0D component concept

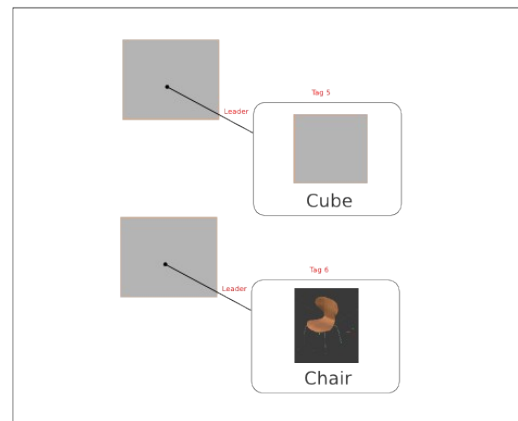
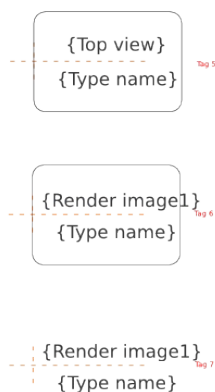
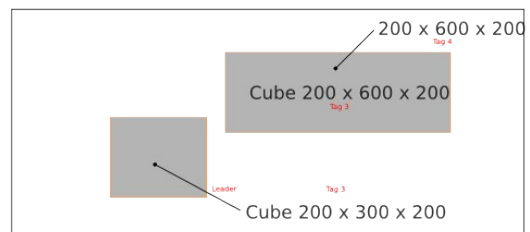
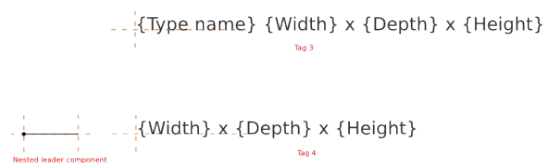
2.3. 2D PO – an object in a plane. This type of PO can be used for dimension markings, tags, 2D details, tables and other UI markings and elements (figure 5.8.). The element itself is very similar to the 0D component, except it would have graphical representation and manipulation tools. Thus, the PO mechanism should have to be supplemented with such a functionality. It should be similar to other objects (e.g. meshes) in Blender, such as a planes. As a reference for 2D component

implementation current Blender implementation details could be found in */blender/source/blender/blenkernel/ intern/ module file mesh.c*. Changes would also be needed in *makesdna* and *makesrna* for storing custom data of different 2D POs. If certain 2D POs are expected to be treated differently (e.g. tags – bound to 2D screen and stay in the same 2D position on screen when rotating the scene), further categorizing and changes to *operators* (*editors* module files) are required. For drawing borders and grid-lines it is advised to look at the *Interface* module, especially *view2D* part [20].

Edit component mode

Component loaded into project

Tag



Dimension

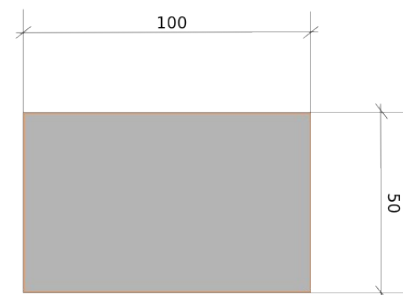
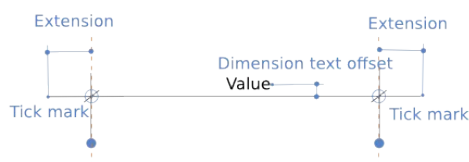
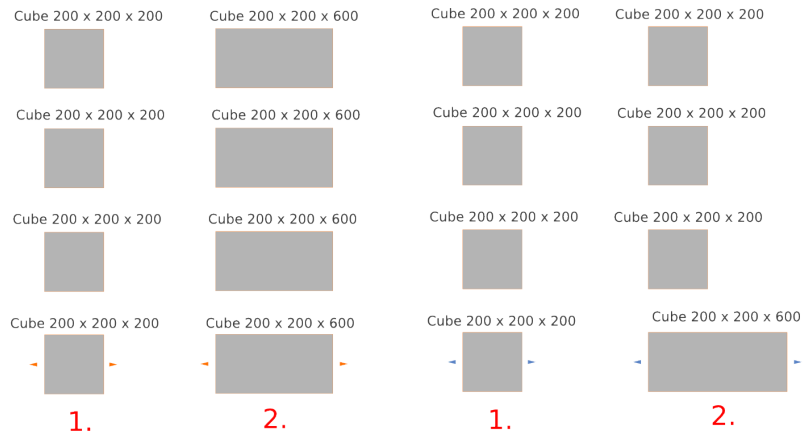


Fig 5.9. *2D component concept*

The functionality for modification of 2D PO's should provide separate cases for changing template and changing an object of that template. The use cases of these features are displayed in the figure 5.9. The first one (orange color in the picture) changes all objects of that template. The second (blue color) changes solely the selected object.

This functionality should be rather easy to implement, since all changes to RNA and DNA would already be made when creating PO base functionality. The only major thing to do is to create a mechanism for iterating through a list of POs, checking POs template and changing the objects appropriately. This iteration cycle depends on the implementation of POs storage and subsidiary structures, allowing to reach certain POs by some attribute value. When changing a selected object, the operation is even simpler. The switch between template change/object change is trivial, although might require creating a new UI element, thus there would be a need for some changes to the *Interface* module. The panel should be created as a python script. The whole mechanism for applying changes should be added to the previously mentioned PO mechanism or a new module in */blender/source/blender*, which would still have to cooperate with that mechanism.

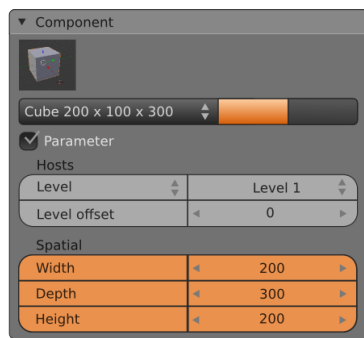


ALT key + Click and drag

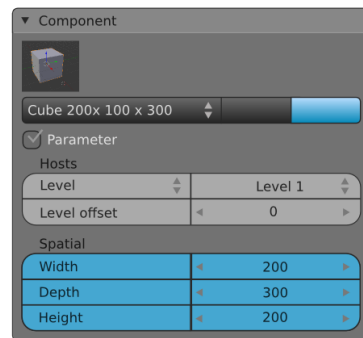
This action edits egisting type parameters

Click and drag

This action creates new type



3.



4.

1.

Name	Count
Cube 200 x 200 x 200	4

1.

Name	Count
Cube 200 x 200 x 200	4

2.

Name	Count
Cube 200 x 200 x 600	4

2.

Name	Count
Cube 200 x 200 x 200	3
Cube 200 x 200 x 600	1

5.

6.

Fig 5.10. Component types concept

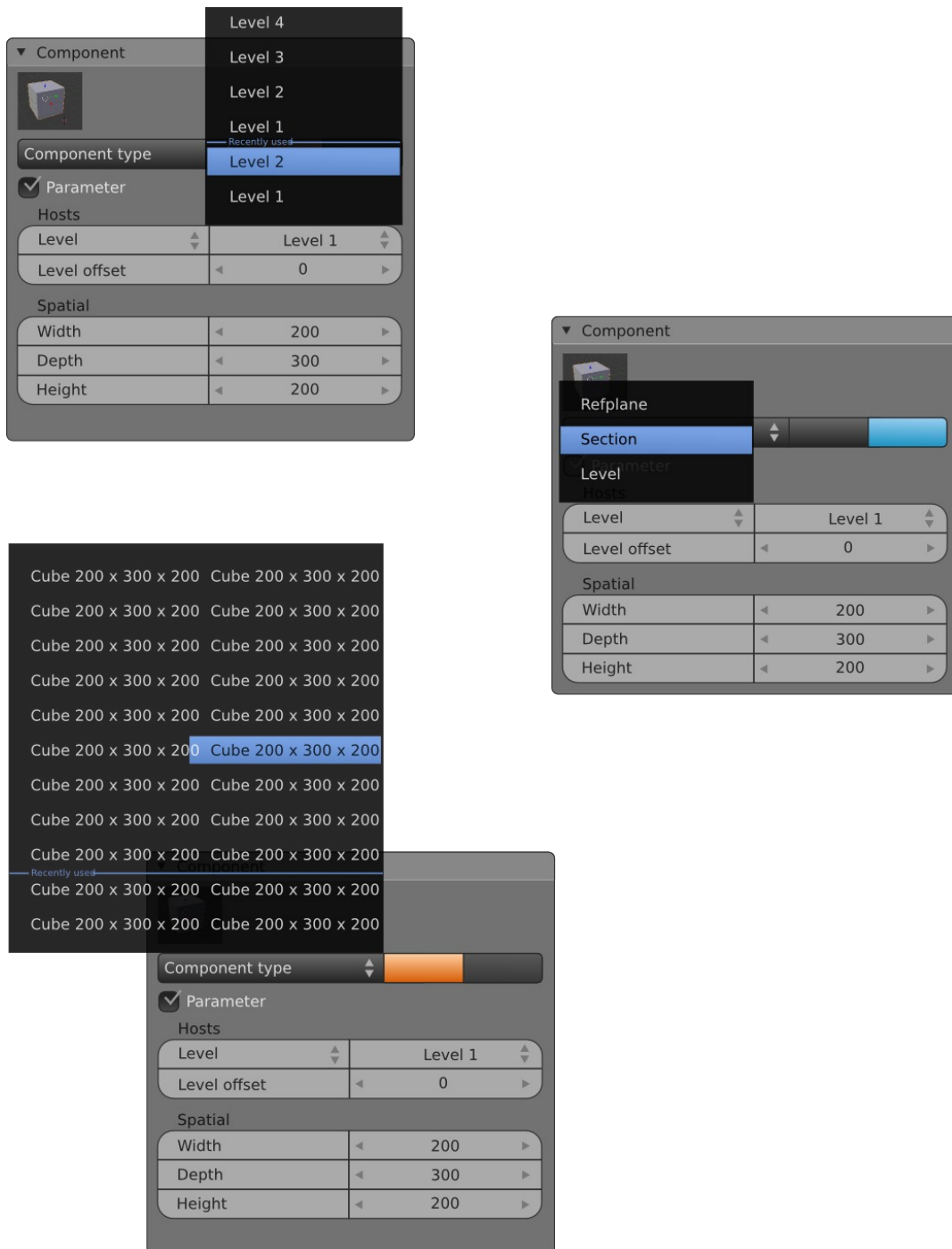


Fig 5.11. PO components GUI

2.4. A TAG. A Tag is a PO, displaying certain parameter values of another POs. In a 2D view when adding a tag onto another PO, the names of parameters in a tag are replaced by the values of the pinned PO. If the PO does not have those parameters, the tag remains blank. Parameter values also include views from different perspectives (top, bottom, etc.).

The mechanism for tags involves several parts. The first one, is to draw the tag and replace its fields with the parameter values of a PO. This involves functionality of general PO mechanism – retrieving information of a PO by a certain PO ID or other field. Thus, functions for updating the tag should be easy to implement in the general mechanism in C language.

The other part, which consists of event handling (drag, drop and others), activating the right PO is a bit more difficult. It involves changing `/blender/source/blender/windowmanager/`, especially its event system part, also the interface module. Some references to the possible changes could be found in the Blender's *Ghost* system (`/blender/intern/ghost`) for event management in lower level. Since tags will be added in view3D and possibly in certain new editors, changes to these editors and interface modules will also be required, e.g. for adding new object (tag), displaying and updating it.

A tag displaying an object view from a certain perspective will probably require a low resolution image rendering. Since rendering is already implemented, probably the only thing needed is to position the camera to the right direction and location. This may be implemented with python scripts, although it is recommended to create it in C for better performance and chances to expand it in the future.

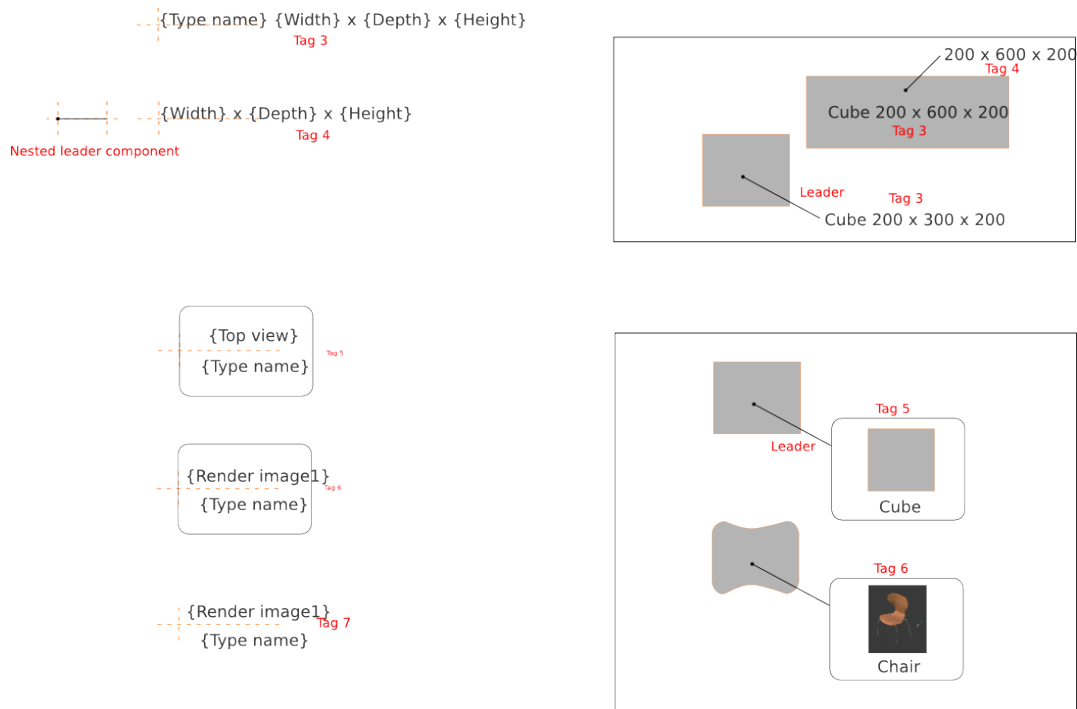


Fig 5.12. TAG component concept

2.5. *The schedule grid PO* – is almost a tag with extended features (figure 5.12). It involves displaying several parameter values for multiple POs. On the top of the grid there are parameter names displayed.

Since representing the schedule grid and its data is in fact analogous to the one for tags, there are some different aspects. The other thing is selecting multiple objects. One of the options is the scope box. It is a PO defining a certain space in which PO reside. Its functionality is believed to be similar to the existing Blender *box-select* tool. An operator for this tool named as `<editor_name>_OT_select_border` can be found in many editor modules in files named `<editor_name>_select.c`. These files should provide a good start point for implementation of the scope box, probably even adding the functionality to these files, which would be a recommended solution in order to preserve the consistency of Blender.

Another way to select objects is to use a constraining objects. It would need a particular panel and operator for selecting the constraining objects. This should be done with python scripts. Highlighting the selected constraining objects should be programmed in the *Interface* module.

In order to select only objects meeting certain criteria (filter objects), the selection functions just have to use PO-related data (e.g. parameter values) from DNA/RNA to decide whether the criteria are met and the objects have to be selected. It is also a matter of design and implementation of PO structures.

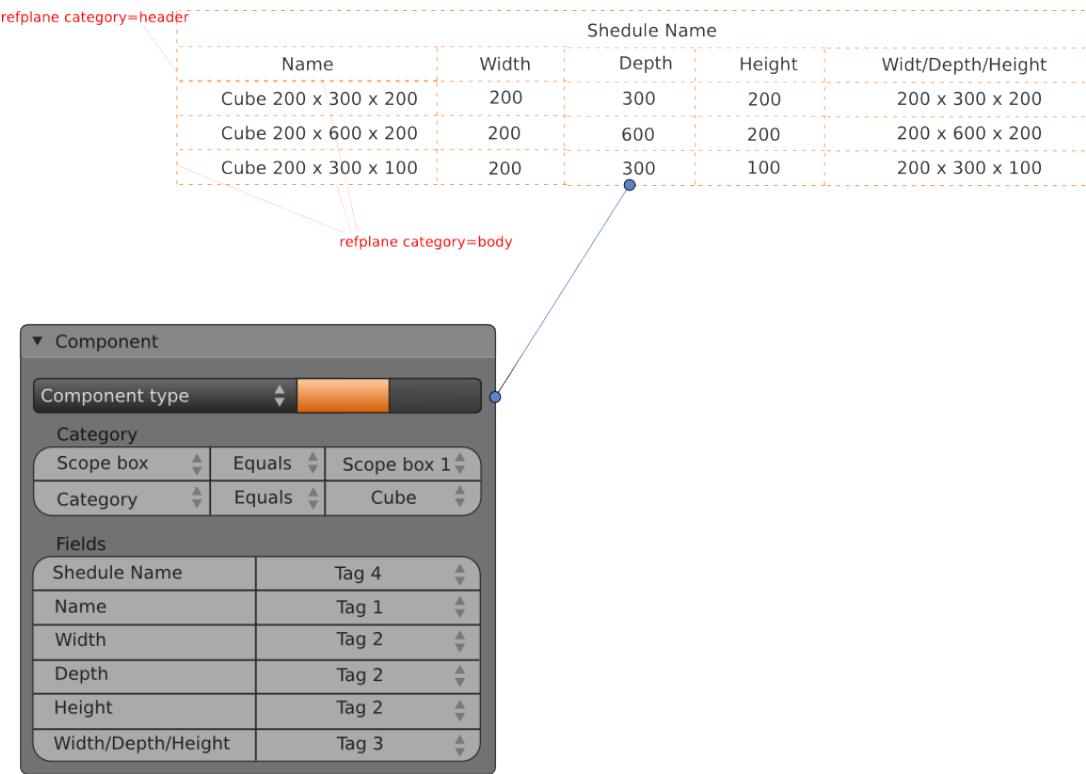


Fig 5.13. *Schedule grid component concept*

Schedule grid component with nested cell components

Cube shedule				
Name	Width	Depth	Height	Widt/Depth/Height
Cube 200 x 300 x 200	200	300	200	200 x 300 x 200
Cube 200 x 600 x 200	200	600	200	200 x 600 x 200
Cube 200 x 300 x 100	200	300	100	200 x 300 x 100

▼ Component

Cube shedule

Category

Scope box Equals Scope box 1

Category Equals Cube

Fields

Schedule Name	Tag 4
Name	Tag 1
Width	Tag 2
Depth	Tag 2
Height	Tag 2
Width/Depth/Height	Tag 3

▼ Nested component

Header cell

Cell apearence

Border line	Line 1
Background color	Gray



▼ Nested component

Body cell

Cell apearence

Border line	Line 2
Background color	White

Schedule grid component with nested cell components and tag7

Chair shedule	
Name	Picture
Chair 200 x 300 x 200	 Chair
Chair 200 x 500 x 200	 Chair

Cell components

▼ Component

Header cell

Cell apearence

Border line	Line 1
Background color	Gray

▼ Component

Body cell

Cell apearence

Border line	Line 2
Background color	White

refplane category=header

refplane category=body

Fig 5.14. Schedule grid with nested cells component concept

2.6. *Dimension* – a PO used to show distances between objects in 2D view. This object is actually another kind of a tag, only retrieving data from two (or maybe even more) parameters. Thus, implementation should be based on tags and could be integrated into the same mechanism. Distance calculations currently have minimal implementation in the mainstream version of Blender, thus will also require some work time. It involves basic mathematical calculations based on PO (refplanes mostly – explained in 5.2.2.) size, location and orientation data.

2.7 *3D and 4D PO's* 3D is a PO understandable naturally as an object that has width, height and length dimensions. 4D PO adds the fourth dimension – time. The latter is used to define object

position in a particular moment of time. Creating these objects should not differ from 2D objects. However, manipulating them, e.g. describing object movement in time, is another issue, which could utilize current Blender animation system. Since the dimension data would still be stored in DNA/RNA system, it is possible to create this functionality both as the python scripts (add-ons) and C functions in editors module.

2.8. *Architectural material* – a PO that has particular parameters and types. The material can be placed onto the object surface and also fill its volume. Thus it is possible to precalculate in the schedule grid the total quantity of the material used.

Architectural material consists of 3 main parts:

1. Material view for rendering – standard Blender material
2. Material view for drawings – a surface view and object cut view. In the drawing the material is presented using vector lines and points.
3. Physical features.

The basis of this kind of object is also very similar to standard POs, so the implementation of the PO parameter table, loading/saving involves changing the same modules (*makesdna*, *makesrna*, *blenloader*, *interface*, etc.). The new thing at this point would be the patterns for surface and cuts. It would require new UI elements from the interface module. It will also require to display patterns in the view3D and other editors. The easiest way to do it is to wrap objects with particular textures at a certain period of time (before rendering or instantly after pattern selection). Another, but more difficult way would be to improve editors, add new functions to them which will allow to draw patterns on the fly through OpenGL shaders functionality for example. In both cases changes to the editor modules would be required. It is recommended to use the first option (textures), because it will require less working time and is believed to sustain Blender consistency. For drag-n-drop and other actions functionality from basic 2D objects and tags should be taken. In fact, it could also be integrated into a single consistent PO system.

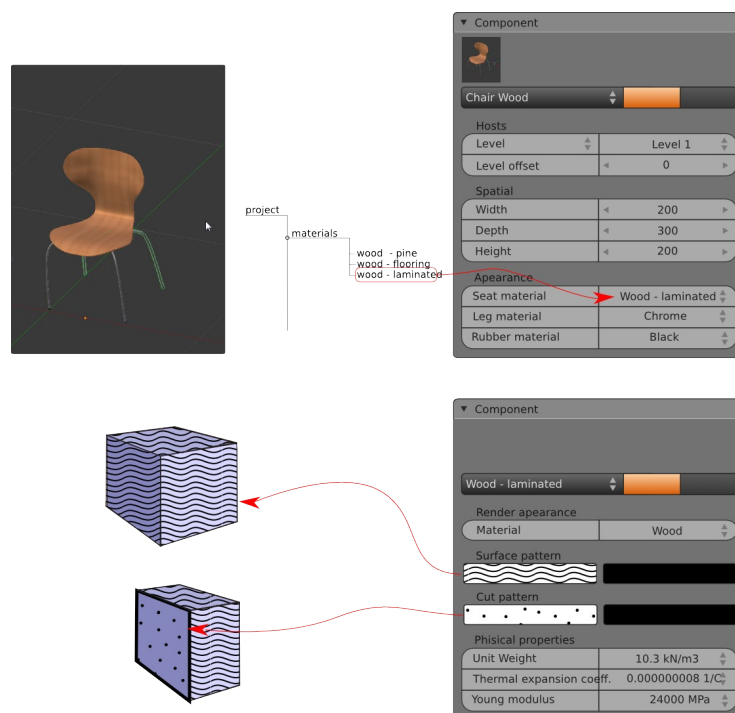


Fig 5.15. *Adaptive materials concept*

2.9. *Adaptive component* – a PO which can adapt to the geometry of other objects. The desirable features:

1. The component does not have fixed length, width and weight parameter values. Placed into a certain space constrained by other objects, such as walls, the component fills that space. Thus it is possible to find out certain information about the space: its volume, surface areas and other
2. Only the width (thickness) of the component is fixed. When placed on a surface, the component is applied to it and the other geometry is taken from this surface, e.g. a wall is painted. This allows covering objects with paint or materials and retrieving information about it.

Blender currently does not have auto-expanding or adapting to object geometry functionality. The basis of such PO should be simple to implement, since the information that is needed for component adaptation can be accessed through objects parameter values. Since these values are expected to be stored in Blender DNA, they should be easy to access through the RNA. The mechanism for analyzing the retrieved values depends on the objects. If they are generally only walls (rectangulars and other regular shapes), parameters such as height and width will be sufficient. However, if the objects, the adaptive component is applied to, are unusual, e.g. they are constructed as surfaces, the latter parameters won't be enough. It will require surface analysis methods application. From this point of view the system might be rather complex to implement, but it is believed to be useful in architectural design. This new system should be placed in `/blender/source/blender/`. Certain parts of it could also be treated as a modifier object (`/blender/source/blender/modifiers/`). Some features of this system are also common to other POs mentioned in this chapter, thus using them and probably implementing all this common functionality as a single module is a good idea.

The mainstream version possibly will not benefit from the adaptive component's functionality because calculation of volumes typically are not required in animation and 3D modelling process.

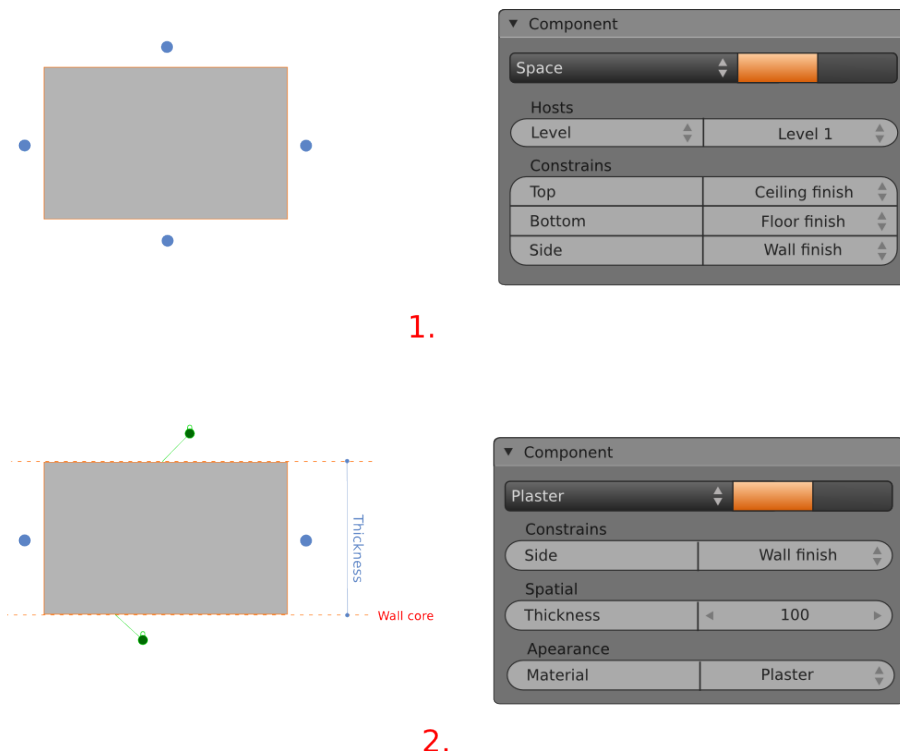


Fig 5.16. *Adaptive components concept*

3. *UI element: project tree.* UI Element that shows views, their hierarchy, POs and their types. Enables to group POs by parameter values, perform component search.

This element is similar to the *Outliner* editor which is implemented in the current, mainstream version of Blender, but has a few differences (figure ...). That the reason why the project tree element will require a new editor. As a reference it is recommended current version of *Outliner*. Drawing lines and showing the results is a matter of design, but will be basically located in the editor module for this element. Actions with POs, such as those mentioned in the introduction of this element, will mostly consist of iterating through PO lists, checking their parameter values for parents, children (also a certain list in DNA) and category values. Depending on the additional structures created, it can be differ though. To sum up, a new editor with some operators and DNA/RNA types as well as functionality for their manipulation will be needed.

Since this functionality is geared towards PO, their hierarchies and project management, it is unsure if it would be useful in the mainstream version. Only small parts and changes could be usefull.

4. *PO behavior programming ability.* This is generally covered in most sections of this chapter – tags, adaptive components and other. If any new features are needed, they should be integrated into the main PO system in `/blender/source/blender/`. Changes to notification and event system might be needed, for tracking changes of PO parameter values and change others accordingly [17].

Solutions development estimation and planning

Table 5.6. *PO components implementation estimation*

Task description	Estimated MT
Parametric object base integration	120
0D parametric components integration	60
2D parametric components integration	60
Tags	30
Schedule grid	30
Dimension	90
3D parametric	30
4D parametric	30
Parametric materials	90
Adptive component	90
Project tree	90
Total:	720

5.2.2. Setting the detail level of a parametrized object (PO)

1. *The ability to select POs level of detail parameter.* In effect this is another feature for PO behavior affected by its parameter value. Since it only involves changing visibility of PO details, it should be easy to implement this even through python scripts (plug-ins), since it can collaborate with Blender RNA. The most simple way could be to create PO hierarchy (already mentioned in the project tree), and assign a parameter (probably a flag), defining in which level which element must appear and also the current level set. In the mainstream version this could also be helpful, along with the object hierarchy feature.

2. *The ability to automatically set the level of detail for a view of a certain scale.* This feature mainly is an extension of PO level of detail feature. However, certain changes to *view3D* editor might be needed, such as processing an event of zooming, rotating or otherwise changing the view perspective. The editor could still call operators from python plug-in, but if the number of PO is large, changing the perspective might become slow. Thus it is then recommended to implement operators in the editor itself.

Solutions development estimation and planning

Table 5.7. PO detail level implementation estimation

Task description	Estimated MT
PO detail level functionality	30
Total:	30

5.2.2. PO Tools

Requirements

Indicate which tools of the latest Blender version can be used for PO creation and modification. Evaluate the convenience of these tools for a user. Blender platform extension with PO-specific tools:

1. *Reference planes – refplanes.* Refplanes are objects that enable aligning other objects to them. They are also used when setting dimension tool, adaptive component and others. They are used to indicate the substantial geometry of a PO – the center, borders and other. The refplanes enable fixing geometry of other objects to them (see figure 5.16 for illustration).

A specific host parameter in a PO should allow locating newly created objects to the object that is directed by the parameter. Thus, the object becomes a refplane.

Currently there is no known tool that has functions similar to the refplanes. Since refplanes are in fact parametrized objects, their implementation is probably going to be similar to the other POs, including supplementation of new DNA/RNA types and functions [13, 18], changes to *blenloader* and other. For further explanation see chapter 5.2.1. In order to create a refplane, a user should be provided with a certain tool. Most probably it could be a choice in a “Add Object” menu (Shift+A), but also it should be possible to convert a simple PO (e.g. plane) into a refplane. For this, certain python scripts should be created introducing new operators and menu options. For other tools to be implemented more simply, the refplanes should have a certain manager, which could quickly access the objects that are refplanes, also providing necessary information about them, such as their location, limits and other. Thus, a new module or a part of the large PO module in */blender/scripts/blender/* should be created. In order to mark refplanes differently from other objects in Outliner editor, the latter should have to be changed as well, introducing new object category and certain things related to its representation and actions (module */blender/source/blender/editors/space_outliner/*). The further details, such as which refplanes should be activated for certain objects and which shouldn't is a matter of design of the refplanes manager and therefore is not covered in this study.

The refplanes could be beneficial for mainstream Blender because it might allow work-flow optimization by aligning several objects in the scene.

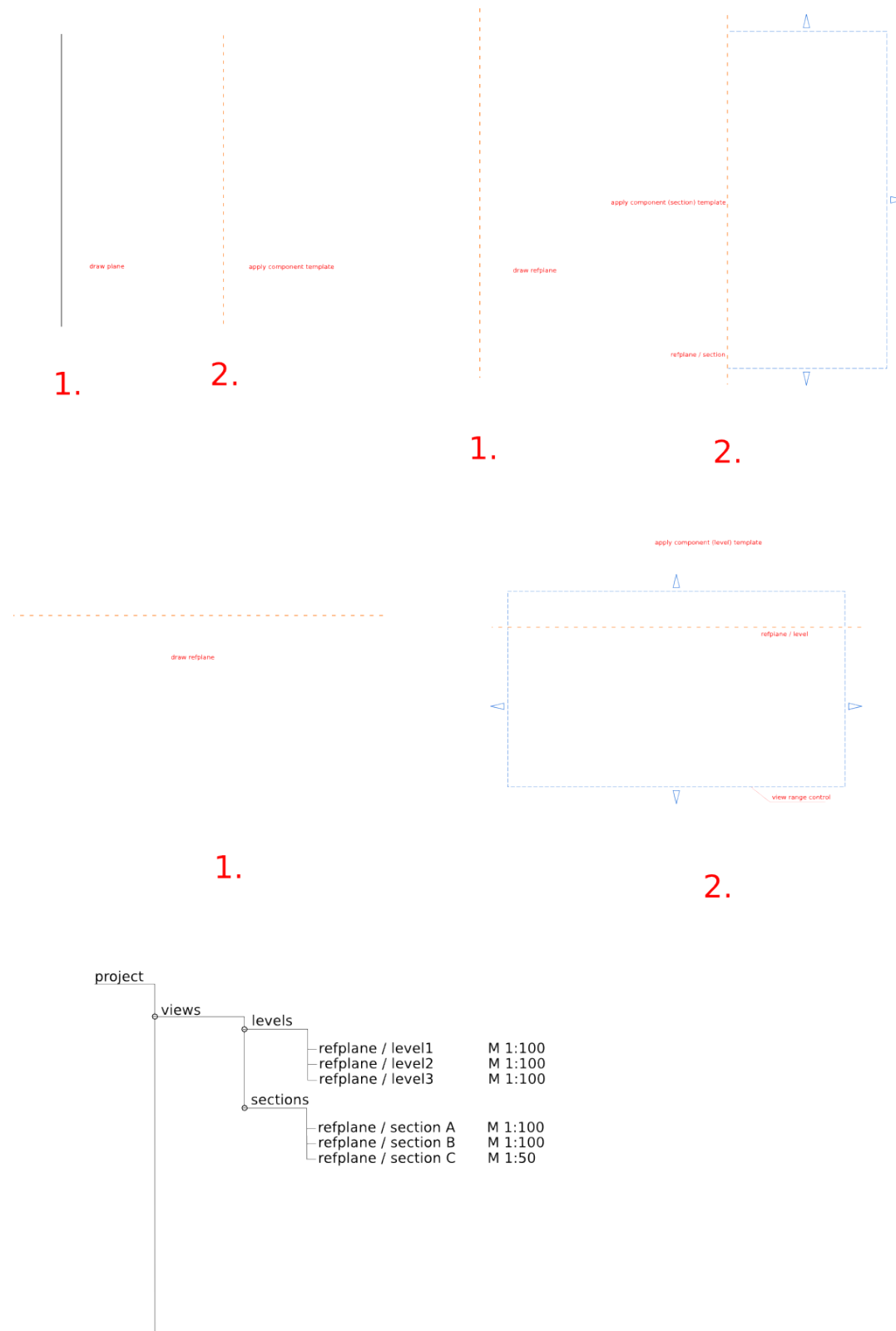


Fig 5.17. *Reference planes concept*

2. *Aligning reference planes of separate POs.* The way to implement this feature depends on the developer. In general case this functionality should reside in the refplanes manager mentioned previously. Since it is easy to reach PO data through RNA system, already known mathematical

formulas and methods should enable introducing the functionality rather easily. It is recommended to consider making reflanes independent from other POs, allowing more than one object to use it as a host. These objects should not be children to other POs, but for the whole project or a scene instead.

3. *PO Controls*. PO controls – graphical elements that enable to change PO parameters and its geometry, without entering 'edit component' mode.

Different control types are presented in figure 5.17.:

1. Control for changing the distance between reflanes
2. Control for creation of a new component as a certain component mirror view
3. Yes/No parameters
4. Same as #3
5. Control allowing to select the type of a component, residing in another component.

Edit component mode

Component loaded into project

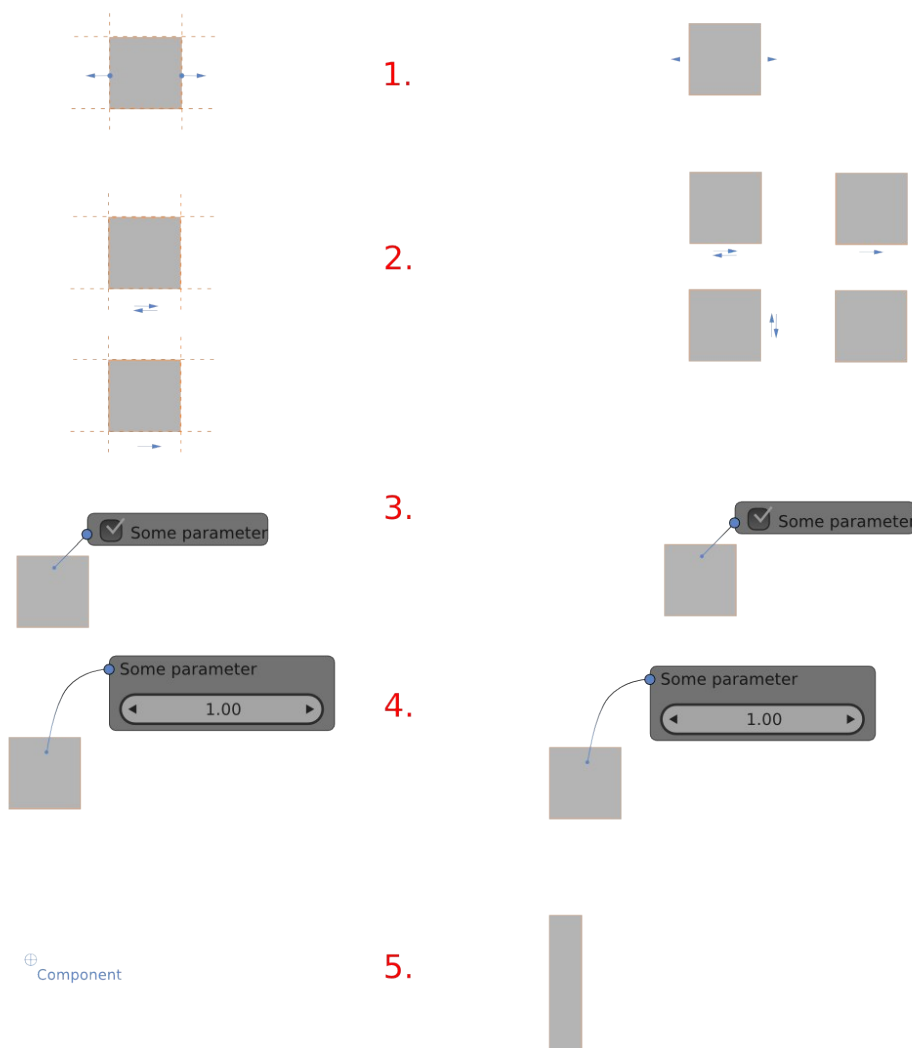


Fig 5.18. *Control types concept*

A way of using a refplane for constraining geometry and further modification with controls is presented in figure 5.18.

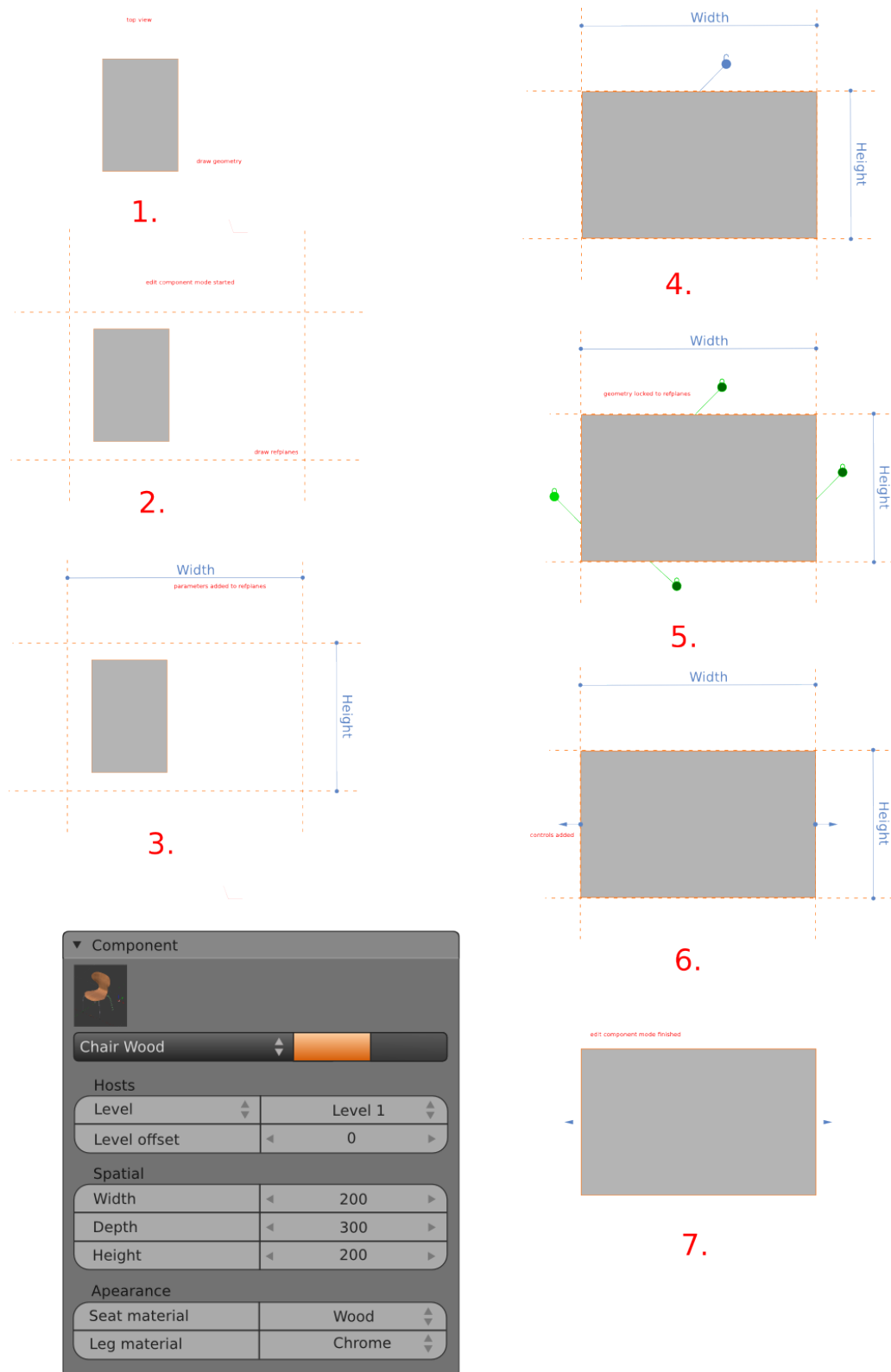


Fig 5.19. Constraining geometry concept

Steps explained:

1. Geometry is drawn
2. Refplanes are drawn
3. Refplanes are related to object's parameters
4. Geometry is fixed (locked) to the refplanes
5. same as #4
6. controls are added to refplanes
7. The component is added to the project (edit component mode is turned off). The controls allow modifying the component in an ordinary mode (not edit component mode).

As it can be seen, controls are somewhat similar to tags, although not only presenting a value, but also allowing to change it (like the PO parameter table). It is again worth noting that although Blender allows using floating UI elements in the source code, there are none in the mainstream version. These floating elements should at least be bound to a dedicated editor. Depending on the editor which would host these controls, changes should be made to them to allow new features. Interface module will have to be changed in any case. Accessing PO data should not be a problem because of the convenient RNA system, allowing to access DNA structures both from C and python code.

Since Blender currently does not have active editor elements such as the ones presented in the geometry constraining example (except for those red, green and blue direction arrows in view3D that allow moving the object or its certain parts), it might require changing many parts of Blender view3D to achieve the required functionality. Firstly, the *Ghost* system might need changes, similar to the previously mentioned drag-n-drop function improvements for tags. Secondly, interface would have to be changed for the new UI elements to be added. Changing an existing editor (such as *view3D*) would require quite a few changes. Probably the best way to start is analyzing the functionality of the arrow keys mentioned.

Such a functionality is hardly usable in the mainstream version of Blender and is related to architectural design only. It may be quite problematic to implement at the first view. Secondly in complex scenes it may provide additional visual overload. Thus it is recommended that controls should be integrated into panels and menus as much as possible.

Solutions development estimation and planning

Table 5.8. PO Tools implementation estimation

Task description	Estimated MT
Reference planes	60
PO controls	30
Constraining geometry	60
Total:	150

5.2.3. Tools for architectural designing

Requirements

Architectural design-specific functionality:

1. *2D views generation from a 3D model* – cuts and plans These generated 2D views should also be POs of a certain category.

Although these features might need no additional editors, it would still need at least python scripting for input of parameters for 2D view and also functions for generation. If the whole PO system is design will be good enough, it should be easy to access various information about POs, such as its hierarchy, location and other, thereby reducing the complexity of the latter functions.

Figure 5.19. and figure 5.20. present a use case of a space (editor) for 2D views generation and layout management. It is called paper space in this example. In both pictures the Properties/Project browser is displayed. As it can be seen, the browser consists of the Project tree and PO parameter table tools. Since they have already been discussed, the only recommendation for this browser is to use separate spaces for separate tools. The editors can still communicate with each other via events and notifications system, just like view3D and UV/Image editor do when working with texture wrapping. Such an implementation will preserve the consistency of the structure and work-flow with Blender.

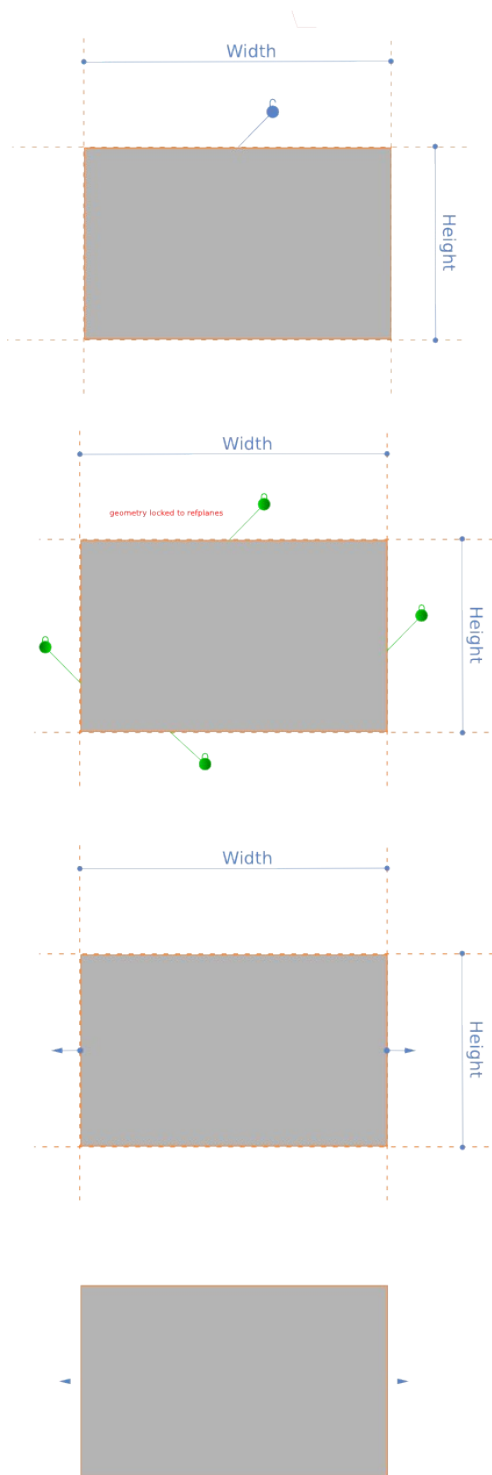
As for the Paper space, it is desired to consist of one or more unlimited paper sheets. In these sheets 2D views should be laid. On the border of the paper certain stamp components should be placed. When exporting the paper to .pdf or other formats only areas constrained by the border should be exported. When a single spread contains several paper border components, they are exported as a .pdf document consisting of multiple pages.

Since the 2D views mentioned are PO's, their management should cause least problems when implementing the functionality. A new editor and several new DNA/RNA types will be required to implement for Paper space. Several new UI elements might be required as well, such as for templates of the stamp mentioned previously. Stamps could also have a separate manager – for their creation, selection, template management and other tasks. This might also require a new editor. The interaction between the Properties/Project browser and the Paper space can and should be implemented through the events and notification system.

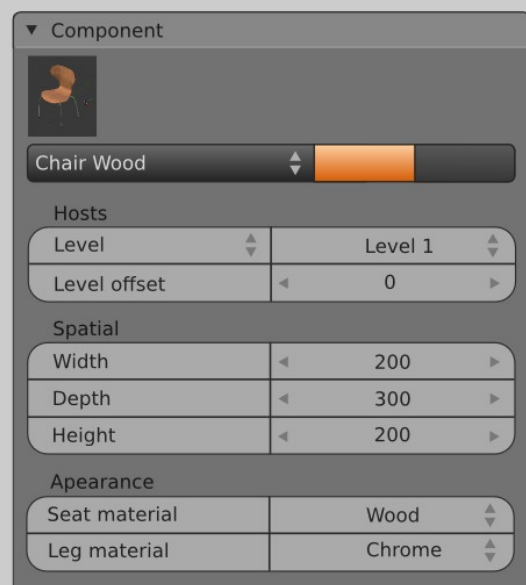
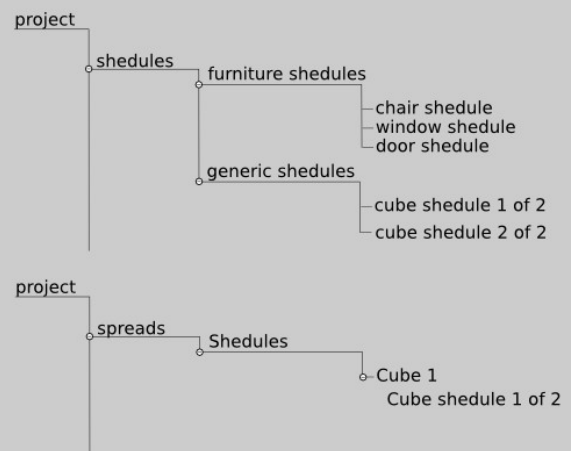
Although it is required to have unlimited space for the layout, technically it would still be limited. The possible solution is to start with a certain minimal space and then increase its size if needed. The Paper space's editor should provide these features. However, this highly depends on editor design.

For export features a certain exporter in python could be implemented. Existing exporters should also provide some help. They are located in `/blender/release/scripts/addons`. The selection of which objects should be exported could be implemented similarly to current Blenders box-select tool (see schedule grid in chapter 5.2.1.)

The introduction of this functionality should not corrupt the structure of Blender. However, it is probably not needed in the mainstream version currently, although in significantly large commercial projects it could be suitable as a documentation tool.

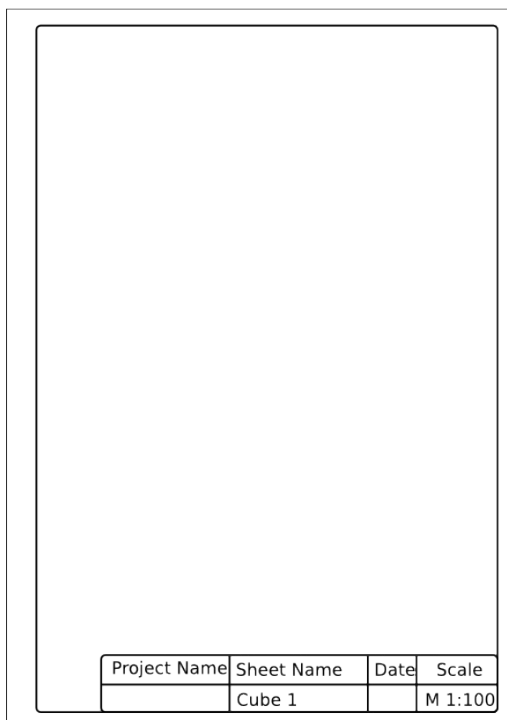
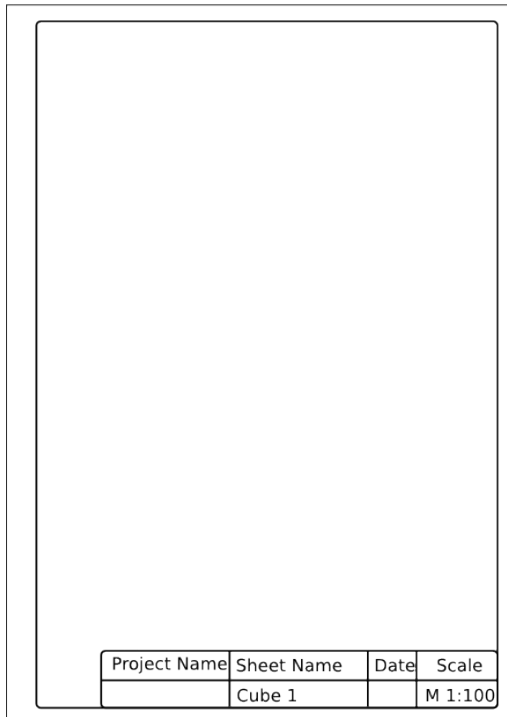


Model space

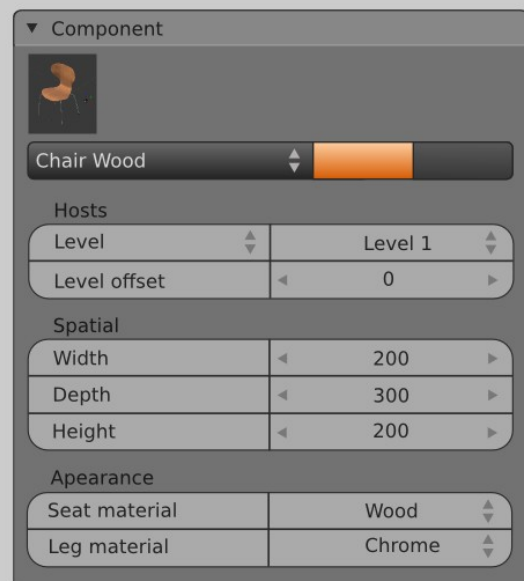
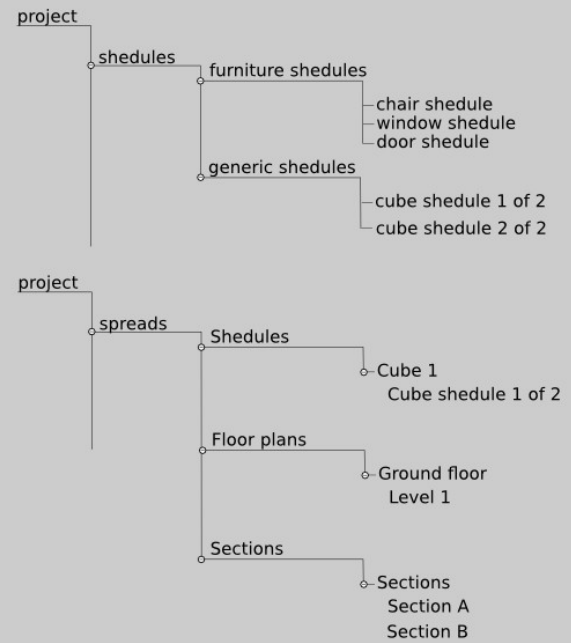


Properties/Project browser

Fig 5.20. Model cuts concept



Paper space



Properties/Project browser

Fig 5.21. *Paper space concept*

2. *Reports generation.* The abilities to generate tables that present parameter values and quantities of PO's in the project and to select PO's, constrained by other PO's are desired.

This feature is actually the same as the schedule grids already described in 5.2.1. The tables containing parameter values and component counts includes scanning the structure of POs and extracting the required data. The majority of the needed interface will be already covered in schedule grid. The only supplemental feature is displaying the table in a Paper space and the ability to customize its position in the sheet. However, since the table would be a PO, there would be no problem moving it around the sheet and the Paper space editor should provide universal tools for 2D items, including these tables, manipulation, such as ordering or rotating. The reports and tables examples are presented in figures 5.21-5.23. How easy these features could be implemented in the same Paper space editor depends on PO modules design of course.

Selecting particular objects, which reside in other objects has already been covered previously. One alternative is to use PO hierarchy and Project tree to find which objects are inside the others. The other alternative is to use space examination features. The first option is similar to that used in schedule grids – using the box-selection tool. The second is to use features of collision detection (*/blender/source/blender/blenkernel/BKE_collision.h* and related). This would also mean that certain data from the POs should be available for access in order to decide whether an object is inside the other objects, intersects its certain faces or is outside an object. It is recommended to use the first option if possible, since it would provide the most logically understandable selection – by a room, a house, etc.

As well as for Paper space, this functionality for mainstream Blender might be necessary only in serious commercial projects, since then documentation tools could decrease the projects cost and time needed.

▼ Component

Box schedule

Category

Scope box

Equals

Scope box 1

Category

Equals

Cube

Fields

Schedule Name

Tag 4

No

Tag 3

Name

Tag 1

▼ Nested component

Header cell

Cell appearance

Border line

Line 1

Background color

Gray

▼ Nested component

Body cell

Cell appearance

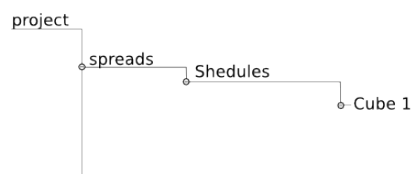
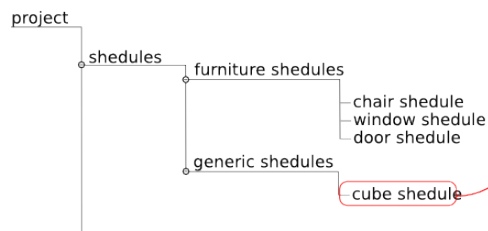
Border line

Line 2

Background color

White

Cube shedule	
No	Name
1	Cube 200 x 300 x 200
2	Cube 200 x 600 x 200
3	Cube 200 x 300 x 100
4	Cube 200 x 300 x 200
5	Cube 200 x 600 x 200
6	Cube 200 x 300 x 100



Cube shedule	
No	Name
1	Cube 200 x 300 x 200
2	Cube 200 x 600 x 200
3	Cube 200 x 300 x 100
4	Cube 200 x 300 x 200
5	Cube 200 x 600 x 200
6	Cube 200 x 300 x 100

Project Name	Sheet Name	Date	Scale
	Cube 1		M 1:100

Fig 5.22. Schedule formatting concepts 1

Cube shedule	
No	Name
1	Cube 200 x 300 x 200
2	Cube 200 x 600 x 200
3	Cube 200 x 300 x 100
4	Cube 200 x 300 x 200
5	Cube 200 x 600 x 200
6	Cube 200 x 300 x 100
7	Cube 200 x 300 x 100
8	Cube 200 x 300 x 200
9	Cube 200 x 600 x 200
10	Cube 200 x 300 x 100
11	Cube 200 x 300 x 100
12	Cube 200 x 300 x 200
13	Cube 200 x 600 x 200
14	Cube 200 x 300 x 100
Project Name	Sheet Name
16	Cube 200 x 300 x 100
17	Cube 200 x 600 x 200
18	Cube 200 x 300 x 100
19	Cube 200 x 300 x 100

Cube shedule	
No	Name
1	Cube 200 x 300 x 200
2	Cube 200 x 600 x 200
3	Cube 200 x 300 x 100
4	Cube 200 x 300 x 200
5	Cube 200 x 600 x 200
6	Cube 200 x 300 x 100
7	Cube 200 x 300 x 100
8	Cube 200 x 300 x 200
9	Cube 200 x 600 x 200
10	Cube 200 x 300 x 100
Project Name	Sheet Name
	Cube 1
Date	Scale
	M 1:100

▼ Component

Box schedule

Category

Scope box

Equals

Scope box 1

Category

Equals

Cube

Fields

Schedule Name

Tag 4

No

Tag 3

Name

Tag 1

Schedule break

Break schedule at row

10

Cube shedule	
No	Name
11	Cube 200 x 300 x 100
12	Cube 200 x 300 x 200
13	Cube 200 x 600 x 200
14	Cube 200 x 300 x 100
15	Cube 200 x 300 x 100
16	Cube 200 x 300 x 200
17	Cube 200 x 600 x 200
18	Cube 200 x 300 x 100
19	Cube 200 x 300 x 100

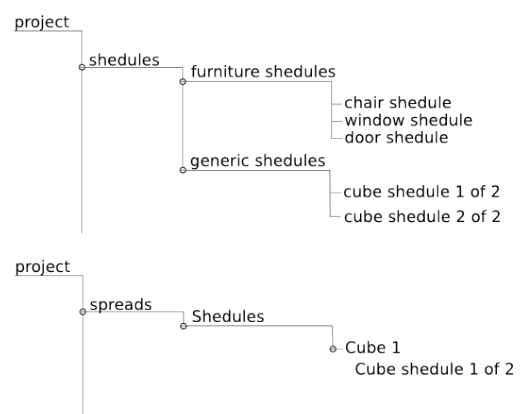
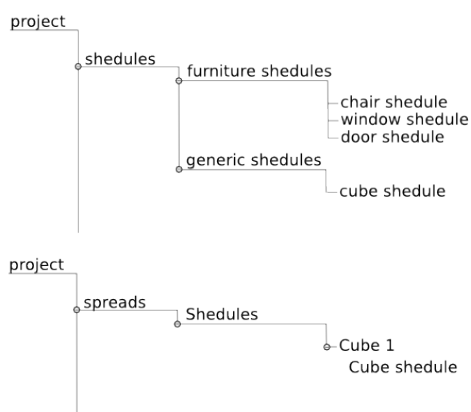
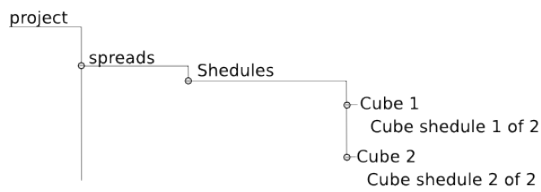
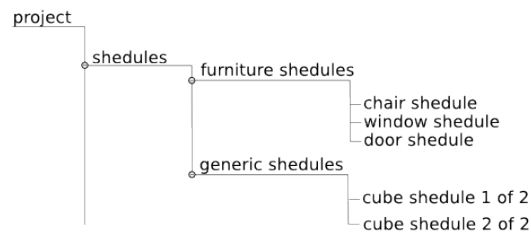


Fig 5.23. Schedule formatting concepts 2



Cube shedule	
No	Name
1	Cube 200 x 300 x 200
2	Cube 200 x 600 x 200
3	Cube 200 x 300 x 100
4	Cube 200 x 300 x 200
5	Cube 200 x 600 x 200
6	Cube 200 x 300 x 100
7	Cube 200 x 300 x 100
8	Cube 200 x 300 x 200
9	Cube 200 x 600 x 200
10	Cube 200 x 300 x 100

Project Name	Sheet Name	Date	Scale
	Cube 1		M 1:100

Cube shedule	
No	Name
11	Cube 200 x 300 x 100
12	Cube 200 x 300 x 200
13	Cube 200 x 600 x 200
14	Cube 200 x 300 x 100
15	Cube 200 x 300 x 100
16	Cube 200 x 300 x 200
17	Cube 200 x 600 x 200
18	Cube 200 x 300 x 100
19	Cube 200 x 300 x 100

Project Name	Sheet Name	Date	Scale
	Cube 2		M 1:100

Fig 5.24. Schedule formatting concepts 3

3. Arrangement management of 2D objects in sheets.

Drawings formation – variant A:

1. Ability to insert several views into the paper space
2. Ability to change the distance between them.

Inserting views into paper space should be no different from inserting objects into view3D space, such as using Shift+A or simple drag-n-drop from the view3D. Thus, when implementing the

functionality it would be wise to look at the similar implementation in the latter editor. It would be convenient to insert POs by just dragging and dropping them into the editor from the Project tree. This would require extending the functionality for events (*Ghost system*), already described in tag implementation suggestions and other previously explained components.

The second feature could also use the extended *Ghost system* for moving views in the paper space. Since the example of this feature includes dimension components, it is necessary to send notifications about changed positions, to which dimension POs should react by recalculating the distance. Thus, new event and notification types might be needed. In general, this and the further functionality should be implemented in `/blender/source/blender/editors` as a paper space editor.

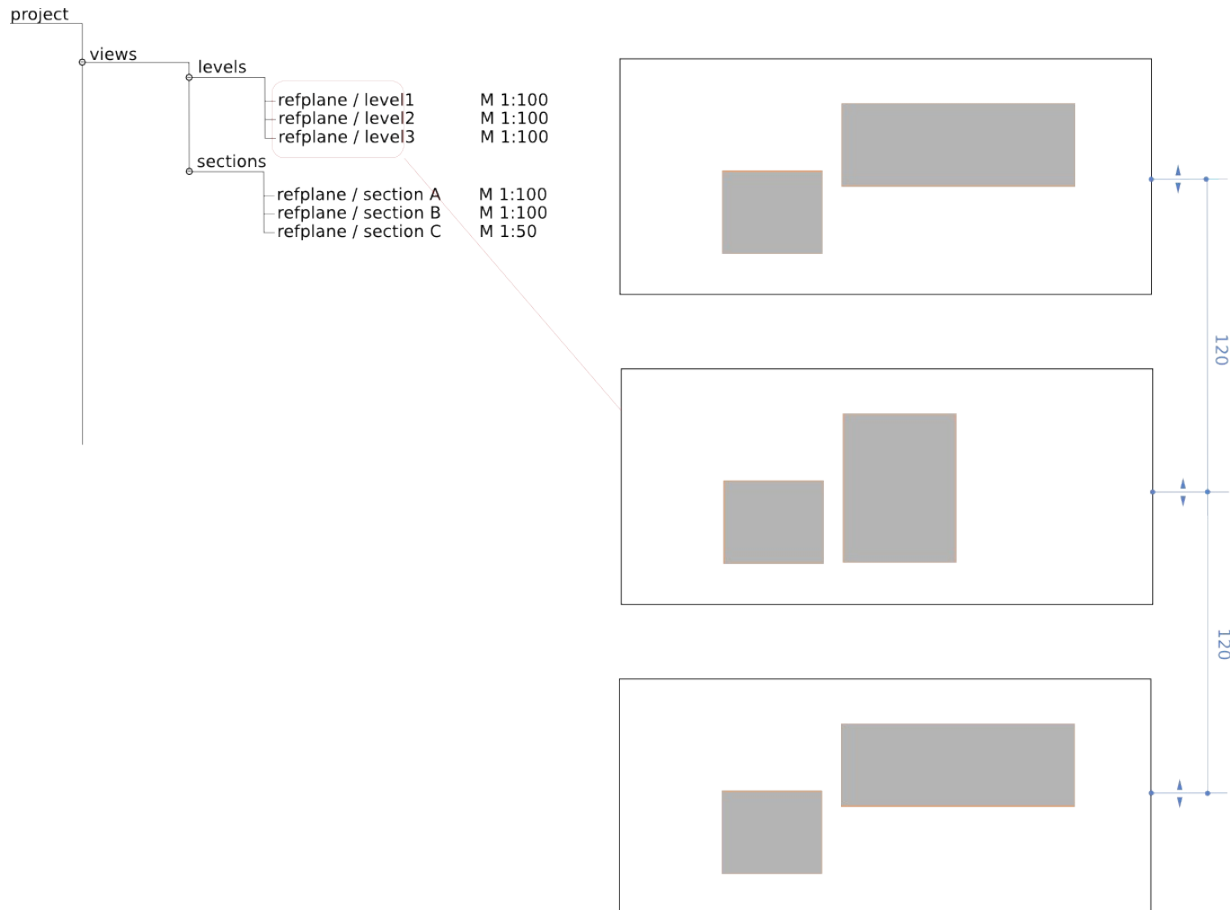


Fig 5.25. *Layout engine concept*

Drawings formation – variant B:

1. Creating a schedule grid;
2. A tag in the table's cell can represent parameter textual or digital value, as well as it's view
3. The table is divided into separate lines, each of which becomes a separate page border's component. Thus, creating a new view that meets the criteria of a table's filter the table will be increased, adding a new page if necessary. The page will be removed on view's deletion if necessary.

To preserve the consistency of Blenders usage, the creation of the views table could be performed either by a hotkey combination (Shift+A) or by a certain manager tools and panels. As mentioned before, it should be implemented in the similar way like tools for Paper space editor. The

panels should be implemented with python, adding new UI elements in *Interface module* where C language will be needed. Moving the table should be allowed in the same way and actually using the same mechanism as for the 2D views. Further details depend on the Paper editor design and implementation.

The tag component mentioned here has been already discussed in 5.2.1 and as it was said, is similar to the schedule grid and thereby the views table.

Making each line separate and managing them this way is a logical and common solution helping to make further system design process as simple as possible. Calculating the elements total size in a page and deciding whether a new page is required to put the items that don't fit in the existing page is a design issue both for the paper space editor and the export script.

See layout examples figures 5.26. – 5.27. for explanation

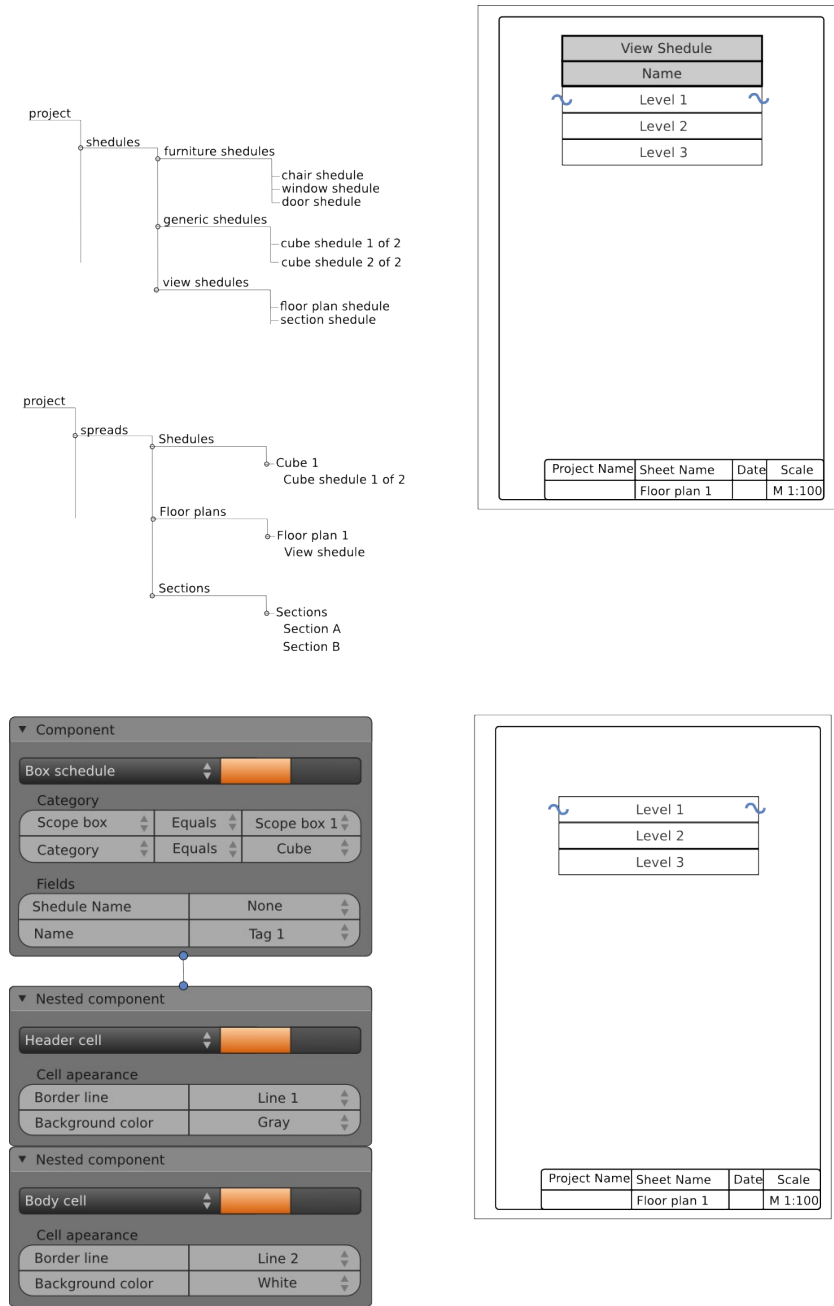


Fig 5.26. Layout concept 1

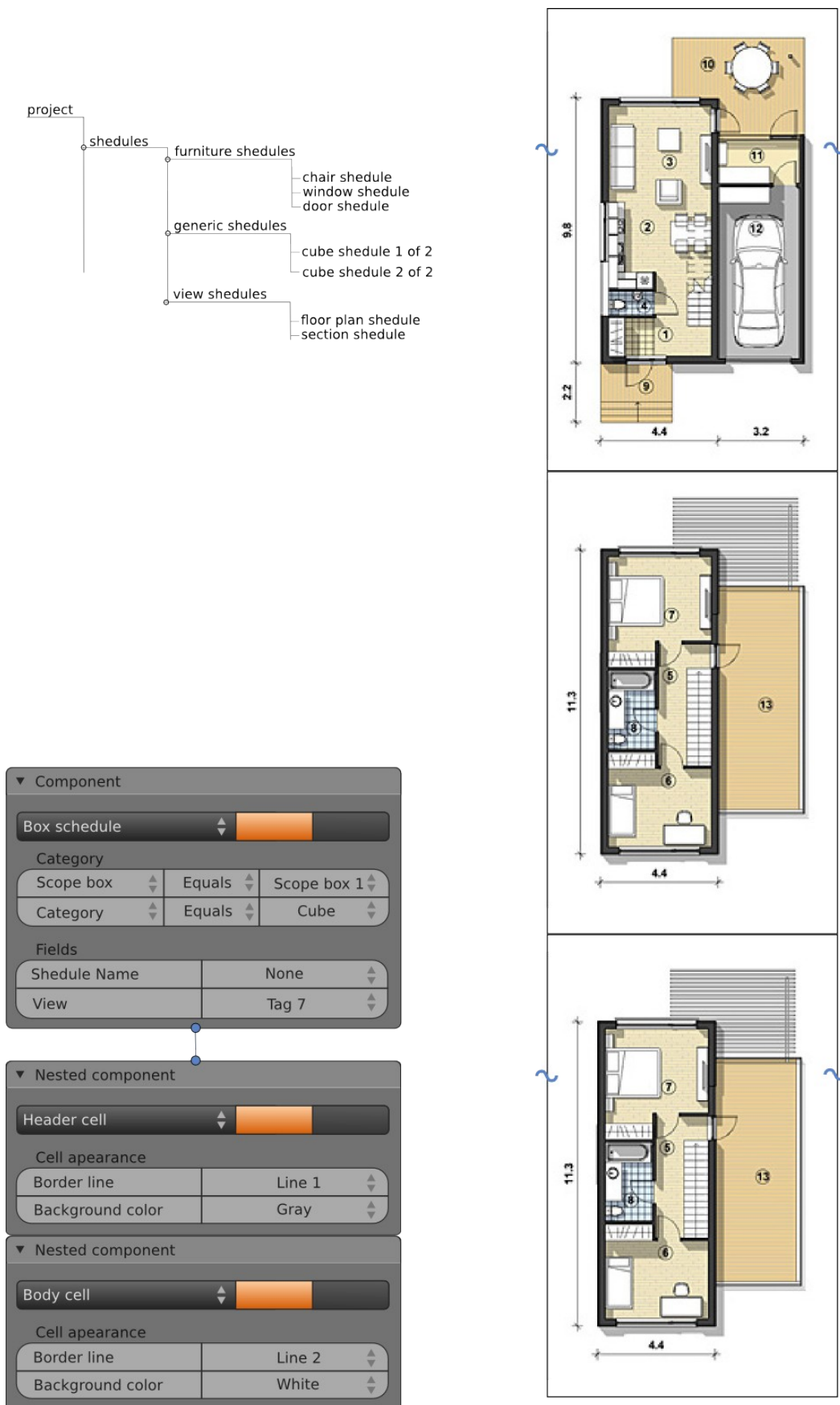
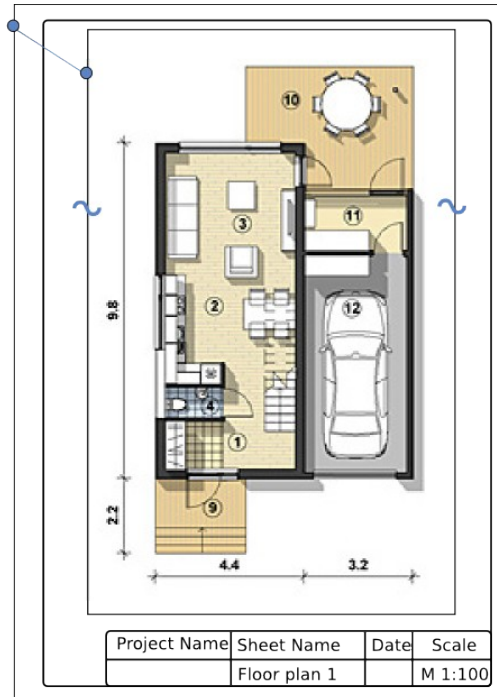
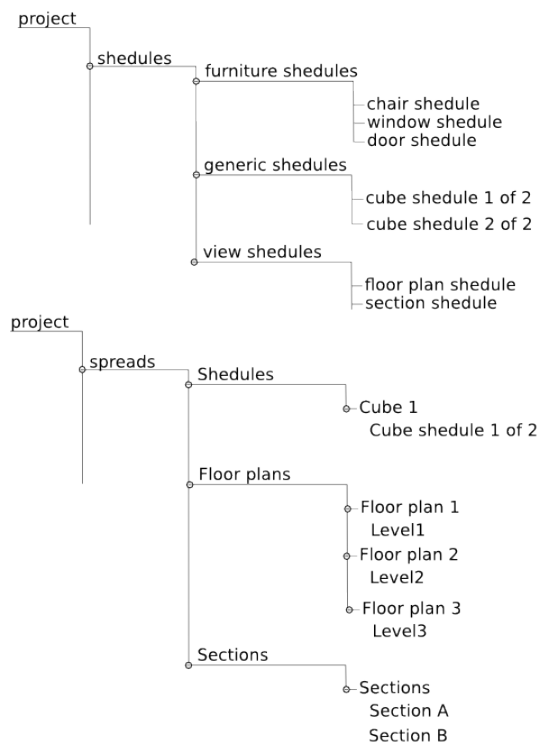


Fig 5.27. *Layout concept 2*



▼ Component

Box schedule

Category

Scope box	Equals	Scope box 1
Category	Equals	Cube

Fields

Schedule Name	None
View	Tag 7

Schedule break

Break schedule at row	1
-----------------------	---

▼ Nested component

Header cell

Cell appearance

Border line	Line 1
Background color	Gray

▼ Nested component

Body cell

Cell appearance

Border line	Line 2
Background color	White

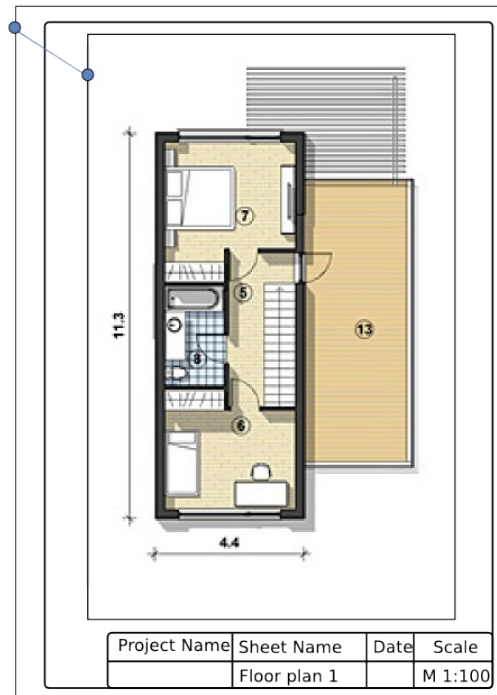


Fig 5.28. Layout concept 3

4. *Exporting sheets to other formats such as PDF, DWG.* This functionality should be similar to exporting ordinary Blender files to other formats. However, there are currently neither PDF nor DWG format exporters, although there is an importer for PDF. Since the specification for both formats is freely available, creating python exporters should be rather easy, despite the time needed to analyze the specifications. As mentioned before, PDF python importer and DXF python importer and exporter should provide the substantial amount of implementation example for similar functionality. Periodical updates for DWG importer/exporter might be needed, in order to provide the ability to export to the latest DWG version.

5. *Graphic marker tool.* Graphic marker tool is a PO that enables showing components parameter values in views. In its intended purpose it is very similar to previously discussed tag component and PO control, only without the ability to change the values. It is recommended to associate these components, extracting the common functionality into a separate module (this functionality would be needed by schedule grid components as well). The appearance features and a way of usage for this component can be customizable, but it could still remain a PO, which would obey the same rules as other POs. To achieve the desirable appearance and the way of usage the *interface* module along with the *window manager* and probably *Ghost* system could have to be supplemented by the new features.

6. *Dimension tool.* Tool that enables one to set dimensions in views. See chapter 5.2.1. for more detail explanation.

Solutions development estimation and planning

Table 5.9. Estimation of tools for architectural design implementation

Task description	Estimated MT
2D projections generation	180
Reports generation	120
Layouting	60
PDF, DWG export	60
Total:	420

5.3. Parametric objects distribution and synchronization

This chapter presents conditions and recommendations for the development of the systems, in order to facilitate the exchange of pre-made parametrized objects among users. It is important to consider what technical means should be included in the systems that is going to be improved including features for teamwork, collaboration via a special server, which would help managing modifications of commonly used objects and would provide functionality for revision control in the object library. The interface to server should probably include SOAP/XML based web-service to manage 3D models in models library and a web-based front-end to display the model information in a web-page.

5.3.1. Server platform and architecture

Requirements

Find the optimal architecture and platform for the server.

Alternative solutions

There are many ways of combining various architectures and platforms. When choosing the right one, it is important to analyze the purpose of the server, i.e. the functionality it will have to provide. Since in the current case the server is expected to store and manage parametrized objects, it should provide functions required for a database server (with DBMS). Thus, database server architecture is believed to be the optimal choice. As a possible solution for the architecture might be Mimer SQL [68]. According to its developers, it is truly multi-threaded, with requests being dynamically allocated to the different request threads. As threads scale very well over multiple CPUs, Mimer SQL is well suited for symmetric multiprocessor (SMP) environments. By the use of threads within the Database Server, optimal efficiency is achieved when context-switching in the Database Server. The scheme of the architecture is presented in figure 5.28.

It is worth considering recommendations for a workstation running Graphisoft's BIM server [34]. According to the developer, the server software was designed to run on any standard CAD workstation (32 bit systems) with at least 4GB RAM. However, if a larger number of decent-sized projects is going to be managed within the server, it is advised to use 64 bit system with at least 8 GB RAM and even more if possible. It should also be multiprocessor computer with at least two CPUs. The computer should not run any other server applications, such as mail server. It is recommended to follow these instructions in order to maintain speed, stability and management of the models server and the system itself.

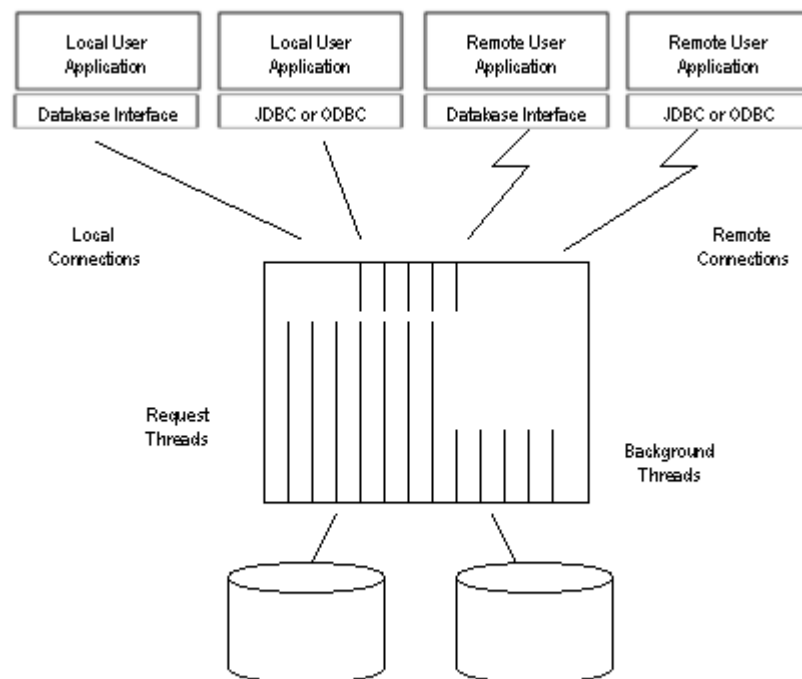


Fig 5.29. *Mimer SQL database server architecture*

As for the platform (OS) there are quite a few options. Choices range from free to expensive systems. Since the overall system (application clients and the server) is going to be implemented as a client/server model, the platform, as well as the architecture, of the server does not affect application clients. The most popular operating systems and their features are presented in the table below [23, 46, 47, 52, 57].

Table 5.10. Most popular server OS features

Platform	Windows Server	Red Hat Enterprise Linux	CentOS (Linux)	Ubuntu Server (Linux)	Solaris (Unix)
Latest version	2008 R2	6.1	6.0	11.04	11
Platform support	IA-32 (until R2, excluding it), x86-64, IA-64	x86, x86-64, IA-64, POWER, S/390, z/Architecture	x86, x86-64	x86, AMD64	SPARC, IA-32, x86-64, PowerPC (Solaris 2.5.1 only)
Approximate prices	\$1,029 for Standard \$2,999 for Datacenter \$3,919 for Enterprise	Various: \$799 (Standard 1 year service, up to 2 sockets, 5x12 support) \$2,499 (Premium 1 year service, unlimited sockets, 7x24 support)	free	free	Various: \$720 (Standard 1 year service, up to 2 sockets, 5x12 support) \$1,980 (Premium 1 year service, unlimited sockets, 7x24 support)
RAM limit	2008 Standard: 4GB(32 bit) / 32GB(64 bit) 2008 Enterprise and Datacenter: 64GB (32 bit) / 1TB (64 bit) 2008 R2 Standard: 32GB (64 bit) 2008 R2 Enterprise and Datacenter: 2TB (64 bit)	16GB (x86) 2TB/64TB(x86_64)	16GB (x86) 2TB (x86_64)	64GB (32 bit with PAE) 1TB (64 bit)	4 TB (for Solaris 10)
File size limit	16 TB (NTFS)	2TB (Ext3), 16TB (Ext4), 100TB (GFS2)	2TB (Ext3), 16TB (Ext4)	8TB (Ext3), 16TB (Ext4)	16 EB (for Solaris 10)
Maximum file system size	256 TB (NTFS)	16TB (Ext3), 16TB (Ext4), 100TB (GFS2)	16TB (Ext3), 16TB (Ext4)	16TB (Ext3), 16TB (Ext4)	16 EB (for Solaris 10)
Maximum logical CPUs	256	32 (x86), 160/4096 (x86_64)	32 (x86), 64/255 (x86_64) for CentOS 5	32 (x86), 64 (AMD64)	512 (for Solaris 10)

Note from the Windows web-page: Devices have to map their memory below 4 GB for compatibility with non-PAE-aware Windows releases. Therefore, if the system has 4GB of RAM, some of it is either disabled or is remapped above 4GB by the BIOS. If the memory is remapped, X64 Windows can use this memory. X86 client versions of Windows don't support physical memory above the 4GB mark, so they can't access these remapped regions. Any X64 Windows or X86 Server release can.

The table presents only several popular server operating systems. Although choosing the right option depends on the business plan, the company staff, it is recommended to narrow the range of choices to Linux/Unix based OS. They provide stability, performance, hot-swapping, open source advantages and other features. Besides, they are widely used in server and mainframe fields. In web-servers, currently the most popular is CentOS (about 30% of web-servers).

In mainframes inside companies, Unix distributions are used mostly. Unix systems, in our example – Solaris, usually provide abilities to process bigger files, manage a larger number of CPU threads and generally is created for the highest and continuous workloads (for large corporations, such as banks). However, these distributions usually come for price, thus it is worth considering whether there will be such workloads and how frequently they will occur.

The server for exchanging objects and managing projects is not expected to experience such intensive loads, thus it is recommended to deploy one of the Linux distributions. At least starting with them is a good way to examine the market, the popularity of the service and the need for expensive software. It is important to note that Linux systems can perform rather resource and memory hungry tasks quickly and smoothly. Both CentOS and Ubuntu Server are recommended for use. These also provides professional consulting and community sites that offer free support for all sorts of issues. Red Hat Enterprise Linux is a quality commercial product and is suitable for businesses. It also provides online support.

Solutions development estimation and planning

Table 5.11. Estimation of server platform and architecture design

Task description	Estimated MT
Server platform and architecture design	60
Total:	60

5.3.2. Model exchange protocol requirements

Requirements

Present the requirements for model exchange protocol.

Currents situation and solutions

Currently there are no public available protocols specifications used by models distribution servers due to commercial nature of such services. However OpenDWG or STEP could be used as basis for such protocol design. Standard for the Exchange of Product also know as “STEP” is an ISO standart (ISO10303) for computer-interpretable representation and exchange of product manufacturing information.

Recommended solutions

Such a protocol highly depends on parametric objects data structures design and implementation in Blender's RNA so it is recommended to develop it's design during Blender Architect design process. As mentioned before – OpenDWG and STEP specifications could be used as basis.

5.3.3. Model exchange portal requirements

Requirements

Present the requirements for model exchange portal.

Currents situation and solutions

Currently Blender does not provide such a portal.

Alternative solutions

There are a number of 3d objects exchange portals in the market. The biggest of them is Turbosquid.com [66]. In this website can be found not only free and commercial 3D models, but also things related to them, including textures, 3ds Max Plug-ins, 3d tutorials and more. Every material can be presented with optional files, such as low-res or high-res renderings, texture previews, videos and more. All models are sorted into categories. To find a desired model more quickly, search functionality is provided within the system. There are 3d products available for 3ds Max, Maya, Revit, Mixamo, Softimage, Photoshop and more. There is a turbosquid artist team, which offers rendering, rigging, modeling and other services for a price. Independent artists can also sell their 3d models and other material in the site. A conversion option, which is free for rather simple models, is also provided in the system. Downloading any object requires registering and logging in first. Analogical functionality for finding and downloading 3d materials is provided in exchange3d.com [26].

Another website, 3dmodelfree.com [1] offers a library of free 3d models, textures and AutoCAD blocks. There is no registration required, thus downloading a file basically selecting one and choosing the download option. Finding a suitable object is quite easy because of the categorization of all the objects. There is also an integrated Google search option, although the list of Google results for objects, displays no images of objects and is not as convenient as it could be. However, the results point directly to respective category, thus no scanning through category list is needed. Most of the models are available for 3ds Max only, but some are also provided in other formats.

Recommended solutions

The main parts for the portal functionality are explained further. The first one is database management system. This piece of functionality includes storage of objects, providing results by categories and performing custom search. Since models are expected to include version data, finding and listing different model versions is also required.

The second part of portal functionality should be user management. This includes user registration, authentication, creation of user groups/categories. Since 3d material is expected to come for price, a certain collaboration with banking institutions is also required. Interacting with banking systems requires extra precaution and security features on the system. Preview mode of

uploaded 3d material is also advised considering. This could be done like in turbosquid.com website, which could be basically presenting images, videos or other files that artists created and uploaded together with the materials for sale.

A forum, where artists and clients could communicate is also required. Along with public area, where every registered user could post and read messages, private sections could be also used for private groups, whose members would be identified by their categories or other identification fields.

It is worth considering integrating the portal into the Blender itself, thus allowing to browse, download and upload models in Blender.

Hosting the portal might cause some technical issues. Transferring and storing the objects as well as processing queries, and video streaming might need sufficient hardware resources, including fast internet connection, storage space and high-performance processors.

Solutions development estimation and planning

Table 5.12. Estimation of model exchange portal implementation

Task description	Estimated MT
Models exchange portal	300
Total:	300

5.3.4. External models integration in Blender

Requirements

Evaluate the ability to integrate external models in Blender.

Currents situation and solutions

At present, Blender does not have functionality for managing external models added to a scene. Thereby, there is no system to track changes and get updates from a remote library where the model is taken from. The only similar option is file linking. It generally means keeping organizing and modifying objects in separate files. More information about file linking as an option for teamwork management can be found in chapter 5.3.8.

Alternative solutions

No functionality allowing to track versions and changes in an external library was found in ArchiCAD or Revit. As a similar, but still different option in ArchiCAD is model comparison feature, when several files (different versions) are explicitly provided to the manager. Then it can show changes, deleted parts, etc.

Revit also has a tool for models and drawings comparison. They have to be presented explicitly. The comparer displays which parts are equal, which ones are different or non-existing.

Recommended solutions

Currently there are no existing server that could be used for required functionality. Thus, probably the only way is to write system from scratch. In Blender it will be needed to perform

minor changes in DNA and RNA (*/blender/source/blender/makesdna* and */blender/source/blender/makesrna* modules), introducing new attributes to objects for version and parent external library address, which will potentially provide updates or changes. ID field might also be needed to identify the type of an object. According to these changes, modifications to file loader will be needed in */blender/source/blender/blenloader* module. A new module for managing the external models and tracking their versions will be also required. It should probably reside in */blender/source/blender/*. A new editor might be needed (*/blender/source/blender/editors/* module), allowing user to choose the file of a necessary version and download it instantly. However, this functionality might be implemented by creating certain python add-ons for menus and panels. In most cases it will be sufficient, although this might limit the potential of further development of this system.

The server side - models library - should be kept and managed in a server. Models can take a lot of space, thus downloading or uploading them into the server might require a decent bandwidth. The server should work as a database and a DBMS, providing information about available models, their versions and other relevant information.

The client-server part should not be difficult to implement in Blender, although its potential in mainstream version is questionable, since it does not have unified models, models are created locally and usually have short-term significance. However it may fit well for production studios where teamwork solutions are required. The technical part would be rather expensive due to the required bandwidth and storage resources.

Solutions development estimation and planning

Table 5.13. Estimation of external models integration

Task description	Estimated MT
External models integration	90
Total:	90

5.3.5. Model exchange in P2P method

Requirements

Evaluate the ability to exchange models in P2P approach.

Current situation and solutions

Currently there are no P2P implementation in Blender. However, .blend files can be shared in P2P method just like any other files, such as E2DK. Other communication technologies, e.g. IRC and Skype, also provide a way to exchange all sorts of files, including .blend files.

Alternative solutions

There were no BIM applications found that use P2P method to exchange objects or object families during this research.

Recommended solutions

There are several Python libraries that can be used to implement P2P functionality in Blender. For example “Twisted” - an event-driven networking engine [63, 67] written in Python, provides a framework used to create network-related Python software. It includes event loop, callback system and networking framework. Thus, it is not intended directly for P2P, but can be used to create needed functionality. “CSpace” can be an alternative. It provides a platform for secure, decentralized, user-to-user communication over the internet. It should be possible to implement the system as an add-on. However, since it is implied that the service should be able to run in background, traditional blender add-ons probably won't solve the problem. Thus, the functionality might have to be implemented in C and other libraries might have to be used. In this way it will work in a separate thread and would no longer be similar to an Blender add-on. A new editor will be needed to implement. Notification and event management system will have to be supplemented with new events and notifications. The implementation of this system would require additional effort and it is not entirely clear whether this functionality is really important, since a centralized library is expected to be implemented for exchanging objects. Furthermore, although this P2P system should work smoothly for different objects, same objects with different revisions (not registered in the central server) might result in inconsistency of object libraries. To sum up, it is recommended to choose either P2P (decentralized version) and refuse the idea of revisions or stay with the centralized library model with revision control, which is believed to be the preferred choice.

Solutions development estimation and planning

Table 5.14. Estimation of model exchange via P2P implementation

Task description	Estimated MT
Model exchange via P2P	60
Total:	60

5.3.6. Replication-synchronization with other distribution servers

Requirements

Evaluate the ability of replication-synchronization with other distribution server installations.

Currents situation and solutions

Currently there are no information about such solutions.

Recommended solutions

This depends on other servers architecture so should be analyzed individually.

5.3.7. *Integration of Jabber protocol*

Requirements

The integration of Jabber protocol into Blender (for instant messaging – IM).

Current situation and solutions

At the moment there is no chat functionality in mainstream version of Blender. There may, however, exist implementations of communication systems in computer games created with Blender Game Engine, although none examples of this kind were found during this research. To circumvent the lack of this feature, Blender artists use IRC channels or other technologies, such as Skype.

Alternative solutions

Jabber (especially XMPP part) [54] is a cross-platform standardized protocol that has many implementations and is used in a great number of applications. Despite the latter, there exist quite a few alternatives. As an example, SIP SIMPLE (*Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions*) [2] is an instant messaging (IM) and presence protocol suite based on Session Initiation Protocol (SIP) messaging protocol, used by Microsoft's Office Communicator. SIMPLE is an open standard and could possibly be used as an alternative to introduce communication functionality in Blender. SIP as a stand-alone protocol is also worth considering. It allows VoIP as well, thereby extending the communication abilities. There exist python libraries for SIP or SIP SIMPLE protocol clients (e.g. python-sipsimple), most of them for Linux operating systems, though it should be easy to adapt them for Windows.

Another potential alternative is ICQ client library for Python [53]. Although ICQ software, and thereby the protocol, is widespread, it is licensed with a proprietary license, which makes the ability to use it in Blender rather limited and less tempting.

Yet another alternative is IRC utilization in Blender. There are python libraries created for IRC functionality for Linux, Windows, other OS and even cross-platform. It is mainly designed for group communication in discussion forums, called channels, but also allows one-to-one communication via private message as well as chat and data transfer, including file sharing, which could be used in Blender for model sharing.

Thus, there are many alternatives to Jabber, which could be used in Blender. It is worth noting that they all work and are used in different ways, as well as they are licensed under various licenses. However, the standards mentioned in this section provide the functionality in python libraries, which make them easy to integrate into Blender system due to the benefits of tightly python and Blender architecture integration.

Recommended solutions

As mentioned previously, there are python modules for Jabber itself. Thus, the easiest way to integrate it into Blender would be merely through the add-on system. The new editor will also be required and it should have an area displaying contacts and their status, a region allowing to write a message and a region to display the messages of the conversation. Of course, it is a design issue whether some regions should be joined together and if there will be needs for other regions. The closest editor here is *python console*. It allows typing and executing a command and also presents results – status and error messages. It would be best to create new editor starting with the code of

python console editor's module, which is basically located in *blender/source/blender/editors/space_console* folder. Additional functionality will be required from *blender/source/blender/editors/interface* module. To register the new editor, its keymap and the required operators, additional code should be inserted in */blender/source/blender/editors/space_api/spacetypes.c* as well. In order to make code as much compatible as possible with mainstream version of Blender, mentioned features must be registered in the same way as for other editors. The python part of Jabber functionality will be easy to implement as python operators (add-ons), which will call the library functions. It is possible to implement the functionality in a deeper level of Blender, i.e. creating the C language operators to call Jabber library functions, but this is not necessary, because instant messaging is not expected to take much of the computational resources.

These changes should be suitable for future Blender versions, if the architecture of Blender does not radically change. It might be wise to consider integrating the functionality to the mainstream version, although this should be first proposed and discussed in Blender forums.

Solutions development estimation and planning

Table 5.15. Estimation of Jabber protocol integration

Task description	Estimated MT
Jabber protocol integration	40
Total:	40

5.3.8. Work-flow formation, tracking and confirming changes. Version Control. Teamwork.

Requirements

Evaluate the abilities for work-flow formation, tracking and confirming changes, version control mechanism and teamwork.

Current situation and solutions

Currently, Blender has no internal mechanisms to control versions of objects, neither is it able to track and confirm changes. Current way to manage and perform teamwork is to use multiple .blend files that are linked together in a certain way. The project generally consists of one or several files which compose the scene or overall object, and cover many smaller objects, which are linked from files to it. Modifying one object in its dedicated file also affects files linked to it. Thus, many contributors can work with the same project simultaneously. Teamwork specific processes and version control of files typically are managed through widely know SVN or GIT version control systems.

Alternative solutions

In Revit Architecture there is a project version control system, which is based on units called Revision Clouds [8]. It is not automatic, which means that Revit does not store new project version every time the project is saved or other frequent actions are performed. However, it allows saving revisions manually, including the modified/created components in a Revision Cloud. It also allows entering additional information about the revision, like description and date of changes. The system

is geared towards revisioning of the whole project and not its separate objects. Analogous Revision Cloud system is used in ArchiCAD.

As for teamwork, Revit Architecture has a fairly sufficient system for that, called Revit Server [11]. It consists of several major elements – a central server, local servers and team members computers. Revit server enables working with other team members in different places and simultaneously. If there is a model needed to modify, one can lock it and no other member can work with it at the period of time until it is locked. A drawback is that updates must be manually downloaded to the computer – no automatic updates are allowed. This teamwork mechanism does not have the built-in functionality for communicating with other team members. As an alternative, Revit Architecture also offers teamwork in the same manner as it is currently possible in Blender – file linking.

A similar solution for ArchiCAD is Graphisoft BIM Server [34]. This package provides functionality to assign roles for team members, thus giving rights to perform only particular actions. Locking (reserving) and releasing objects, notifying about changes and receiving updates is also available and is similar to Revit's mechanism.

Recommended solutions

While implementing revision control functionality in Blender other revision control systems can be very handy. Since Revit and ArchiCAD source code is not available due to the licenses, it would be best to look at external version control systems, such as Subversion (SVN) or GIT. The source code for these systems can be easily accessed and even used in other projects, as long as license terms are not violated. All of these systems provide delta compression for both text and binary files. Delta compression examines the new files, compares them to the old ones and saves only the data related to the changes performed. Thus, the project size depends not on the number of revisions, but on the number of changes performed in the project. Two, most popular, open source, version control systems already mentioned are SVN [4] and GIT. There is a difference between SVN and GIT [25]. The former uses a centralized revision control model – revisions are saved in a certain master server. While the latter features a distributed revision control model – no master server is required – people can share their new versions with coworkers almost directly – similar to P2P). If it is believed that the improved system should present functionality similar to Revit's, the better choice here will be SVN - for its more suitable model. Since there is a python interface to SVN, it should be rather easy to create a python script (plug-in) for committing and checking out a revision, consisting of the whole blend file.

Development of revisions mechanism for individual objects in .blend file would be a harder task and most probably require to create a dedicated Blender Architect revision control system from scratch. The key elements would be attributes to a parametrized object – id field (by which a certain kind of PO would be identified) and a revision number. In order to preserve current Blender architecture, these elements would require changes in Blender's DNA [18] and RNA (API to DNA [13]). Thus, in modules */blender/source/blender/makesrna* and */blender/source/blender/makesrna* proper functions and structures should be created, including the ones that register these structures in the system. It should not be forgotten that these files will have to be possible to load and save to a .blend file. For this reason, and depending on the changes made to DNA and RNA modules, */blender/source/blender/blenloader* module might need changes as well. The revision control module will have to be created from scratch (with the potential support of SVN source code) and should probably be located in *blender/source/blender*. The revision manager might also be implemented as python plug-in, since it is possible to reach RNA structures from scripts, but technically it is not recommended, because plug-ins are more meant to perform tasks, such as modify certain data, add objects or draw menus. The manager will require a new editor and maybe

new regions, thus */blender/source/blender/editors* module and some other related packages, such as */blender/source/blender/editors/interface* will require modifications. To sum up, this system has great potential, should not be complicated to implement and might be accepted to the mainstream version for the benefits it would provide in various kinds of projects.

A harder task would be to implement teamwork functionality, i.e. allowing multiple team members to work on the same file simultaneously. As it was said before, the only possible way to collaborate at the moment is by using many separate .blend files and linking them together in a suitable manner. It is an important to mention that the task discussed here would require changes in many code parts and there is a risk of corrupting current Blender structure. The first question to discuss is the way of reserving/releasing objects and receiving updates. Reserving/releasing objects would require ability to identify if an object is reserved. This could be done by manipulating object fields defined in */blender/source/blender/makesdna/DNA_object_types.h*. When manipulating DNA of the file simultaneously by several people in a network, it is critical to preserve the consistency of the file, i.e. the same object must have the same id value in all workstations and one id cannot be used for more than one object. To solve potential conflicts a proper manager system would have to be created, which would either notify all the computers working in a network about a change or (and) could manage merging operation on received changes with the central or a local file. However, this remains problematic. New editors probably will not be required since disired functionality could be implemented using add-on's API. A concerning thing is making Blender menu options and key presses inactive/active when an object is reserved/released. This might include changing most of the menus and operators in */blender/source/blender/editors* module. This would include checking whether a newly selected object is allowed to perform a certain operation. This could be done with Blender events and notification system (*/blender/source/blender/windowmanager* module files *wm_event_system.h*, *./intern/wm_event_system.c* and related files).

As an alternative, it should be possible to split a single Blender file into several files, containing non-intersecting sets of objects due to the Blender's DNA structure, which is basically a set of categorized lists of objects that have various combinations of the attribute values. Module */blender/source/blender/blenloader* could provide a lot of help in this case, both showing how Blender's DNA is manipulated and providing the most part of the code required to present the functionality for splitting. However, how to sort objects to files in Blender Architect system remains a question of design. It could be done by additional attribute in DNA structure and an analyzer for this value. The new functionality would reside in a module located most probably in */blender/source/blender/*. SVN also has functionality for locking files, thus analyzing its implementation might also provide ideas for Blender's teamwork functionality development.

Solutions development estimation and planning

Table 5.16. Estimation of teamwork and version control functionality implementation

Task description	Estimated MT
Teamwork functionality	80
Version control	40
Total:	120

6. CONCLUSIONS

1. Blender API is based on C/C++ and Python scripting language. C/C++ is a common language where time critical computations solutions are required. Python is powerful scripting language with large number of libraries and user base. Blender has well established modular, MVC based design and well developed, Python driven API which makes integration and extensions easy. Blender is open source, has no commercial licensing restrictions and is the best starting point for open source architectural design project. For architecture application developers it is recommended to follow mainstream Blender API coding guidelines and use currently available ways to extend application – Python scripts to create new operators, menus and special low computation power required features. For more complex tasks, where computation speeds are critical it is recommended to use Blender C/C++ code as a basis for extension, new features support and modification.
2. Performance problems are expected in 3D view editor where large number of complex parametric objects will be displayed. Solutions here could be: better PO scene management, robust level of detail algorithms implementation and utilization of parallel computing (OpenCL) in time critical parts. Some parts of specific operators will also require better data management algorithms or partial C/C++ implementation.
3. For open source architecture design project it is recommended to use a cross-platform and free implementation of OpenPGP . GPG should be integrated into Blender as a Python plugin using a GPG Python wrapper.
4. PO components, level of detail and tools should be implemented as separate editors. Existing Blender platform editors and their coding guidelines should be used as reference to keep consistency with the mainstream Blender base. Python scripting should be used as much as possible and only time critical parts should be implemented in C/C++. It is recommended that open source architectural software was created as separate software package from current Blender mainstream version. It should follow existing Blender coding guidelines and structure consistent with mainstream Blender version which will allow to import/export possible new features from Blender to architecture design software and vice versa with minimal efforts. It is also recommended while starting open source architecture design project to clean up existing Blender mainstream base and decide which features may be required for architects and which not. RNA structure should also be reviewed and adopted to architectural needs and PO during design process.
5. Tools for architectural design should be implemented as separate editors for reports creation and sheets generation. Current Blender editors may be used as reference points. Consistency with Blender mainstream version API and User interface design should be kept as much as possible
6. Multi-threaded, scalable Mimer SQL based database server should be used as a model exchange server. It is recommended to use CentOS, Ubuntu server or other Linux based operating system.
7. Models exchange portal should support user management, 3D models database, video streaming capabilities, modern community tools and on-line payments processing through secure transfer protocols. It should handle high data loads and support scalable architecture such as server clustering for better stability. Integration with open source architecture software is also recommended.
8. Currently there are no external models integration in the mainstream version of Blender. Such a system should be developed from scratch. Blender DNA and RNA should be changed to support new objects attributes, ID fields, version control, external libraries address. A new module for managing the external models and tracking their versions will be

also required. The server side - models library - should be kept and managed in a server. Models can take a lot of space, thus downloading or uploading them into the server might require a decent bandwidth. The server should work as a database and a DBMS, providing information about available models, their versions and other relevant information .

9. “Twisted” - an event-driven networking engine written in Python or “Cspace” could be used as P2P model exchange solution. However version control problems may arise using P2P as a solution for teamwork and version control so it is recommended to use centralized version management model like SVN or GIT with revision control over P2P (decentralized, without version control) solution.
10. There are python modules available with Jabber implementation. The recommended way to integrate Jabber protocol for communication into Blender would be through the add-on system. The new editor will also be required and it should have an area displaying contacts and their status, a region allowing to write a message and a region to display the messages of the conversation. It is recommended to create new editor for communication functionality. Current Blender scripting editor could be used as reference point.
11. Development of revisions mechanism for integrated teamwork, version control and individual objects version control in .blend file will require to create a dedicated revision control system from scratch. It is recommended to integrate this system into architectural software by using open source version control system SVN as basis in project design. The key elements would be attributes to a parametrized object – id field and a revision number. In order to preserve current Blender architecture, these elements would require changes in Blender DNA and RNA .

REFERENCES

1. 3dModelFree, [online] Available at: <<http://www.3dmodelfree.com/>> [Accessed 20 September 2011]
2. Adrian Georgescu, SIP SIMPLE Client SDK [online] Available at: <<http://sipsimpleclient.com/>> [Accessed 25 August 2011]
3. Andrew Kator, Distributed Blender Rendering with Xgrid [online] Available at: <<http://www.atpm.com/10.06/blender.shtml>> [Accessed 3 June 2011]
4. Apache Software Foundation, Apache Subversion Features [online] Available at: <<http://subversion.apache.org/features.html>> [Accessed 22 September 2011]
5. Architect, 10 April 2011. BIM on a Budget [online] Available at: <<http://www.architectmagazine.com/design/bim-on-a-budget.aspx>> [Accessed 28 May 2011].
6. Autodesk, Autodesk Revit Architecture 2012 Product Information - Autodesk eStore Online Store [online] Available at: <<http://store.digitalriver.com/DRHM/store>> [Accessed 28 May 2011].
7. Autodesk, How to Buy [online] Available at: <<http://usa.autodesk.com/autocad-architecture/how-to-buy/>> [Accessed 28 May 2011].
8. Autodesk, 19 June 2009. Revision Clouds (hitting Preprint Construction Documents, then Revisions, then Revision Clouds) [online] Available at: <<http://docs.autodesk.com/REVIT/2010/ENU/Revit%20Architecture%202010%20Users%20Guide/RAC/index.html>> [Accessed 22 September 2011]
9. Autodesk, 22 March 2011. AutoCAD Architecture – Architectural CAD Features and Demos [online] Available at: <<http://usa.autodesk.com/autocad-architecture/features/>> [Accessed 28 May 2011].
10. Autodesk, 22 March, 2011. Revit Architecture - Features and Demos [online] Available at: <<http://usa.autodesk.com/revit-architecture/features/>> [Accessed 28 May 2011].
11. Autodesk, 28 September 2010. Revit Server Overview [online] Available at: <<http://docs.autodesk.com/subscription/REVIT/2011/ENU/filesUsersGuide/WS1a9193826455f5ff-7d5f8a95129ad2c3b01-7fc4.htm>> [Accessed 22 September 2011]
12. Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato, 15 September 2008. Locking [online] Available at: <<http://svnbook.red-bean.com/en/1.5/svn.advanced.locking.html>> [Accessed 14 September 2011]
13. Blender Foundation, Data API [online] Available at: <<http://wiki.blender.org/index.php/Dev:2.5/Source/Architecture/DataAPI>> [Accessed 25 August 2011].
14. Blender Foundation, Import and Export [online] Available at: <<http://www.blender.org/download/python-scripts/import-export/>> [Accessed 18 June 2011]
15. Blender Foundation, (n.d.). Layouts [online] Available at: <<http://wiki.blender.org/index.php/Dev:2.5/Py/Scripts/Guidelines/Layouts>> [Accessed 25 August 2011].
16. Blender Foundation, MVC Model [online] Available at: <http://wiki.blender.org/index.php/Dev:2.5/Source/Architecture/Window_Manager> [Accessed 25 August 2011].
17. Blender Foundation, Notes and Updates [online] Available at: <<http://wiki.blender.org/index.php/Dev:2.5/Source/Architecture/Notifiers/Updates>> [Accessed 25 August 2011].
18. Blender Foundation, Notes on SDNA [online] Available at: <http://wiki.blender.org/index.php/Dev:Source/Architecture/SDNA_Notes> [Accessed 25 August 2011].

- August 2011].
19. Blender Foundation, Quickstart Introduction — Blender v2.59.0 - API documentation [online] Available at: http://www.blender.org/documentation/blender_python_api_2_59_0/info_quickstart.html [Accessed 10 June 2011]
 20. Blender Foundation, View2D [online] Available at: <http://wiki.blender.org/index.php/Dev:2.5/Source/UI/View2D> [Accessed 25 August 2011].
 21. Blender Foundation, 17 May 2011. Blender Code Layout [online] Available at: <http://www.blender.org/bf/codelayout.jpg> [Accessed 14 September 2011].
 22. Blender Foundation, 2 May 2011. blender.org - Features [online] Available at: <http://www.blender.org/features-gallery/features/> [Accessed 14 May 2011].
 23. CentOS, 4 March 2010. CentOS Product Specifications, [online] Available at: <http://www.centos.org/product.html> [Accessed 10 September 2011]
 24. Dassault Systemes, DraftSight: Free CAD software* for your DWG files [online] Available at: <http://www.3ds.com/products/draftsight/overview/> [Accessed 28 May 2011]
 25. Derek Robert Price, Ximbiot and Free Software Foundation, Inc. 3 December 2006. CVS - Concurrent Versions System [online] Available at: <http://www.nongnu.org/cvs/> [Accessed 25 September 2011]
 26. Echange3d, [online] Available at: <http://www.exchange3d.com/> [Accessed 20 September 2011]
 27. Free Software Foundation, Inc. , 14 September 2011. The GNU Privacy Guard – GnuPG.org [online] Available at: <http://www.gnupg.org/> [Accessed 10 June 2011].
 28. Glare Technologies Limited, Blender | Indigo Renderer [online] Available at: <http://www.indigorenderer.com/blender/> [Accessed 28 September 2011].
 29. Glare Technologies Limited, Glare Technologies Store [online] Available at: <http://store.glaretechnologies.com/> [Accessed 28 September 2011].
 30. Glare Technologies Limited, Technical Specifications | Indigo Renderer [online] Available at: <http://www.indigorenderer.com/features/technical/> [Accessed 28 September 2011].
 31. Google, Download SketchUp C++ SDKs [online] Available at: <http://code.google.com/apis/sketchup/docs/downloadsdksubmit.html> [Accessed 14 June 2011]
 32. Google, Purchase Google SketchUp Pro 8 [online] Available at: <https://sketchupprostore.appspot.com/index.ep> [Accessed 28 May 2011].
 33. Google, What makes SketchUp great? [online] Available at: <http://sketchup.google.com/intl/en/product/features.html> [Accessed 28 May 2011].
 34. Graphisoft, 1 August 2011. ArchiCAD Revolutionizes BIM Collaboration [online] Available at: <http://www.graphisoft.com/products/archicad/teamwork.html> [Accessed 15 August 2011]
 35. GreenButton, Applications – Blender [online] Available at: <http://www.greenbutton.net/Applications/Blender> [Accessed 12 June 2011]
 36. GreenButton, GreenButton – Overview [online] Available at: <http://www.greenbutton.net/Solutions> [Accessed 12 June 2011]
 37. Hannes Högni Vilhjálmsson, 9 August 2005. Writing a Sequence Plugin [online] Available at: <http://www.ru.is/kennarar/hannes/useful/blendermanual/htmli/ch27s06.html> [Accessed 5 June 2011]
 38. John Surewicz, Revit vs Archicad vs Microstation [online] Available at: <http://www.scribd.com/doc/2231948/Revit-vs-Archicad-vs-Microstation> [Accessed 28 May 2011].
 39. Kent Mein, Tips for Coding Blender [online] Available at:

- <http://wiki.blender.org/index.php/Dev:Doc/Tools/Tips_for_Coding_Blender> [Accessed 10 June 2011].
40. Khronos Group, OpenCL - The open standard for parallel programming of heterogeneous systems [online] Available at: <<http://www.khronos.org/openccl/>> [Accessed 12 June 2011]
 41. LuxRender, 14 September 2011. Features – LuxRender Wiki [online] Available at: <<http://www.luxrender.net/wiki/Features>> [Accessed 28 September 2011].
 42. LuxRender, 14 September 2011. Introduction to LuxRender – LuxRender Wiki [online] Available at: <http://www.luxrender.net/wiki/Introduction_to_LuxRender> [Accessed 28 September 2011].
 43. LuxRender, 14 September 2011. LuxRender and OpenCL – LuxRender Wiki [online] Available at: <http://www.luxrender.net/wiki/Luxrender_and_OpenCL> [Accessed 28 September 2011].
 44. LuxRender, LuxRender – blender 2.5x [online] Available at: <http://www.luxrender.net/en_GB/blender_2_5> [Accessed 5 June 2011].
 45. Matthew Allum, 29 June 2004. jabber.py - A Python Jabber library [online] Available at: <<http://jabberpy.sourceforge.net/>> [Accessed 14 June 2011].
 46. Microsoft, Memory Limits for Windows Releases [online] Available at: <<http://msdn.microsoft.com/en-us/library/aa366778%28v=vs.85%29.aspx>> [Accessed 10 September 2011]
 47. Microsoft, 15 September 2011. Windows Server 2008 R2 How to Buy [online] Available at: <<http://www.microsoft.com/en-us/server-cloud/windows-server/2008-r2-buy.aspx>> [Accessed 15 September 2011]
 48. NVIDIA ARC GmbH, (n.d.). Products with mental ray [online] Available at: <<http://www.mentalimages.com/products/mental-ray/availability.html>> [Accessed 10 June 2011]
 49. NVIDIA Corporation, CUDA [online] Available at: <http://www.nvidia.com/object/cuda_home_new.html> [Accessed 10 June 2011]
 50. OpenMP ARB, 13 January 2011. About the OpenMP ARB and OpenMP.org [online] Available at: <<http://openmp.org/wp/about-openmp/>> [Accessed 10 June 2011]
 51. OpenPGP Alliance, About OpenPGP [online] Available at: <http://www.openpgp.org/about_openpgp/> [Accessed 19 June 2011]
 52. Oracle, Oracle Solaris and Oracle SPARC Servers—Integrated and Optimized for Mission Critical Computing, [online] Available at: <<http://www.oracle.com/technetwork/server-storage/solaris/documentation/index-jsp-137508.html>> [Accessed 10 September 2011]
 53. OStatic, YAPCQ [online] Available at: <<http://ostatic.com/yapcq>> [Accessed 25 August 2011]
 54. Peter Saint-Andre, About – jabber.org [online] Available at: <<http://www.jabber.org/about/>> [Accessed 25 August 2011]
 55. RandomControl, SLU, Arion [online] Available at: <<http://www.randomcontrol.com/arion>> [Accessed 28 September 2011].
 56. RandomControl, SLU, RandomControl Shop [online] Available at: <<http://www.randomcontrol.com/arion-purchase>> [Accessed 28 September 2011].
 57. Red Hat, Inc., Red Hat Enterprise Linux Technology capabilities and limits (supported[/theoretical]) [online] Available at: <<http://www.redhat.com/rhel/compare/>> [Accessed 10 September 2011]
 58. Refractive Software LTD, 16 October 2010. Octane Render – Features [online] Available at: <<http://www.refractivsoftware.com/features.html>> [Accessed 5 June 2011].
 59. RibbonSoft GmbH, 17 March 2011. QCAD [online] Available at: <<http://www.qcad.org/qcad.html>> [Accessed 14 June 2011]
 60. Solid Iris Technologies, Hybrid Engine (Biased+Unbiased) [online] Available at:

- <<http://www.thearender.com/cms/index.php/features/tech-tour.html>> [Accessed 28 September 2011].
61. Solid Iris Technologies, Application + Plugins (Hitting the Buy Now button) [online] Available at: <<http://www.thearender.com/cms/>> [Accessed 22 September 2011]. {22}
 62. Solutions in Computer Graphics, Blender OpenCL released [online] Available at: <http://sicg.atmind.nl/index.php?option=com_content&view=article&id=30> [Accessed 15 June 2011].
 63. Tachyon Technologies, 3 August 2010. CSpace [online] Available at: <<http://cspace.in/>> [Accessed 25 August 2011]
 64. Thomas L. „Blender Insider“, Sparkling publishing, 2010
 65. Ton Roosendaal, Blender Architecture Overview [online] Available at: <<http://wiki.blender.org/index.php/Dev:Source/Architecture/Overview>> [Accessed 10 June 2011].
 66. TurboSquid, [online] Available at: <<http://www.turbosquid.com/>> [Accessed 20 September 2011]
 67. Twisted Matrix Labs, Twisted Matrix Laboratories Projects [online] Available at: <<http://twistedmatrix.com/trac/wiki/TwistedProjects>> [Accessed 18 June 2011]
 68. Upright Database Technology, AB, 22 April 2001. Database Server Architecture [online] Available at: <http://developer.mimer.com/documentation/Mimer_SQL_Technical_Description/Performance2.html> [Accessed 5 June 2011]
 69. Vinay Sajip, 2 January 2011. *python-gnupg* - A Python wrapper for GnuPG [online] Available at: <<http://packages.python.org/python-gnupg/>> [Accessed 19 June 2011]
 70. YafaRay, Home | YafaRay [online] Available at: <<http://www.yafaray.org/>> [Accessed 24 September 2011].