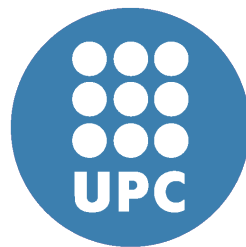


A reusable web hosting control panel with billing system



Marc Aymerich Gubern
Universitat Politècnica de Catalunya

A thesis submitted for the degree of
Diploma in Computer Systems

Fall semester 2011-2012

Contents

1	Introduction	1
1.1	Project overview	1
1.2	Context	3
1.2.1	What is Pangea?	4
1.2.2	Pangea members description	4
1.2.3	Pangea staff description	5
1.2.4	Pangea software stack	5
1.3	Requirements analysis	7
1.3.1	Contacts management	8
1.3.2	Services	8
1.3.3	Advanced pricing configurations	10
1.3.3.1	Calculate the <i>metric</i> of the service	11
1.3.3.2	How to calculate the price	12
1.3.4	Billing System	13
1.3.5	Payment gateway	15
1.3.6	Resource limiting and accounting	15
1.3.7	Internationalization	16
1.3.8	Target software to support	16
1.3.9	Multi server support	17
1.3.10	Easy to use for unskilled users	17
1.3.11	Easy to add new functionalities	17
1.3.12	Reusable	17
1.3.13	Open source friendly	17
2	State of the art	19
2.1	Current related software at Pangea	19
2.1.1	Members and billing management	19
2.1.2	User control panel	20

2.1.3	Administration scripts	21
2.2	Outstanding existing solution	21
2.2.1	SysCP	23
2.2.1.1	Requirements fit	23
2.2.2	Domain Technologie Control (DTC)	25
2.2.2.1	Requirements fit	25
2.2.3	ISPConfig 2	27
2.2.3.1	Requirements fit	27
2.2.4	Summary	28
2.3	Final decision	29
3	Selected tools	31
3.1	Django framework	31
3.1.1	Framework advantages	31
3.1.2	Django advantages	32
3.2	Celery distributed task queue	34
3.3	Pisa PDF generator	35
3.4	Django admin tools	35
4	Design principles	37
4.1	Reusability	37
4.1.1	Separation of Concerns (SoC)	37
4.1.2	Loose coupling	38
4.2	Orthogonality	39
4.3	Adaptability	40
4.4	Extensibility	41
4.5	DRY Don't repeat yourself	42
5	Architecture design and implementation	43
5.1	General considerations	43
5.2	Common	48
5.3	Daemons	50
5.3.1	Daemons Model	50
5.3.2	Daemons Admin	53
5.4	Resources	55
5.4.1	Resources Model	55
5.4.2	Resources Admin	58

5.5	Contacts	59
5.5.1	Contacts Model	59
5.5.2	Scheduling cancellations and deactivations	61
5.5.3	Contact Admin	63
5.6	Ordering	68
5.6.1	Ordering Model	68
5.6.2	Ordering Admin	69
5.7	Billing	72
5.7.1	Billing Model	72
5.7.2	Billing Admin	74
5.8	Service converter	76
5.8.1	django.contrib.auth a succesfull example	76
5.9	Mail	79
5.9.1	Mail Model	79
5.9.2	Mail Admin	80
5.10	Web	81
5.10.1	Web Model	81
5.10.2	Web Admin	82
5.11	PHP Plugin	83
5.11.1	PHP Plugin Model	83
5.11.2	PHP Plugin Admin	84
5.12	DNS	84
5.12.1	DNS Model	84
5.12.2	DNS Admin	85
5.13	Jobs	87
5.14	Jobs Model	87
5.14.1	Jobs Admin	87
5.15	Extra fields	88
5.15.1	Extra fields Model	88
5.16	Extra fields Admin	89
5.17	Global interactions	90
5.17.1	Create, update and delete a service	90
5.17.2	Bill a contact	92

6	Evaluation	93
6.1	Development effort	93
6.2	Global economic analysis	96
6.3	Evaluation and testing of the project implementation	97
6.4	Reusability evaluation	98
6.5	Requirements compliance	100
6.6	General evaluation	101
6.7	Future work	102
A	Examples	105
A.1	Template example	105
A.2	Invoice example	105
A.3	Fee example	105
B	Manuals	109
B.1	Installation	109
B.2	Create and install new service	113
B.3	Admin use guide	114

List of Figures

1.1	Pangea software stack	6
1.2	Caption for LOF	7
1.3	Bill life cycle	15
2.1	SysCP screenshot	23
2.2	DTC screenshot	25
2.3	ISPConfig 2 screenshot	27
3.1	Celery architecture overview	34
3.2	Django admin tools drag and drop detail	36
3.3	Django admin tools menu detail	36
4.1	Django Model-Template-View overview diagram	40
5.1	Admin home page screenshot	44
5.2	Admin change list screenshot	45
5.3	Admin actions screenshot	45
5.4	Admin change form screenshot	46
5.5	Applications architecture overview	47
5.6	Screenshot of list add form with default looking	50
5.7	Daemons application model	51
5.8	Daemons admin change list	53
5.9	Django-celery admin change list	54
5.10	Daemons admin change form	54
5.11	Resources application model	55
5.12	Monitoring dynamic inline	58
5.13	Monitor admin change list	58
5.14	Monitor admin change form	59
5.15	Contacts application model	60
5.16	Contacts application model	61

5.17	Diagrams legend	62
5.18	Related contracts graph	62
5.19	Related objects dependency graph	62
5.20	Scheduling cancellation and deactivation structure	63
5.21	Contact link inserted on the service change list page	64
5.22	Contact and contract links inserted on the service change form	64
5.23	Filter objects by related contact	64
5.24	Add service cycle	65
5.25	Contact admin change list	66
5.26	Contact admin change form	66
5.27	Contract admin change list	67
5.28	Pack contraction page	67
5.29	Contract service page	67
5.30	Ordering application model	68
5.31	Service admin change list	70
5.32	Order admin change list	70
5.33	Service admin change form	71
5.34	Billing application model	73
5.35	Billing admin change list	74
5.36	Billing admin change form	75
5.37	Billing amend line page	75
5.38	Admin login form	77
5.39	User change list	77
5.40	User change form	78
5.41	Mail application model	79
5.42	Mail admin change list	80
5.43	Mail admin change form	80
5.44	Web application model	81
5.45	Web admin change form	82
5.46	PHP Plugin application model	83
5.47	PHP plugin dynamic form example	84
5.48	DNS application model	85
5.49	DNS admin change form	86
5.50	DNS admin change list	86
5.51	Jobs application model	87
5.52	Jobs app admin add form	88

5.53	Extra fields application model	89
5.54	Extra fields app admin add form	89
5.55	Extra fields dynamic form inserted on contacts change form	90
5.56	Apps interaction on save() service	90
5.57	Apps interaction on bill() contact	92
6.1	Loc and Churn	94
6.2	Commit Activity Index	94
6.3	File Count	94
6.4	Test environment	98
A.1	Invoice example	106
A.2	Fee example	107
B.1	Creating a chroot with deploy_dev.sh	110
B.2	Configuring MySQL server with deploy_dev.sh	111
B.3	Screenshot of the last deploy_dev.sh steps	111

List of Tables

1.1	Price definition example	12
1.2	Bill life cycle	14
2.1	Main Linux web hosting control panels comparison	22
2.2	Requirements compliance summary	29
6.1	Requirements compliance summary (with our solution)	101

Chapter 1

Introduction

This project report describes the development process of a reusable web hosting control panel with billing system. The document reflects the chronological order of the development steps followed such as identifying the requirements, checking if an existing project fits these requirements and finally designing and implementing the new solution.

This first introductory chapter describes the general goals of the project, explains its context and presents an in deep requirements analysis.

1.1 Project overview

A web hosting control panel is a web-based interface provided by a hosting company that allows customers to manage their hosted services accessing from a single web site. It is also used by the hosting company as a support tool for their administrators, who use it as a resource to facilitate their daily tasks.

The project is sponsored by pangea.org, a non-profit Internet hosting provider¹. They require this kind of system in order to help its staff and members to manage their servers and hosted services. The most remarkable reasons that have lead Pangea to decide on making use of a complete and integrated web hosting control panel are:

1. To ease sysadmin's work by automating the most repetitive tasks.
2. To support on the accountancy operations by managing, in a centralized way, all the members information, contracted services and other billing details.

¹These organizations provide services that run on "Internet servers", allowing other organizations and individuals to serve content to the Internet. The most common kind of offered services are web, email, DNS and database hosting.

3. To offer the members the ability to contract and manage their services on their own.
4. To keep services and resources under control.

As we will see on chapter 2, *State of the art*, there are a lot of existing open source control panels that Pangea could use in order to satisfy its needs. Unfortunately after widely testing the three more promising panels none of them was convincing enough so we have decided to implement our own solution.

The objective of this project is to provide a collection of independent and reusable applications that cover, as much as possible, the needs of any hosting organization, in such a way that each hosting organization can create its own control panel just by picking and configuring a subset of these reusable applications. Django, the web framework selected for this project, fits perfectly with this objective since its philosophy and design encourages to split a software project into several reusable applications.

We have made an extra effort in order to ensure that this project can be reused by others. The following key feature list gives a glance on the main reuse-based features that have not been seen before in an open source control panel.

- **Easy to extend.** Services can be easily created and extended by means of inheritance. Also third party Django applications can be automatically converted to new services of the control panel.
- **Extremely modular.** Any component of this project can be substituted by another one providing the same kind of functionality, and no source code modification is needed.
- **Dynamic resource control.** There is no assumption on what kind of resources are needed to be controlled for each service. This decision is entirely up to each hosting organization. For instance, someone may need to only consider the traffic consumption for a web site and other's might need to take into account the number of running fcgid² processes as well as the disk, memory and CPU usage.
- **Dynamic service ordering.** For each service the panel gives to each hosting organization the decision of choosing, in a very generic way, what parts of its services do they want to charge.

²fcgid is a persistent process that handles web CGI requests

- **Multiplatform and multiserver.** Our control panel is multiplatform because it can be installed on any platform with Python³ support and it can control servers with any operating system. We say that our control panel is multiserver because it can control any type of service architecture; services centralized on a single server, and decentralized or distributed over multiple servers.
- **Multi-database support.** The control panel database backend can be changed at any time and without touching a single line of code.
- **Easy to integrate with an existing servers infrastructure.** The control panel can be configured to match virtually any servers infrastructure.

Other remarkable and unique hosting control features that this project has, are:

- **Featured billing and pricing systems.** Allowing more low level billing control and more pricing configurations than any existing open source control panel.
- **Message queuing for task execution.** Crontab is not used for executing task like other control panels do. Instead we take advantage of a message queue component that provides a more reliable and scalable way to perform tasks on background.
- **Scheduling of service cancellations and deactivations.** Our solution implements an scheduling mechanism for future service cancellation or deactivation. This mechanism is fully compatible with our pricing system, allowing to discount future cancellations or deactivation periods.
- **Manage large number of customers.** The existing control panel interfaces are not designed to handle large number of customers.

1.2 Context

To properly understand the developed software it is necessary to introduce its context, so that the reader can get a precise idea of which are the needs, priorities and circumstances that have influenced and shaped this concrete solution. It will also help to understand under which similar situations the control panel can be useful, with or without modifications.

³Python is a general-purpose, high-level programming language whose design philosophy emphasizes code readability. Python is a remarkably powerful dynamic programming language that is used in a wide variety of application domains.

1.2.1 What is Pangea?

Pangea.org or Pangea is a non-profit supporting organization, which has been working for social change since 1993. It “facilitates communication” through the use of information and communication technologies, such as e-mail, web content, free software and any other Internet service. Although it works with the Association for Progressive Communications in international projects its activity is focused on Europe / Spain / Catalonia following the principle of “act locally, think globally”.

This network began with an idea in early 1993 with the goal to enable electronic communication and information dissemination before the Internet appeared in Spain. That was the time of modems, terminal emulators, email, conferences, data networks. Later on, Pangea offered, promoted and provided training on the use of Internet, Linux, free software, web pages and applications to civil society organizations, social movements and non-profits. Furthermore, Pangea promotes critical thinking about the use, development, participation and politics of the Internet, ICT4D, and the Information Society in general. Pangea offers is self-sustained by the membership.⁴

Today Pangea is formed by a group of more than 700 members, including NGOs and individuals, who contribute to this project and use their services. Its core is composed by 10 employees and voluntaries who take care of the daily work, including system administration, software developing, member support and entity’s accountability.

1.2.2 Pangea members description

The Pangea’s members are persons and organizations like associations, NGO’s and foundations. They usually work on the third sector, in topics such as development, peace, ecology, cooperation, education and so on. They are people from all ranges of age, including old people who usually has a huge lack of IT related basic knowledge, so they appreciate the good support provided by Pangea as well as services with simple and intuitive program interfaces.

⁴Source en.wikipedia.org/wiki/Pangea.org

1.2.3 Pangea staff description

Currently there are 4 persons working at Pangea who are affected by the control panel subject in one way or another. Find below each person's responsibilities, so the reader will get a general idea on how a control panel can improve and ease its everyday work.

- Support (Lorena). Responsible of the help-desk support for the members via email and telephone. The usual members requests are changes on their services or ask for aid in any IT related topic, such as configuring their mail client or the best CMS choice for their new web space. She also manages their bills and membership fees, and it is on this point were the control panel can be very useful for she.
- System administrators (Carlos and I). Our responsibility is improve and maintain up and running all the IT infrastructure. Also we have the responsibility of solving non trivial members requests that Lorena can not do by her own. For example it could be configure the Apache server to fit the needs of a member or create a new virtual machine. The control panel intends to change this situation and make this part of our work easy enough to be done by Lorena.
- Accountant (Maite). She is responsible for reporting the Pangea's financial results, in accordance with government and regulatory authority rules. She also measures, disclosure and provision assurance about financial information that helps make decisions about allocating resources, like buying new servers or reducing IT expenses.

1.2.4 Pangea software stack

The Pangea software architecture is all based on Open Source and it runs on top of a Proxmox VE⁵ cluster composed by three physical servers (dell.pangea.org, sol.pangea.org and backup.pangea.org). Pangea uses OpenVZ⁶ for the virtualization layer and all the services run on top of a Debian squeeze Linux distribution isolated inside OpenVZ containers.

⁵Proxmox Virtual Environment is an easy to use Open Source virtualization platform for running Virtual Appliances and Virtual Machines

⁶OpenVZ (Open Virtualization) is an operating system-level virtualization technology based on the Linux kernel and operating system. OpenVZ allows a physical server to run multiple isolated operating system instances, known as containers, Virtual Private Servers (VPSs), or Virtual Environments (VEs).

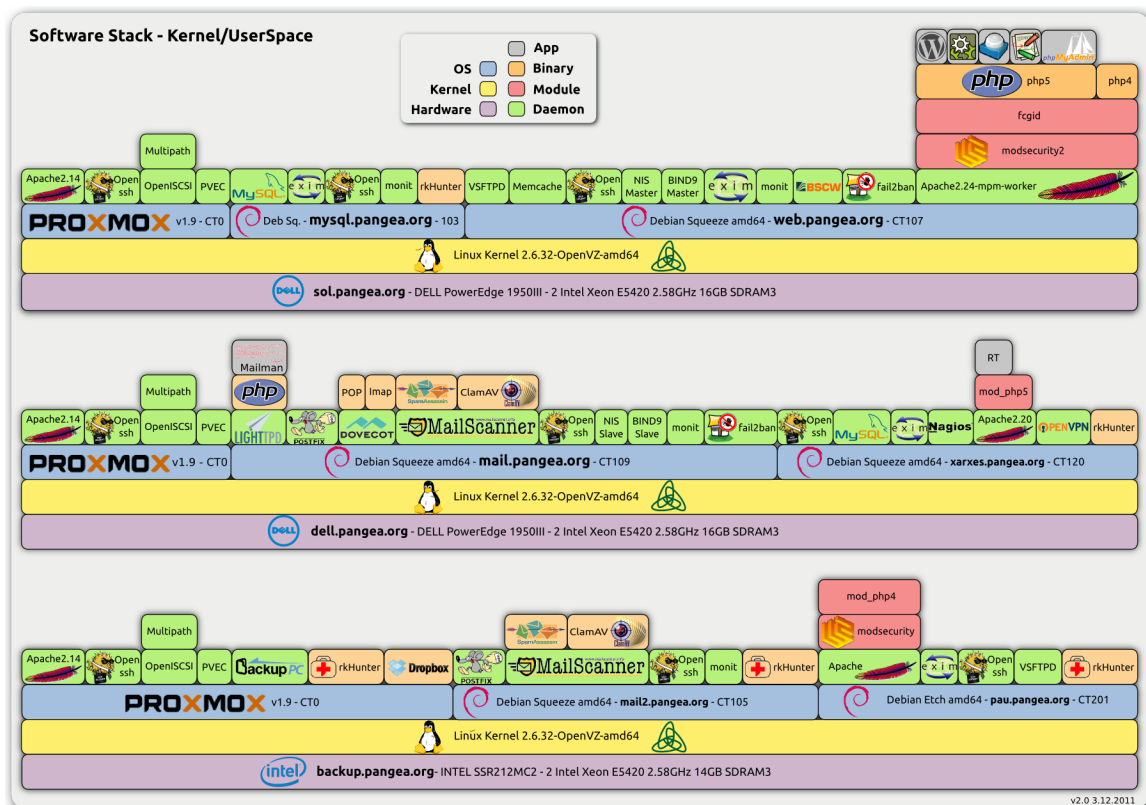


Figure 1.1: Pangea software stack

Basically there are 3 containers that offer services directly to their members. These containers are the main target of the control panel.

- **web.pangea.org** this server offers all the web-related functionalities providing shared hosting services through Apache2, SuExec, fcgid, php4-cgi and php5-cgi. There is a Wordpress and a Dokuwiki with multisite capabilities that allow members create their own blogs or wikis. Also this server provides the web interfaces of the services that are running on other servers, such as phpmyadmin to manage the MySQL databases and two webmails clients: RoundCube and Horde/IMP.
- **mysql.pangea.org** is the MySQL database server.
- **mail.pangea.org** is the main mail server of Pangea running with Postfix as a MTA⁷ and MailScanner as an antipsam gateway. This server stores the mailboxes and provides IMAP and POP3 access through the Dovecot daemon. The mailing list service lives on this server too, and it is offered by Mailman.

⁷Mail transfer agent (MTA) or mail relay is software that transfers electronic mail messages from one computer to another using a clientserver application architecture.

1.3 Requirements analysis

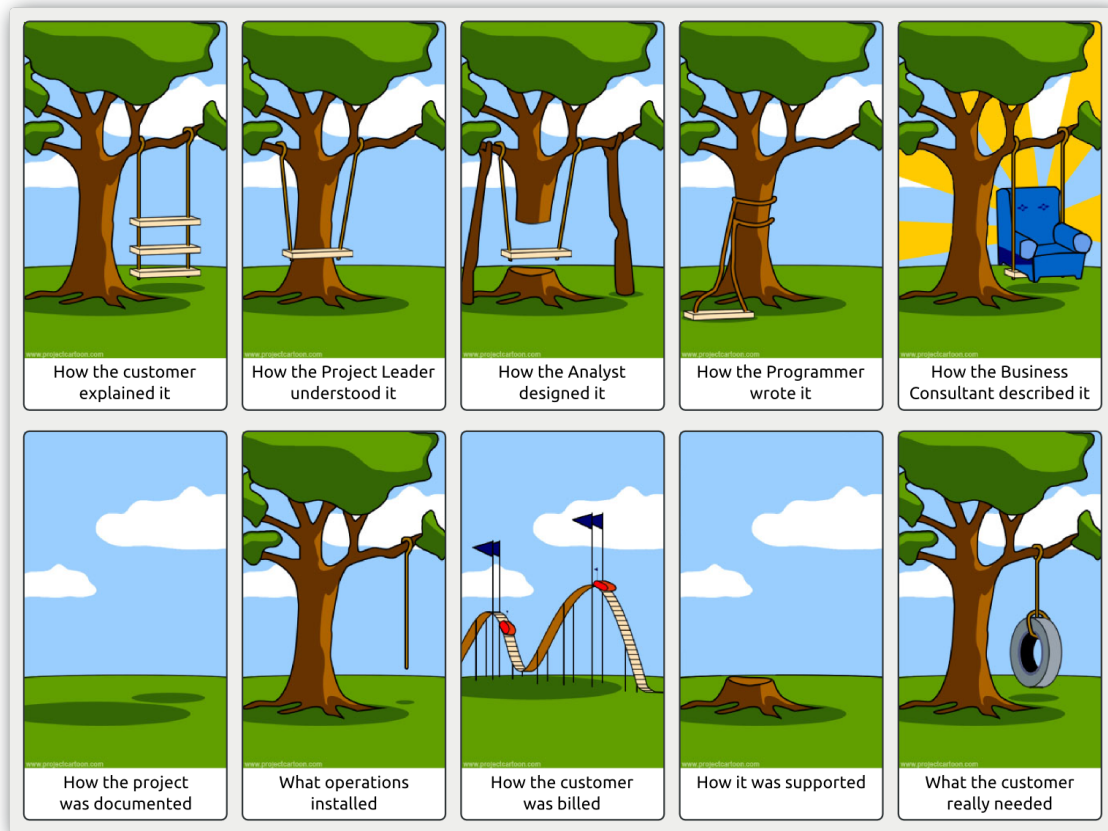


Figure 1.2: Requirements analysis parody⁸

An accurate analysis of the requirements is important if we want to satisfy the expectations of the users. Also it is one of the most complicated tasks during the development of a project since it involves identifying all the actors involved and understand their needs. So even for somebody, like in my case, that has been working for Pangea the last 4 years, it has not been a trivial matter since there are many underlying details which only the person who is daily working with them is aware of.

Most of the efforts on the requirements analysis have been made on the accountancy and billing related functionalities, since the ones related with the system administrator's tasks are much simpler for me. The work described on the following lines is the result of a lot of communication and discussions with the rest of the Pangea's team.

⁸From <http://www.projectcartoon.com>

Please notice that the presented requirements are not limited to Pangea's context. We have tried to identify them in a generic way, avoiding to exclude any potential use case that other organizations could have. The presented requirements can be applied to any commercial hosting company.

1.3.1 Contacts management

Contact management is about managing the information stored on the system regarding the members (customers in a business context) that make use of the service provided. Typical information that is stored can be the phone numbers, email, post address, postal code, national ID, and so on. Also it can be possible to provide alternates contact information for technical contact and billing purpose. The basic necessary operations are register and unregister a contact and maintains each registered service related with a contact.

1.3.2 Services

The most common hosting services that any hosting provider organization could offer are:

1. Shared web hosting⁹ with optional SSL
2. Mail Accounts
3. MySQL databases
4. DNS¹⁰
5. Mailing Lists
6. Blogs and Wikis
7. Virtual Private Servers (VPS)
8. Internet transit consumption
9. Web development and maintenance

⁹Refers to a web hosting service where many websites reside on one web server connected to the Internet. Each site "sits" on its own partition, or section/place on the server, to keep it separate from other sites. This is generally the most economical option for hosting, as many people share the overall cost of server maintenance.

¹⁰http://en.wikipedia.org/wiki/Domain_Name_System

10. Training courses

11. Packs of services

The control panel must implement the mechanism that talks to the daemons¹¹ behind these services (detailed ahead on requirement 8. *target software to support*) as well as provide a web interface for creating and managing these different services.

These services can be differentiated based on two *bussines model*, (1) the *subscription model* and (2) the *one time model*.

1. The ***subscription model*** is applied in a service like shared hosting, databases, VPS or DNS. It implies that the customer must contract a subscription for a certain amount of time and when it expires the subscription must be renewed in order to keep using the service. For the subscription services it must be possible to schedule future cancellations and deactivations.
2. The ***one time model*** is used on those services that are contracted just for a one particular moment (eventual services). The services from the above list that fall into this category are Web development and Training courses.

Two variants of the *subscription services* can be identified based on the service *renewal point* (the future date when a subscription service should be renewed). The renewal point can be (1) *fixed* or (2) *variable*.

1. A ***fixed renewal point*** means that the moment of renovation is fixed for all services and, for example, it should be renewed every April of each year. Most of the services offered by a hosting provider uses this approach since they used to invoice their subscription services only once a month or once a year.
2. A ***variable point*** means the renovation should happens just one month (or one year) after the moment when the service was contracted. Maybe DNS is the only service that uses this variable point since most DNS providers use it.

There are some situations where it is desirable to be able to configure some discounts for the *subscription services*:

- ***Discount on cancel or deactivation***. In some service it is desired to discount the canceled or deactivated proportional part of the subscription period. For example we want to discount the time when a mail account was deactivated.

¹¹A daemon is a computer program that runs as a background process, rather than being under the direct control of an interactive user.

- **Discount on register.** For some services with a fixed renewal point it is interesting to discount a quantity proportional to the time between the fixed renewal point and the register date. For example if we order a new mail account on February and the billing point is in April, maybe we want to discount April to February part of the entire year.

We can identify two more variants of the *subscription model* depending if the service should be payed before (**prepay**) or after their use/consumption (**postpaid**). An example of postpaid service is the Internet transit consumption, since there is no way to know in advance how much transit a member consumes in a month. For prepay services such as shared hosting, databases or DNS it should be possible enable some additional flags:

- When the subscription of the service is effective or scheduled for future cancellation or deactivation we might want to **refund** the proportional part of the payed period
- When a discounted future cancellation or deactivation has been revoked we might want to **recharge** this proportional part

1.3.3 Advanced pricing configurations

The pricing configuration defines how a price is calculated for a particular service. The hosting providers usually have multiple pricing options, most of them have (1) *hosting plans* where a pack or group of services are available for a cheapest price than picking these services one by one. (2) *Tiered prices* are other pricing option, with it the unitary price of a service decays according to the total of services of the same kind that a customer has contracted. The tiered price is useful to provide discounts based on bulk service contracting. Another pricing option is (3) the *flat rate* where a customer always pay the same price no matter how much resources consume.

Since we want to provide a highly flexible control panel, we have came up as a requirement with a pricing mechanism that, even being pretty complicated, it will cover almost any possible configuration that can be thought of. The following subsections explain the different elements and parameters that should be taken into consideration by the control panel when calculating the prices. First we introduce the *metric* concept and how to calculate it for a given service, and then we will see how to calculate the price using this calculated *metric*.

1.3.3.1 Calculate the *metric* of the service

A service can have two different *metrics*: (1) the **number of contracted services per customer** or (2) the **amount of resources consumed by a service**. Also we need to define which time period is relevant for calculating the metric, it can be a month or a year. And the start point of the period can be fixed or variable (same meaning as explained at the previous section).

1. In case it is a **subscription service** and the metric calculated is the **number of contracted services** we can calculate the metric based on the state of the service during the period. It can be:
 - Active or disabled
 - Registered (service registration moment)
 - Renewed (service renovation moment)
 - Registered and renewed
2. Otherwise in case we use the amount of resource consumed by a service we have two options:
 - Use the amount of resource for each order or,
 - use the amount of resource of all the contact orders of the same type of service.

Moreover with a subscription service two different approaches for calculating the metric can be used:

- Split the entire billing period in small pricing pieces and calculate the price based on the metric for each piece. For example this is possible with an annual billing period and a monthly pricing period, so even it is an annual service the metric used for calculate the price may vary each month.
- Just use the first month of each renovation to calculating the metric of the entire billing period.

1.3.3.2 How to calculate the price

Apart from the calculated metric there is another factor used to determinate the price of a service. It is the contracted *hosting plans* or **packs** that a customer or member has. The contracted packs are used for make different prices depending on, for example, the membership type. For instance the membership organization include more services with price 0 than a personal membership account. Or the members can contract for example 50GB Internet transit prepay pack, so with this pack the first 50GB of traffic are covered by the price of the pack.

Additionally we have considered three different rating algorithms to determinate the price of a service based on its metric and the contracted packs. The different algorithms are (1) the best price approach, (2) the progressive approach and (3) the match price approach. The three different approaches are presented by examples using the following price definition.

Table 1.1: Price definition example

Service	Pack	Metric	Unitary price
Domain .ORG	Default	1 - 4	15 €
Domain .ORG	Default	5 - 9	10 €
Domain .ORG	Default	10 - ∞	5 €
Domain .ORG	Organization member	1	0 €
Domain .ORG	Organization member	2 - ∞	10 €
Domain .ORG	Personal member	1 - ∞	10 €

Now lets imagine that we have contracted an Organization member pack, so we need to consider the default pack and the *Organization member pack*. This is how each algorithm should obtain the price for 10 domains:

1. *Best price* method returns the best total price considering the given metric (10). It should return 1 at 0€ and 9 domains at 5€
2. *Progressive price* returns the cheapest prices with accumulation of the pack entries. It is easy to understand with an example: With 10 domains it should return: 1 at 0€, and 9 at 10€. But notice that if we request 20 domains it should return a completely different values: 1 at 0€, 4 at 15€, 4 at 10€ and 11 at 5€ (total price: 145€) because this path is cheapest than the more 'intuitive' 1 at 0€ and 19 at 10€ (total price: 190€)

3. *Match price* returns the cheapest price that match with the metric. For 10 domains it should return 10 at 5€, for 9 domains it should return 9 at 10€ and for 1 domain 1 at 0€.

1.3.4 Billing System

The billing system must be able to generate and manage five kinds of bill documents:

1. **Invoice.** It is a commercial document issued by the service provider (seller) to the customer (buyer) related to their contracted services. This document describes the contracted services, quantities, and agreed prices for the services. The invoice indicates how much the buyer must pay to the seller, according to some payment terms. The buyer has 60 days to pay for these services. The invoice must implement support for the Spanish invoicing regulatory rules¹², but we need to provide hooks to allow future support for other regulations. You can see an invoice example on Appendix A.1.
2. **Membership Fee.** It is a document issued by a non profit organization and indicates that the receiver is a member of that organization and an amount of money should be payed for that reason. The membership fees have an independent numeration and no TAX should be applied. This kind of document does not have any regulatory rules. You can see a membership fee example on Appendix A.2.
3. **Amendment Invoice.** It is a document that indicates a rectification for a previous issued invoice with incorrect information, such as errors in the price, taxes, the services contracted or the contact information. An special request for the accountant of Pangea is that the amendment invoices should be adapted to the previous Spanish regulatory rules (RD 1496/2003¹³) rather than the current ones RD 87/2005¹⁴. We will provide both of them.
4. **Amendment Fee.** It is a document that describes a rectification for a previous issued membership fee, the amended error can be incorrect contact information or incorrect price value. This kind of document does not have any regulatory rules.

¹²descargas.sri.gov.ec/download/pdf/regfactura.PDF

¹³www.boe.es/boe/dias/2003/11/29/pdfs/A42537-42556.pdf

¹⁴www.boe.es/boe/dias/2005/02/01/pdfs/A03397-03401.pdf

5. **Budget.** It is a document requested by a customer (or potential customer) which reflects the detailed prices that a group of service will cost if the member contract it. This kind of document does not have any regulatory rule.

An independent numeration must be used for each type of bill. Also the numeration must be correlative and must restart each financial year (1st of January).

For subscription services it can be possible to invoice multiple periods at once, also it can be possible to ignore/discard some orders that for any reason we do not want to bill. For the international bills the taxes should be excluded. Finally the system should be able to create a PDF version of each document and handle its entire life cycle.

Table 1.2: Bill life cycle

Status	Performed process	Available actions
Open	Generate open ID ¹ Generate creation date	Download as PDF Move bill line Add bill line Delete
Closed	Generate Q19 Change ID Generate HTML ² Generate PDF Update creation date Generate due date	Discard Modify HTML Regenerate PDF Send Download PDF Amend line
Send	Send e-mail Send Q19	Mark as returned Mark as payed Amend line Download as PDF
Payed		Download as PDF
Returned		Discard Resend Regenerate PDF Amend line Download as PDF
Discarded		Undo (Close) Download as PDF

¹ Open ID is a provisional bill ID needed because until the bill is not closed it can be deleted and a correlative ID must be guaranteed

² An HTML version of the bill is stored in order to allow future modifications of the PDF

The presented table 1.3.4 and following figure 1.3 summarize the performed operations and the available actions for each bill status.

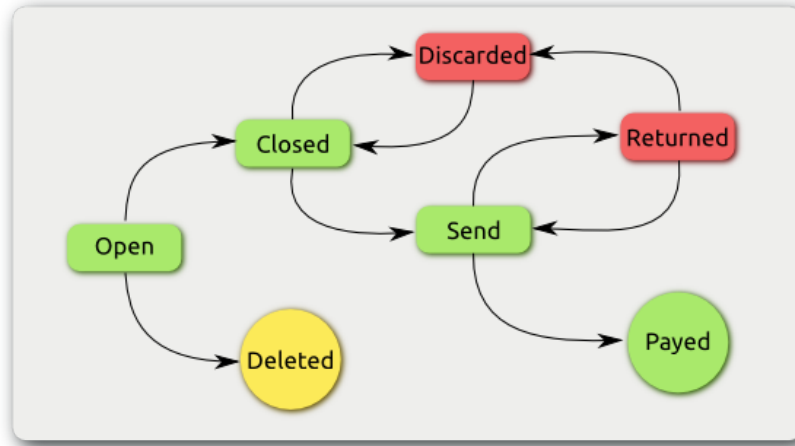


Figure 1.3: Bill life cycle

1.3.5 Payment gateway

There are lot of different available payment gateways, therefore we can not implement all of them, instead we plan to implement the one used at Pangea and provide hooks for future inclusion of new gateways. The payment methods currently available at Pangea are two: (1) bank transfers and (2) direct debit. The direct debit is performed through the Spanish Q19 bank procedure¹⁵. It is a communication mechanisms that uses a text file where the bank operations are described in a regulated format. In the case of bank transfer is performed by the member we only need to notify the member by email when a new bill has been emitted attaching it, this way the client can go directly to its bank and pay it through a bank transfer.

1.3.6 Resource limiting and accounting

Some of the services offered by a hosting company consume resources like CPU cycles, memory or Internet traffic. The panel should be able to limit the consumption of these resources and make a historical logging for posterior charge. Also the resources should not be hardcoded to each service since each organization can have their own specific needs for resource accounting and limiting.

¹⁵www.grupobes.es/auxfiles/aeb_19.pdf

1.3.7 Internationalization

Some users give high value on interfaces within their maternal language, for example in our case the most common are Catalan, Basque, Galician or Spanish. Therefore the control panel should be capable of supporting multiple languages. Also the control panel should support multiple currencies in order to allow reusing for organizations from countries outside the EU.

1.3.8 Target software to support

The priority target software to support is the software that is running right now on Pangea's servers, but we need to focus on the abstract functionalities provided by these services rather than the software specifics, so switching to an alternative software can be done easily. The control panel have to provide CRUD (Create, Read, Update and Delete) operations for each one of the following services.

1. HTTP: Apache2-mpm-worker with SuExec and fcgid
Create, modify and delete virtual hosts.
2. MTA: Postfix with MailScanner
Create and delete mail accounts, and handle per account Spam-filter configuration.
3. IMAP/POP: Dovecot with sieve
Create, modify and delete mail vacations, mail forwards and mail alias.
4. FTP: Vsftpd
Create and delete FTP accounts.
5. DB: Mysql
Create, modify and delete databases and database users, and their permissions.
6. DNS: Bind9
Create, configure and delete domain zones.
7. VPS: Proxmox (OpenVZ)
Create and delete virtual private servers and manage their resource limits.
8. MailingList: Mailman
Create and delete mailing lists.

9. Application as a service: Wordpress and Dokuwiki

Deploy new Wordpress, Dokuwiki or any other widely used open source web application.

1.3.9 Multi server support

We have to consider three different server architectures, since some organizations have (1) all the services in a single machine, others, like us, have (2) each services on a different servers, see figure 1.1, and others can have (3) the same service distributed over multiple machines. So in the worst case, for each service the panel must be able to know in what specific server it is hosted and implements a mechanism that allows remote task executions in order to perform the management operations over the servers.

1.3.10 Easy to use for unskilled users

We like to provide a user interface suitable for unskilled users, but at the same time feature rich enough that satisfies the more advanced users. Maybe the interface could hide the advance options or we can allow multiple user interface implementations and let the user chooses which one is more comfortable for them.

1.3.11 Easy to add new functionalities

In short periods of time new services may become available for the users. The control panel must facilitate the integration of these new services as soon as possible.

1.3.12 Reusable

The design of the whole project must be focuses on reusability, making things as generic as possible in order to ensure that other organizations or persons can use this project for their own needs. Also in order to guarantee their reusability, the source code must be clean and well structured to facilitate future developments.

1.3.13 Open source friendly

Due the Pangea's philosophy and compromise with the open source¹⁶ community, the adopted solution must be under an open source license and not dependent of commercial products.

¹⁶Open source definition at <http://opensource.org/docs/osd>

Chapter 2

State of the art

Following section presents the current available solutions, both specifically developed for (and by) Pangea and existing open source hosting control panels. Each tool is analyzed and evaluated to decide whether it should be reused or integrated to the final solution.

2.1 Current related software at Pangea

Pangea has been providing its services for 18 years, during which the same necessities were already present. As expected, Pangea has developed different software solutions along time in order to support its users and employees. Currently the used solution is composed by three different pieces of software developed for and by Pangea that try to cover up most of the functionalities that should be provided by a hosting control panel.

2.1.1 Members and billing management

During two years (2004 and 2005) Pangea developed an account management and billing system called GISP (Gestió ISP, ISP Management in English). But the project has been disastrous because after all the work it lacks of basic Pangea's requirements, just a glance:

- GISP does not have any pricing configuration, so in most cases the price must be calculated by hand by the *sales personal*
- Each new year the source code needs to be modified because the current year is hard coded
- Does not make amended invoices nor amended fees

- Each year the state of the orders of the subscription service must be updated via raw SQL queries
- Very poor search options. GISP can not perform search over bills, services or anything else than contact information
- Q19 bank procedure is not fully support, GISP can not handle the Q19 file with returned charges delivered by the bank entity

Moreover the poor design makes it nearly impossible to maintain or reuse. For example these are some of the design principles applied on the GISP database (notice the irony):

- Don't trust on auto-incremental primary keys functionality of your database engine, it is way better create your own tables to do the job
- If a relational database is 'not enough' just create a text field and embed what you need in XML format
- You don't want an intermediary table for many-to-many relations, just use a sequence of primary keys in a typical varchar field

2.1.2 User control panel

During 2006 a student, for his final thesis, tried to develop a control panel working on top of GISP. The project is called AGE¹ (Auto Gestión de Clientes en Pangea, which means clients self-management for Pangea). Unfortunately the only functionality provided by AGE is allowing members to list their invoices and their contact information. We tried adding new functionalities to this panel but in the end, only some basic functionalities were implemented:

- Login system (the original release does not have a working one)
- Web form for creating automated mail responses during holidays (vacation) using Sieve
- Web form for creating blog accounts that worked under Wordpress and wikis with DokuWiki

¹<http://upcommons.upc.edu/pfc/handle/2099.1/5372>

2.1.3 Administration scripts

Currently we are using a set of Perl scripts developed by previous system administrators that helps Pangea's work of creating and deleting accounts for FTP and email services. And that's it, so most of the rudimentary work must to be done manually by sysadmins.

As result of the analysis of the current tools, since their design is not clear and extensible, we have considered that the best option is move forward with a completely new approach. So the next logical step seems to take a look at the existing open source control panels, which is detailed at the following section.

2.2 Outstanding existing solution

As a web hosting control panel is a common need of any hosting company, so before deciding on implementing our own control plane, we should check out if there is an existing solution that fits the requirements or can be easily extended to fit them. A lot of control panels with different properties can be found with a quick search.

Table 2.1 shows a comparison with basic information of the most popular existing control panels compatible with Linux servers.

We are principally interested in the ones that are focused on hosting management (customers and billing support), support Linux servers and are released under an open source license.

Among all of them we consider that there are 3 projects that stand out since they satisfy more or less the Pangea's requirements: (1) SysCP, (2) Domain Technologie Control and (3) ISPConfig. We have installed these three candidates in our test servers and during a few months we have been made an exhaustive evaluation of them. The following sections detail the results and conclusions extracted after this evaluation.

Table 2.1: Main Linux web hosting control panels comparison

Name	License	Billing System	Language	Plugin Support	Multiserv Support
Baifox	GPL	?	PHP, SQLite	?	?
cPanel	Proprietary	Agora	Perl	Yes	Partial
Direct Admin	Proprietary	No	C++, PHP	Yes	?
Domain Technologie Control	GNU LGPL	Yes	PHP	?	No
Gnupanel	GPL	?	PHP	?	?
H-Sphere	Proprietary	Miva	Java	Yes	Yes
HDE Controller X	Proprietary	?	PHP	?	Yes
Hosting Controller	Proprietary	No	.NET, MSSQL	Yes	?
i-MSCP	GPL	?	PHP, MySQL	Yes	No
InterWorx	Proprietary	No	PHP, MySQL	Yes	Yes
ISPConfig	BSD	No	PHP, Perl	Yes	Yes
ispCP	GPL	Yes	PHP, MySQL	No	No
Kloxo	AGPL	Planned	PHP, C++	?	Yes
OpenPanel	GPL	Miva	C++	Yes	Planned
Plesk	Proprietary	Yes	PHP, MySQL	?	?
SysCP	GPL	Yes	PHP, MySQL	Yes	Partial
Froxlор	GPL	Yes	PHP, MySQL	Yes	Partial
Usermin	BSD	No	PHP	?	?
Virtualmin	GPL	Yes	Perl	Yes	Partial
Virtualmin Pro	Proprietary	No	Perl	Yes	Partial
Webmin	BSD	No	Perl	Yes	Yes

Main source from http://en.wikipedia.org/wiki/Comparison_of_web_hosting_control_panels
 More at <http://web-hosting-top.com/directory.control-panels>

2.2.1 SysCP

SysCP² is an open source web hosting control panel written in PHP and with MySQL as a storage backend. The project started in autumn 2003 by Florian Lippert, and two more developers. On 2010 the founder and another core developer left the project. The fact that only a single core-developer continues working on the project gives us little certainty about the future developments of this project³.



The screenshot shows the SysCP web interface. At the top, there is a logo for 'SysCP SERVER MANAGEMENT' and a navigation bar with the text ': : MANAGE YOUR ACCOUNT'. The main content area is divided into a left sidebar and a right main panel. The sidebar contains several sections: 'Main' with links for 'Logged in as: customer1', 'Change password', 'Change language', and 'Logout'; 'E-mail' with links for 'Addresses', 'WebMail', and 'Create e-mail-address'; 'Support tickets' with a link for 'Support tickets'; and 'MySQL' with a link for 'Databases'. The main panel displays two tables. The first table, titled '>> Customer Details', shows the following information: Name: Mustermann Max, Company: Musterfirma, Street: Mustergasse 1, Zipcode/City: 12345 Musterstadt, Email: max@example.org, and Customer ID: (empty). The second table, titled '>> Account Details', shows: Username: customer1, Domain: (empty), Sub-Domain: 0 (unlimited), WebSpace (MB): 0 (1024), Traffic (GB) (Jun 2008): 0 (2), E-mail-Addresses: 1 (5), and E-mail-Accounts: 1 (5).

Figure 2.1: SysCP screenshot

2.2.1.1 Requirements fit

1. Contacts management **Ok**

The contact system fits our requirements.

2. Services **Partial**

Supports most of the required services directly or indirectly through third-party pluggins, except for the complete lack of VPS service support. Also there is no support for shared services. A shared service is a property for some services

²Official SysCP web page <http://www.syscp.org>

³And we were right, no active development has seen since then.

that allows their use and exploitation for multiple different members. An example of this sort of service could be *pangea.org* domain name, if *pangea.org* is configured as shared service other members can create subdomains of it, like *member1.pangea.org* or *member3.pangea.org*.

3. Advanced pricing configurations **Fail**

Syscp does not support all the pricing requirements.

4. Billing system **Fail**

Syscp can be extended through a patch developed by a third-party developer that it supports billing. The billing system is very basic and does not support membership fees and amendment that follow the Spanish regulatory rules.

5. Payment gateway **Fail**

There is no Q19 support.

6. Internationalization **Ok**

SysCP fits the internationalization requirements. It's translated to 8 different languages including Catalan and Spanish.

7. Resources limiting and accounting **Ok**

Good resource control support including Internet traffic, web and mail account disk quota.

8. Target software to support **Partial**

Except OpenVZ, all the other required software is compatible with SysCP.

9. Multi server support **Partial**

Multi-server support is not natively supported in SysCP, but you can split services among different servers with little modifications, unfortunately the same service can not be split between multiple servers.

10. Easy to use for unskilled users **Ok**

The interface is clear and simple.

11. Easy to add new functionalities **Partial**

SysCP has a plugin system for including third-party plugins but it has some limitations since is not posible to change the behaviour of existing parts.

12. Reusable **Fail**

The SysCP is designed for a very specific server's architecture and could be difficult to adapt in an existing infrastructure. Also most core developers complain about the code quality (it is not object oriented) and at the time of the decision they were considering to perform a complete code refactoring.

13. Open source friendly **Ok**

SysCP is released under GNU GPL licence.

2.2.2 Domain Technologie Control (DTC)

Domain Technologie Control (DTC)⁴ is a control panel aiming at commercial hosting. DTC is free software released under the GNU LGPL v2.1 license. It is also the first web hosting control panel that has reached inclusion in major distributions like Debian (since Lenny in 2009), Ubuntu (since 2008) and FreeBSD.

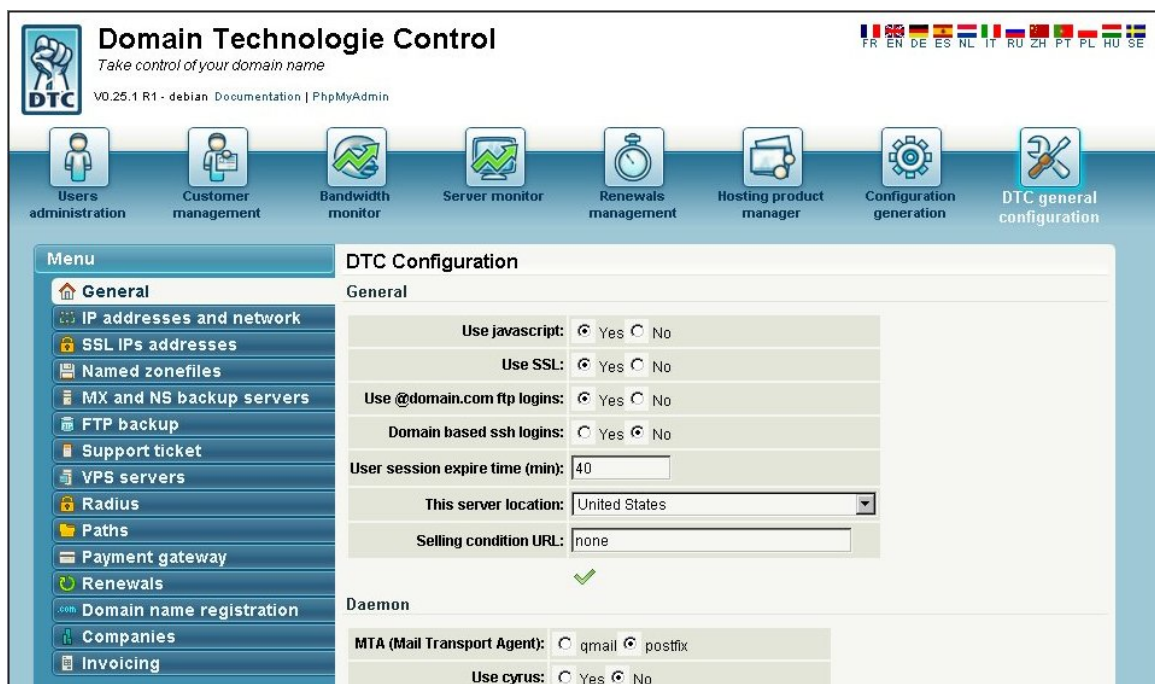


Figure 2.2: DTC screenshot

2.2.2.1 Requirements fit

1. Contacts management **Ok**

The contact system features fits our requirements.

⁴DTC project page <http://www.gplhost.com/software-dtc.html>

2. Services **Partial**
Supports most of the required services except the installation of apps like Blogs or Wikis, and support for shared services is missing too.
3. Advanced pricing configurations **Fail**
DTC does not support all the pricing requirements.
4. Billing system **Fail**
The billing system is very basic and does not support membership fees and fee's amendment according to Spanish regulatory rules.
5. Payment gateway **Fail**
There is no Q19 support.
6. Internationalization **Ok**
DTC is translated up to 11 languages including catalan and spanish.
7. Resources limiting and accounting **Ok**
The resource control support seems to be enough.
8. Target software to support **Partial**
As SysCP, OpenVZ support is missing on DTC.
9. Multi server support **Fail**
Due to the directory structure used on DTC it is not possible to have different services distributed between multiple servers.
10. Easy to use for unskilled users **Fail**
Despite the interface is well designed it has too much options and it is too complicated for most of Pangea's members.
11. Easy to add new functionalities **Fail**
There is no plugin-like architecture on DTC.
12. Reusable **Fail**
DTC has strong server architecture needs and it is really hard to use in an existing infrastructure. Also the source code is a complete mess and most of the best practices on software development seems to be ignored: it is not object oriented, the directory structure is messy and it is impossible to find any kind of documentation oriented to developers.

13. Open source friendly **Ok**
Licensed under LGPL.

2.2.3 ISPConfig 2

ISPConfig 2⁵ is a control panel for Linux server licensed under BSD license. The current release of ISPConfig is ISPConfig 3 and it has a lot more functionalities and it fits more or less the proposed requirements, unfortunately at the time of the decision it was on early alpha development stage and we did not consider it as an option.

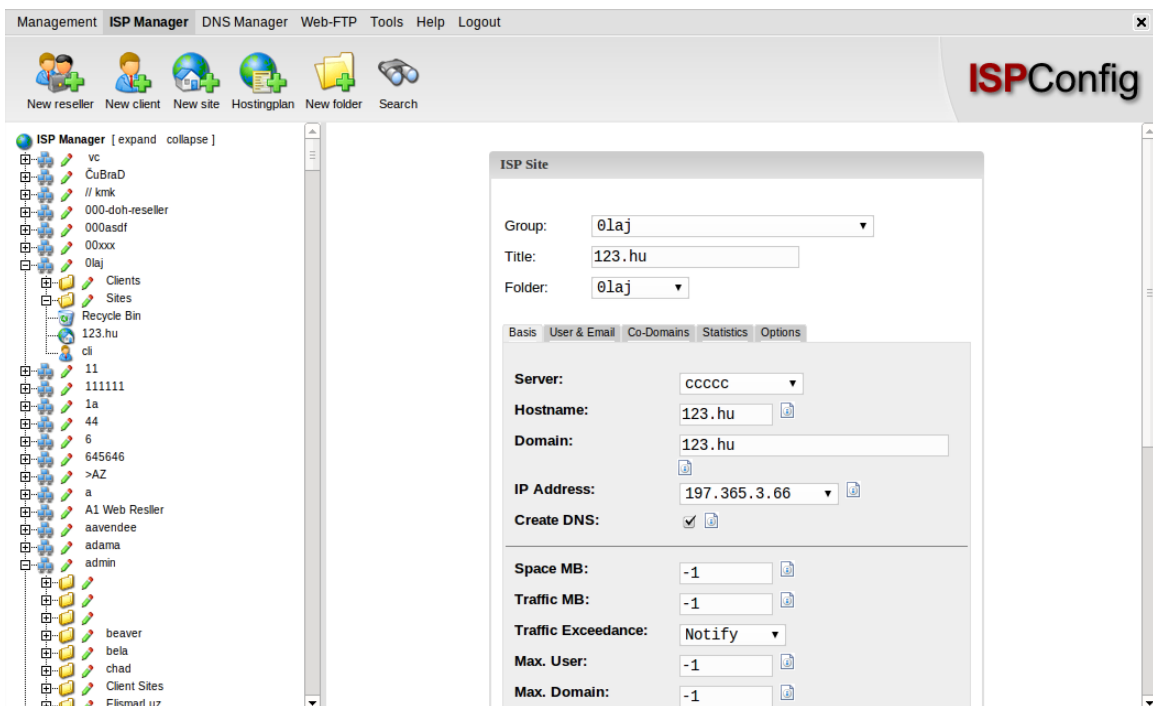


Figure 2.3: ISPConfig 2 screenshot

2.2.3.1 Requirements fit

1. Contacts management **Ok**

The contact system is quite complete.

2. Services **Fail**

Unfortunately, like the other valuated control panels, there is no support for shared services.

3. Advanced pricing configurations **Fail**

DTC does not support all the pricing requirements.

⁵Official ISPConfig2 web page <http://www.ispconfig.org/ispconfig-2>

4. Billing system **Fail**
There is no billing system support on ISPconfig2.
5. Payment gateway **Fail**
Since there is no billing support, neither is payment gateway module.
6. Internationalization **Partial**
ISPConfig2 is translated up to 13 languages but it is not translated to Catalan.
7. Resources limiting and accounting **Ok**
The resource control is enough for most hosting providers.
8. Target software to support **Partial**
Missing support for OpenVZ containers, Mailman mailing lists, Wordpress and Dokuwiki applications.
9. Multi server support **Fail**
There is no multi server support for ISPconfig2.
10. Easy to use for unskilled users **Ok**
One advantage of the lack of functionalities on ISPConfig 2 is that the user interface can not be much complicated.
11. Easy to add new functionalities **Fail**
There is no plugin-like architecture.
12. Reusable **Partial**
It is difficult to install on an existing infrastructure but the source code quality is pretty good and there is a lot of development documentation.

2.2.4 Summary

None of the presented solutions fits all or nearly all requirements, therefore in case we decide on one of those solutions a lot of development should be done in order to achieve the objectives of the project.

Table 2.2 summarizes how the requirements are fit by the reviewed control panels. The easiest project to contribute seems to be ISConfig 2 because of the good quality code and extensive development documentation, but instead is the solution with less features implemented. On the other hand, we have SysCP, wich has most of the functionalities already implemented but its future is completely uncertain because

two of three main core developers has left the project. Finally, DTC is the less preferred solution since it does not support multi server architectures and because the code is quite unfriendly.

Table 2.2: Requirements compliance summary

	Requirement	SysCP	DTC	ISPCConfig2
1	Contacts management	Ok	Ok	Ok
2	Services	Partial	Partial	Fail
3	Advance pricing configurations	Fail	Fail	Fail
4	Billing system	Fail	Fail	Fail
5	Payment gateway	Fail	Fail	Fail
7	Resource limiting and accounting	Ok	Ok	Ok
8	Target software to support	Partial	Partial	Partial
9	Multi server support	Partial	Fail	Fail
10	Easy to use for unskilled users	Ok	Fail	Ok
11	Easy to add new functionalities	Partial	Fail	Fail
12	Reusable	Fail	Fail	Partial
13	Open source friendly	Ok	Ok	Ok

2.3 Final decision

As the current solutions implemented at Pangea are poor designed and difficult to reuse, and at this point the available open source hosting control panels are not convincing, we decide to develop a new control panel that fits all the presented requirements and takes care of the common shortages of the reviewed panels, such as:

- Limited pricing system
- Hard to fit on an existing server infrastructure
- No shared services
- Lack of multi-server support
- No membership fee support
- No Spanish billing regulatory rules
- Not scalable: not suitable for distributed architectures and no any interface seems appropriate for organizations with more than 40 customers

Chapter 3

Selected tools

The developed control panel relies on a set of tools of different kinds. In this chapter each of them is listed and described, as well as the reasons that has lead to the selection of those concrete tools instead of other available ones.

This chapter is presented before the design principles section because Django framework, which has an strong design philosophies behind¹, has made an deep influence on my final design decisions. So in order to preserve the chronological order of the events, let's introduce the Django framework and their friends.

3.1 Django framework

Django² is an open source web application framework, written in Python, which follows the model-view-controller architectural pattern³. It was originally developed to manage several news-oriented sites for The World Company of Lawrence, Kansas, and was released publicly under a BSD license in July 2005.

3.1.1 Framework advantages

We decide on a framework instead of directly using a language and their libraries because it is more easy and faster to create web sites with the correct framework. This is because a web-related framework already integrates all the common parts needed for building web sites and it also helps us on the decision taking of some critical and difficult design decision. Another reason is that we can get new features for free in each new framework release. Also a project developed using a popular framework is

¹Django design philosophies <http://docs.djangoproject.com/en/dev/misc/design-philosophies>

²Official Django site <http://djangoproject.com>

³Discussed ahead on chapter 4 *Design principles* section 4.2 *Orthogonality*.

easy to maintain because it is possible to find developers that are already familiarized with the parts of the project design which belongs to the framework.

3.1.2 Django advantages

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes on reusability and "pluggability" of components and rapid development. Moreover, it comes with a pretty useful functionalities out of the box:

- **Object-relational mapper**

It allows to define your data models⁴ entirely in Python providing a rich, dynamic database-access API for free, but it is still possible to write raw SQL queries if needed.

- **Automatic admin interface**

Django can create an admin interface based on the application model description. It is completely automatic and it is production-ready and very customizable. We base our admin interface entirely on this feature.

- **Elegant URL design**

Allowing to design pretty and cruft-free URLs⁵ with no framework-specific limitations. Giving you the possibility of being as flexible as you like.

- **Template system**

Use Django's powerful, extensible and designer-friendly template language to separate design, content and Python code. Apart from generating HTML pages, this template system is interesting for create the PDF version of the bills as well as for dynamically generate shell scripts that performs the management tasks on the hosting servers.

- **Internationalization**

Django has full support for multi-language applications, letting you specify translation strings and providing hooks for language-specific functionality.

⁴Data models provide a structure for data used within information systems by providing specific definition and format. The data model will normally consist of entity types, attributes, relationships, integrity rules, and the definitions of those objects.

⁵Without the characters which are relevant or meaningful only to the people who created the site, such as implementation details of the computer system which serves the page, such as the .php or .html file extensions or internal organizational details such as /public/ or / users/john/work/drafts/.

- **Reusable Apps**

In Django context the projects are build putting different applications together, normally an application encapsulates some kind of functionality, in our case each app will implement an entire subsection of our *1.3 requirements analysis*: one app for *1.3.1 contacts management*, another for *1.3.4 billing system*, and so on. A reusable application are those apps designed in such a way that it could be directly used in any project that requires this kind of functionality. There are plenty of open source reusable applications out there, so we can take advantage of the existing third-party Django apps integrating them into our project, avoiding reinventing the wheel.

- **Multiplatform and multi-database support**

With these features the control panel can run on any operating system that has Python support and switch the database back-end without changing a single line of code.

- **Database transactions**

A database transaction is a logical unit of database operations which are executed as a whole to process user requests for retrieving data or updating the database. Django provides a few ways to control how database transactions are managed, if you are using a database that supports transactions. We can rely on this Django feature in order to implement the, sometimes difficult, task of ensuring data consistency in case of hardware or software failures.

- **Extensive documentation**

In contrast to other open source frameworks Django has a very detailed and up-to-date documentation. Also, the Django users' community is very active on their own blogs or on the Django users mailing list, that is full of developers sharing knowledge and helping others.

Finally you do not get tied to Django since it is extremely modular and any of the components provided by the framework can be easily switched. The ORM or the template system can be changed for other ones, such as SQLAlchemy⁶ or Jinja2⁷. Also Django can work together with other frameworks like jQuery⁸.

⁶SQLAlchemy is the Python SQL toolkit and Object Relational Mapper

⁷Jinja2 is a modern and designer friendly templating language for Python.

⁸jQuery is a cross-browser JavaScript library designed to simplify the HTML client-side scripting

3.2 Celery distributed task queue

Celery⁹ is an open source asynchronous task/job queue based on distributed message passing¹⁰ used in production systems to process millions of tasks a day. It is focused on real-time operation, but supports scheduling as well. Celery supports multiple broker agents but it is specially designed to work with RabbitMQ.

Celery is easy to integrate with Django using django-celery app. This application uses the Django ORM and cache backend for storing results. Figure 3.1 represents a high level overview of its architecture.

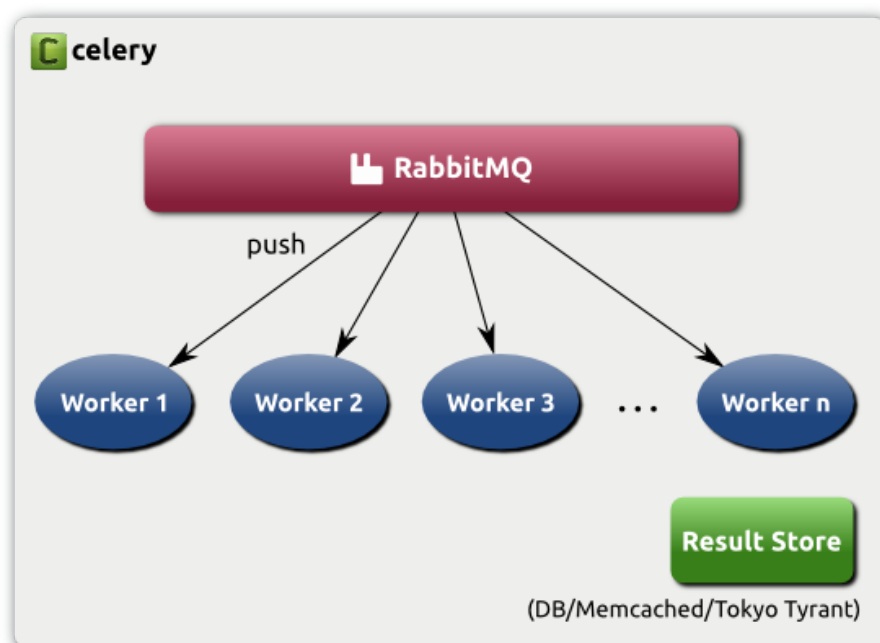


Figure 3.1: Celery architecture overview

Basically the broker delivers tasks to the worker nodes. A worker node is a networked machine running `celeryd`. This can be one or more machines depending on the workload. The result of the task can be stored for later retrieval¹¹.

- **Why do we need an asynchronous task queue?**

The panel has to execute some tasks that are quite expensive computationally speaking, such as executing scripts on the server side and gathering the results, calculating prices and generating invoices. In order not to impact the user

⁹Celery web site <http://ask.github.com/celery>

¹⁰More at youtu.be/o3TuRs9ANhs Google I/O 2009 - Offline Processing on App Engine

¹¹Source at <http://ask.github.com/celery/getting-started/introduction.html>

experience with long locks waiting for a task to end, we want to run those tasks asynchronously.

- **Why not using cron jobs?**

The use of crontab for the execution of asynchronous tasks is widely used in other control panels. They normally schedule a procedure that every 10 minutes polls the database looking for changes and triggers specific scripts in order to reflect these changes on the server. However this approach has some drawbacks:

- It is not triggered when changes happen
- It is not bundled on the application, there is a need to have some definitions on the OS crontab
- If a cron job fails it is not repeated until the next schedule comes
- The tasks are performed on a single machine, therefore it is hardly scalable
- Less control: it is not possible to revoke pending operations and reschedule faulty ones

3.3 Pisa PDF generator

Pisa¹² is a HTML to PDF converter that uses the ReportLab Toolkit, HTML5lib and pyPdf. Pisa is easy to use and well integrated with Django. We use Django templates and HTML/CSS in order to define how the bills look like. Then the template is rendered with an specific context¹³ and finally passed to the `pisa.pisaDocument()` that creates a PDF version of it.

3.4 Django admin tools

Django-admin-tools¹⁴ is a collection of tools that improve some parts of the Django administration interface, it includes:

- a fully featured and customizable dashboard

¹²Pisa web page <http://xhtml2pdf.com>

¹³A context is a “variable name”: “variable value” mapping that is passed to a template. A template renders a context by replacing the variable ”holes” with values from the context and executing all block tags.

¹⁴Django-admin-tools project page <http://bitbucket.org/izi/django-admin-tools>



Figure 3.2: Django admin tools drag and drop detail

- a customizable menu bar

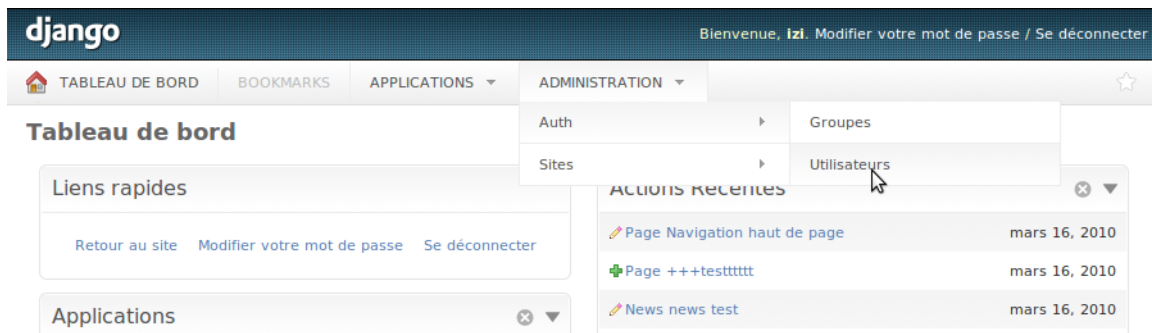


Figure 3.3: Django admin tools menu detail

- tools to make admin user interface theming easier

Chapter 4

Design principles

In this section we describe the main design patterns and decisions on which the implementation is based. As we early discuss in this document some of the design decisions has been already taken by choosing the Django framework. The design decisions have been made prioritizing the pluggability and reusability of the system. We have designed the project with the idea of creating a framework for the easy creation of web hosting control panels more than designing a control panel itself.

4.1 Reusability

The control panel design is highly focused on reusability. We understand reusability as the ability of the system, or some of their parts, to be used in other projects. We implement reusability using two design patterns: (1) Separation of concerns and (2) Loose coupling.

4.1.1 Separation of Concerns (SoC)

Django has well integrated this pattern on its project design. With Django you are encouraged to split your project into more specific and focused applications, commonly known as a Django reusable applications.

A reusable Django app, is an application that is easily plugged into any project, providing a very specific piece of functionality. They should be focused and follow the Unix philosophy of “do one thing and do it well”. James Bennett¹ likes to say that an application encapsulates some particular feature².

¹James is Django's release manager, and a big contributor to the Django documentation

²James Bennett at DjangoCon 2008: Reusable Apps. youtu.be/A-S0tqpPga4

4.1.2 Loose coupling

The different applications of the project should not know about others, unless it is absolutely necessary, that is, loose coupling and tight cohesion. If we write applications with this principle in mind we ensure that each app will be independent from each other. This independence gives us two potential benefits:

1. Easy development and future maintenance
2. Make it reusable for other projects

Some underlying patterns of this principle that we widely use on this project are:

- **Signals** (publish/subscribe)

Django includes a “signal dispatcher” which helps on allowing decoupled applications to get notified when actions occur elsewhere in the framework. In a nutshell, signals allow certain senders (publishers) to notify a set of receivers (subscribers) that some action has taken place. They are especially useful when many pieces of code may be interested in the same events.

- **Registration pattern**

The registration pattern³ is used on an application in order to notify their existence to other applications. That way the other application can take some decisions based on this information. For example, with this pattern we can enable or disable the admin interface for a certain application.

- **Backend**

The goal is to create a *driver* for an specific app that knows how to talk with the API of another application. This is used normally to decouple two applications that have a highly coupled behavior, such as, for example, Order and Billing apps. With the purpose of bill an order the Order application must know how to tell the billing system that it wants to create a bill, this should go on the backend file. If you want to switch your billing application the only thing that you should do is change the backend file in order to match with the new billing API.

- **Generic relations**

Common foreign keys⁴ can only “point to” one other model, which means that

³More on <http://charlesleifer.com/blog/looking-registration-patterns-django/>

⁴In the context of Django, a foreign key is a referential constraint between two models. It is implemented with a database field in a relational table that matches a candidate key of another table

if a model uses a *ForeignKey* it would have to choose one and only one model to point to.

The Django ContentTypes⁵ application provides a special field type (`GenericForeignKey`) which works around this and allows a relationship to point to any model.

- **Modify the behavior of another app at runtime**

Python is a language with a very high level dynamic classes, therefore it is very easy to modify classes and their methods at runtime. One application can deeply change the behavior of classes from another application, and without knowledge of it.

The idea behind the reusability principle is to create a generic set of reusable applications that covers the different needs of a web hosting control panel. Then others can build their own control panel reusing a subset of these apps.

4.2 Orthogonality

“Orthogonality is a system design property facilitating feasibility and compactness of complex designs. Orthogonality guarantees that modifying the technical effect produced by a component of a system neither creates nor propagates side effects to other components of the system.”⁶

The Django apps are designed using a variant of a model-view-controller (MVC)⁷ pattern called model-template-view (MTV). As the MVC pattern does, MTV isolates the business logic from the user interface, permitting independent development, testing and maintenance of each part.

Figure 4.1 shows the Django representation of the MTV design pattern.

⁵Django includes a contenttypes application that can track all of the models installed in your Django-powered project, providing a high-level, generic interface for working with your models

⁶http://en.wikipedia.org/wiki/Orthogonality#Computer_science

⁷More at <http://en.wikipedia.org/wiki/Model-view-controller>

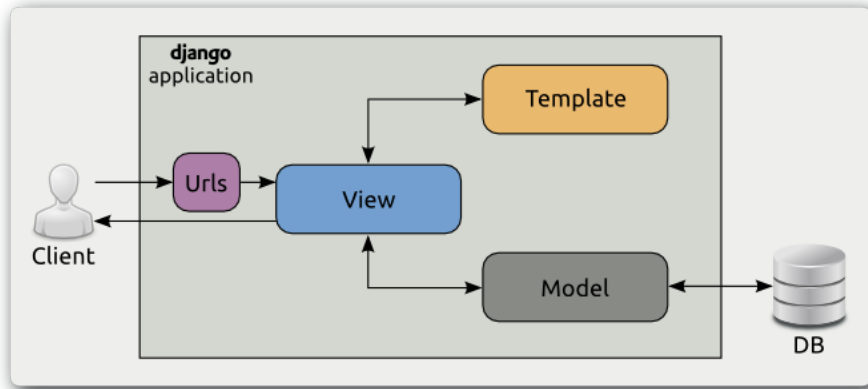


Figure 4.1: Django Model-Template-View overview diagram

- The **Model** layer implements a description of the database table, represented by a Python class. It is responsible for creating, retrieving, updating and deleting records in the database using simple Python code rather than writing repetitive SQL statements. This layer is the same as the MVC pattern and it is implemented on the `models.py` application file.
- The **View** layer performs the business logic⁸ for the web pages. It is implemented on the `views.py` application file.
- The **Urls** module is responsible to map each URL to a certain view function. It should be specified on the `urls.py` file. The Urls system together with View layer does the same job as the controller on the traditional MVC pattern.
- The **Template** layer is implemented in different files within the template app directory. These files describe the design of the page so this layer is the same as the View layer on the MVC pattern. A template file uses the Django template language with basic logic statements.

4.3 Adaptability

This project is designed with the philosophy of *must be the control panel which suits IT infrastructure instead of infrastructure fitting the panel*. The downside of this approach is that in most cases the sysadmins will need to modify the server management scripts in order to make it fully compatible with their software. This is not necessarily

⁸Business logic, or domain logic, is a non-technical term generally used to describe the functional algorithms that handle information exchange between a database and a user interface

a big deal since most system administrators love writing their own shell scripts, and moreover we will use the Django template system to ease this task.

4.4 Extensibility

The control panel is designed to provide the basic and generic functionalities by default and move the specifics to extensions. Extensions can be through (1) the modification of existing functionality or through (2) the addition of new functionality. “The central theme is to provide for change, atypically enhancements, while minimizing impact to existing system functions.”⁹

(1) In systems architecture, extensibility means the system is designed to include hooks and mechanisms for expanding/enhancing the system with new capabilities without having to make major changes to the system infrastructure.

We have been implemented a plugin mechanism that allows to easy extend the control panel services. For example, the web app only brings support for the basic VirtualHost configuration, and support for *mod_php* or *mod_proxy* configuration can be provided though a plugin. We provide two base classes that helps with the plugin creation task: `PluginServiceModel` and `PluginServiceAdmin`.

Also the system must facilitate as much as possible the task of adding new services. We provide an inheritance based approach that makes easier the task of implementing support for new types of services. You only need to inherit from `BaseService` class and define the service specifics, `BaseService` does the rest.

(2) Extensibility can also mean that a software system’s behavior is modifiable at runtime, without recompiling or changing the original source code. Our control panel has an admin interface that allows to modify the behavior of the control panel without touching any single line of code. For example it is possible to describe the resources used for a concrete service and the web interface forms will be modified at runtime in order to reflect these resource descriptions. Another example is the ability of our panel to dynamically add an arbitrary extra field to the services, useful when an organization has a very specific need for a concrete service, like store an additional field on the contact information.

⁹Quote from <http://en.wikipedia.org/wiki/Extensibility>

4.5 DRY Don't repeat yourself

This software development principle aims at reducing repetition of information of all kinds. The DRY principle is stated as “Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.”

When the DRY principle is applied successfully, a modification of any single element of a system does not require a change in other logically-unrelated elements. Additionally, elements that are logically related all change predictably and uniformly, and are thus kept in sync. This principle is especially useful on model-view-controller architectures since both pursue guarantee that changes in one part of the system does not affect others.

Chapter 5

Architecture design and implementation

In this chapter we are going to see how we have implemented the control panel requirements reviewed on section 1.3 using the development tools that we have presented on chapter 3 and following the design principles presented in the previous chapter.

As one of the design principle was to split the project into small applications this chapter is also divided in the defined applications. Also an overview of different applications working together is presented on the last section: *5.17 Global interactions*.

5.1 General considerations

Before dealing with the different applications that compose the project we want to present some general points that are used on the implementation of several developed apps.

- Commit on success. The views susceptible to perform any database change run under a commit on success database transaction scope. It means that if all the code runs successfully, the panel will commit the work done. Otherwise if the code raises an exception then the panel will roll back the transaction.
- The admin interface relies entirely on `django.contrib.admin`. We reuse the `django.contrib.admin` application for the web interface by subclassing its `ModelAdmin` class. `ModelAdmin` class is very flexible, it has several options for dealing with customizing the interface¹. All the admin interface related code lives on the `app_name/admin.py` file. The automatic Django admin interface is composed by three main parts:

¹Check available configurations at <https://docs.djangoproject.com/en/dev/ref/contrib/admin>

1. The admin **home** page, figure 5.1. This page lists all the available types of data (models) that can be edited through the admin site.
2. Admin **change list**, figure 5.2. Each type of data in the Django admin site has a change list that displays all the available objects in the database. The basic workflow of Django's admin is “select an object, then change it.”, so the change list page lets you to perform a certain action over a set of selected objects as you can see on figure 5.3.
3. **Change** and **add forms**, figure 5.4. Used to modify existing objects and create new ones. Each field defined in the application model appears here, and fields of different types get different widgets. Also every change made through the admin interface is logged. You can examine this log by clicking the history link in the upper-right corner of the window.

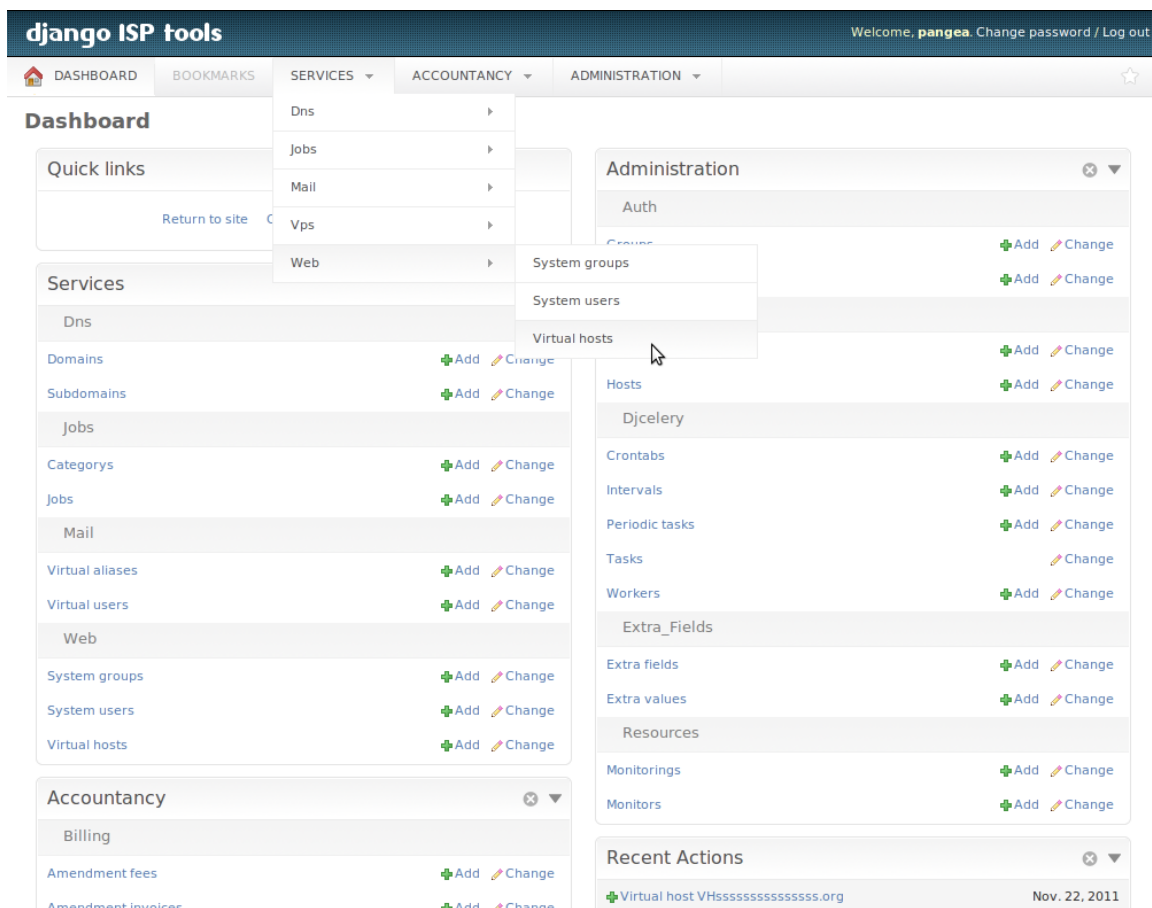


Figure 5.1: Admin home page screenshot

django ISP tools Welcome, **pangea**. [Change password](#) / [Log out](#)

[DASHBOARD](#) [BOOKMARKS](#) [SERVICES](#) [ACCOUNTANCY](#) [ADMINISTRATION](#)

Home > Dns > Domains

✔ The domain "t1000.org" was added successfully.

Select domain to change

[Recover deleted domains](#) [Add domain](#) **+**

Action: Go 0 of 12 selected

<input type="checkbox"/>	Name	Extension	Is mail	Is hosted	Is purchased	Active	Contact Link
<input type="checkbox"/>	ssssssssssssss	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Federacio Catalana de sucs de taronja
<input type="checkbox"/>	holaaa	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Jaume prats
<input type="checkbox"/>	samcro	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	SAMCRO
<input type="checkbox"/>	fringe_division	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Walter Bishop
<input type="checkbox"/>	wmca	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Olivia Dunham
<input type="checkbox"/>	lost	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Dharma Initiative
<input type="checkbox"/>	pangea	es	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Pangea
<input type="checkbox"/>	futurama	net	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Bender Doblador Rodriguez
<input type="checkbox"/>	multiplexer2001	es	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Bender Doblador Rodriguez
<input type="checkbox"/>	dellcomputers	info	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Planet Express
<input type="checkbox"/>	moriarty	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Olivia Dunham
<input type="checkbox"/>	t1000	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Walter Bishop

12 domains [Save](#)

Filter

By active
All
Yes
No

By extension
All
org
net
es
info

By is mail
All
Yes
No

By is hosted
All
Yes
No

By is purchased
All
Yes
No

Figure 5.2: Admin change list screenshot

django ISP tools

[DASHBOARD](#) [BOOKMARKS](#) [SERVICES](#) [ACCOUNTANCY](#) [ADMINISTRATION](#)

Home > Contacts > Contracts

Select contract to change

Search

< All dates **October 2011** November 2011 December 2011

Action: Go 5 of 41 selected

<input type="checkbox"/>	Contract	Contact	Content type	Content Object
<input checked="" type="checkbox"/>	con...	Pangea	contact	Pangea
<input checked="" type="checkbox"/>	use...	Pangea	user	pangea
<input checked="" type="checkbox"/>	sys...	Pangea	system group	pangea
<input type="checkbox"/>	system user(pangea)	Pangea	system user	pangea
<input type="checkbox"/>	virtual host(VHssssssssssss.org)	Pangea	virtual host	VHssssssssssss.org
<input checked="" type="checkbox"/>	subdomain(www.pangea.es)	Pangea	subdomain	www.pangea.es
<input type="checkbox"/>	domain(pangea.es)	Pangea	domain	pangea.es
<input checked="" type="checkbox"/>	contact(Jaume)	Jaume	contact	Jaume

Bill contracts

Delete selected contracts

Cancel contract

Schedule cancelation

Annul cancelations

Bill contracts

Schedule deactivation

Annul deactivations

Figure 5.3: Admin actions screenshot

Change domain of contact Walter Bishop Contract History

Name: Extension:

Expire: Today |

Active

Is mail

Is hosted
Host the DNS Zone

Is purchased
Purchase this domain

Optional Fields (Show)

Subdomains

Active	Is mail	Name	Delete?
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="text" value="www"/>	<input type="checkbox"/>

+ Add another Subdomain

Records

Type	Value	Delete?
t1000.org web.pangea.org. NS	<input type="text" value="web.pangea.org."/>	<input type="checkbox"/>
t1000.org mail.pangea.org. NS	<input type="text" value="mail.pangea.org."/>	<input type="checkbox"/>
t1000.org 77.246.179.81 A: IPv4 Address	<input type="text" value="77.240.170.78"/>	<input type="checkbox"/>
t1000.org 10 mail.pangea.org. MX	<input type="text" value="10 mail.pangea.org."/>	<input type="checkbox"/>
t1000.org 20 mail2.pangea.org. MX	<input type="text" value="20 mail2.pangea.org."/>	<input type="checkbox"/>

+ Add another Record

Figure 5.4: Admin change form screenshot

- Each application has its own `settings.py` file where all the configuration specifics are defined. This way we can provide generic applications that can be reused by others after editing the settings file.
- We make extensive use of Django templates, not just for generating the HTML web pages but also for the bills and the server side scripts. See appendix A.1.
- The dynamic forms are those web forms created at runtime because until then we do not know how they should look like. We use Python *metaclasses* for creating them. A *metaclass* is a class whose instances are classes. Just as an ordinary class defines the behavior of certain objects, a metaclass defines the behavior of certain classes and their instances.

- Usually each application is presented broken down into two subsections: (1) Model of the application and (2) Admin interface of the application.

Figure 5.5 presents an overview diagram of the most visible applications that compose the panel.

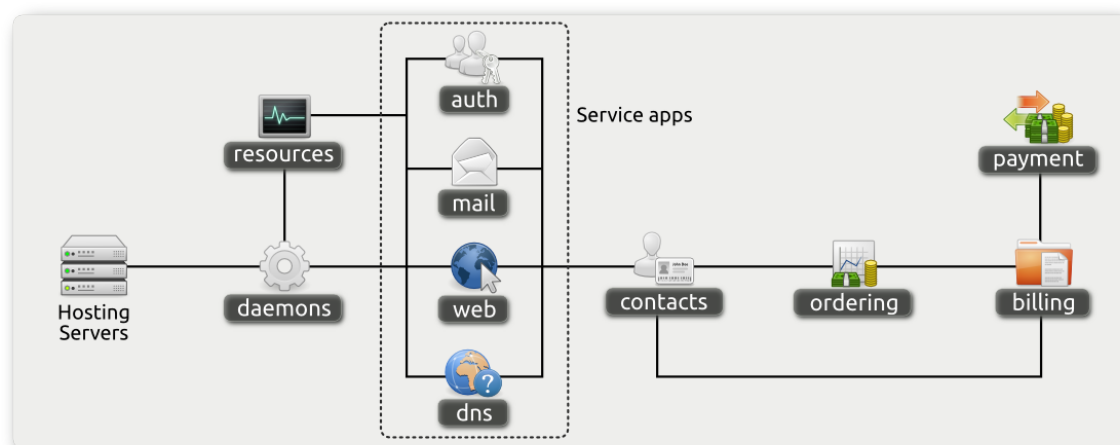


Figure 5.5: Applications architecture overview

First we are going to present the core application called common (not represented on the diagram 5.5) responsible of sticking together the other project applications. The next set of applications that we present are those with system related work: daemons and resources. Then we dive into the accountant-related apps, such as contacts, ordering and billing. Finally we will take a look at the services applications, which includes the authentication system, mail, web, DNS, and jobs. Please notice that we make references to this set of applications down the line using the term *service applications* or *service apps*.

Generally speaking we think that documentation goes against DRY principle, since the same concept is in two places at the same time (in the source code and in the documentation). Therefore, documentation might easily become outdated because of that. Also, considering that in software development the time is an scarce resource, we think that it is worth spending more time creating a high quality code or tests than spend it writing an extensive documentation, good code with good naming convention does not need to be extensively documented. We'd like to just provide a high level description of each module and leave the low-level details on the source code. If you find this documentation too general please review the source code for more details. Also you can use the python `help()` utility to review the API of each application.

5.2 Common

Common app is the core of the project, since it provides the common utilities for creating the service applications and other tools commonly used by the different panel apps. Therefore it is mandatory to install it if you want to reuse some other project app.

This core application consists on:

- **BaseService**, which is a base model class² for all service models
- **BaseService**, which is a base model class for all service plugin models
- **ServiceAdmin** and other base admin classes for service providing admin support through means of inheritance
- A collection of utilities and common functions used by multiple control panel apps, for example the factory class that builds dynamic forms

BaseService and **BasePlugin** abstract models implement the plugin mechanism that allows extending a service with new functionalities. The service activation and deactivation feature is implemented by **BaseService** base class. Also inheriting from **BaseService** is like labeling the subclass, because some applications rely on the fact that if one of your parents is **BaseService** then you are a service and you should be treated consequently.

ServiceAdmin model implements admin interface hooks for service applications. A practical example can be the **ServerName** field from the *add form* when adding a *VirtualHost*. By default Django shows a select widget with all existing domains, but if you have the contacts application installed it overrides this behavior and shows only the domains owned by the concerned customer.

ServiceAdmin model also provides some useful signals that alert other applications that something has changed in the service through the admin interface:

- **service_created()** Sent when a new service has been created, e.g. a new e-mail account
- **service_updated()** Sent when a service has been modified
- **service_deleted()** Sent when a service has been deleted

²A superclass, base class, or parent class is a class from which other classes have been derived

How easy is to create a new service application using this classes?

1. Execute `python manage.py startapp startapp lists` in order to create the skeleton of the new application, a mailing list in our example case
2. Edit `lists/models.py` and add the following lines:

```
from common.models import BaseService
from dns.models import Name

class List(BaseService):
    name = models.CharField(max_length=32)
    domain = models.ForeignKey(Name)
    admin_password = models.CharField(max_length=16)
```

3. Create the file `lists/admin.py` with this content:

```
from django.contrib import admin
from lists.models import List
from common.admin import ServiceAdmin

class ListAdmin(ServiceAdmin):
    pass

admin.site.register(List, ListAdmin)
```

These few lines of code just have created a new service with a full admin panel integration. On figure 5.6 you can see how the admin interface for this new service looks like. Please notice that this is the default appearance and it can be completely customized with few lines of code³.

³docs.djangoproject.com/en/dev/ref/contrib/admin/

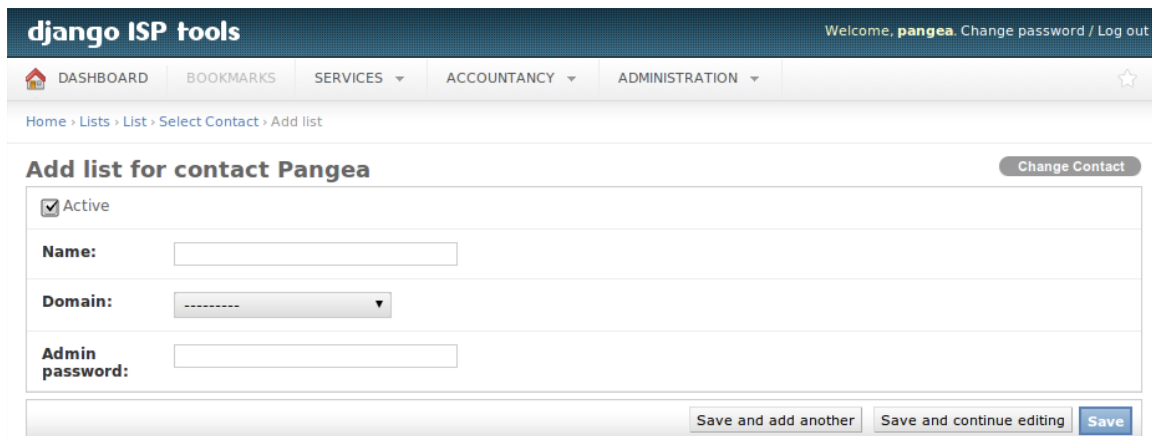


Figure 5.6: Screenshot of list add form with default looking

5.3 Daemons

The Django ORM uses two different data modification methods: save and delete. This application performs these data modifications on the server side.

In a UNIX like operative system a daemon is a computer program that runs as a background process. Practically all of the process running on the servers providing services on a hosting company are daemons. So we have decided to name this application daemons in order to avoid confusion with other usual names like services or servers.

5.3.1 Daemons Model

The model of this application consists of:

- **Daemon** model, that is the representation of a running process that is responsible of the service provisioning on the server side, so a **Daemon** defines how to perform the panel save and delete operations on the servers
- **Host** model, which identifies the machine (remote or local) where **Daemon** is running

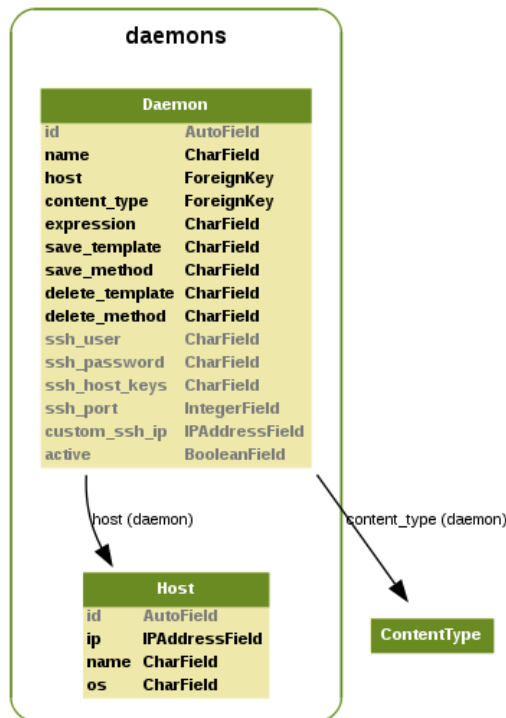


Figure 5.7: Daemons application model

The workflow of this application can be defined by three steps: (1) detect save and delete service operations, (2) identify the daemon and host based on service object and (3) execute the scripts.

1. Detect save and delete service operations

The daemon application takes advantage of the signals that `AdminService` sends every time that a service is created, deleted or modified through the admin interface. When the daemons app receives a `service_inserted()` or `service_update()`, it is interpreted as a save operation, and when the signal is `service_deleted()` as a delete operation.

2. Identify the daemon and host based on service object

We could have the web sharing hosting service split into 3 different servers. It is the daemon application responsibility to identify in what server resides each web, for example `www.pangea.org` is hosted on `web.pangea.org` and `ucp.pangea.org` is hosted on `web3.pangea.org`. To work around this the server lookup is done using a combination of `ContentType` (Model level) framework and `expressions` (Instance level).

- Model Level: The *ContentType* framework is used to identify which is the service model represented by the Daemon, `webs.VirtualHost` for our web hosting example.
- Instance Level: When we have a service distributed over multiple servers multiple daemon instances must be defined too, one for each server. Therefore the system must be able to identify in what server lives each service instance. An *expression* approach is used to solve this problem. It gives the flexibility to make rules by service instance attributes combined with Python logical and arithmetic operations. Let's introduce the *expression* mechanism with an example:

`0['pk']%3==0`

0 is a model instance converted to a Python dictionary so we have access to their attributes using this syntax `0['attribute_name']`. This expression example will match all *VirtualHosts* with primary key (pk) mod 3. So with other 2 daemons with expressions `0['pk']%3==1` and `0['pk']%3==2` we get nothing more than a round-robin distribution between those 3 daemons.

3. Execute the scripts

This application performs the data modifications on the server side executing the `save_template`⁴ and the `delete_template`. These Django templates can be executed using three different methods:

- (a) Execute as script over SSH.

99.92% of the *nix⁵ servers around have the SSH service installed. So we take advantage of this, offering the ability to execute the scripts through SSH without the need of installing anything on the server side. We use `paramiko`⁶ for managing SSH connections with Python.
- (b) Execute as python code.

This option is interesting if we want to run some Python code that wants to access at the control panel model using Django ORM.
- (c) Execute the template as local script.

This is useful if we want to run any other remote access different than SSH,

⁴See an example on Appendix A.1

⁵A Unix-like (sometimes referred to as UN*X or *nix) operating system is one that behaves in a manner similar to a Unix system

⁶<http://www.lag.net/paramiko/>

or just run the script locally because the server that we want to manage is *localhost*.

This approach is multi platform friendly because we can execute virtually anything. Bash, Perl, Ruby, Python, C, .NET. The only thing we need to do is to write the scripts using the Django template system, see appendix A.1. The script must be idempotent, that is, written in such a way that you may run the same operation more than once and achieve the same result. Otherwise you can run in troubles if the script fails and it is running partially multiple times.

The execution of the templates is performed asynchronously using Django Celery message queue. So the user request does not get stuck waiting for the task's end and the administrator can easily manage this task execution via the control panel interface [see figure 5.9].

5.3.2 Daemons Admin

The actions provided for the daemons are available on its admin change list page [fig. 5.8]:

- Delete selected daemons
- Enable selected daemons
- Disable selected daemons

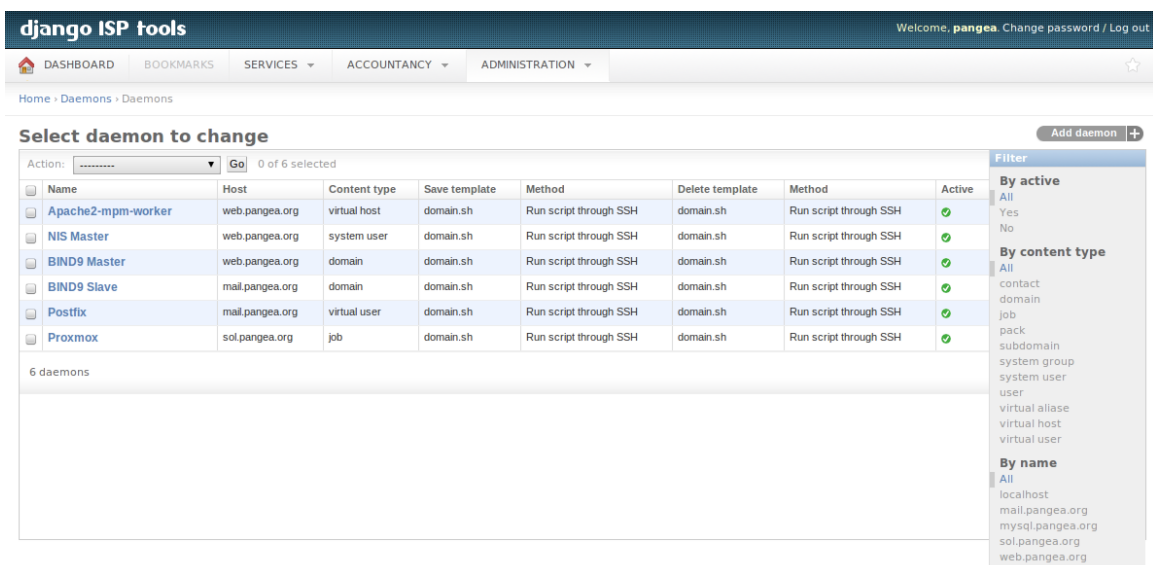


Figure 5.8: Daemons admin change list

UUID	State	Name	Args	Kwargs	ETA	When	Worker
02749f53-8862-4060-a888-6300c6e194df	FAILURE	LOCAL-Script	{'*/bin/bash\\n\\necho "ssssssssssss.org. IN SOA web.pan...	{}	none	just now	w1.bestia
736f51dd-fe2d-462b-aec2-a0dd3ac25585	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "ssssssssssss.org. IN SOA web.pan...	{}	none	7 minutes ago	w1.bestia
a25b6787-cbb5-41ff-98f9-7c586666c591	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "ssssssssssss.org. IN SOA web.pan...	{}	none	8 minutes ago	w1.bestia
5a8e5726-2bda-4307-8da5-25e22215f245	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "potato.org. IN SOA web.pangea.ORG. ...	{}	none	10 minutes ago	w1.bestia
99a1bb33-b4bb-4a0f-8526-78baa2a5727	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "kakas.org. IN SOA web.pangea.ORG. m...	{}	none	10 minutes ago	w1.bestia
5334c9cf-383a-4004-a232-e57c4fcaa498	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "jax_teller_m.org. IN SOA web.pangea...	{}	none	10 minutes ago	w1.bestia
dae5c833-bfcb-4e1e-8794-0fc5aa920c9b	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "t1000.org. IN SOA web.pangea.ORG. m...	{}	none	10 minutes ago	w1.bestia
3b813259-4887-46fa-9bfa-52ab103b80a	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "moriarty.org. IN SOA web.pangea.ORG...	{}	none	10 minutes ago	w1.bestia
40f4064f-d8b-4470-a5cd-6ba52dde1e52	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "dellcomputers.info. IN SOA web.pang...	{}	none	10 minutes ago	w1.bestia
0a39c5a4-0d13-46e2-b22d-c36f1c15253c	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "dellcomputers.info. IN SOA web.pang...	{}	none	10 minutes ago	w1.bestia
323c9c05-ac0-443d-b4b4-eaf32af15162	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "multiplexer2001.es. IN SOA web.pang...	{}	none	10 minutes ago	w1.bestia
20f8609c-b0ea-43e5-a53d-d7c760e4069d	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "futurema.net. IN SOA web.pangea.ORG...	{}	none	10 minutes ago	w1.bestia
d09a48f3-9152-4fe1-aa4a-e801fa87e164	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "pangea.es. IN SOA web.pangea.ORG. m...	{}	none	10 minutes ago	w1.bestia
1aa84eb2-2795-40d9-a4be-a4c81fb9fa88	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "lost.org. IN SOA web.pangea.ORG. ma...	{}	none	10 minutes ago	w1.bestia
fcbbcd93-810c-420a-824b-f23029560833	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "mmsc.org. IN SOA web.pangea.ORG. ma...	{}	none	10 minutes ago	w1.bestia
0bd4146-ff3c-4d25-829d-04f0a09b6498	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "holaaa.org. IN SOA web.pangea.ORG. ...	{}	none	10 minutes ago	w1.bestia
fe71f61b-cae5-48ec-9e60-0a5b7e08cb21	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "fringe_division.org. IN SOA web.pan...	{}	none	10 minutes ago	w1.bestia
ba8e59d5-aa63-4a40-8c4e-f3f292cd4d4	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "ssssssssssss.org. IN SOA web.pan...	{}	none	10 minutes ago	w1.bestia
8a083dbb-088e-43e3-af7b-e1cdabb31d792	SUCCESS	LOCAL-Script	{'*/bin/bash\\n\\necho "samcro2.org. IN SOA web.pangea.ORG....	{}	none	11 minutes ago	w1.bestia

Figure 5.9: Django-celery admin change list

django ISP tools		Welcome, pangea. Change password / Log out
DASHBOARD BOOKMARKS SERVICES ACCOUNTANCY ADMINISTRATION		
Home > Daemons > Daemons > Apache2-mpm-worker		
Change daemon		History
Name:	Apache2-mpm-worker	<input checked="" type="checkbox"/> Active
Manager		
Host:	web.pangea.org	
Content type:	virtual host	
Expression:	True	
Driver		
Save template:	domain.sh	Method: Run script through SSH
Delete template:	domain.sh	Method: Run script through SSH
SSH Options (Hide)		
SSH User:	root	SSH Password: *****
SSH Host Keys:	/home/glic3rinu/.ssh/known_hosts	
SSH IP:		SSH Port: 22
<input type="button" value="Delete"/> <input type="button" value="Save and add another"/> <input type="button" value="Save and continue editing"/> <input type="button" value="Save"/>		

Figure 5.10: Daemons admin change form

5.4 Resources

This application is responsible of monitoring, recording and limiting the resource consumption of an specific service on the IT infrastructure. A common resource could be CPU cycles, memory usage, number of process, Internet transit or swap space.

5.4.1 Resources Model

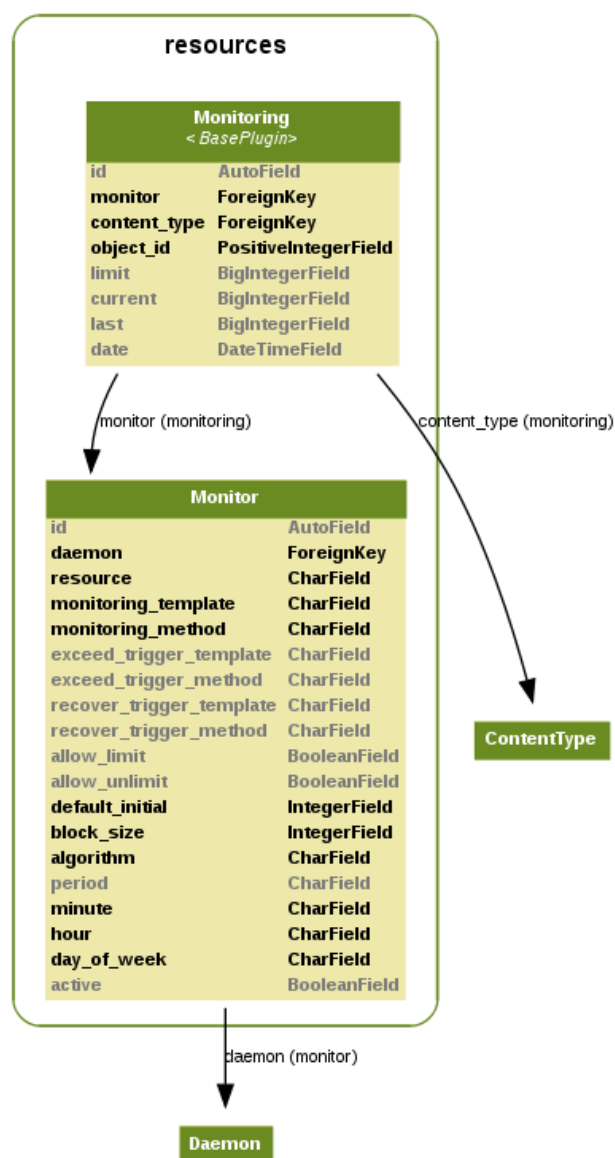


Figure 5.11: Resources application model

The resources application consist on two models:

- **Monitor**. Defines the properties of the resource (like if it is limited or which is the algorithm used to postprocess the monitored data) and establishes how and when the monitorization should be performed.
- **MonitoringData**. Maintains a record of the monitorization data.

The resource abstraction on a single application is something intuitive but seems that it is not easy to come up with this concept, since there is no single existing control panel that does it. The existing control panels have hard-coded resources for each application susceptible to consume them, making the control panel more difficult to reuse.

This lack of resource abstraction on the other control panels might be because a classic relational model does not have the concept of generic foreign key that allows us to have a model with foreign key relations with multiple models. Therefore we have this powerful tool allowing us to manage all the resources as a single model.

The workflow of this application is: (1) execute the monitorization script periodically, (2) interpret and store monitorization data and (3) limit the resource consumption, by running predetermined actions.

1. **Execute the monitorization script periodically**

We use django-celery for scheduling the execution of monitorization scripts, allowing crontab like configuration: minute, hour and day of week.

Like daemons application we provide three methods for the template execution:

- (a) Execute as a script over SSH. (Relies on Daemon SSH connection details)
- (b) Execute as python code
- (c) Execute the template as a local script

The templates are polled from `./resources/templates/scripts` directory and the server where the template will be executed is determined using the daemons application.

2. **Interpret and store monitorization data**

The Monitor configuration provides customizations for the monitorization data interpretation: (1) period, (2) algorithm and (3) block size.

- `period`. Is the time period used for calculating the current state of the resource consumption.
- `algorithm`. Is the operation performed over the period. It can be:
 - Last monitorization result
 - Average during the period
 - Sum of the period values
 - Maximum or minimum of a period
- `block_size`. Defines the allowed tolerance on the measured value to avoid reiterative executions of the `limit_exceeded` template when the measured unit presents an oscillatory behavior, creating a hysteresis⁷ cycle for the thresholds that trigger the scripts execution. This is specially useful for a limit on monthly average disk usage.

3. Limit the resource consumption

We can define a monitor with or without a limit. If we define a monitor with a limit two different ways of ensuring the resource limitation of a daemon are provided: (1) with the resource `limit_exceeded` template and (2) with the Daemon `save_template` execution.

(a) Resource `exceeded_template`.

This approach is achieved by executing the `exceeded_template` when a resource reaches the given threshold limit. For example, this is useful for limiting the monthly consumption of web traffic per `VirtualHost`. Imagine that you only want to allow 1GB consumption per month for an specific `VirtualHost`, so when at the middle of the month this `VirtualHost` has reached the limit, the `exceeded_template` is executed and the `VirtualHost` is disabled. When a new month comes the `recovery_template` will be executed and the `VirtualHost` will be reenabled.

(b) Daemon `save_template` execution.

The limit is ensured by the daemon `save_template`. As the resource application acts as a plugin for a Service, the service knows which is its limit value, so when we write the `save_template` script of the related Daemon we can perform the limitation. A use case would be the web file system quota, because it is set by the command `edquota`.

⁷<http://en.wikipedia.org/wiki/Hysteresis>

5.4.2 Resources Admin

Python's metaclasses are used to dynamically create the resource control forms since their definition is stored on the database and we do not know how the monitors are defined until runtime. The Django forms are based on classes. With metaclasses we can create these classes only by defining how they look like. Once we have this form class created we insert it into the admin change list page of the related service. Figure 5.12 shows how a *resource dynamic* form looks like, and figure 5.45 on page 82 shows this form embedded on a service change form.

Monitorings

Monitoring: #1

Disk limit:
Limit for Disk.0 means unlimited.

Current disk: No data

Current traffic: No data

Figure 5.12: Monitoring dynamic inline

The available actions provided for the monitor on its admin change list page [fig. 5.13] are:

- Delete selected monitors
- Disable selected monitors
- Enable selected monitors

django ISP tools Welcome, pangea. Change password / Log out

DASHBOARD BOOKMARKS SERVICES ACCOUNTANCY ADMINISTRATION

Home · Resources · Monitors

Select monitor to change Add monitor +

Action: ----- Go 0 of 7 selected

Monitor	Content type	Resource	Allow limit	Allow unlimited	Default initial	Block size	Algorithm	Period	Crontab	Active
Apache2-mpm-worker(Traffic)	virtual host	Traffic	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="text" value="0"/>	<input type="text" value="1"/>	Sum	Monthly	0 06 * (m/h/d)	<input checked="" type="checkbox"/>
Apache2-mpm-worker(Disk)	virtual host	Disk	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="text" value="0"/>	<input type="text" value="1024"/>	Average	Monthly	00 07 * (m/h/d)	<input checked="" type="checkbox"/>
Postfix(Disk)	virtual user	Disk	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="text" value="0"/>	<input type="text" value="1"/>	Last	-----	00 ** (m/h/d)	<input checked="" type="checkbox"/>
Proxmox(CPU)	job	CPU	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="text" value="0"/>	<input type="text" value="1"/>	Last	-----	00,20,40 ** (m/h/d)	<input checked="" type="checkbox"/>
Proxmox(Disk)	job	Disk	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="text" value="0"/>	<input type="text" value="1"/>	Last	-----	00 ** (m/h/d)	<input checked="" type="checkbox"/>
Proxmox(Memory)	job	Memory	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="text" value="0"/>	<input type="text" value="1"/>	Last	-----	00 ** (m/h/d)	<input checked="" type="checkbox"/>
Proxmox(Swap)	job	Swap	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="text" value="0"/>	<input type="text" value="1"/>	Last	-----	00 ** (m/h/d)	<input checked="" type="checkbox"/>

7 monitors Save

Filter

- By active: All, Yes, No
- By allow limit: All, Yes, No
- By allow unlimited: All, Yes, No
- By resource: All, Disk, Traffic, CPU, Memory, Swap
- By algorithm: All, Last

Figure 5.13: Monitor admin change list

Change monitor History

Daemon: Apache2-mpm-worker +
Used for find the object: content_type + Host + expression

Resource: Disk

Templates

Monitoring template: apache_traffic.sh **Method:** Run script through SSH

Exceed trigger template: **Method:** Run script through SSH

Recover trigger template: **Method:** Run script through SSH

Allow limit This monitor *can* be limited Allow unlimited This monitor *can* have unlimited resources **Default initial:** 0 0 means unlimited, x>0 means limited

Block size: 1024 **Algorithm:** Average **Period:** Monthly

Crontab

Minute: 00

Hour: 07

Day of week: *

✘ Delete Save and add another Save and continue editing Save

Figure 5.14: Monitor admin change form

5.5 Contacts

This application stores the contact information and provides mechanisms to keep each service instance related to a certain contact.

5.5.1 Contacts Model

The contacts system consists on the following models:

- **Contact** information with optional **BillingContact** and **TechnicalContact**
- **Contract** relations between contacts and services
- Contract **CancellationDate** and **DeactivationPeriod** scheduling capabilities

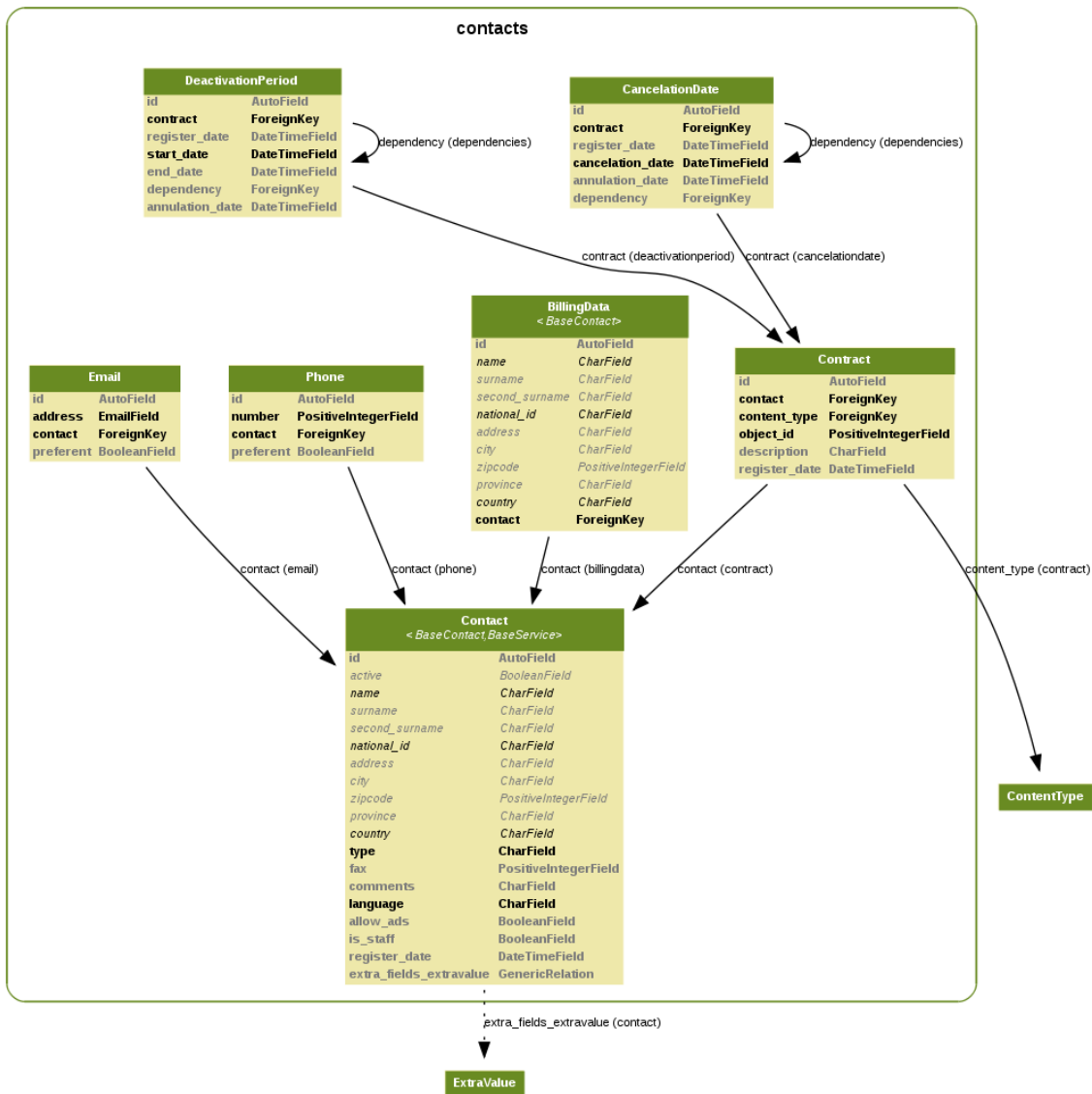


Figure 5.15: Contacts application model

Main contact information is stored on the `Contact` model class, and also there are two auxiliary model classes that make possible to have an arbitrary number of email and phone numbers per contact. Optional billing contact information is handled by the `BillingData` model. Contract relations are managed by the `Contract` model. This class uses generic relations provided by `django.contrib.ContentType` application. This generic relationship allows us to decouple the contacts application from the others, since it provides a Foreign Key emulation that can point to multiple models. This `Contract` abstraction is pretty intuitive but there is no existing control panel that takes a similar approach.

5.5.2 Scheduling cancellations and deactivations

The scheduling of cancellations and deactivations is supported by `CancellationDate` and `DeactivationPeriod` models. Since services can depend on others, the data structure is a n-tree, where the root of the tree is a object without dependencies.

This structure is generated and maintained by an algorithm that uses Django introspection methods provided by `_meta`⁸ attribute. The algorithm is able to discover by itself the underlying relationships between the services and follows them detecting which are the related services susceptible of being canceled or disabled.

We explain on the lines below how the algorithm works with a practical example. Imagine that we have a `Contact` called `Pangea` that owns the `Domain` `pangea.org` with their two subdomains: `www.pangea.org` and `carlos.pangea.org`. Also the `Contact` `Pangea` has a `VirtualHost` that uses `pangea.org` and `www.pangea.org` and a `SystemUser` called `pangea` that is member of a `SystemGroup` called `equipo`. The figure 5.16 represents these relationships and figure 5.17 shows the legend of this and following pictures.

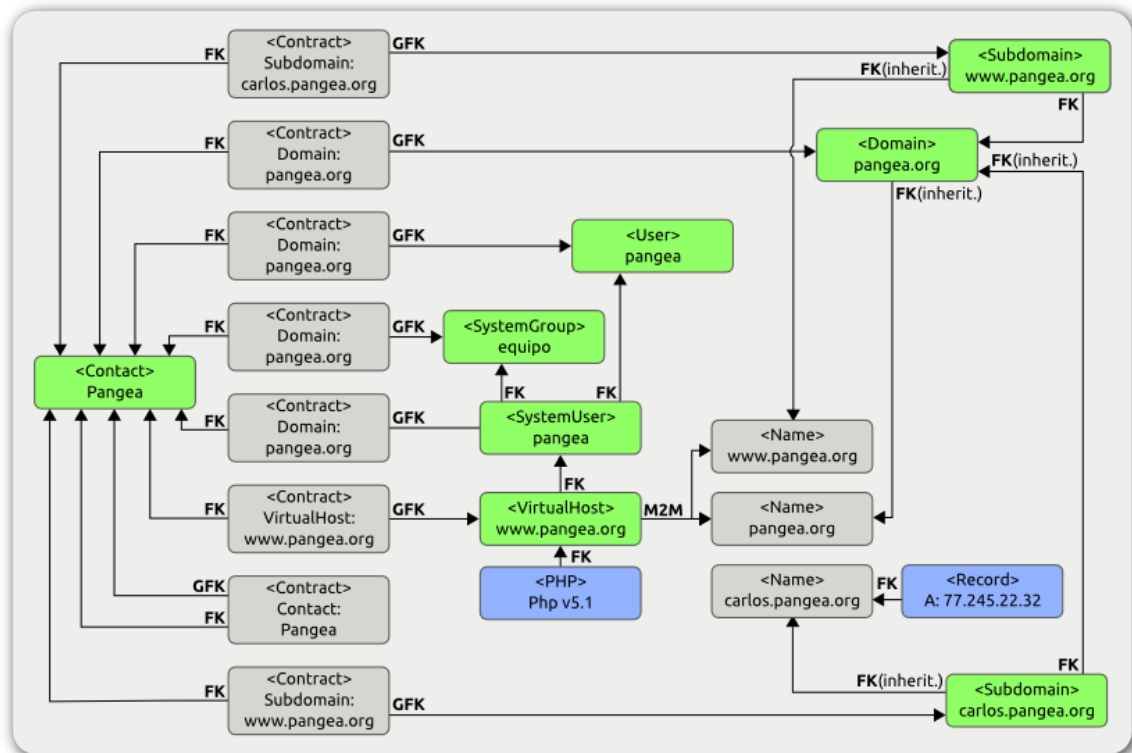


Figure 5.16: Contacts application model

The green boxes are services and the blue boxes are plugins of these services

⁸ `_meta` is a Django internal attribute used for model introspection

To cancel or disable a service, the algorithm must check the related services, propagating on cascade the deactivation or cancellation of the services whose all dependencies have been canceled (or deactivated). For instance, on the case of cancellation of the domain `pangea.org`, the algorithm will obtain the related contracts [fig. 5.18] and objects [fig. 5.19] and will check if any of them is affected by the cancellation. As we can see in these figures, in this case the two subdomains and the `VirtualHost` will also be canceled. The result of the algorithm will be the creation of a `CancellationDate` which points to the affected contract, as shown in figure 5.20



Figure 5.17: Diagrams legend

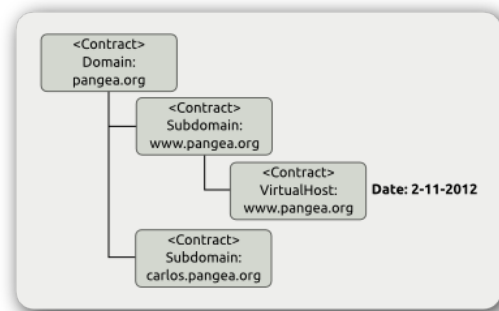


Figure 5.18: Related contracts graph

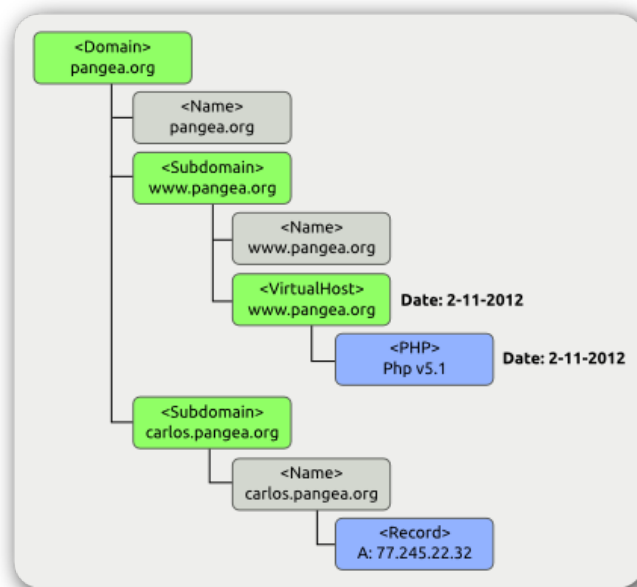


Figure 5.19: Related objects dependency graph

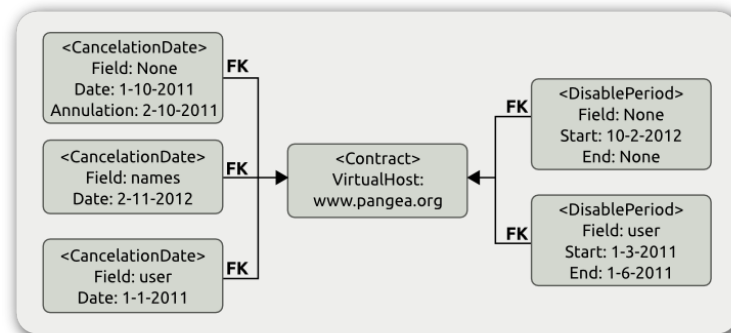


Figure 5.20: Scheduling cancellation and deactivation structure

5.5.3 Contact Admin

The available actions provided for contacts through its admin change list page [figure 5.25] are:

- Delete selected contacts
- Send bulk email. Allows create and send an email to the selected contacts. This is very useful combined with the `allow_ads` filter
- Bulk service contracting. For those services suitable to be created in bulk mode because do not have any specific configuration (like packs)
- Contract service. A single contact must be selected
- Unsubscribe selected contacts
- Bill selected contacts

The provided actions for contracts [fig. 5.27] are:

- Delete selected contracts
- Cancel selected contracts, which will delete the related service
- Disable selected contracts, which will disable the related service
- Schedule cancellation date
- Schedule disable period
- Bill selected contracts

Also, when the contact application is installed, it modifies the behavior of the control panel by:

1. Inserting a contact link on each service change list page [Figure 5.21].



Address	Contact Link
pangea@pangea.es	Pangea
pepito@holaaa.org	Jaume prats

Figure 5.21: Contact link inserted on the service change list page

2. Inserting both, contract and contact links at the top of each service change form [Figure 5.22].



Figure 5.22: Contact and contract links inserted on the service change form

3. Filtering automatically the related objects by contact. For example, if we want to create a new e-mail account, the available domains will be those contracted by the same contact [Figure 5.23].

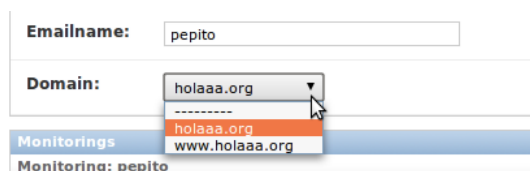


Figure 5.23: Filter objects by related contact

Apart from allowing the management of contact information, the contacts application must maintain the relations that identify the services belonging to each contact. So the admin user must provide this information each time that a new service is created.

The approach that we use to solve this problem without coupling with the service applications is divided into two parts: (1) alter the `ServiceAdmin` class at runtime and (2) creating the contract using the signals sent by `ServiceAdmin`.

1. Altering the `ServiceAdmin` class at runtime in such a way that each time an admin user requests creating a new service this hook responds with an intermediary page with a list of possible contacts, and the admin user must select one of them. This way the contract system knows to what contact belongs the new service.

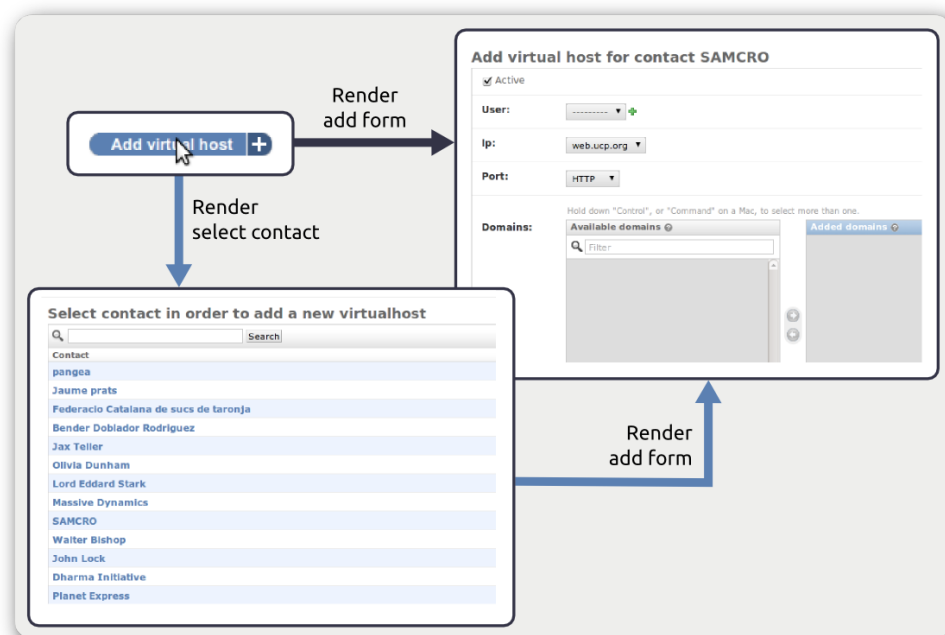


Figure 5.24: Add service cycle

The normal *add service* cycle is represented by the black arrow, when we want to add a new service the add form should appear. The *add service* cycle with the contacts app installed follows the blue arrow because a contact must be provided before creating the service.

2. Create the contract using the signals provided by `ServiceAdmin`. Every time that a service is created, updated or deleted using the admin interface `ServiceAdmin` send a signal attaching the request and the related object. With that information the contacts application is able to create or cancelate the related

contract. The ContactAdmin also sends its own signals in order to warn other apps that something has happened, these signals are:

- (a) `contract_created()`
- (b) `contract_updated()`
- (c) `contract_deleted()`

Following we present some screenshots of the contact app admin pages:

The screenshot shows the 'Select contact to change' page in the Django ISP tools admin interface. The page has a navigation bar with 'DASHBOARD', 'BOOKMARKS', 'SERVICES', 'ACCOUNTANCY', and 'ADMINISTRATION'. The main content area displays a table of 13 contacts with columns for Name, Surname, Type, Email, Phone, Fax, National id, Address, City, Zipcode, and Register date. A search bar and a filter sidebar are also visible.

Name	Surname	Type	Email	Phone	Fax	National id	Address	City	Zipcode	Register date
pangea		Organization	suport@pangea.org	938545936	(None)	ES5161536		Barcelona	(None)	Oct. 5, 2011, 5 p.m.
Jaume	prats	Person	memo@prats.com	63457854	(None)	72783728G		Barcelona	(None)	Oct. 14, 2011, 1:07 p.m.
Federacio Catalana de sucres de taronja		Organization	sucs@pangea.org	936845745	(None)	ES2823849		Barcelona	(None)	Oct. 14, 2011, 1:07 p.m.
Bender	Doblador	Person	bender@futurama.org	101010011	(None)	101010010011		Barcelona	(None)	Oct. 14, 2011, 1:08 p.m.
Jax	Teller	Person	jax@sonsofanarchy.us	555182812	(None)	628718718		Barcelona	(None)	Oct. 14, 2011, 1:09 p.m.
Olivia	Dunham	Person	o.dunham@fringe.us	617839211	(None)	23222232		Barcelona	(None)	Oct. 14, 2011, 1:09 p.m.
Lord Eddard	Stark	Person		2323331233	(None)	00000032H		Barcelona	(None)	Oct. 14, 2011, 1:10 p.m.
Massive Dynamics		Organization	suport@massivedynamics.com	627262372	(None)	721038362		Barcelona	(None)	Oct. 14, 2011, 1:10 p.m.
SAMCRO		Organization			(None)	28118811		Barcelona	(None)	Oct. 14, 2011, 1:11 p.m.
Walter	Bishop	Person	walter@fringe.us	87322	(None)	72647827B		Barcelona	(None)	Oct. 14, 2011, 1:20 p.m.
John	Lock	Person			(None)	62739288		Barcelona	(None)	Oct. 14, 2011, 1:21 p.m.
Dharma Initiative		Organization	contact@lost.net	27123822	(None)	US928291		Barcelona	(None)	Oct. 14, 2011, 1:22 p.m.
Planet Express		Organization	contact@futurama.org	8237287	(None)	US28272822		Barcelona	(None)	Oct. 14, 2011, 1:23 p.m.

Figure 5.25: Contact admin change list

The screenshot shows the 'Change contact' form in the Django ISP tools admin interface. The form includes fields for Name, Surname, National id, Type, Language, and Active status. Below the main form, there are sections for 'Optional Fields (Show)', 'Emails', 'Phones', and 'Billing Datas'. The 'Emails' section shows a table with columns for Address, Preferent, and Delete?. The 'Phones' section shows a table with columns for Number, Preferent, and Delete?.

Figure 5.26: Contact admin change form

Select contract to change

Filter

By content type

All
contact
domain
job
list
pack
subdomain
system group
system user
user
virtual alias
virtual host
virtual user
vps

By cancellation date

Any date
Today
Past 7 days
This month
This year

Contract	Contact	Content type	Content Object	Description	Register date	Active	Cancel date
contact(Pangea)	Pangea	contact	Pangea	(None)	Oct. 5, 2011, 5 p.m.	✔	None
user(pangea)	Pangea	user	pangea	(None)	Oct. 5, 2011, 5 p.m.	✔	None
system group(pangea)	Pangea	system group	pangea	(None)	Nov. 22, 2011, 1:20 p.m.	✔	None
system user(pangea)	Pangea	system user	pangea	(None)	Nov. 22, 2011, 1:20 p.m.	✔	None
virtual host(VHssssssssssss.org)	Pangea	virtual host	VHssssssssssss.org	(None)	Nov. 22, 2011, 1:20 p.m.	✔	None
subdomain(www.pangea.es)	Pangea	subdomain	www.pangea.es	(None)	Dec. 9, 2011, 9:55 p.m.	✔	None
domain(pangea.es)	Pangea	domain	pangea.es	(None)	Dec. 9, 2011, 9:55 p.m.	✔	None
virtual user(pangea)	Pangea	virtual user	pangea	(None)	Dec. 12, 2011, 4:23 p.m.	✔	None
contact(Jaume)	Jaume	contact	Jaume	(None)	Oct. 14, 2011, 1:07 p.m.	✔	None
subdomain(www.holaaa.org)	Jaume	subdomain	www.holaaa.org	(None)	Dec. 9, 2011, 9:54 p.m.	✔	None
domain(holaaa.org)	Jaume	domain	holaaa.org	(None)	Dec. 9, 2011, 9:54 p.m.	✔	None

Figure 5.27: Contract admin change list

Home > Ordering > Packs

Contract Pack for contact Pangea

Add pack +

Action: [dropdown] Go 2 of 5 selected

Name	Contracted	Number	Allow multiple
Member Single	⊖	0	⊖
Traffic prepay	⊖	0	⊖
Special offer #1	⊖	0	⊖
Special offer #2	⊖	0	⊖

5 packs

Figure 5.28: Pack contraction page

Home > Contacts > Contacts > Contract service for contact Massive Dynamics

Contract service for contact Massive Dynamics

Choose a service from the list below for contract.

App	Model
auth	user
contacts	contact
dns	domain, subdomain
web	system user, system group, virtual host
mail	virtual user, virtual alias
llists	list
ordering	pack
jobs	job
vps	vps

Figure 5.29: Contract service page

5.6 Ordering

This application defines the different services that must be charged, keeps track of them, and determines the cost of each one existing on the system in a flexible and powerful way.

5.6.1 Ordering Model

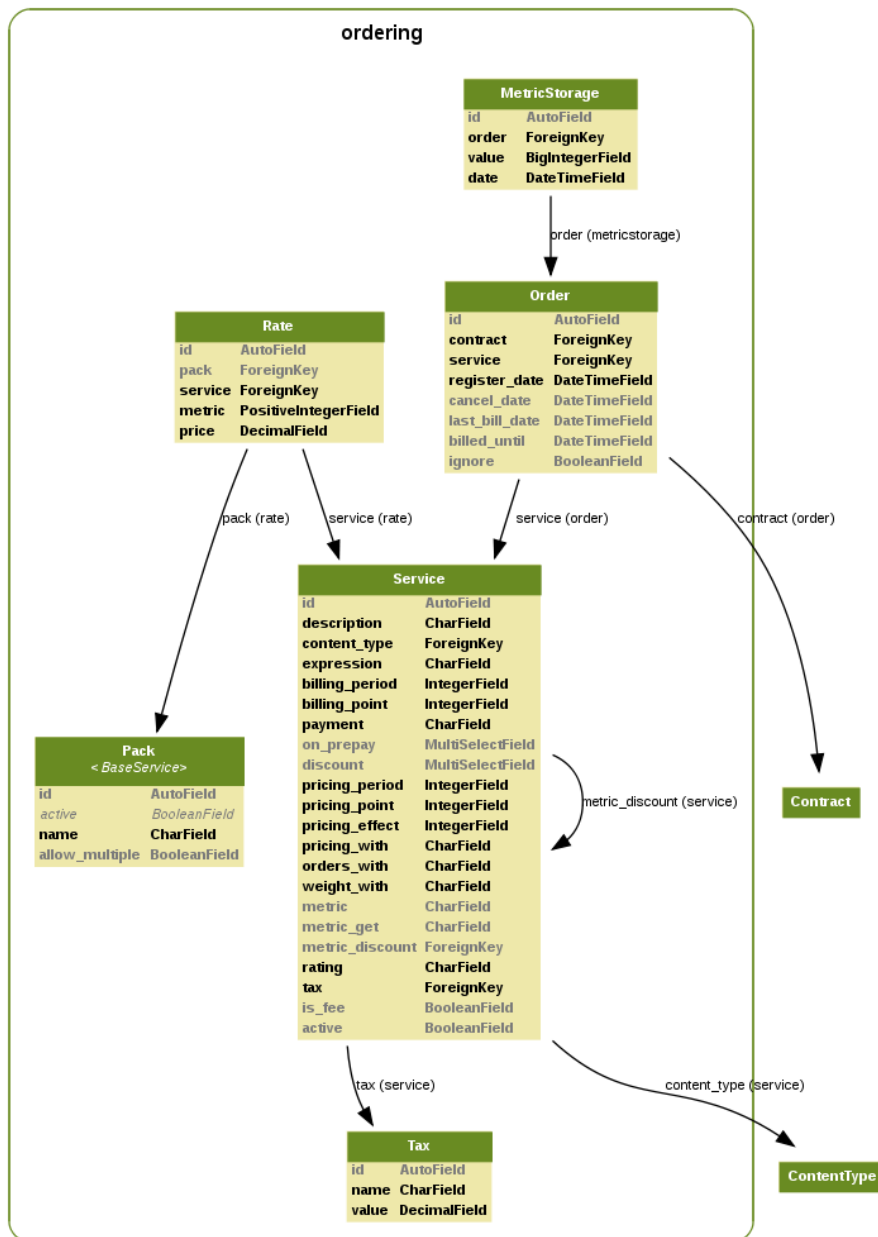


Figure 5.30: Ordering application model

The ordering application implements all the requirements discussed on chapter 1 sections *1.3.2 Services* and *1.3.3 Advance pricing configurations*. This app consists on the following data models:

1. **Service**, that defines precisely how a service is going to behave in terms of pricing and billing
2. **Order**, which keeps track of the instances that are going to be be charged
3. **Pack**, that defines service bundles⁹ with included services and/or special offers
4. **Rate** defines the price of a service

The Ordering app is conceptually quite coupled with billing application, so to solve this the backend approach has been used. In a file called `billing_backend.py` we define how and with which application we want to perform the creation of the bills. Notice that it is not mandatory to use a Django related application for generating the invoices, since we can define on the backend file how to create and retrieve an invoice with any existing tool.

5.6.2 Ordering Admin

The available actions provided for orders through its admin change list page, figure 5.32, are:

- Ignore selected orders. The orders marked with an ignored bit will not be billed
- Bill orders
- Bill orders (default behavior). Will bill the orders with the behavior defined on `settings.py`
- Budget selected orders. This will generate budgets of the selected orders. This action does not perform any change on the order status

Actions provided for the services, figure 5.31:

- Delete selected services
- Disable selected services

⁹A services bundle is a package of services, it is used for offering several products for sale as one combined product

- Enable selected services
- Check selected services, it update orders when a service has been updated or created

Three screen-shots of the admin interface are presented below: (1) the service change list page on figure 5.31, (2) the order change list page on figure 5.32, and finally, (3) the figure 5.33 is an screenshot of the service add form. As you can see, it contains all the configuration details discussed on chapter 1 section 1.3.3 *Advance pricing configuration*. It has a lot of options, but, fortunately for the admin user, we provide a bunch of predefined services that will suit most of the use cases, at most only one or two parameters will need to be tweaked to precisely fit your needs: the billing or pricing period.

Select service to change Add service +

Action: Go 0 of 7 selected

Service description	Content type	Is membership fee?	Pricing period	Billing period	Payment	Tax	Is active?
<input type="checkbox"/> FTP User	system user	<input checked="" type="checkbox"/>	No period	Annual	Prepay	base 18.00%	<input checked="" type="checkbox"/>
<input type="checkbox"/> System User	system user	<input checked="" type="checkbox"/>	No period	Annual	Prepay	(None)	<input checked="" type="checkbox"/>
<input type="checkbox"/> Domain ORG	domain	<input checked="" type="checkbox"/>	Annual	Annual	Prepay	(None)	<input checked="" type="checkbox"/>
<input type="checkbox"/> Domain NET	domain	<input checked="" type="checkbox"/>	Annual	Annual	Prepay	(None)	<input checked="" type="checkbox"/>
<input type="checkbox"/> Domain ES	domain	<input checked="" type="checkbox"/>	Annual	Annual	Prepay	(None)	<input checked="" type="checkbox"/>
<input type="checkbox"/> Domain INFO	domain	<input checked="" type="checkbox"/>	Annual	Annual	Prepay	(None)	<input checked="" type="checkbox"/>
<input type="checkbox"/> Mail User	virtual user	<input checked="" type="checkbox"/>	No period	Annual	Prepay	(None)	<input checked="" type="checkbox"/>

7 services

Filter

By is active?
 All
 Yes
 No

By content type
 All
 contact
 domain
 job
 list
 pack
 subdomain
 system group
 system user
 user
 virtual alias
 virtual host
 virtual user
 vps

By billing period
 All

Figure 5.31: Service admin change list

Select order to change Add order +

Search

< 2011 December 12

Action: Go 0 of 6 selected

Contract	Contact	Service	Metric	Register date	Cancel date	Billed until	Ignore
<input type="checkbox"/> contact(John)	John	FTP User	1	Dec. 12, 2011, 5:55 p.m.	(None)	April 1, 2012, midnight	<input type="checkbox"/>
<input type="checkbox"/> contact(John)	John	FTP User	1	Dec. 12, 2011, 5:55 p.m.	(None)	April 1, 2012, midnight	<input type="checkbox"/>
<input type="checkbox"/> contact(Walter)	Walter	FTP User	1	Dec. 12, 2011, 5:54 p.m.	(None)	April 1, 2012, midnight	<input type="checkbox"/>
<input type="checkbox"/> contact(Walter)	Walter	FTP User	1	Dec. 12, 2011, 5:54 p.m.	(None)	April 1, 2012, midnight	<input type="checkbox"/>
<input type="checkbox"/> contact(Walter)	Walter	FTP User	1	Dec. 12, 2011, 5:54 p.m.	(None)	April 1, 2012, midnight	<input type="checkbox"/>
<input type="checkbox"/> user(pangea)	Pangea	FTP User	1	Dec. 12, 2011, 4:09 p.m.	(None)	April 1, 2012, midnight	<input type="checkbox"/>

6 orders Save

Filter

By ignore
 All
 Yes
 No

By content type
 All
 contact
 domain
 job
 list
 pack
 subdomain
 system group
 system user
 user
 virtual alias
 virtual host
 virtual user
 vps

Figure 5.32: Order admin change list

Change service History

Service description:

Content type:

Expression:
format: O['field1']=='value' and (O['field2']=='value2' or O['field3'] == True)

Billing Period

Billing period:
Renew period for recurring invoicing

Billing point:

- No Period
- Fixed
- Variable

Payment:

- Eventual
- Postpay
- Prepay

On prepay:

- Refund on Cancel
- Recharge on Cancel
- Refund on Disable
- Recharge on Disable
- Refund on Weight
- Recharge on Weight

Discount:

- Discount on Register
- Discount on Cancel
- Discount on Disable

Before billing

After billing

Pricing Period

Pricing period:
In what period do you want to lookup for calculating price?

Pricing point:

- No Period
- Fixed
- Variable

Pricing effect:

- No period
- Current on register or renew
- Every period

Pricing

Pricing with:

- Number of Orders
- Weight of Service

Metric:
format: O['field1'] - O['field2']. If null metric is the number of orders.

Metric get:

Metric discount:
make a discount, for example on traffic prepay, target needs to have pricing_period, bill period is required for both, only with single order

Weight with:

- Pricing with Orders
- Single Order
- All Contact Orders

Orders with:

- Pricing with weight
- Active
- Renewed
- Registred
- Registred or Renewed

Rating:

Tax:

Is membership fee?
select this if this service is a membership fee

Is active?

Rates

Pack	Metric	Price	Delete?
default--FTP User--1	<input type="text" value="1"/>	<input type="text" value="12.00"/>	<input type="checkbox"/>
default--FTP User--10	<input type="text" value="10"/>	<input type="text" value="10.00"/>	<input type="checkbox"/>

Add another Rate

Delete

Figure 5.33: Service admin change form

5.7 Billing

The Billing app generates and administrates budgets, invoices, membership fees and their amendments. This application implements all the requirements presented on chapter 1 section *1.3.4 Billing system*.

5.7.1 Billing Model

The billing application consists on the following models:

- **BaseBillLine**, which is a bill line abstract model
- **BudgetLine** is a bill line for budget
- **BillLine**, which is a base class for the models above
- **FeeLine** is a bill line for fees
- **InvoiceLine** is a bill line for invoices
- **AmendedBillLine** is a base class for amended lines
- **AmendedInvoiceLine**, which is a amended line for invoices
- **AmendedFeeLine** is an amended line for fees
- **BaseBill**, which is a base bill abstract model
- **Bill** is a base class for the models above
- **Invoice** is an invoice
- **AmendmentInvoice**, which is an amended invoice
- **Fee** is a fee
- **AmendmentFee** is an amended fee
- **Budget** is a budget (inherit from BaseBill)

An *aggressive* inheritance pattern is adopted on the implementation of the billing application models. We are not sure yet if this approach is better than one providing a `bill_type` field. At least the adopted solution works really well with the Django automatic admin interface.

The billing system is heavily coupled with Contacts application because each bill needs to have a foreign key relation with a contact. To work around with this issue we provide a setting value allowing you to choose what is the model that you want to use for storing the contact information, as with the ordering app. By default it uses the `Contact` model from contacts application but the adopted approach lets you use any other application that encapsulates the same contact concept, like the `User` model from Django auth application.

A bill can be created from a subset of contacts, contracts or orders.

5.7.2 Billing Admin

The available actions on the admin change list page [fig 5.35] are:

- Delete selected bills
- Close selected bills
- Send selected bills
- Amend selected bills
- Merge selected bills, must be bills issued to the same contact
- Mark as returned
- Mark as paid
- Mark as irrecoverable

Follows a bunch of screenshots of the billing admin interface is presented.

BILL	Date	Modified	Comments	Status	Num lines	Total	Contact	Contact Link
OPENF20110001	Dec. 12, 2011, 4:22 p.m.	Dec. 12, 2011, 4:22 p.m.	aaaa	Open	0	0	Federacio Catalana de sucs de taronja	Federacio Catalana de sucs de taronja
OPENF20110002	Dec. 12, 2011, 5:53 p.m.	Dec. 12, 2011, 5:53 p.m.	(None)	Open	1	4.27	Pangea	Pangea
OPENF20110003	Dec. 12, 2011, 5:55 p.m.	Dec. 12, 2011, 5:55 p.m.	(None)	Open	3	12.78	Walter	Walter Bishop
OPENF20110004	Dec. 12, 2011, 5:55 p.m.	Dec. 12, 2011, 5:55 p.m.	(None)	Open	2	8.52	John	John Lock

Figure 5.35: Billing admin change list

The actions early described on table 1.3.4 are available on the billing change form through links on the right corner of the page, see figure 5.36.

django ISP tools Welcome, pangea. [Change password](#) / [Log out](#)

[DASHBOARD](#) [BOOKMARKS](#) [SERVICES](#) [ACCOUNTANCY](#) [ADMINISTRATION](#)

Home > Billing > Bill > OPENF20110004

Change invoice of contact John Lock

[Close](#) [Manage Lines](#) [Get PDF](#) [History](#)

Ident: OPENF20110004

Status: Open

Date: 2011-12-12 17:55:38

Due date:

Comments:

HTML (Show)

Line Number	Order ID	Description	Initial Date	Final Date	Price	Metric	Tax
OPENF20110004:1							
1	6	John (FTP User)	2011-12-12 17:55:08	2012-04-01 00:00:00	3.61	1	18.00
OPENF20110004:2							
2	5	John (FTP User)	2011-12-12 17:55:00	2012-04-01 00:00:00	3.61	1	18.00

[Delete](#) [Save and add another](#) [Save and continue editing](#) [Save](#)

Figure 5.36: Billing admin change form

The bill amendment is performed through the page displayed on figure 5.37. It supports two amend modes: (1) manual, that lets you specify the amended price value for each line, and (2) automatic, assuming the amendment of the entire value.

Amend invoice lines of F20110001

Action: Go 2 of 3 selected

Line	Description	Initial date	Final date	Price	Amount	Tax	Bill	Amended Bill
1	John (FTP User)	Dec. 12, 2011, 5:54 p.m.	April 1, 2012, midnight	3.61	1	18.00	F20110001	(None)
2	Walter (FTP User)	Dec. 12, 2011, 5:54 p.m.	April 1, 2012, midnight	3.61	1	18.00	F20110001	(None)
3	Walter (FTP User)	Dec. 12, 2011, 5:54 p.m.	April 1, 2012, midnight	3.61	1	18.00	F20110001	(None)

3 bill lines

Figure 5.37: Billing amend line page

5.8 Service converter

Service converter is an application with the aim of providing full integration for third parties Django applications. That way we can pick up any interesting Django app and use it as a new service of our control panel without the need of touching a single line of code.

All you need to do is adding the new application through two configuration parameters of the Service converter app `settings.py` file. Just like we have done for `django.contrib.auth.User`:

```
# (module, class)
BASESERVICE_CONVERTER = (
    ("django.contrib.auth.models", "User"),
)

# ((model module, model class),
# (modeladmin module, modeladmin class))
SERVICEADMIN_CONVERTER = (
    (("django.contrib.auth.models", "User"),
     ("django.contrib.auth.admin", "UserAdmin")),)
)
```

At this point your new service application has the same control panel support as any service application that we have developed, like DNS, mail, web... At this moment this app will bring support for contacts, resources, daemons and orders, but this system is fully compatible with any future features of the control panel.

5.8.1 `django.contrib.auth` a succesfull example

The control panel user accounting and authentication subsystem relies entirely on `django.contrib.auth` application. Please notice that this is not an application developed by this project, but an official Django reusable application. We have considered very interesting to present this application on this subsection because it is a successful example on how the control panel can perfectly integrate third-party apps without modifying their code.

Screen-shots of the user interface are presented on the next two pages. Figure 5.38 shows the login form displayed when we try to access the panel without an active session.

Figure 5.38: Admin login form

We can confirm that the `UserAdmin` class has been successfully adapted to the `ServiceAdmin` looking at the *contact link* on user change list, figure 5.39. Apart from this little detail of the contact support provided to the Auth application, other service-related support are also provided, such as support for resources, daemons, ordering and billing.

Username	E-mail address	First name	Last name	Staff status	Contact Link
dunkan				⊖	Massive Dynamics
jaume				⊖	Jaume prats
jeremies				⊖	Federacio Catalana de sucs de taronja
John				⊖	John Lock
pangea	pangea@pangea.org			⊕	Pangea
pepino				⊖	SAMCRO

Figure 5.39: User change list

Finally on figure 5.40 we can see how the User support for `SystemUsers` and `VirtualUsers` has been provided through two forms dynamically inserted on the User change form, one (`SystemUser`) from web app and the other (`VirtualUser`) from mail application.

Change user of contact Pangea

[Contract](#) [History](#) [View on site](#) →

Username:	<input type="text" value="pangea"/>
	Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.
Password:	<input type="text" value="sha1\$YEUKuXlyrqKo\$007935e7de9d37"/>
	Use "[algo]\${salt}\${hexdigest}" or use the change password form .
Personal info	
First name:	<input type="text"/>
Last name:	<input type="text"/>
E-mail address:	<input type="text" value="pangea@pangea.org"/>
Permissions	
<input checked="" type="checkbox"/> Active	Designates whether this user should be treated as active. Unselect this instead of deleting accounts.
<input checked="" type="checkbox"/> Staff status	Designates whether the user can log into this admin site.
<input checked="" type="checkbox"/> Superuser status	Designates that this user has all permissions without explicitly assigning them.
User permissions:	<p>Hold down "Control", or "Command" on a Mac, to select more than one.</p> <div><p>Available user permissions</p><input type="text" value="Filter"/><ul style="list-style-type: none">admin log entry Can add log entryadmin log entry Can change log entryadmin log entry Can delete log entryauth group Can add groupauth group Can change groupauth group Can delete groupauth permission Can add permissionauth permission Can change permissionauth permission Can delete permissionauth user Can add userauth user Can change userauth user Can delete userbilling amended fee line Can add amenc<p>Add all Remove all</p></div> <div><p>Added user permissions</p><ul style="list-style-type: none"></div>
Important dates	
Last login:	Date: <input type="text" value="2011-12-12"/> Today <input type="text" value="13:27:04"/> Now <input type="text"/>
Date joined:	Date: <input type="text" value="2011-10-05"/> Today <input type="text" value="17:00:06"/> Now <input type="text"/>
Groups	
Groups:	<input type="text"/>
	In addition to the permissions manually assigned, this user will also get all permissions granted to each group he/she is in. Hold down "Control", or "Command" on a Mac, to select more than one.
System Users	
System User: pangea <input type="button" value="Delete"/>	
<input checked="" type="checkbox"/> Active	
Shell:	<input type="text" value="/bin/false"/>
Uid:	<input type="text" value="212"/>
Primary group:	<input type="text" value="pangea"/> <input type="button" value="Add"/>
Homedir:	<input type="text" value="/home/pangea/"/>
<input type="checkbox"/> Only ftp	
Virtual Users	
Virtual User: #1	
Emailname:	<input type="text"/>
Domain:	<input type="text" value="-----"/>
<input checked="" type="button" value="Delete"/>	<input type="button" value="Save and add another"/> <input type="button" value="Save and continue editing"/> <input type="button" value="Save"/>

Figure 5.40: User change form

5.9 Mail

This app provides mail account support for `django.contrib.auth` application.

5.9.1 Mail Model

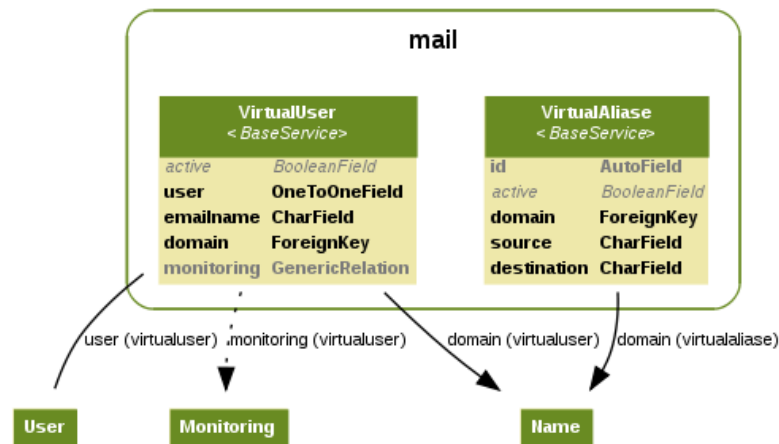


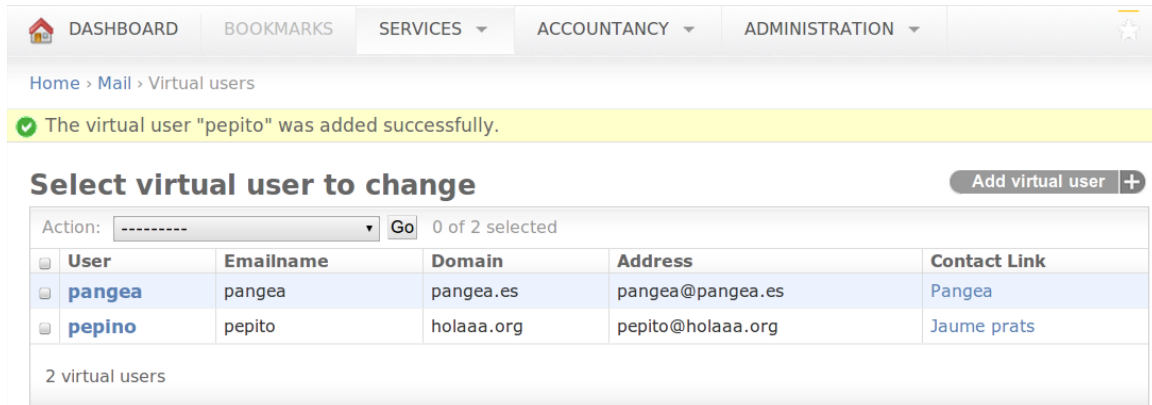
Figure 5.41: Mail application model

The mail application is composed by two models:

- **VirtualUser** which is responsible for storing the mail account properties. Notice that it has a OneToOne relation with **User** model, that is because the **VirtualUser** is an extension of an existing **User**. So rather than having independent users for mail accounts we bring **VirtualUser** support to an existing user, so each user uses only a single credential for accessing all the services.
- **VirtualAlias**, which is an alias table used in mail servers that rewrites recipient addresses for local, virtual, and remote mail destinations. The main applications of virtual aliasing are:
 - To redirect mail to one address to another one or more addresses
 - To implement virtual alias domains where all addresses are aliased to addresses in other domains
 - To define a “catch all” of the emails addressed to a domain that do not exist in the mail server

5.9.2 Mail Admin

Following two screen-shots of the mail app admin interface are provided. On figure 5.42 we can see how a message is displayed on the top of the page notifying that a new virtual user has been successfully created.

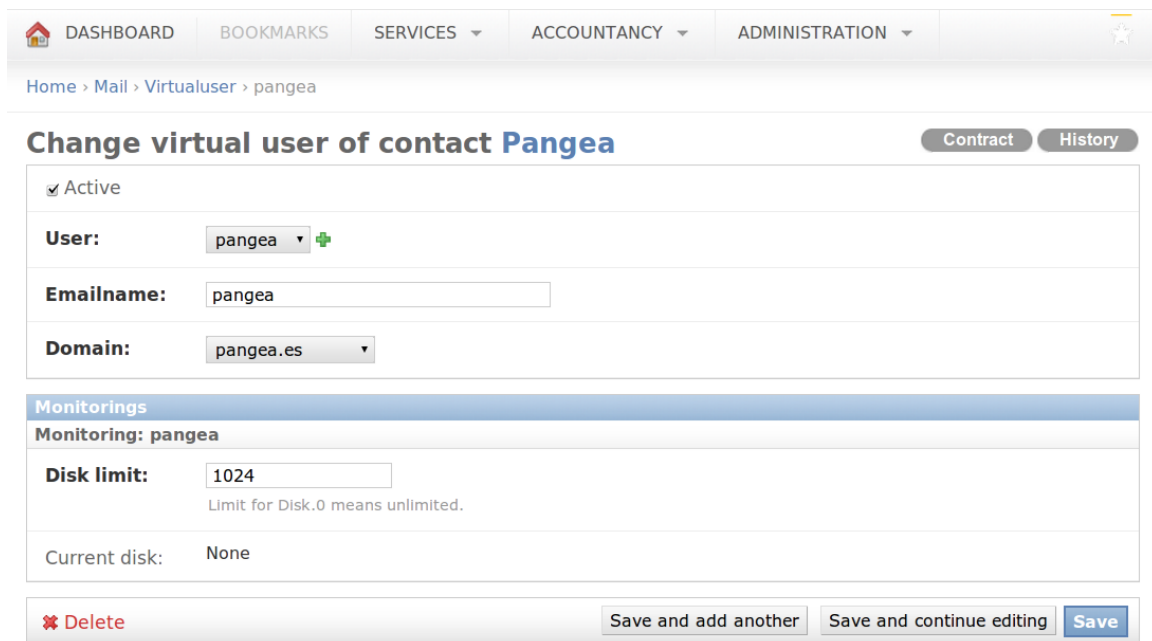


The screenshot shows the Mail Admin interface. At the top, there is a navigation bar with tabs for DASHBOARD, BOOKMARKS, SERVICES, ACCOUNTANCY, and ADMINISTRATION. Below the navigation bar, there is a breadcrumb trail: Home > Mail > Virtual users. A yellow notification banner at the top states: "The virtual user 'pepito' was added successfully." Below the notification, there is a section titled "Select virtual user to change" with a button "Add virtual user +". Underneath, there is a table with columns: User, Emailname, Domain, Address, and Contact Link. The table contains two rows: one for 'pangea' and one for 'pepino'. Below the table, it says "2 virtual users".

User	Emailname	Domain	Address	Contact Link
<input type="checkbox"/> pangea	pangea	pangea.es	pangea@pangea.es	Pangea
<input type="checkbox"/> pepino	pepito	holaaa.org	pepito@holaaa.org	Jaume prats

Figure 5.42: Mail admin change list

On the next figure 5.43 the mail change form is displayed together with a disk limit resource form.



The screenshot shows the Mail Admin interface for editing a virtual user. The navigation bar is the same as in Figure 5.42. The breadcrumb trail is: Home > Mail > Virtualuser > pangea. The main section is titled "Change virtual user of contact Pangea" with buttons for "Contract" and "History". Below the title, there is a form with the following fields: "Active" (checked), "User:" (dropdown menu with "pangea" selected), "Emailname:" (text input with "pangea"), "Domain:" (dropdown menu with "pangea.es" selected). Below the form, there is a section titled "Monitorings" with a sub-section "Monitoring: pangea". This section contains a "Disk limit:" field with a text input containing "1024" and a note "Limit for Disk.0 means unlimited." Below this, it says "Current disk: None". At the bottom of the form, there are three buttons: "Delete" (with a red X icon), "Save and add another", and "Save and continue editing" (disabled), and a "Save" button.

Figure 5.43: Mail admin change form

Also remember that this application inserts a form on the user change form page, just as we early saw on the previous user adapter section, on figure 5.40.

5.10 Web

This application implements the web shared hosting support for `django.contrib.auth` application.

5.10.1 Web Model

This web app consists of two models:

1. **VirtualHost**, which provides basic Apache-like virtual host configuration. It supports `ServerName`, `ServerAlias`, `DocumentRoot` and `Redirect` directives.
2. **SystemUser** and **SystemGroup** Unix compatible definitions.

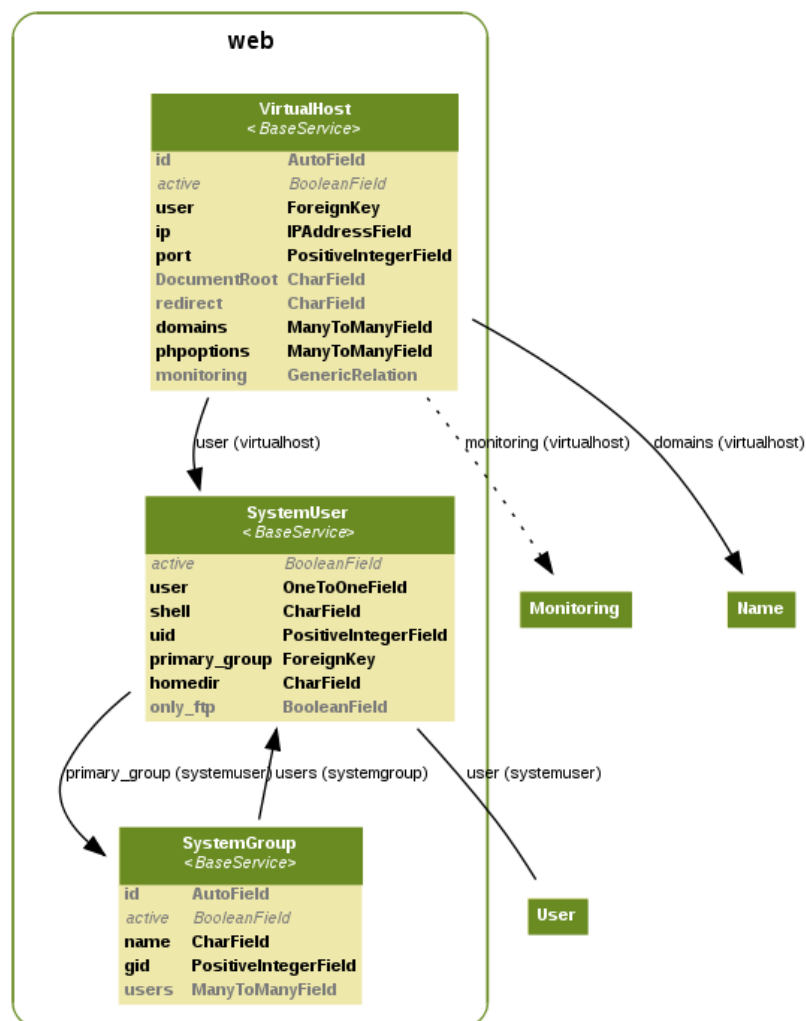


Figure 5.44: Web application model

5.10.2 Web Admin

Figure 5.45 is a screenshot of the Virtual Host change form with three dynamically inserted forms two from the PHP plugin and one by the resource application.

Change virtual host of contact Pangea Contract History

Active

User: +

Ip:

Port:

Domains: Hold down "Control", or "Command" on a Mac, to select more than one.

Available domains

Filter

- www.lost.org
- pangea.es
- www.pangea.es
- futura.net
- moriarty.org
- www.moriarty.org
- t1000.org
- www.t1000.org

Added domains

- ssssssssssss.org
- www.futura.net
- multiplexer2001.es
- www.multiplexer2001.es
- dellcomputers.info
- www.dellcomputers.info

Add all **Remove all**

DocumentRoot:

Redirect:

Virtual host php options

Phpoption	Value	None	Delete?
VirtualHostPHPOption object			
<input type="text" value="pangea"/> +	<input type="text" value="30"/>	possible values, depending on option	<input type="button" value=""/>
<input type="text" value="....."/> +	<input type="text"/>	possible values, depending on option	

+ Add another Virtual Host Php Option

Php versions

Value	Delete?
<input type="text" value="PHP5"/>	

Monitorings

Monitoring: VHssssssssss.org

Disk limit:
Limit for Disk.0 means unlimited.

Current disk: None

Current traffic: None

Figure 5.45: Web admin change form

5.11 PHP Plugin

the PHP plugin app extends the web application providing PHP¹⁰ and SuExec¹¹ support for Apache-like virtual hosts.

5.11.1 PHP Plugin Model

The PHP plugin models are composed by:

- **PHPOption** model class, which defines possible `php.ini` directives¹² and allowed values for them
- **VirtualHostPHPOption** model, which stores the custom values for each virtual host
- **PHPVersion** model, that provides support for different PHP versions

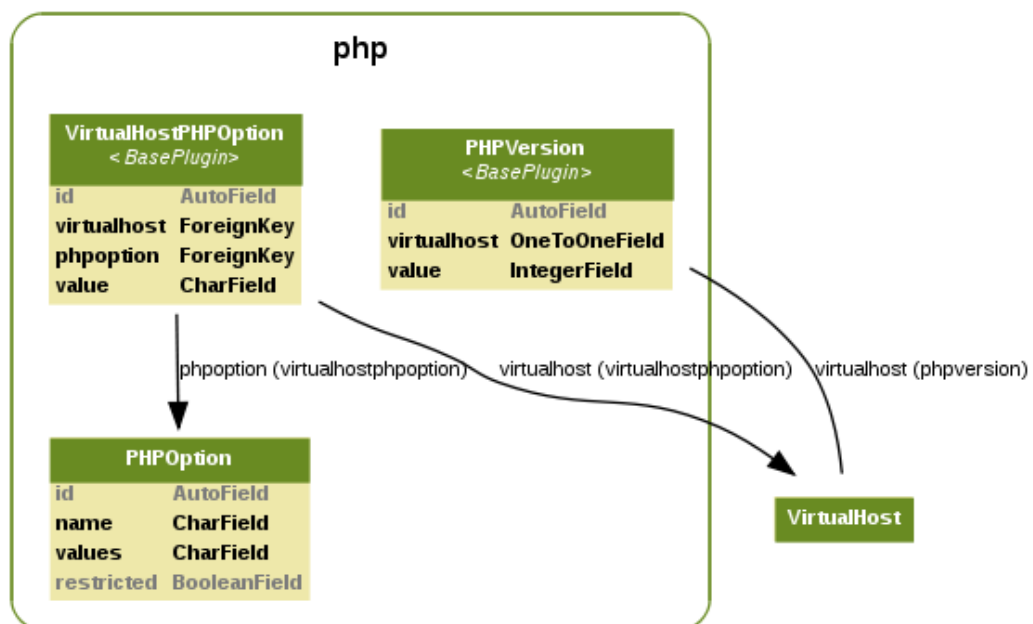


Figure 5.46: PHP Plugin application model

¹⁰<http://www.php.net/>

¹¹<http://httpd.apache.org/docs/trunk/suexec.html>

¹²List of `php.ini` directives at <http://www.php.net/manual/en/ini.list.php>

5.11.2 PHP Plugin Admin

Figure 5.47 shows in detail the form inserted on web change form, figure 5.45, when the PHP plugin is installed.

The screenshot displays two sections of a web interface. The top section, titled "Virtual host php options", contains a table with columns "Phpoption", "Value", "None", and "Delete?". It lists two options: "pangea" with a value of "4444" and a note "possible values, depending on option", and a second option with a value of "" and the same note. Below the table is a link to "Add another Virtual Host Php Option". The bottom section, titled "Php versions", contains a table with columns "Value" and "Delete?". It lists "PHP5" as the current version.

Virtual host php options			
Phpoption	Value	None	Delete?
pangea	4444	possible values, depending on option	
-----		possible values, depending on option	✕

[+ Add another Virtual Host Php Option](#)

Php versions	
Value	Delete?
PHP5	

Figure 5.47: PHP plugin dynamic form example

5.12 DNS

The DNS app is used to manage DNS zones and DNS server configurations.

5.12.1 DNS Model

The DNS model application is composed of:

- **Domain**, which is a main domain registered by a customer. It is usually is a second-level domain, for example, in the domain `www.mydomain.org` it will be `mydomain`.
- **Subdomain**, which is an extension of a Domain. `www` in our previous example.
- **Name**, which is the parent model class for Domain and Subdomain. The model Name (domain name) can be a **Domain** or a **Subdomain**. This inheritance pattern exists in order to allow no practical distinction between them. This is useful to allow domains and subdomains be part of the `ServerName` or `ServerAlias` statements of a `VirtualHost` definition.
- **Record**. A DNS resource record is the basic data element in the domain name system. Each record has a type (A, MX, etc.), an expiration time limit, a class, and some type-specific data. Configuring a default set of records that will be created for each new domain is possible through `dns/settings.py` file.

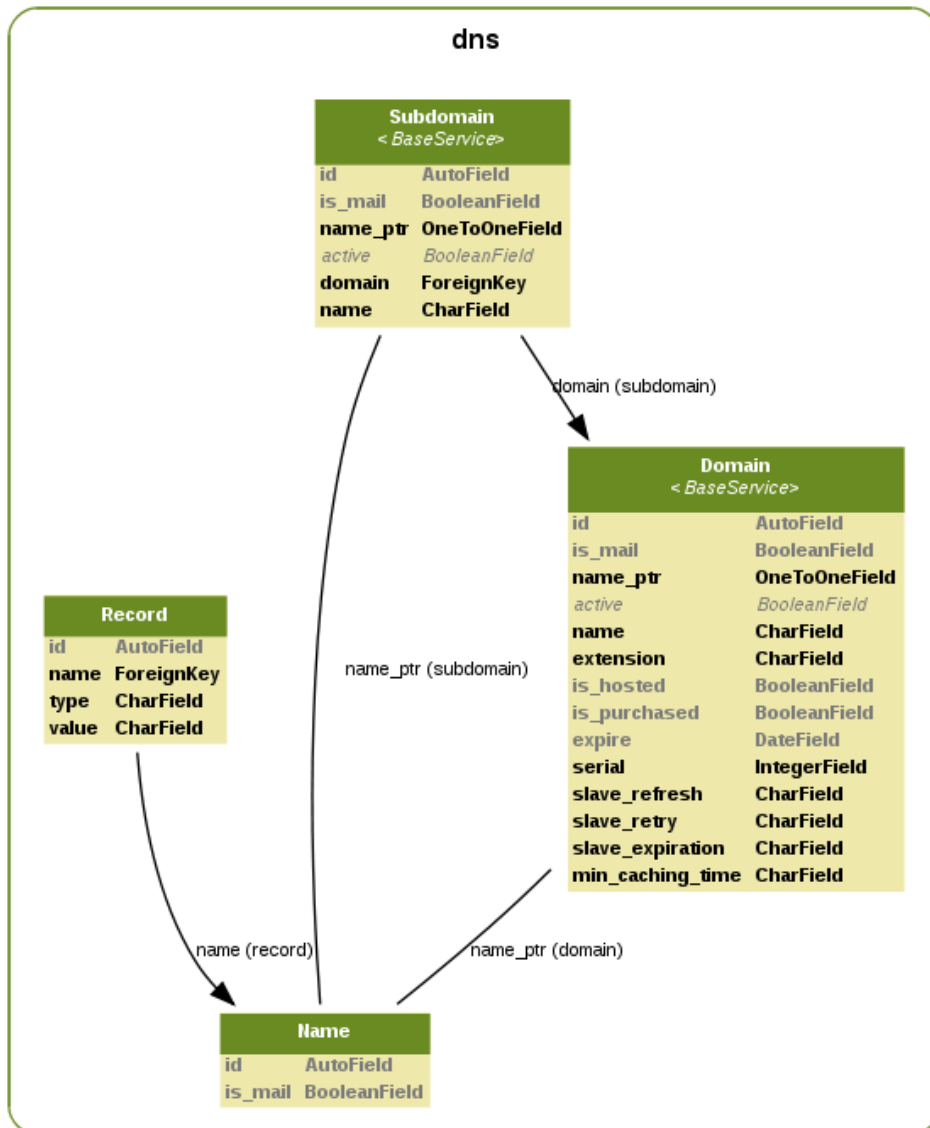


Figure 5.48: DNS application model

5.12.2 DNS Admin

The admin change form of the DNS is represented on the figure 5.49. The DNS records are filled automatically according to DOMAIN_DEFAULT_RECORDS settings value. Figure 5.50 shows the admin change list of the domains, notice that it is possible filter the domain list using the filters located on the right part of the page.

Change domain of contact **Walter Bishop** Contract History

Name: **Extension:**

Expire: Today |

Active

Is mail Is hosted Host the DNS Zone Is purchased Purchase this domain

Optional Fields (**Show**)

Subdomains

Active	Is mail	Name	Delete?
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="text" value="www"/>	<input type="checkbox"/>

[+ Add another Subdomain](#)

Records

Type	Value	Delete?
fringe_division.org web.pangea.org. <input type="text" value="NS"/>	<input type="text" value="web.pangea.org."/>	<input checked="" type="checkbox"/>
fringe_division.org mail.pangea.org. <input type="text" value="NS"/>	<input type="text" value="mail.pangea.org."/>	<input checked="" type="checkbox"/>
fringe_division.org 77.246.179.81 <input type="text" value="A: IPv4 Address"/>	<input type="text" value="77.226.139.83"/>	<input type="checkbox"/>
fringe_division.org 10 mail.pangea.org. <input type="text" value="MX"/>	<input type="text" value="10 mail.pangea.org."/>	<input type="checkbox"/>
fringe_division.org 20 mail2.pangea.org. <input type="text" value="MX"/>	<input type="text" value="20 mail2.pangea.org."/>	<input type="checkbox"/>

[+ Add another Record](#)

[✖ Delete](#) [Save and add another](#) [Save and continue editing](#) [Save](#)

Figure 5.49: DNS admin change form

Select domain to change Recover deleted domains Add domain +

Action: 0 of 12 selected

<input type="checkbox"/>	Name	Extension	Is mail	Is hosted	Is purchased	Active	Contact Link
<input type="checkbox"/>	ssssssssssssss	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Federacio Catalana de sucs de taronja
<input type="checkbox"/>	holaaa	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Jaume prats
<input type="checkbox"/>	samcro	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	SAMCRO
<input type="checkbox"/>	fringe_division	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Walter Bishop
<input type="checkbox"/>	wmca	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Olivia Dunham
<input type="checkbox"/>	lost	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Dharma Initiative
<input type="checkbox"/>	pangea	es	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Pangea
<input type="checkbox"/>	futurama	net	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Bender Doblador Rodriguez
<input type="checkbox"/>	multiplexer2001	es	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Bender Doblador Rodriguez
<input type="checkbox"/>	dellcomputers	info	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Planet Express
<input type="checkbox"/>	moriarty	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Olivia Dunham
<input type="checkbox"/>	t1000	org	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Walter Bishop

12 domains

Filter

By active
 All
 Yes
 No

By extension
 All
 org
 net
 es
 info

By is mail
 All
 Yes
 No

By is hosted
 All
 Yes
 No

By is purchased
 All
 Yes
 No

Figure 5.50: DNS admin change list

5.13 Jobs

Jobs application provides support for those services that have no representation on the IT infrastructure, such as courses, web maintenance, web development, complex service installations and so on. This application is just for recording and accounting purposes.

5.14 Jobs Model

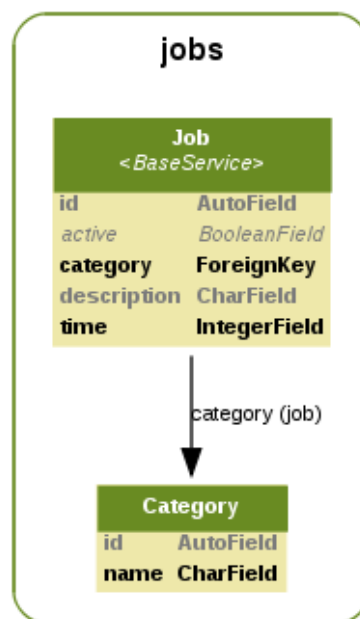


Figure 5.51: Jobs application model

The jobs application models are:

- **Category** model, which allows creating and deleting jobs type definitions
- **Job** model, which are the job instances belonging to a category and composed by a description and the number of hours that the job took

5.14.1 Jobs Admin

Figure 5.52 is how the jobs admin form looks like.

Add job for contact Jaime prats Change Contact

<input checked="" type="checkbox"/> Active
Category: Web development ▼ +
Description: Creation of the new web
Time in hours: 20
<input type="button" value="Save and add another"/> <input type="button" value="Save and continue editing"/> <input type="button" value="Save"/>

Figure 5.52: Jobs app admin add form

5.15 Extra fields

The extra fields application is used to add custom extra fields on your service apps. Recall that the control panel service applications are designed with the philosophy of implementing the basic functionality on a base application and move the specifics to a plugin application. Extra fields is a plugin app that allows you to insert a custom field to a base application.

This application uses dynamic forms pattern in order to insert the extra fields on the corresponding service at runtime. As an interesting concept, this application is not so far from an application that allows to entirely create new service applications with the admin interface.

5.15.1 Extra fields Model

The extra fields application is composed by two models:

- **ExtraField** model, which is responsible of defining an extra field for a concrete service. Letting to choose the field name, the description and the default initial value.
- **ExtraValue** models, that stores the extra field instances.

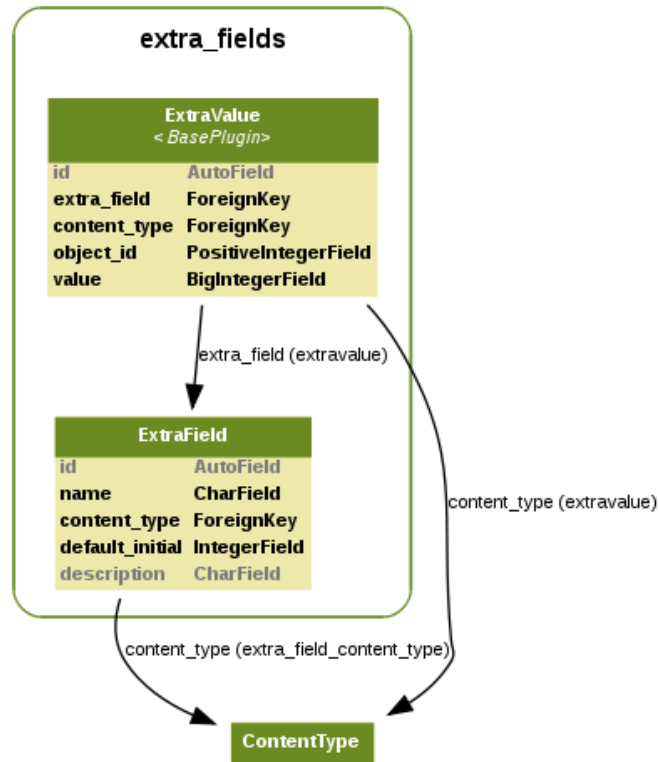


Figure 5.53: Extra fields application model

5.16 Extra fields Admin

Figure 5.54 shows the admin add form page and figure 5.55 displays the form that will be inserted on the change/add form page of the related application, in this case the one related is the contacts app.

Add extra field

Name:	<input type="text" value="likes_chocolate"/>
Content type:	<input type="text" value="contact"/>
Default initial:	<input type="text" value="Null"/>
Description:	<input type="text" value="Contact likes chocolate?"/>

Figure 5.54: Extra fields app admin add form

Extra Values	
Extra Value: #1	
likes_chocolate:	<input type="checkbox"/> True
Contact likes chocolate?	

Figure 5.55: Extra fields dynamic form inserted on contacts change form

5.17 Global interactions

This section provides an overall workflow overview in order to give a better understanding of how the different applications work together. Two use cases, or situations, are presented (1) Create, update and delete a service and (2) Bill a contact.

5.17.1 Create, update and delete a service

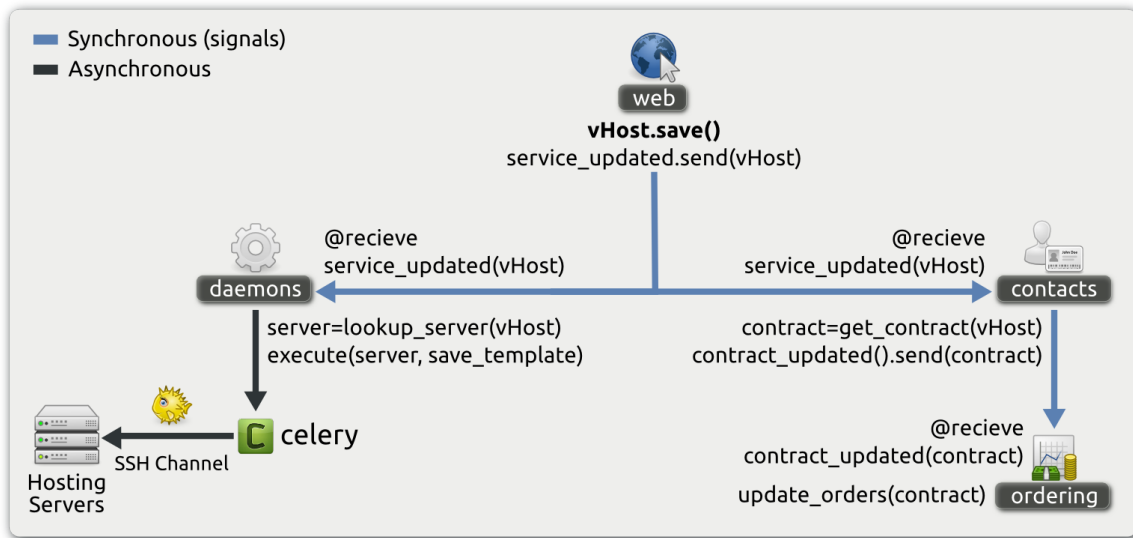


Figure 5.56: Apps interaction on save() service

Considering that the global interactions performing a create, update or delete over a service are very similar with each other, only an example of updating a VirtualHost is analyzed. Figure 5.56 summarize this process.

As we have seen before a service can send 3 different signals depending on the operation type:

1. `service_created()`
2. `service_updated()`

3. `service_deleted()`

In our example case `service_updated()` is sent because the object `vHost` is changed successfully using the web interface. At this point there are two applications listening to these signals: (1) `contacts` and (2) `daemons` app.

1. The `contacts` application behaves a bit different depending on the kind of signal that receives:
 - (a) `service_created()`: creates a new contract and sends `contract_created()` signal
 - (b) `service_updated()`: sends `contract_updated()` signal
 - (c) `service_deleted()`: cancels the related contract and sends a `contract_deteled()` signal

These signals are sent basically for the ordering application who is listening to them in order to keep track of possible changes on the contract related orders. The ordering application will perform an `update_orders(contract)` whenever receives one of those contract signals. This operation creates new orders and cancels others depending on the changes performed on the service.

2. In the other hand when the `daemons` application receives one of the `service_XXX()` signals it makes a lookup in order to determine what is the hosting server which manage the current service and then executes the `save_template` or the `delete_template` on the given server. The `save_template` will be executed on `service_created()` and `service_updated()` and the `delete_template` on `service_deleted()`. This operations on the servers are executed asynchronously using `django-celery`, and in most cases over an SSH channel.

5.17.2 Bill a contact

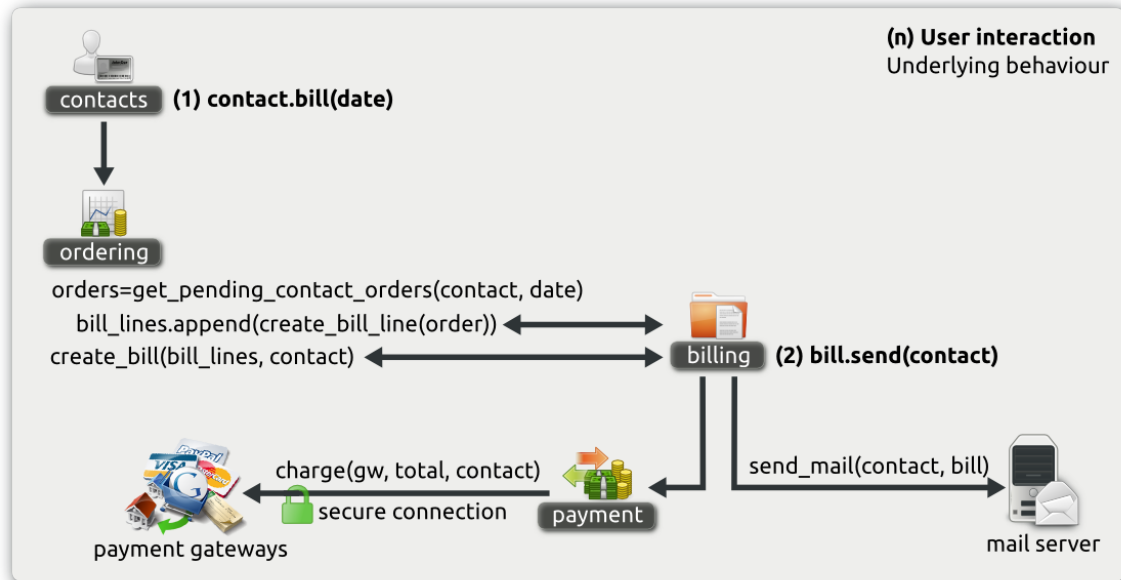


Figure 5.57: Apps interaction on bill() contact

1. When the admin user performs the billing action over a set of contacts the contacts application tells to the ordering application that some contacts must be billed. The date parameter is provided by the admin user and it indicates until what date wants to bill. This is useful to bill multiple periods in advance. The ordering application asks to its billing backend (billing app on our case) to create, first the bill lines, and then the bill with these bill lines.
2. Once the invoice is created the admin user can perform a send action over it. This will (1) send a mail with an electronic copy of the bill to the contact and (2) charge the bill value to the contact through the correct payment gateway. This last step is not yet implemented.

It is planned to run all these operations with the task queue (celery).

Chapter 6

Evaluation

In this chapter an evaluation of the whole project is presented: the dedicated work in terms of hours and economic costs, the implementation with tests and requirements fit (special attention to the reusability since is the main requirement of this project) and at the end of the chapter you can find the planned work to do beyond this thesis.

6.1 Development effort

In this section we are going to show the whole development process in terms of dedicated work. From the very beginning a Subversion (SVN¹) repository has been set up for tracking all the changes on the source code and other related files of the project. Now we can take advantage of this version control system creating some graphs, using `svnplot`², for visual representation of the activity on the repository. The repository activity can show us different development phases, like learning, design, coding, refactoring and documentation.

The presented graphs are:

1. Figure **6.1 Loc and Churn** represents the lines of code (blue line) and code churn (red lines, and it means the lines added, modified or deleted to a file from one version to another). For instance, a high churn might be caused by a code refactoring; or a high increase on the lines of code may indicate that a new project branch has been created.
2. Figure **6.2 Commit Activity Index** reflects the number of commits per day.
3. Figure **6.3 File Count** shows the number of existing files. It might be the sign of the start of a new code branch or the creation of a new Django application.

¹<http://subversion.tigris.org>

²<http://code.google.com/p/svnplot>

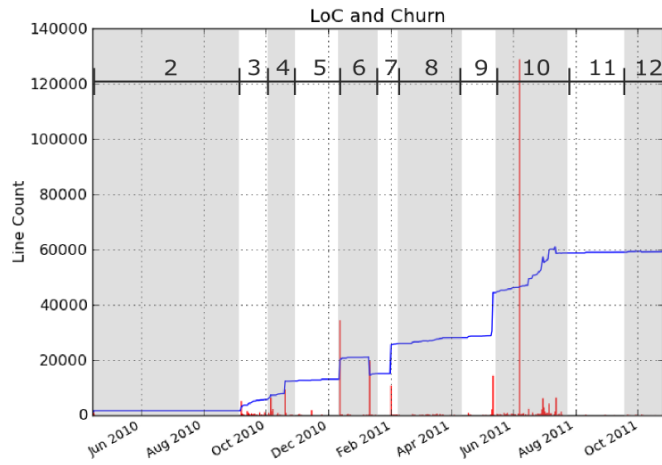


Figure 6.1: Loc and Churn

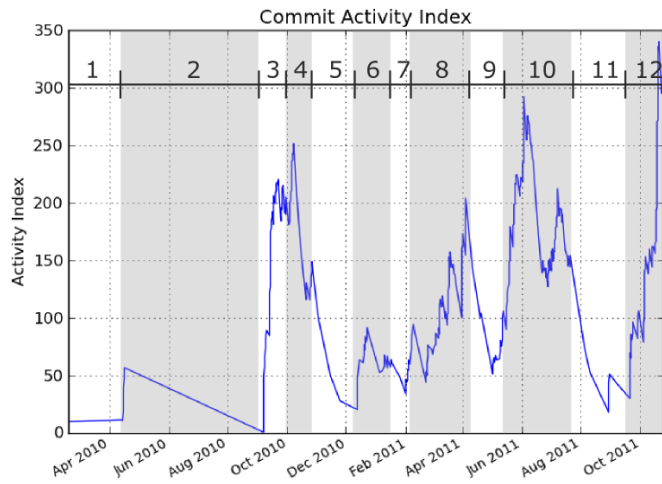


Figure 6.2: Commit Activity Index

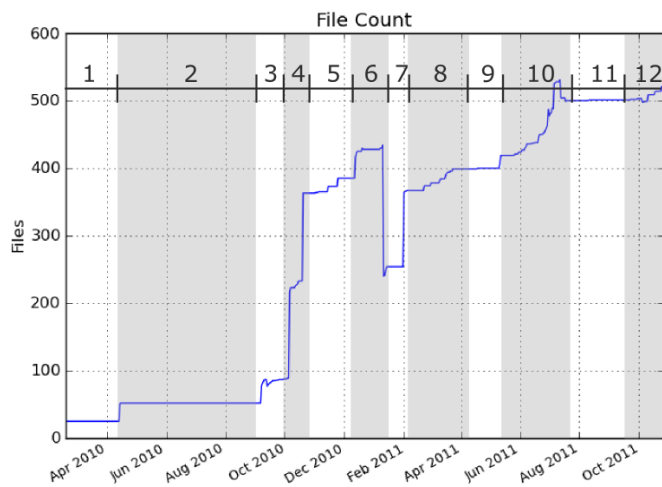


Figure 6.3: File Count

By analyzing the graphs we can differentiate the different phases of the project:

1. Learning Django and Python.
2. Design of the whole system.
3. Writing a prototype based on GISP requirements. I based my initial development on GISP with the idea of implementing the rest of the requirements later. At this point I realized how unfeatured GISP is and how hard was to implement the missing requirements. I spent few weeks asking a lot of questions to my coworkers regarding billing and pricing issues.
4. Refactor with the aim of split behavior into applications.
5. Design refactoring adding contracts and resources concepts.
6. Code refactoring adding contracts and resources concepts.
7. Ordering application refactoring (in deep requirements analysis and design). One of the hardest parts of the project is the ordering application and their featured service definition. It took me few months to come up with a complete and consistent solution.
8. Refactor ordering application (code).
9. Refactor billing design.
10. Code the new billing design and solve some general bugs before deliver a demo version to Pangea.
11. Summer holidays.
12. Writing this report.

Django framework has a steep learning curve so it was very difficult to me to get a deep knowledge of all the framework parts and concepts. Moreover I have not attended any formal course related to software engineering, therefore I had to learn those concepts by myself. It took me 4-5 months to learn Django, Python, design patterns and coding best practices. After this initial learning time I spent most of the time thinking and designing the different parts of the system rather than coding it. Once you have the base knowledge on Django and Python writing the design in code is really fast and fun.

Following the best practices on writing code, making it clean, modular and reusable is fundamental for the success of any open source project. Making it right has a huge long time benefits since it is easy to maintain and other developers can easily join to the project. The downside of writing quality code is that it becomes multiple times more difficult than writing messy code, at least in a short time point of view.

6.2 Global economic analysis

In order to estimate the economic resources needed for a software development project we must consider two expenses. The salary of the developers and the resources needed for them (workstations, repository servers, test servers, office space...). The software licenses are excluded because one of the requirements is that it must be non dependent of commercial solutions.

Our salary estimation is based on an average Spanish developer that works 30 hours a week during a year (including taxes and holidays).

$$1340\text{€}/\text{month} * 14 \text{ months (2 extra)} = 18.900\text{€}$$

The cost of the resources are only 1 workstation valued on 400€ with amortization on 4 years, it is 100€ year.

The total development cost of the first year is up to 19.000€ (deployment and maintainability not included). The development cost may be expensive at the first moment but in long term the facilitation to add new functionalities and the easy maintainability should compensate and make this solution cheaper than others.

The benefits for the organization are on the automation of work and the billing process that takes a high amount of hours from our personal, making it possible to reduce them dramatically with this solution because it pretends to fit all the requirements. I estimate that the panel could save 1 hour of work a day for Lorena, Carlos and I, So as we work in Pangea 4 hours a day, the first year of development should be amortized in the next 2 years.

Notice that the project is not production ready yet as described in the Future work section, so I estimate that at least two more months of development should be applied to the equation.

6.3 Evaluation and testing of the project implementation

We have made continuous testing during the development process. Each time that a new commit was made on the source code we ensured its proper functions by extensively testing the related components assuring that changes did not break anything and new features worked proper. For those parts more tedious to test by hand we have wrote tests with Python unittest³. Also before coding any decision the design has been widely evaluated by Pangea's team.

Maybe the more relevant tests are those tests that involve multiple applications. This is a brief list of some remarkable tests that we have done:

- Install and uninstall applications. For example it is a critical test for the contacts application since it deeply changes the behavior the whole control panel.
- Check the billing system by generating invoices, amendments... and checking the result with Pangea's accountant.
- Check if the services are correctly handled on save and delete operations. This test involves:
 1. Check if the contract is created or canceled.
 2. Check if the correct daemon is trigged.
 3. Check if the correct orders are created or canceled.

When the project was mature enough we have set up a test environment based on *clones*⁴ of our production virtual machines at Pangea (see figure 6.4) with the idea of making anyone with interest on the project able to test it and give us feedback.

The test infrastructure deployment details are: (1) create copies of our three main machines (web.pangea.org, mysql.pangea.org and mail.pangea.org) using *vzdump*⁵ and (2) deploy these *clones* on our lab.

³Python test automation, more at <http://docs.python.org/library/unittest.html#module-unittest>

⁴Exact machine copies of the original

⁵Vzdump is an utility to make consistent snapshots of running OpenVZ containers. It basically creates a tar archive of the container private area, which also includes the container configuration files.

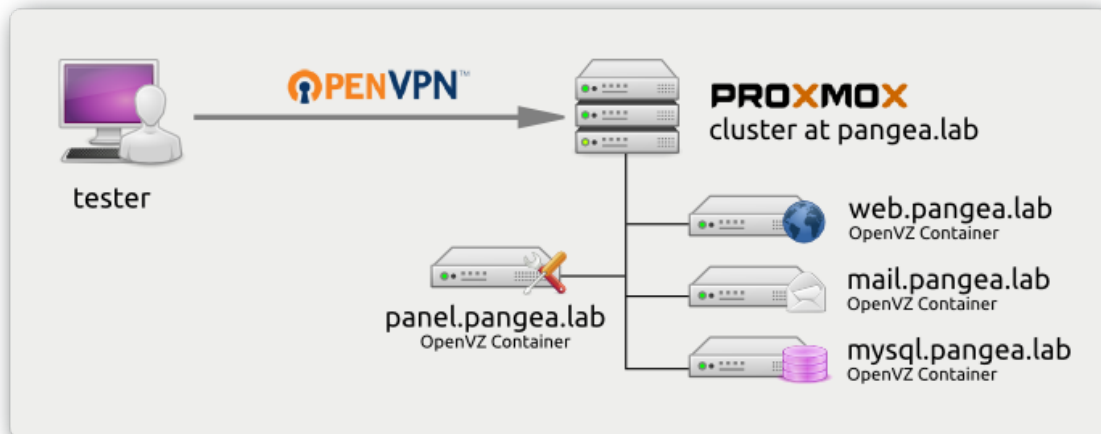


Figure 6.4: Test environment

At this point anyone with a web browser, Internet access and the admin control panel credentials is able to try it, just browsing `django.pangea.org`. But one more step is needed if you want to really check the changes on the DNS zones, virtual hosts, mail accounts and so on. To do that you must get access to our lab VPN⁶ and use `web.pangea.lab` as a primary DNS server.

6.4 Reusability evaluation

The reusability is one of the main requirements of this project. In this section we are going to summarize the reasons that make us believe that we have accomplished this objective.

As we early discussed on this document the fact that this project is released under an open source license, the consistent design, the internationalization already provided by Django framework and the possibility of reuse existing Django applications can make it attractive for people and organizations to use it on their system. In this section we are going to focus on analyzing why the different developed applications have achieved this point too.

Daemon application

1. It is compatible with any Django application out of the box, there is no need to design or adapt an existing applications in order to provide daemon control functionalities.

⁶Virtual Private Network using OpenVPN <http://en.wikipedia.org/wiki/Vpn>

2. The scripts are Django template-based so it is easy for a sysadmin to create or adapt the templates in order to fit their systems.
3. We provide facilities to execute these scripts through SSH because it is widely used on *nix environments. But nothing stops you to use any other remote execution mechanism on your scripts.

Resource application

1. All the points given for the *Daemon* app are valid for this application too.
2. This app does not make any assumption on what a resource is, we provide the configuration parameters needed to let each organization define each resource in a generic way.

Contacts application

1. This app provides generic contact information that should be enough for most organizations. In case of the contact fields provided by this application are not enough to cover the needs of an specific organization the *Extra_field* application can be used in order to dynamically provide the missing fields.
2. This application can provide contract functionalities to any Django application without the need of adapting the Django app.

Ordering application

1. This app does not make any assumption on what a service is, it lets you define your services in a very flexible way, in such a way that anything stored on the control panel database is susceptible to become a service.
2. We provide a huge number of configuration parameters in order to fit the maximum number of possible scenarios.
3. The settings file lets to each organization the ability to configure the default value for these parameters.
4. If you do not want to use our *Billing* application you can still use any other billing system by just editing the `billing_backend.py` file.

Billing application

1. The billing system is designed to be compatible with Spanish billing regulatory rules but all of these regulatory specifics are moved on the settings file and each organization should be able to adjust these configuration parameters in order to satisfy the regulations of their country.
2. Also for each bill type the following attributes can be configured on the settings file: which billing template do you want to use, prefix and number of digits of your bill id's, due date in days and the directory where to store the PDF version of your bills.

Service applications

- The overall reusable decisions for all the service applications is that we put the generic attributes on the main service application and moved the extra stuff to service plugins and the specifics to a settings file. This way we ensure maximum reusability of each component.
- As we have seen in the 5.2 *Common* app section we have extremely reduced the complexity of creating a new service in such a way that you do not need to have advanced programming skills to be able to develop a new service or create an extension. Also the fact that we have moved the common and basic service functionalities on separate applications makes it not necessary to care about the resource control of the new application, neither the contract feature nor billing or ordering. You only need to focus on your application specifics.
- Third party Django applications can be easily reused as a new service for our control panel by using our *Service converter* application, previously seen on page 76 section 5.8.

6.5 Requirements compliance

The presented table 6.5 is an update of the requirements fit summary, table 2.2, presented before on chapter 2, page 29. Our panel is included in order to have a requirement compliance comparison with other existing open source solutions.

Table 6.1: Requirements compliance summary (with our solution)

	Requirement	SysCP	DTC	ISPConfig2	Our
1	Contacts management	Ok	Ok	Ok	Ok
2	Services	Partial	Partial	Fail	Ok
3	Advance pricing configurations	Fail	Fail	Fail	Ok
4	Billing system	Fail	Fail	Fail	Ok
5	Payment gateway	Fail	Fail	Fail	Fail ¹
7	Resource limiting and accounting	Ok	Ok	Ok	Ok
8	Target software to support	Partial	Partial	Partial	Partial ²
9	Multi server support	Partial	Fail	Fail	Ok
10	Easy to use for unskilled users	Ok	Fail	Ok	? ³
11	Easy to add new functionalities	Partial	Fail	Fail	Ok
12	Reusable	Fail	Fail	Partial	Ok
13	Open source friendly	Ok	Ok	Ok	Ok

¹ The Payment gateway is not yet implemented.

² Some software still remains without support, like mailing lists or databases

³ The user interface is not yet implemented, so we can not say if it is easy to use or not :)

6.6 General evaluation

As we have seen on the previous section *6.4 Reusability evaluation*, we have came up with a solution that can be reusable beyond Pangea organization. The solution can be easily adopted in many different contexts, such as (1) corporations that provide hosting services, (2) any non profit hosting organization like Pangea, e.g. Nodo50⁷ or Greenet⁸, (3) for personal use and (4) also for reusing in other projects that have nothing to do with hosting services (think about how many organizations need contact management or a billing system).

During the development of this project we have used some powerful concepts that have not been seen before on an open source web hosting control panel. Like dynamic resource control, truly multi server support, extremely easy creation of new services, message queuing for asynchronous tasks execution, oriented to be easy to integrate on an existing IT infrastructure, a lot of pricing configurations and so on.

This project does not end with this thesis, I will continue working on it until the control panel becomes production ready. In the next section we discuss the planned work to do on the next months.

⁷<http://info.nodo50.org/>

⁸<http://www.gn.apc.org/>

6.7 Future work

Unfortunately it still remains some work to be done before releasing an stable version of this software to the open source community. Below you can find a detailed list of the work that is planned to be done on the short run and in long term too.

Planned work for the first stable release.

1. Some code and APIs should be re-factored and cleaned up before the release.
2. Write Python unittests for all applications.
3. Create the databases application which adds support for database management, also for database user management and permissions.
4. Implement the payment application as an extensible payment interface for multiple payment gateways, implement the Q19 bank procedure and provide hooks to **reuse** third party payment applications such as:
 - `django-paypal`⁹, which is a pluggable application that implements PayPal Payments Standard and Payments Pro.
 - `django-authorizenet`¹⁰, which implements Django integration with Authorize.NET payment gateway. Includes SIM and AIM implementations
 - Satchmo¹¹ payment application. Satchmo is an ecommerce framework built on the Django framework, it includes a payment application with support for several different payment modules including Authorize.net, TrustCommerce, CyberSource, PayPal, Google Checkout, Sage Pay (Formerly Protx) and Sermepa.
5. Implement the APPS app. This application should implements the concept of SaaS “Software as a Service¹²” one click application installer. This application should be able to automatically install any kind of software, previously packaged. For example the packages provided by www.apsstandard.org.
6. Develop the Mailing lists app, which provides support for mailing lists.

⁹`django-paypal` project page <http://github.com/johnboxall/django-paypal>

¹⁰`django-authorizenet` project site <http://github.com/zen4ever/django-authorizenet>

¹¹Satchmo project web page <http://www.satchmoproject.com>

¹²http://en.wikipedia.org/wiki/Software_as_a_service

7. The VPS app, which provides support for creating and managing Virtual Private Servers, also known as virtual machine service.
8. Asynchronous tasks on billing system.
9. Create the remaining daemons and resource control scripts.
10. Give a name to this project.

Long term interesting improvements.

1. Create an automatic interface for panel users like `django.contrib.admin`.
2. Integrate a ticket system.
3. Also it would be interesting to set up a web site for hosting the project and some community tools, such as a ticket system, source code browsing, mailing list, and so on. A sharing templates web page will be interesting too.
4. Reuse `django-reversion` which allow undo delete and change operations through the admin interface.
5. Provide REST API using `django-piston` or `django-rest-framework` in order to allow communication and management from other programs.

Appendix A

Examples

A.1 Template example

```
#!/bin/bash
echo "{{ object.name }}. IN SOA web.pangea.ORG. mail.pangea.ORG. (
    {{ object.serial }}
    {{ object.slave_refresh }}
    '{{ object.slave_retry }}'
    {{ object.slave_expiration }}
    {{ object.min_caching_time }})
{% for record in object.record_set.all %}@
    {{ record.type }} IN {{ record.value }}
{% endfor %}
{% for subobject in object.subobject_set.all %}
    {% for record in subobject.record_set.all %}
        {{ subobject }} {{ record.name }} IN {{ record.value }}
    {% endfor %}
{% endfor %}" > /etc/bind/master/{{ object.name }}

if [[ $(grep "^{{ object.name }}" /etc/bind/primarios.conf)]; then
    echo {{ object.name }} >> /etc/bind/primarios.conf; fi
rndc reload
```

A.2 Invoice example

A.3 Fee example

Figure A.1: Invoice example



Pangea
Internet solidari

Coordinadora Comunicació per a la Cooperació Pangea
 ES32342423
 Pl. eusebi guell 6-7
 Edifici vertex, planta 0
 08034 - Barcelona

934015664
 suport@pangea.org
 pangea.org

Invoice
F20110231

El petit mosntre
 ES83229101
 c/ yoquese, 43
 08034 - barcelona

INVOICE DATE	TOTAL	DUE DATE
Oct 20, 2011	1340,00 €	Nov 20, 2011

id	description	hrs/qty	rate/price	subtotal
232	Mail account, monstre@pangea.org (10, 2011 - 04, 2012) Discount incomplete period Discount per volume	1		20,00 € -10,00 € -20,00 €
245	Domain elpetitmonstre.org	1	15,00 €	15,00 €
33	MySQL database, monstre (11, 2011 - 04, 2012) Discount incomplete period	1	24,00 €	24,00 € -5,00 €
33	MySQL database. monstre (04, 2012 - 04, 2013)	1		24,00 €
33	MySQL database, monstre (04, 2013 - 04, 2014)	1		24,00 €

	subtotal 1200,00 €
	taxes 140,00 €
	total 1340,00 €

Page 1 of 1

COMMENTS The comments should be there.

PAYMENT You can pay our invoice by bank transfer. Please make sure to state your name and the invoice number. Our bank account number is **3432.323.23.23123 (Caixa d'Enginyers)**.

QUESTIONS If you have any question about your bill, please feel free to contact us at your convenience. We will reply as soon as we get your message.

Figure A.2: Fee example



Coordinadora Cooperacio per a la Comunicacio Pangea
ES32342423
Pl. eusebi guell 6-7
Edifici vertex, planta 0
08034 - Barcelona

934015664
suport@pangea.org
pangea.org

El petit monstre
ES83229101
c/ yoquese, 43
08034 - barcelona

Membership Fee
Q20110231
Nov 20, 2011

€ 1342,34
To pay before Oct 20, 2011
on 232.2323.23.23233

from **Apr 1, 2010** to **Apr 1, 2011**

Con vuestras cuotas, además de obtener conexión y multitud de servicios, estáis apoyando un proyecto solidario de comunicación tanto en nuestro país como en otros: Colombia, México, Perú, ... Pangea forma parte de la Asociación para el Progreso de las Comunicaciones (APC) con nodos asociados en más de 60 países, trabajando para fomentar la cooperación y la solidaridad.

COMMENTS The comments should be there.

PAYMENT You can pay our invoice by bank transfer. Please make sure to state your name and the invoice number. Our bank account number is **3432.323.23.23123 (Caixa d'Enginyers)**.

QUESTIONS If you have any question about your bill, please feel free to contact us at your convenience. We will reply as soon as we get your message.

Coor. Com. per a la Coop. Pangea Pl. Eusebi Guell 5-6 Edifici Vertex Planta 0 08034 - Barcelona - Spain	Tel (+34)934015664 Fax (+34)934015664 Email suport@pangea.org Web pangea.org	NIF G23324134 Bank 2232.2323.23.23213 IBAN 23123.232322.2321.3.123 BIC 2313.21323
---	---	--

Appendix B

Manuals

B.1 Installation

On our SVN repository you can find a bash script called `deploy_dev.sh`¹ that automatizes the tedious work of installing and setting up a software project and their dependencies. Please notice that the deployment performed by this script is not suitable for production, it is only for development and testing purpose.

As you can see on figure B.1 `deploy_dev.sh` script provides support for three different deployment types (only for Linux platforms):

1. Deploy on your local Debian like distribution, most common are: Debian, Ubuntu and Linux Mint. This option will install the control panel on your own system.
2. Deploy on a Debian chroot (recommended). The control panel and all its dependencies will be installed inside a Debian chroot. A chroot jail is like a virtualized machine but only ensuring filesystem isolation, this allows to install a self contained environment where applications can run. This is useful when you want to install some software with a lot of dependencies and you do not want to mess your system.

With this option two additional and optional scripts can be installed on your machine. One on `/etc/init.d/ucp_chroot` that makes easier the task of start and stop the chroot. The second will be installed on `/usr/sbin/ucp`, this is useful to enter inside the chroot with all the environmental state ready.

¹http://ucp.pangea.org/svn/ucp/trunk/ucp/scripts/deploy_dev.sh

3. Deploy on a LXC container². With this option all the environment is encapsulated inside a Debian LXC container. This option requires some background knowledge of LXC in order to be able to configure the network and manage the container life cycle.

One consideration to take into account is that the script only has support for MySQL databases (Fig. B.2). Also it will install a useful `init`³ script on `/etc/init.d-django_server` that starts the Django development server inside an `screen`. `Screen` is a UNIX application that can be used to multiplex several virtual consoles, allowing a user to access multiple separate terminal sessions inside a single terminal window or remote terminal session.

```
~/tmp$ sudo /home/debian/rootfs/home/ucp/trunk/ucp/scripts/deploy_dev.sh
This script only supports development deployment with MySQL server.
If you want to use any other db backend please cancel this script and make the installation manually

Hit enter to continue...

1. Chose one of the following deployment option:
  1) Deploy on local environment (only Debian) (not tested)
  2) Deploy on Debian Chroot (any distro.) (Recommended)
  3) Deploy on Debian LXC container (any distro.)
Choice: 2

Provide a target directory: /tmp/Control_Panel
Creating chroot environment...
chroot: cannot change root directory to /tmp/Control_Panel: No such file or directory
I: Retrieving Release
W: Cannot check Release signature, keyring file not available /usr/share/keyrings/debian-archive-keyring.gpg
I: Retrieving Packages
I: Validating Packages
I: Resolving dependencies of required packages...
I: Resolving dependencies of base packages...
I: Found additional required dependencies: insserv libbz2-1.0 libdb4.8 libslang2
I: Found additional base dependencies: libnfnetlink0 libsqlite3-0
I: Checking component main on http://ftp.us.debian.org/debian...
I: Retrieving libacl1
I: Validating libacl1
I: Retrieving adduser
I: Validating adduser
I: Retrieving apt-utils
I: Validating apt-utils
I: Retrieving apt
```

Figure B.1: Creating a chroot with `deploy_dev.sh`

²LXC (Linux Containers) is an operating system-level virtualization method for running multiple isolated Linux systems (containers) on a single control host. It is similar OpenVZ that we have seen early on chapter 1 section 1.2.4.

³Init (short for initialization) is a program for Unix-based computer operating systems that spawns all other processes.

```

I: Configuring gpgv ...
I: Configuring gnupg ...
I: Configuring debian-archive-keyring ...
I: Configuring apt.
I: Configuring libept1 ...
I: Configuring apt-utls...
I: Configuring aptitude...
I: Configuring taskel-data...
I: Configuring taskel...
I: Base system installed successfully.

2. MySQL configuration
  1) Install and configure new MySQL server
  2) Use an existing MySQL server
Choice: 2

You chose use an existing MySQL server
Please before provide the connection parameters be sure that the connection is gonna be working
Database name: ucp
Database user name: ucp
Database password: ucp
Database server address: localhost

3. Installing dependencies
Hit enter to continue...

```

Figure B.2: Configuring MySQL server with `deploy_dev.sh`

```

U: /usr/share/django
Checked out revision 17171

5. Create and configure a system users for the panel
Please enter the password for users root and ucp
Default password will be 'ucp' (just pres Intro key).
Password: ucp
User configured.

6. Installing and configuring the control panel

7. Configuring celery as a daemon...
update-rc.d: using dependency based boot sequencing

8. Installing custom scripts...
Enter a port number for the dev web server [8080]: 8080
update-rc.d: using dependency based boot sequencing
Insserv: warning: script 'django_server' missing LSB tags and overrides

As you chose chroot deployment we highly recomend install an initt.d script on your system that automatically starts the chroot on each power on.
Do you want to install this script? yes
update-rc.d: warning: /etc/initt.d/ucp_chroot missing LSB information
update-rc.d: see <http://wiki.debian.org/LSBInitScripts>
System start/stop links for /etc/initt.d/ucp_chroot already exist.

9. Starting the control panel...
You can execut 'ucp' command as root in order to enter on the chroot

Installation Complete
The web dev server runs within an SCREEN under the 'ucp' user.
You can stop/start this server using /etc/initt.d/django_server script

Check if the panel is working browsing http://localhost:8080/admin

```

Figure B.3: Screenshot of the last `deploy_dev.sh` steps

If you want to install the control panel on other platforms not supported by our `deploy_dev.sh` script, or you just like to take care of the installation by yourself, you just have to follow these general steps:

1. Create a new database on your favourite DB server officially supported by Django:

- PostgreSQL
- MySQL
- SQLite
- Oracle

Or supported for third parties:

- Sybase SQL Anywhere
- IBM DB2
- Microsoft SQL Server 2005
- Firebird
- ODBC
- ADSDB

Additional note: to take advantage of the control panel transaction management you must use a database engine with transaction support, like for example PostgreSQL or MySQL with InnoDB tables.

2. Install the control panel dependencies by this order:
 - (a) Python \geq 2.6, python-paramiko, python-pisa and python-reportlab
 - (b) RabbitMQ (or your favorite broker supported by Celery)
 - (c) Django framework⁴, python-reportlab and django-celery

3. Install the control panel from our SVN repository:

```
svn co http://ucp.pangea.org/svn/ucp/trunk/ucp
```

4. Configure PYTHONPATH and DJANGO_SETTINGS environment variables:

```
export PYTHONPATH=/base_dir/ucp
export DJANGO_SETTINGS=ucp.settings
```

5. Go inside the *installation dir*: `cd install_dir`

6. Configure the connection with your database through `settings.py` file.

7. Create the database tables with

```
python manage.py syncdb
```

8. Make sure that you have rabbitmq, celeryd and celerymon working.

9. Run the Django development server:

```
python manage.py runserver 127.0.0.1:8080
```

10. And try if it is working browsing `http://127.0.0.1:8080/admin`

⁴How to install Django <https://docs.djangoproject.com/en/dev/topics/install/>

B.2 Create and install new service

1. Execute `python manage.py startapp startapp lists` in order to create the skeleton of the new application, a mailing list in our example case.
2. Edit `lists/models.py` and add the following lines:

```
from common.models import BaseService
from dns.models import Name

class List(BaseService):
    name = models.CharField(max_length=32)
    domain = models.ForeignKey(Name)
    admin_password = models.CharField(max_length=16)
```

3. Create the file `lists/admin.py` with this content:

```
from django.contrib import admin
from lists.models import List
from common.admin import ServiceAdmin

class ListAdmin(ServiceAdmin):
    pass

admin.site.register(List, ListAdmin)
```

Optionally you can customize the admin interface by defining `ModelAdmin` attributes, see <https://docs.djangoproject.com/en/dev/ref/contrib/admin/>

4. Add the new applications to the `INSTALLED_APPS` on `project/settings.py`.

```
INSTALLED_APPS = (
    'djcelery',
    'admin_tools.theming',
    ....
    'web',
    'php',
    'mail',
```

```
        'lists',
        .....
    )
```

5. Run django syncdb command:

```
python manage.py syncdb
```

6. Create the daemon save and delete templates if needed.

7. Create the resource control templates if needed.

8. Restart your server

9. Ensure that all the steps on *B.3 Admin use guide* are applied for the new application.

B.3 Admin use guide

We consider that the panel admin interface work-flow is intuitive enough so we only are going to provide the description of the initial steps that maybe are not much friendly.

After the installation and configuration the following steps should be followed, at the same order as presented here.

1. Add the hosts that you want to control. `Daemons.Hosts`
2. Create or modify the daemon save and delete script templates.
3. Add the daemons that will be behind your hosting services.
4. Configure your resources and create or modify the resource template scripts if needed.
5. Add extra fields to your services if you want some special information.
6. Create your non-hosted services with jobs applications.
7. Create packs (hosting plans and offers).
8. Configure the services for future billing.

At this point your control panel installation should be ready to add your contacts, start creating new services for them and bill.

Bibliography

- [1] *Design Patterns: Elements of Reusable Object-Oriented Software*. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (the GangOfFour) 0-201-63361-2, November 10, 1994
- [2] *Clean Code: A Handbook of Agile Software Craftsmanship*. Robert C. Martin 0132350882, August 11, 2008
- [3] *Code Complete: A Practical Handbook of Software Construction*. Steve McConnell, 978-0735619678, 2on Edition, July 7, 2004
- [4] *The Pragmatic Programmer: From Journeyman to Master*. Andrew Hunt and David Thomas ISBN: 020161622X Published by Addison-Wesley, Oct 1999
- [5] *Pragmatic Thinking and Learning: Refactor Your Wetware*. Andy Hunt ISBN: 978-1-93435-605-0 2008-09-15
- [6] *Django documentation*. <http://docs.djangoproject.com>
- [7] *Django source code*. <http://code.djangoproject.com/svn/django/trunk>
- [8] *Django book*. Adrian Holovaty and Jacob Kaplan-Moss, <http://djangobook.com>
- [9] *DjangoCon speaks*. <http://blip.tv/djangocon>
- [10] *Wikipedia. The Free Encyclopedia*. <http://en.wikipedia.org>