UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

”GAUDII: Generador Automático de Diseños Inteligente”

Víctor José Martín Ramírez

Febrero, 2010

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Tecnologías y Sistemas de Información

PROYECTO FIN DE CARRERA

GAUDII : Generador Automático de Diseños Inteligente

Autor: Víctor José Martín Ramírez
Director: Carlos González Morcillo

Febrero, 2010

TRIBUNAL:

Presidente:
Vocal 1:
Vocal 2:
Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL 1	VOCAL 2	SECRETARIO
Fdo.:	Fdo.:	Fdo.:	Fdo.:

© Víctor José Martín Ramírez. Se permite la copia, distribución y/o modificación de este documento bajo los términos de la licencia de documentación libre GNU, versión 3.0 o cualquier versión posterior publicada por la *Free Software Foundation*, sin secciones invariantes. Puede consultar esta licencia en http://www.gnu.org.

Este documento fue compuesto con LaTeX.

Resumen

El diseñador gráfico es aquel que concibe, programa y crea soluciones visuales para unos problemas de comunicación específicos mediante el uso de reglas de composición y principios estéticos. Su trabajo es de los pocos empleos que presenta una importante progresión para los próximos años, lo que demuestra que existen una creciente demanda para la creación de obras gráficas

El presente proyecto ofrece un sistema automático de generación de obras gráficas que utiliza principios y técnicas de Computación Evolutiva y Lógica Difusa para capturar, modelar y utilizar el conocimiento experto del diseñador gráfico.

Abstract

Graphic designers conceive, plan and create visual solutions to specific communications problems by using layout rules and aesthetic principles. This kind of work is one of the few that shows an important growth for the next decade, which proves that there is an increasing demand for the creation of graphic works.

The presente work offers an automated system for the generation of graphics items that utilizes the principles and techniques known from the fields of Evolutionary Computation and Fuzzy Logic to capture, model and use the expert knowledge of a graphic designer.

A mis padres, hermanos y tías,
por llevar siempre la sonrisa puesta.

Agradecimientos

A Carlos González, por su ayuda, entrega, ideas y amistad.

A mi familia, porque es su culpa que haya llegado hasta aquí.

A todas las amistades hechas estos años en Ciudad Real y Creta, porque tantos y tan buenos momentos no cabrían aquí.

A Vito, por estar siempre atenta a sujetarme cuando me tambaleo.

Índice general

Índice de figuras

Lista de algoritmos

Capítulo 1

Introducción

1.1. Motivación

Los diseñadores gráficos son aquellos que planean, analizan y crean soluciones visuales a problemas de comunicación [51]. Su trabajo consiste en saber utilizar herramientas como la tipografía, colores, ilustraciones o fotografías para transmitir un mensaje a través de un determinado medio, como puede ser una página web, un cartel o el envase de un producto.

El trabajo de un diseñador gráfico está siendo cada vez más solicitado en áreas como la publicidad, prensa o diseños corporativos. Según la oficina de Estadísticas Laborales de los Estados Unidos [51], la demanda de este empleo crecerá hasta un 13 % en el periodo comprendido entre el 2008 y el 2018, una época donde muchas otras áreas laborales están sufriendo para mantener el número de empleados. Podemos afirmar, por tanto, que existe una clara necesidad del trabajo del diseñador gráfico.

Esta necesidad no siempre está relacionada a tareas de ámbito profesional o ligada a una campaña de publicidad a nivel internacional, sino que también existe otro tipo de usuarios reclamando esta actividad a pequeña escala. Y aunque la disciplina de diseño gráfico depende enormemente de la experiencia y conocimiento del profesional, existen reglas y principios básicos [59] [68] para crear elementos de diseño visualmente atractivos.

Por otro lado, cada vez más Sistemas Expertos forman parte de nuestra vida ayudando al ser humano en múltiples tareas. Estos sistemas tienen como objetivo emular la actividad de un experto humano en un determinado campo. Su labor quizá más importante, y conocida,

está en la medicina, donde asisten a los médicos en el diagnóstico y tratamiento de multitud de enfermedades. Pero existen otros campos que se ven claramente beneficiados por estos sistemas, desde plataformas de vigilancia inteligentes hasta controladores de producción en las fábricas.

A pesar de esto, pocos han sido los intentos de llevar el mundo de los sistemas expertos a alguna disciplina artística. Los casos más notables, y que se estudiarán más adelante, centran sus esfuerzos en el campo musical, un área de conocimiento que ha sido objeto de estudio durante siglos en busca de la armonía melódica. Para las artes gráficas, sin embargo, apenas existen trabajos que traten de capturar y modelar el conocimiento experto de su campo.

Además de la dificultad inherente de capturar el conocimiento experto de una disciplina, existe el problema de que, como todas las artes, no existe una única solución posible. Es decir, lo que muchos usuarios pueden considerar como excelente, otros tantos podrían tacharlo de mediocre.

En definitiva, hay dos factores claves en la creación de este proyecto. El primero es el auge del diseño gráfico, un campo con magnificas perspectivas debido a su creciente demanda; el segundo es la ausencia de trabajos que automaticen tareas relacionadas con esta disciplina.

Así, Gaudii se presenta como un sistema automático para generar elementos gráficos. El sistema utiliza técnicas de Computación Evolutiva y métodos de Optimización Local que buscan soluciones válidas empleando conocimiento experto difuso. De esta manera, el proyecto intenta maximizar la calidad del diseño usando funciones de coste a la vez que sigue los fundamentos del diseño gráfico.

1.2. Estructura del documento

El presente documento se estructura en seis capítulos y diversos anexos. El contenido de estos capítulos se detalla a continuación:

- **Capítulo 1: Introducción:** El capítulo actual, donde se presenta brevemente el problema al que se enfrenta al proyecto y la motivación detrás de su creación. También se describe la estructura que sigue el documento.

- **Capítulo 2: Objetivos del proyecto:** En este capítulo se definen de forma detallada los objetivos marcados por el proyecto.
- **Capítulo 3: Antecedentes, estado de la cuestión:** En este capítulo se expondrán las diversas áreas con las que este proyecto guarda relación, profundizando también en las tecnologías utilizadas en el desarrollo.
- **Capítulo 4: Metodología de trabajo:** Durante este capítulo se especifica el proceso de desarrollo seguido en la implementación del sistema.
- **Capítulo 5: Resultados obtenidos:** En este capítulo se muestran distintos ejemplos de pruebas realizadas con el sistema, así como los tiempos que necesita el sistema en cada etapa del proceso.
- **Capitulo 6: Conclusiones y propuestas:** Por último, en este capitulo se analiza en qué grado se han cumplido cada uno de los objetivos del Capítulo 2. Además, se describen las posibles líneas de investigación que podrían abordarse para mejorar el proyecto.

Capítulo 2

Objetivos del proyecto

El objetivo principal de este proyecto es el modelado y utilización de conocimiento experto en el área de diseño gráfico para la generación automática de composiciones artísticas. En concreto, el proyecto se centrará en la definición del conocimiento experto estrictamente relacionado en el área de diseño gráfico de carteles, aunque el proyecto es fácilmente extensible a otras áreas de diseño gráfico.

La hipótesis de trabajo sobre la que se pretende conseguir el objetivo expuesto es que es posible modelar el conocimiento experto en el área de diseño gráfico, además de definir una serie de funciones de evaluación de la bondad de diferentes instancias para proporcionar diferentes alternativas de diseño que cumplan ciertas restricciones al usuario. De esta forma se generará una población de diseños empleando técnicas de softcomputing que serán evaluados automáticamente por el sistema.

Las funciones de evaluación, reglas de diseño, valoración y ponderación de pesos de cada atributo serán aisladas de los métodos de resolución de forma que puedan trasladarse a otras áreas de trabajo.

De este objetivo general se definen una serie de requisitos que deben cumplirse y que dotarán al sistema de la generalidad necesaria para su posterior aplicación a otras áreas de diseño gráfico.

- **Modelado de reglas de composición gráfica.** La colocación de los elementos en el diseño dará como resultado una creación artística, y por tanto debe realizarse empleando

reglas de composición gráfica. El sistema deberá permitir el trabajo con altos niveles de incertidumbre en el modelado de cualquier conocimiento experto en disciplinas artísticas.

- **Automatización en el proceso de creación**. Se debe minimizar la intervención humana en el proceso de generación de diseños. El usuario proporcionará textos y alguna imagen que quiera utilizar para el diseño. El sistema deberá ser capaz de combinar esta información utilizando las reglas de composición gráfica definidas anteriormente.

- **Adaptabilidad al gusto particular de cada usuario**. La plataforma debe permitir al usuario guiar el proceso de generación de obras, adaptándose a sus gustos generando obras de acuerdo a sus preferencias. Esta información podrá ser usada por el sistema para obtener futuras obras para un mismo usuario.

- **Diversidad y heterogeneidad**. El sistema debe ser capaz de generar distintas obras que serán presentadas al usuario. Éste podrá elegir entre las alternativas que se le presentan o guiar al sistema en la creación de otras parecidas a un subconjunto de éstas.

- **Interfaz multidispositivo**. El sistema podrá ser usado mediante un entorno web sin la necesidad de ningún tipo de instalación en el ordenador o dispositivo del usuario. Se emplearán mecanismos de interacción avanzados para evitar la recarga del interfaz en primer plano mediante tecnologías de petición asíncrona.

- **Sistema multiplataforma**. El sistema debe desarrollarse tal que pueda ser utilizado en distintos sistemas operativos y distinto hardware. Para conseguir este aspecto se desarrollará, como se ha explicando anteriormente, sobre una interfaz web.

- **Desarrollar del sistema empleando estándares y software libre**. Para facilitar su explotación y posterior ampliación y mantenimiento, el sistema se desarrollará empleando estándares y software libre.

Capítulo 3

Antecedentes, estado de la cuestión

3.1. Trabajo relacionado

Existen numerosos trabajos de sistemas expertos que ayudan a diseñar, parcial o completamente, algún proyecto específico. Este tipo de aplicaciones son, en su mayoría, trabajos dedicados a optimizar un diseño cuyas bases son matemáticas o físicas, como puede ser el diseño de circuitos o como ciertas herramientas arquitectónicas.

Por tanto, la mayoría de los trabajos de diseño están enfocados a campos donde el resultado final es fácilmente mesurable. Es decir, existen formas de saber cuando el diseño de la planta de un edificio o el de un circuito es óptimo, mientras que en las disciplinas gráficas este cometido no es tan fácilmente medible. Es por esto que hay pocos sistemas expertos que se dediquen a ésta tarea, por lo que nos encontramos en un área relativamente inexplorada.

Dentro de las artes gráficas, el trabajo de DiPaola et al. [26] sugería la utilización de algoritmos genéticos para buscar la creatividad en los ordenadores sin la necesidad de intervención humana. Hasta entonces los humanos eran los que le decían al programa si una obra era buena o mala, pero el trabajo de DiPaola obtenía la bondad de cada obra mediante una función que medía, por un lado, la exactitud con la que la obra de pintura se asemejaba al original y, por otro lado, el uso de distintas técnicas de pintura que el sistema almacenaba como reglas de creación artística.

Otro trabajo interesante es el de Anderson et al. [18], en el que describían un sistema evolutivo e interactivo (es decir, necesitaba de la intervención humana) para crear diseños

atractivos de baldosas y que estos pudieran evolucionar gráficamente. El sistema usaba distintas figuras geométricas y colores para crear los diseños, y el usuario podía indicar al sistema si una determinada parte del diseño le gustaba para que los siguientes se generasen de esa misma manera. Este sistema es interesante puesto que normalmente, en los diseños genéticos y evolutivos, los hijos no tienen porque mantener ciertas características idénticas que los padres.

El trabajo de Chiba et al.[23] se concentraba en la tarea de adquirir el conocimiento detrás de los diseños. Chiba postulaba que es complicado adquirir el conocimiento de un experto y exponerlo de forma que la máquina pueda entenderlo, por lo que obtenía el conocimiento experto a partir de ejemplos de diseño gráfico. El conocimiento de estos diseños se adquiría en forma de Programación Lógica Inductiva (ILP), un sistema de aprendizaje máquina que permite manejar los elementos de un diseño y sus relaciones.

Chiba entrenaba al sistema con ejemplos positivos o negativos. Según ellos, este tipo de conocimiento se podía dividir en dos partes: la composición, por un lado, y el resto de elementos del diseño por otro (colores, patrones, etc.). Su trabajo se centra en la composición, donde se diferenciaba, por un lado, la bondad de un diseño frente a los principios de composición y, por otro lado, las sensaciones que transmitía el diseño, concentrando su trabajo en los principios de composición.

Otro trabajo relacionado con este campo es el de Mackinlay [44], quien expone una herramienta que diseña presentaciones de forma automática. Estos diseños se crean teniendo en cuenta dos factores: *Expresividad* y *Efectividad*. El primero hace referencia a los tipos de lenguaje gráfico que usamos para representar la información, mientras que el último identifica cual de estos lenguajes es el más apropiado en cada situación.

Los sistemas comentados tratan de modelar el conocimiento experto para acometer tareas concretas. En la siguiente sección definiremos los sistemas expertos y estudiaremos con detalle sus características y aplicaciones.

3.2. Sistemas Expertos

Un Sistema Experto puede definirse, según Siddall [61], *como un algoritmo para acometer decisiones o predicciones de forma automática que está relacionado con algún área específica y que requiere del consejo de algún experto [para su desarrollo]*; según Kandel [37] *como un programa de ordenador que emula el proceso de razonamiento para realizar, tal y como lo haría el experto, una tarea para la que este no está*; o según Petrik [53] como *un sistema basado en el razonamiento que ejecuta tareas dentro de un dominio específico igual o mejor que su equivalente humano*.

Por otro lado, un experto es, según Slade [63], *alguien que tiene un vasta experiencia en una especialidad concreta, que ha sido testigo de numerosos casos en ese dominio y que ha generalizado esta experiencia para aplicarla a situaciones nuevas*.

Un Sistema Experto es, por tanto, un programa, parte de la rama de la Inteligencia Artificial, que permite utilizar el conocimiento y las conclusiones de un experto humano. En esta definición reside toda la importancia de desarrollo de estos sistemas.

Al igual que en la época de la Revolución Industrial la cantidad de máquinas de vapor era el símbolo de la riqueza de las naciones, hoy en día se podría decir que el conocimiento de su población es su mayor patrimonio, tanto para ella misma como la propia raza humana [29], y los Sistemas Expertos son capaces de capturar y modelar ese conocimiento.

3.2.1. Algunos ejemplos de Sistemas Expertos

El 11 de Mayo de 1997, un hombre se levantó resignado de su asiento porque una máquina había conseguido hacer su función mejor que él mismo. Él, que había sido considerado como el mejor en su trabajo, vio con sus propios ojos como un ordenador lo sobrepasaba. Su nombre era Garry Kasparov, campeón mundial de ajedrez, y su némesis tomaba el nombre de Deep Blue, un Sistema Experto diseñado por IBM cuya área de conocimiento era el conocido juego de tablero.

Este acontecimiento revolucionó el campo de la Inteligencia Artificial [25], especialmente por toda la repercusión que obtuvo, pero los Sistemas Expertos ya llevaban mucho tiempo entre nosotros realizando tareas menos mediáticas.

En 1975 se presentó MYCIN [60], un proyecto desarrollado durante 20 años en la Universidad de Stanford para ayudar a los médicos en el diagnóstico y tratamiento de los pacientes con enfermedades causadas por bacterias en la sangre, afecciones que pueden causar la muerte instantánea si no se tratan rápidamente. Este Sistema Experto demostró ser igual o mejor que sus equivalentes humanos en el cuidado de este tipo de enfermedades.

En 1988 se puso en marcha DUSTPRO [27], una plataforma que aconsejaba a los operadores de las minas de carbón cuando había demasiada contaminación en el aire como para considerarse nocivo para su salud (el polvo de carbón en las minas es la primera causa de Neumoconiosis para los mineros). El sistema funcionaba usando 30 Sistemas Expertos, de forma distribuida, que tomaban datos del equipo de ventilación y de los procedimientos de control para asistir a los controladores en el manejo del sistema de ventilación de la mina.

En 1999 se presentó una aplicación [55] que permitía al ordenador actuar como instrumento de acompañamiento al músico solista. Este sistema primero escucha la melodía principal y, con ella aprendida, interpreta una canción que le acompañe de forma armónica.

Actualmente los sistemas expertos están muy asentados en nuestra sociedad, y existen multitud de proyectos que capturan y utilizan el conocimiento experto. En el 2009, Wu et al. [70] trabajaron en un sistema experto que diagnosticaba los fallos producidos en un motor de combustión evaluando el sonido que este emitía. También en el 2009, Karabatak et al. [38] desarrollaron un sistema experto que, utilizando reglas de asociación y redes neuronales, era capaz de detectar cáncer de pecho en un 95,6 % de las muestras.

3.2.2. Componentes de un Sistema Experto

Los elementos principales de un Sistema Experto son una **Base del Conocimiento** (KB), **una Base de Afirmaciones (o hechos)** y un **Motor de Inferencia**. El esquema de estos tres elementos puede verse en la Figura 3.1.

En la **Base del Conocimiento** (KB) se almacena todo el conocimiento que hemos obtenido del experto. Esto se hace durante la fase de adquisición del conocimiento que es, según Liou [43], *el proceso de extraer, estructurar y organizar el conocimiento desde las fuentes de experticidad de forma que sea comprensible para una máquina*. La KB es la parte más

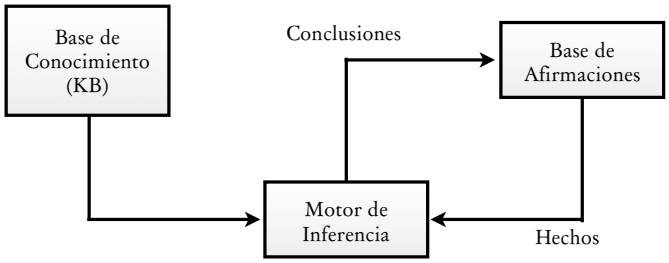

Figura 3.1: Esquema clásico de un Sistema Experto.

importante del Sistema Experto, puesto que es la base sobre la que funciona todo el razonamiento; es por eso que la fase de adquisición es primordial para el buen funcionamiento de un Sistema Experto.

La **Base de Afirmaciones** contiene los hechos específicos que conocemos de antemano acerca problema a analizar, así como las conclusiones a las que ha llegado el **Motor de Inferencia**. Este último utiliza tanto las reglas de la KB como los hechos almacenados en la Base de Afirmaciones para obtener nuevas conclusiones del problema [30].

Por ejemplo, un sistema experto encargado de diagnosticar enfermedades tomaría los síntomas del paciente como la Base de Afirmaciones; en la Base del Conocimiento se almacenarían las enfermedades y sus síntomas relacionados; por ultimo, el Motor de Inferencia sacaría conclusiones, e incluso nuevos hechos, mediante la evaluación de su conocimiento y de los hechos. Es decir, tal y como haría un médico humano.

3.2.3. Taxonomía de Sistemas Expertos

Podemos establecer una taxonomía de tipos de Sistemas Expertos según su funcionamiento [30]:

- **Sistemas Basados en Reglas**

  En los Sistemas Basados en Reglas, y como su nombre indica, la KB contiene el conocimiento específico en forma de reglas lógicas. En este sistema distinguimos dos métodos de inferencia: encadenamiento hacia atrás y encadenamiento hacia delante. El primero de estos métodos trata de encontrar unas conclusiones dada una meta determinada. Es

decir, se parte de una hipótesis a probar.

Por otro lado, el encadenamiento hacia delante no tiene ninguna meta que probar, simplemente trata de socavar toda la información posible a partir de los hechos de un problema. A partir de las conclusiones que obtiene se generan nuevos hechos de los que seguir obteniendo información, de forma que la Base de Afirmaciones acaba conteniendo tanto los hechos principales como todas las conclusiones que han sido inferidas por el motor de inferencia.

- **Sistemas Basados en Casos (CBR)**

  Esta técnica utiliza soluciones pasadas, almacenadas en la KB, para generar una solución al problema [41]. Si el sistema encuentra una caso similar al problema propuesto, entonces ese será la solución para el problema; si no lo encuentra, el problema propuesto dará lugar a un nuevo caso. Esto se hace adaptando el caso conocido que mejor resultado obtenga de la *función de semejanza*; es decir, el caso más similar de la KB y que encaje con el problema. Este nuevo caso, la solución creada, se convertirá en un nuevo caso en nuestro sistema. Por tanto, el sistema puede aprender de nuevas experiencias, y mejorar su eficiencia con el tiempo.

- **Sistemas Basados en Redes Bayesianas**

  Este enfoque [21] aplica conceptos estadísticos provenientes del Teorema de Bayes; utiliza un mecanismo que calcula la probabilidad de un determinado evento con la probabilidad que tenga ese evento *a priori* y las probabilidades condicionadas relacionadas con las situaciones en las que puede ocurrir ese evento. Es decir, dado por ejemplo un evento A (tener fiebre) y un evento B (dolor de cabeza), la probabilidad condicionada de que ocurra A dado B sería la probabilidad de que a alguien le duele la cabeza partiendo de la evidencia conocida de “tener fiebre”. Formalmente sería:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

3.2.4. Incertidumbre en Sistemas Expertos

Una de las principales dificultades en la captura de conocimiento experto se encuentran en el funcionamiento del propio cerebro humano. Éste trabaja de forma global y tiene el conocimiento de una forma compilada; además, la información que utiliza es incompleta, con incertidumbre [50]. Esta incertidumbre puede definirse como la carencia de conocimiento exacto que impide alcanzar una solución perfectamente valida.

Un sistema experto, por su parte, necesita una separación clara entre el conocimiento y la forma de procesarlo (lo que se conoce como *inferencia*). Uno de los mecanismos clásicos de representación del conocimiento es mediante sistemas lógicos. Estos sistemas se basan en la rama de la filosofía que estudia los principios de inferencia.

Estos sistemas tradicionalmente trabajaban con un la lógica clásico, que se caracteriza por incluir principios tradicionales como el principio de contradicción, el principio de explosión o el principio del tercero excluido.

Este último principio de la lógica clásica es el que más problemas causa a la hora de representar el conocimiento de un ser humano, puesto que estipula que un valor solo puede ser *Verdadero* (1) o *Falso* (0), y no ningún otro valor intermedio. Dado que el conocimiento del ser humano puede llegar a ser incompleto, inconsistente e incierto, la lógica clásica no puede afrontar la resolución de problemas basados en el lenguaje natural.

Y es que el problema con el lenguaje humano es que es, por naturaleza, impreciso. Los seres humanos describimos los hechos con términos como “demasiado”, “algunas veces” o “frecuentemente”, el tipo de términos con el que es difícil expresar el conocimiento en forma de reglas de producción.

Por ejemplo, la frase “cuando la sartén esté *muy caliente*, poner la intensidad del fuego *baja*” sería difícilmente representable en un ordenador mediante la Lógica Clásica (o Booleana), porque sería incapaz de representar los distintos umbrales de Temperatura. La lógica clásica trataría de discretizar los valores continuos con límites muy definidos, pero esto daría lugar a situaciones absurdas.

Por ejemplo, en la lógica clásica podríamos definir a las personas altas como aquellas que miden más de 180 cm. Alguien que mida 181 cm sería considerado alto, mientras que otro

individuo de 179 cm no.

Para solucionar todos estos problemas está la Lógica Difusa, o Borrosa, introducida en 1965 por Zadeh [71]. Según Zadeh, *la lógica difusa está formada por un grupo de principios matemáticos de representación basados más en el grado de pertenencia que en la estricta pertenencia de la lógica binaria clásica*.

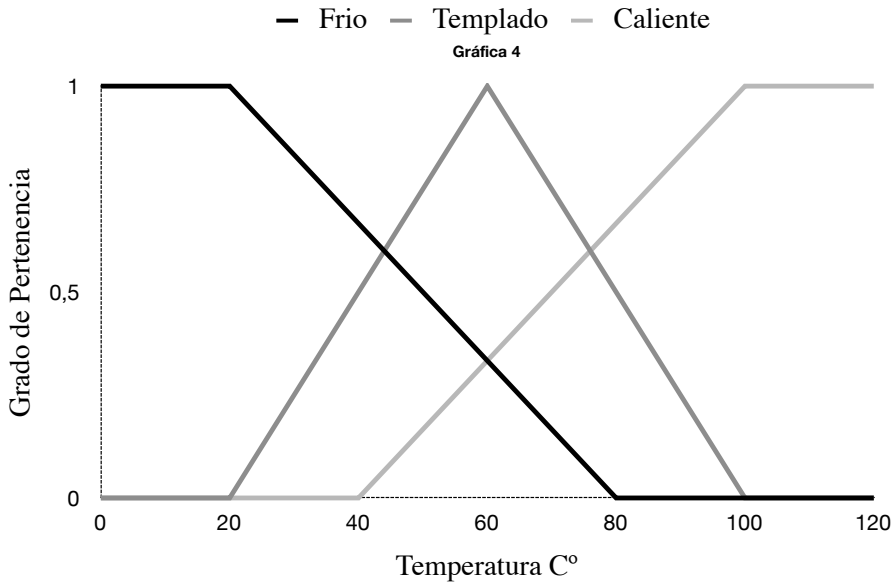

Figura 3.2: Conjuntos difusos del Universo de Discurso Temperatura

Por tanto, en la Lógica Difusa representamos los términos ambiguos con valores entre 0 y 1 que indican el grado de pertenencia a un conjunto difuso [30], permitiendo así que algo sea *completamente verdadero*, *completamente falso* o cualquier cosa intermedia. Por ejemplo, teniendo los conjuntos difusos *Frío*, *Templado* y *Caliente* de la Figura 3.2, todos pertenecientes al Universo de Discurso de Temperatura, una temperatura de 60°pertenecerá a *Frio* y *Caliente* con un grado de 0,25, a la vez que es miembro de *Templado* con un grado de 1. Este proceso es llamado *Borrosificación*, y las funciones se conocen como Funciones de Pertenencia:

$$\mu_A = X \rightarrow [0, 1]$$

Siendo *A* un Conjunto Difuso y *X* un valor del que queremos obtener su grado de pertenencia en el conjunto *A*.

Junto a los subconjuntos difusos tenemos las reglas difusas, que son expresada con la estructura IF-THEN (SI-ENTONCES) [62] y se basan en los sistemas de reglas de producción clásicos. La forma más compacta de representar estas reglas es:

$$IF\ (A)\ THEN\ (B)$$

Siendo *A* el antecedente y *B* el consecuente, ambos en forma de proposiciones difusas atómicas o compuestas. Cada antecedente, o consecuente, está formados por un objeto (un objeto lingüístico) y un valor. Además, una regla puede tener varios antecedentes conectados entre si por operadores lógicos. Por ejemplo:

$$IF\ el\ coche\ no\ arranca$$

$$AND\ el\ tanque\ de\ gasolina\ esta\ vacio$$

$$THEN\ la\ accion\ es\ arrancar\ el\ coche$$

La única diferencia entre estas reglas binarias y las reglas difusas es que las primeras usan términos binarios mientras que las segundas se sirven de la lógica difusa. Por ejemplo, en la lógica clásica podemos representar una regla así:

$$IF\ la\ velocidad\ es\ >\ 100$$

$$THEN\ la\ distancia\ de\ frenado\ es\ larga$$

Mientras que en la lógica difusa tomaría esta forma:

$$IF\ la\ velocidad\ es\ alta$$

$$THEN\ la\ distancia\ de\ frenado\ es\ larga$$

La diferencia es que la lógica clásica utiliza un término binario para la velocidad, que puede ser Alta o Baja dependiendo del límite. Mientras, la lógica difusa define un universo de discurso entre 0 y 220km/h que incluye diversos conjuntos lógicos como Muy Baja, Baja, Media, Alta o Muy Alta. Lo mismo ocurriría con la distancia de frenado en un rango de, por ejemplo, 0 y 300 metros. Los sistemas de reglas de producción se verán con más detalle en la siguiente sección.

La evaluación de las reglas difusas dependerá del operador de lógica difusa que usemos. Los operadores introducidos por Zadeh son [22]:

$$a \wedge b = min\{a, b\}$$

$$a \vee b = max\{a, b\}$$

Un Sistema Experto Difuso trabajará con varias reglas difusas, por lo que cuando obtenemos un antecedente se disparan todas las reglas en las que este está implicado. Hay dos métodos de obtener un único consecuente a partir de varias reglas: o bien se genera el consecuente de cada regla por separado y luego se combinan, o bien se combinan las reglas para generar directamente un único consecuente. Existen, por tanto, diferentes sistemas de razonamiento para obtener el consecuente. Uno de los más utilizados es el Método Directo de Mamdani [45], que es el que se ha implementado en Gaudii.

El proceso de inferencia difusa del Método Directo de Mamdani se realiza en cuatro pasos: borrosificación de las variables de entrada, evaluación de las reglas, agregación de los consecuentes y, por último, la deborrosificación [50]. El proceso entero puede verse en la Figura 3.3.

- **Paso 1. Borrosificación**

  En esta etapa se determina el grado de pertenencia de cada entrada en los correspondientes conjuntos difusos. La entrada será un valor numérico limitado por los rangos de los universos de discurso.

- **Paso 2. Evaluación de las Reglas**

  El segundo paso consiste en tomar las entradas ya borrosificadas y aplicarlas a los antecedentes de las correspondientes reglas. Para esta evaluación utilizamos los operadores de Zadeh: unión de conjuntos para el operador OR (Fórmula 3.1), e insersección para el operador AND (Fórmula 3.2).

$$\mu_{\mathrm{A}\cup\mathrm{B}} = \max\left[\mu_A(x), \mu_B(x)\right] \tag{3.1}$$

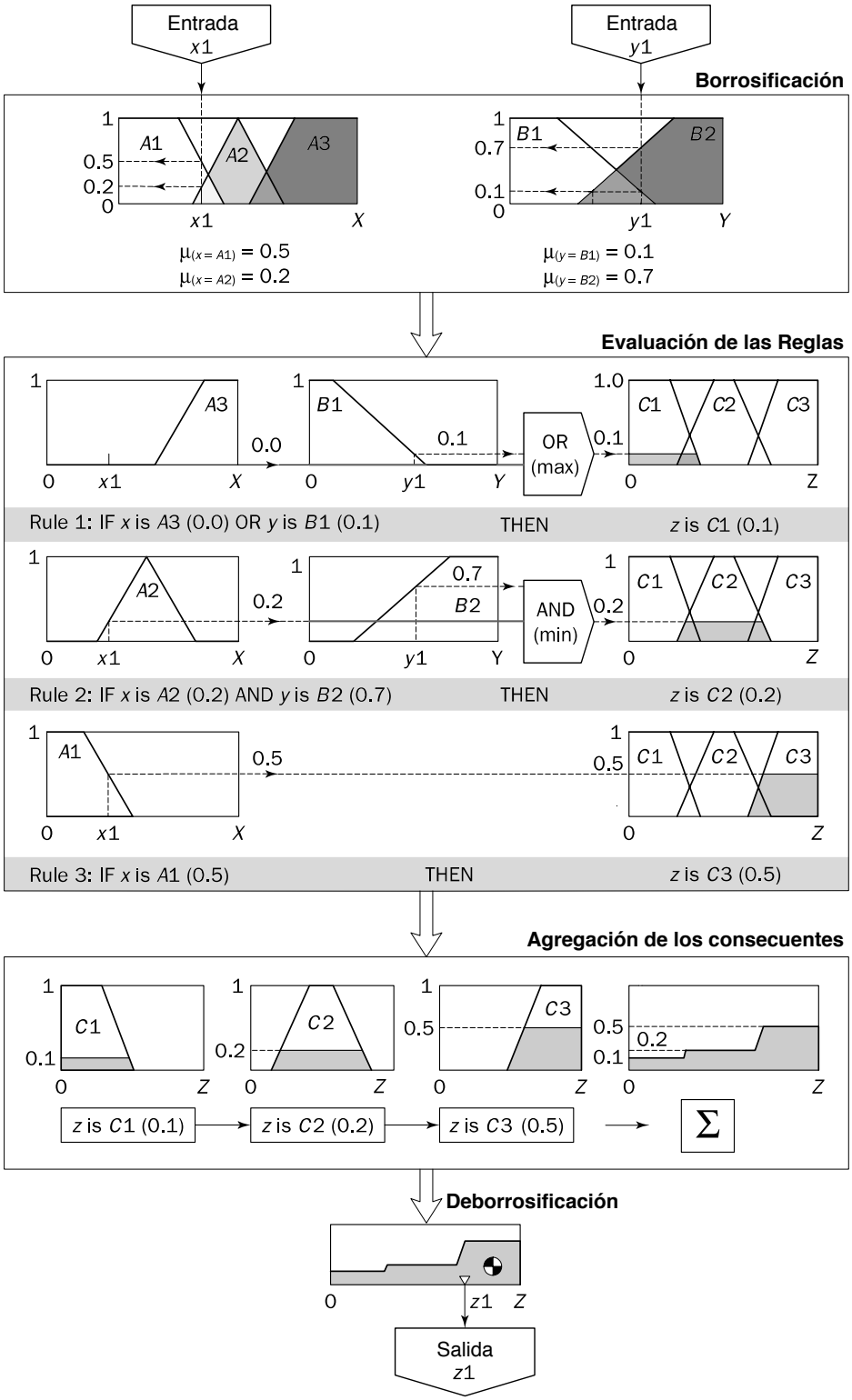

Figura 3.3: La estructura de la inferencia Mamdani [50]

$$\mu_{\mathrm{A}\cap\mathrm{B}} = \min\left[\mu_A(x), \mu_B(x)\right] \tag{3.2}$$

- **Paso 3. Agregación de los consecuentes**

  La agregación es el proceso de unificar todas las salidas de las reglas. Es decir, se obtienen todas las funciones de pertenencia de todas las reglas y se combinan en un único conjunto difuso.

- **Paso 4. Deborrosificación**

  A estas alturas del proceso ya se ha obtenido una salida en forma de conjunto difuso. En el último paso transformamos la salida a un único número.

  Existen diversos métodos para la deborrosificación. En Mandami se utiliza la técnica del centroide. Esta encuentra el punto exacto en el que una linea vertical dividiría el conjunto difuso en dos zonas idénticas (Fórmula 3.3).

$$Centroide = \frac{\sum_{x=a}^{b} \mu_A(X) * X}{\sum_{x=a}^{b} \mu_A(X) * X} \tag{3.3}$$

3.2.5. Representación del Conocimiento Experto

Después de haber capturado el conocimiento experto, este debe ser trasladado a la KB[1] del sistema. Esta fase es llamada Representación del Conocimiento, y es uno de los conceptos básicos en la rama de la Inteligencia Artificial.

Según Davis [24], la Representación de Conocimiento puede entenderse como cinco roles distintos: (1) como un sustituto en sí mismo de la entidad que queremos representar, (2) como un set de compromisos ontológicos, (3) como una fragmento de razonamiento inteligente, (4) como una manera de conseguir una computación eficiente y (5) como un medio de expresión para el ser humano.

Existen diferentes métodos para representar el conocimiento; elegir el más adecuado es vital porque influye directamente en el rendimiento. Lo métodos más utilizados son:

Reglas de Producción

[1] Base de Conocimiento (Knowledge Base)

Las Reglas de Producción son el método más sencillo e intuitivo de entre los mostrados. En su sencillez radica gran parte de su potencial, puesto que son muy flexibles a la hora de representar información.

El problema de este método es que solo puede representar aquel conocimiento que pueda ser escrito en forma de reglas IF-THEN, cuya estructura es:

$$SI\ (A)\ ENTONCES\ (B)$$

Siendo *A* un conjunto de antecedentes que representa las condiciones y *B* un conjunto de consecuentes que representan las acciones a llevar a cabo.

Un Sistema Experto cuenta con un gran número de reglas que se disparan cuando aparece un antecedente. El orden de disparar las reglas no es definido por el programador, sino que es el propio motor del SE quien se encarga de hacerlo. Esto, a ojos del diseñador, proporciona una gran alivio en cuanto a la escalabilidad del sistema, puesto que no tiene porqué preocuparse del orden en el que introduce nuevas reglas.

En una plataforma basada en reglas es importante utilizar un Control de Coherencia[32]. Un sistema con reglas inconsistentes pude dar lugar a conclusiones absurdas y/o contradictorias. Para evitar esto se debe implementar un sistema que controle la consistencia, buscando posibles contradicciones tanto en las reglas como en los hechos.

Las Reglas de Producción ofrecen un buen rendimiento, pero no son recomendables para sistemas complejos que traten de representar con total exactitud un sistema del mundo real [53].

Frames (Marcos)

En 1975 Minsky[49] propuso una estructura de datos para representar en un computador las características típicas de un concepto; a esta estructura la llamó Frame, y es definida según Krishnamoorthy [41] como *una unidad de conocimiento descrita por una serie de slots*, o según Karp et al. [39] como *un objeto con hechos asociados entre sí*. Por ejemplo, y siguiendo el ejemplo que propuso Minsky[49], una pelota es una unidad de conocimiento (el frame) y está descrita por una serie de atributos (slots) que la representan: color, tamaño, dureza, peso, etc.

El concepto de Frame es el de tratar con estructuras de datos que representan situacio-

nes prototípicas o, por decirlo de otra manera, estereotipadas. Minsky[48] se basó en muchas ideas del campo de la Psicología que sugieren que la memoria humana se organiza en relaciones de Frames; según esta teoría, cuando nos encontramos con una nueva situación, cogemos otra similar ya vivida y cambiamos los detalles necesarios para adaptarla a la nueva.

Cada concepto tendría una o más instancias en el sistema[30]. Por ejemplo:

Frame → Instancia del Frame

pelota *pelota de fútbol*

Tamaño → 60 cm.

Color → Amarillo

Frame → Instancia del Frame

Botar1 (Método) → Botar1 (Método)

También se pueden asignar procedimientos (métodos) a los Frames para describir acciones típicas del concepto que queremos representar. Por ejemplo, una pelota puede tener el método de *botar*.

Los Frames pueden conectarse entre sí como un grafo o red. Un slot de una determinada instancia puede utilizarse para almacenar información de esa instancia con otras [41]. Por otro lado, los métodos pueden soportar *paso de mensajes*, lo que permite a los Frames comunicarse entre sí.

Los Frames puede combinarse con reglas [16]. En estos sistemas, los frames incluyen slots de reglas que enlazan cada elemento con unas reglas concretas. El set de reglas incluye un algoritmo de correspondencia de patrones capaz de razonar nuevas reglas a partir de todos los Frames del sistema.

Los sistemas basados en marcos permiten mayor flexibilidad y complejidad que aquellos basados en reglas de producción, por la capacidad de describir tanto la información como el comportamiento de las entidades que representan.

Por esto, los Frames son adecuados para representar problemas de simulación, puesto que este tipo de problemas suelen implicar interacción entre objetos. Son además apropiados si el experto describe el problema haciendo referencia a objetos importantes y sus relaciones, particularmente si el estado de un objeto afecta al resto [28].

Redes Neuronales

Las Redes Neuronales no son, estrictamente hablando, un método de representación de conocimiento, sino que es un método muy extendido de aprendizaje máquina. A pesar de esto, pueden modelar conocimiento experto si se dispone de un conjunto adecuado de ejemplos para el entrenamiento. Es por eso que se ha decidido añadirlas a esta sección.

La investigación del campo de las redes neuronales viene incentivada por modelar en los ordenadores el sistema de razonamiento y procesamiento del cerebro humano[33]. El cerebro tiene la capacidad de organizar estructuras de neuronas para ejecutar ciertos cálculos (como el reconocimiento de patrones o el control motor del cuerpo) mucho más rápido que cualquier máquina. Introduciremos el campo de las Redes Neuronales con la definición de Jordan [36]:

> *Una Red Neuronal es, ante todo, un grafo, con patrones representados como valores numéricos y asociados a los nodos del grafo, así como transformaciones entre los patrones que se llevan a cabo mediante sencillos algoritmos de paso de mensajes. Algunos de los nodos se distinguen entre nodos de entrada y nodos de salida, y el grafo puede verse, en su conjunto, como una representación de una función multivariante que enlaza las entradas con las salidas. Se adjuntan valores numéricos (*pesos*) a los enlaces del grafo, parametrizando así la función de entrada/salida y permitiendo que sea ajustada mediante un* algoritmo de aprendizaje.

Estos nodos (también llamados *neuronas*) se organizan en grupos llamados capas, como lo son la capa de Entrada y la capa de Salida. Entre estas, existen además otras capas ocultas. La Figura 3.4 muestra un ejemplo de una Red Neuronal completamente conectada.

Las neuronas combinan los pesos de las entradas teniendo en cuenta el peso de cada enlace. Esta suma se modifica en una función de transferencia que pasa la información solo si supera un determinado umbral. El procesamiento por sí mismo no parece ofrecer nada, pero es cuando se conectan en redes cuando este sistema muestra todo su potencial.

Una Red Neuronal debe ser entrenada antes de ser utilizada [64]. Este entrenamiento se realiza proporcionando datos al sistema para que los asimile con un algoritmo de aprendizaje. Una vez se han conseguido que el sistema aprenda, los pesos de los enlaces obtenidos en

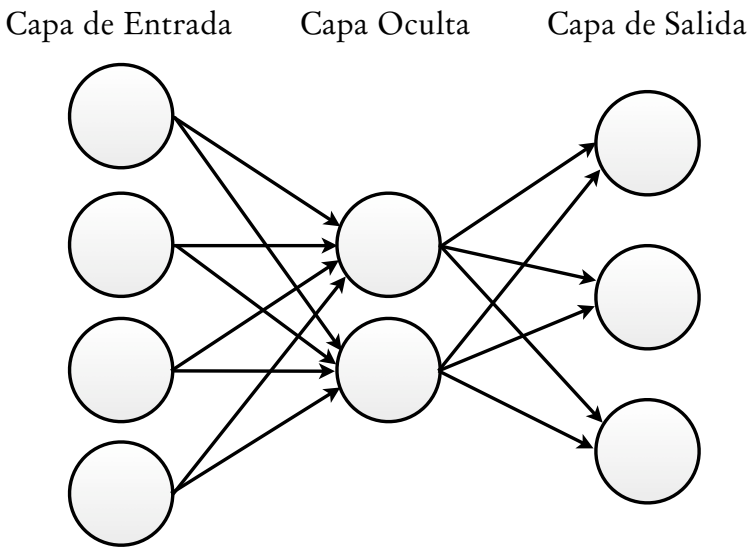

Figura 3.4: Red Neuronal completamente conectada.

el entrenamiento se congelan y el sistema ya puede computar nuevas salidas. Existen tres métodos de entrenamiento para Redes Neuronales [30]: *aprendizaje supervisado*, *aprendizaje no-supervisado* y *aprendizaje por refuerzo*.

En el *aprendizaje supervisado*, posiblemente el método más utilizado, se proporciona a la red unas entradas, y unas salidas como objetivo. Así, la red modifica los pesos de los enlaces para obtener las salidas requeridas.

Al contrario que en el anterior, en el *aprendizaje no-supervisado* no se proporcionan salidas, por lo que los pesos se modifican solo con las salidas. Este tipo de algoritmos suelen implementar una función de agrupamiento para categorizar las entradas en diferentes clases. El *aprendizaje no-supervisado* es muy interesante en problemas donde queremos sacar ciertos patrones a partir de las entradas.

Entre estos dos algoritmos está el *aprendizaje por refuerzo*, donde sí se le proporciona las salidas a la Red Neuronal, pero no las completamente correctas; así, este algoritmo asigna un grado de validez a cada respuesta, dependiendo de cómo de correcta sea esta.

Si el *aprendizaje supervisado* es el método de entrenamiento más extendido, el algoritmo de *Propagación hacia atrás* (o *Retropropagación*) es el algoritmo de aprendizaje más extendido para ese método. La *Propagación hacia atrás* es, según Mehrotra [47], *una generalización*

del algoritmo LMS[2] *que altera los pesos de la red para minimizar el error cuadrático entre las salidas obtenidas y las deseadas*.

La idea básica detrás del algoritmo de *Propagación hacia atrás* es la de hacer grandes cambios en los pesos de los enlaces si esto nos lleva a reducir el error entre la salida deseada y la obtenida, y al contrario cuando la reducción es mínima. Como su nombre indica, este algoritmo comienza alterando el peso de los enlaces de la parte que conecta la capa oculta con la capa de salida y luego con los que conecta las entradas con la capa oculta (ver Figura 3.4).

Las aplicaciones adecuadas para las Redes Neuronales van desde la clasificación hasta la predicción, pasando por el reconocimiento de patrones o algoritmos de agrupamientos [47].

3.2.6. Adquisición del Conocimiento Experto

> *Cuanto más competentes son los expertos en su materia, más complicado les resulta describir el conocimiento que usan para solucionar los problemas.*
>
> *Paradoja del ingeniero del conocimiento*, Waterman [67]

La adquisición del conocimiento no solo es una tarea compleja, puesto que tratamos de capturar todo el conocimiento y razonamiento de un experto para trasladarlo a un sistema computacional, sino que además se trata de un cometido absolutamente crucial del que depende el desempeño final del sistema experto.

Aunque uno de los principales problemas es trasladar con fiabilidad este conocimiento al sistema experto, la mayor problemática de este proceso consiste en tratar con el experto para obtener su conocimiento. El ingeniero del conocimiento puede encontrarse con numerosas dificultades en el transcurso[28]:

- El experto puede ignorar el conocimiento que ha usado para resolver el problema.
- El experto puede ser incapaz de explicar el conocimiento.
- El experto puede proveer conocimiento irrelevante.

[2] Del inglés Least-Mean-Square, el Cuadrado Menos Significativo.

- El experto puede proveer conocimiento incompleto.
- El experto puede proveer conocimiento incorrecto.
- El experto puede proveer conocimiento inconsistente.

Hayes-Roth [31] utilizó el término *Cuello de botella* para describir las dificultad que encontraban los ingenieros del conocimiento en este proceso:

> *La adquisición de conocimiento es un cuello de botella en la construcción de los sistemas expertos. El trabajo del ingeniero del conocimiento es actuar como intermediario para ayudar a construir el sistema experto. Ya que el ingeniero de conocimiento tiene mucho menos conocimiento del dominio de la aplicación que el experto, existen problemas de comunicación que impiden el proceso de transferir la experticidad al programa.*

Además del propio experto, que puede considerarse como la principal fuente de conocimiento en la mayoría de los sistemas expertos, existen otras fuentes de las que obtener conocimiento. Por ejemplo, el usuario final puede ser muy útil, ya que los expertos suelen ver el problema desde un nivel bajo de abstracción, mientras que los usuarios lo hacen desde un nivel más alto. Por supuesto, la literatura también es una importante fuente de documentación.

La adquisición de conocimiento consta de las siguientes fases [28]:

- **Recolección**: Esta fase trata de obtener la información del experto, y es la más problemática de todas. El ingeniero del conocimiento debe tratar de obtener una visión básica del problema en las primeras sesiones, para tratar después de conseguir las partes de información más específicas del dominio de la aplicación.
- **Interpretación**: Después de la fase de recolección, el ingeniero de conocimiento debe interpretar los datos obtenidos e identificar las piezas claves de ese conocimiento. Se debe, con la ayuda del experto, examinar la información y establecer las metas, las restricciones y el alcance del problema.

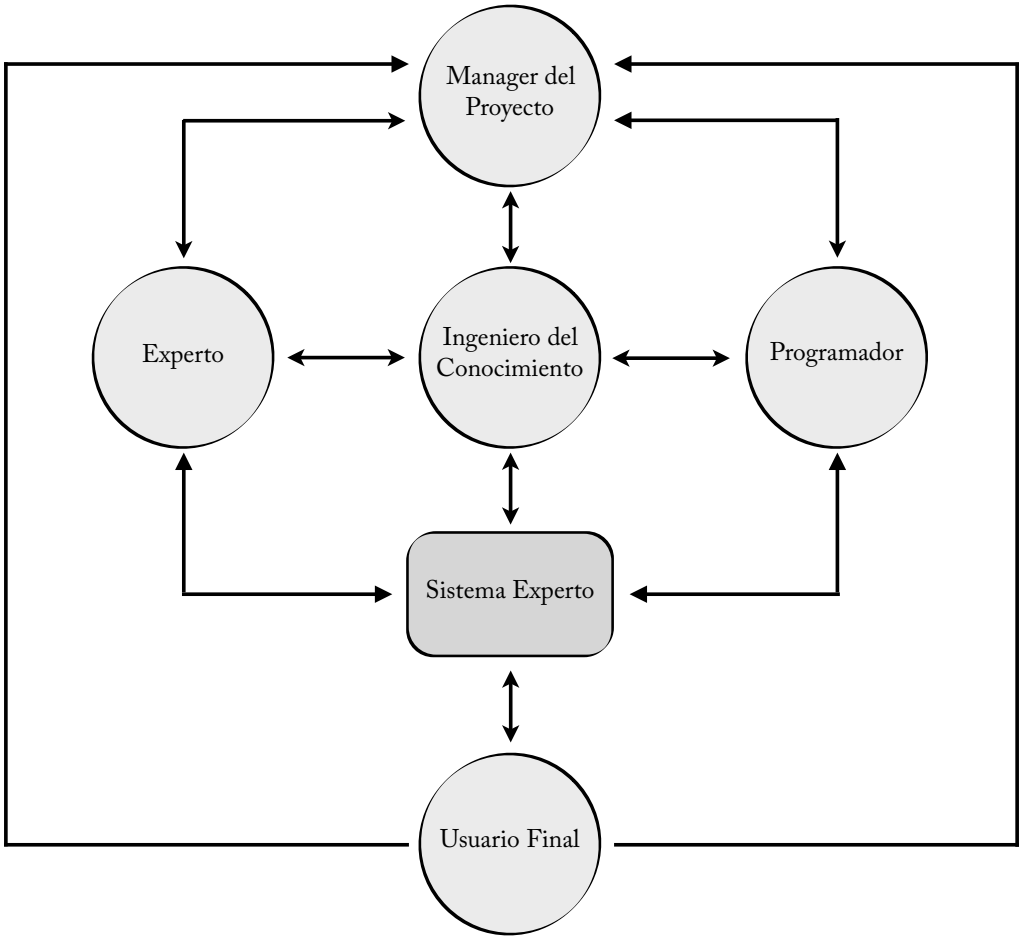

Figura 3.5: Esquema de la adquisición de conocimiento.

- **Análisis**: Posteriormente se utiliza el conocimiento extraido de la fase de interpretación para formar teorías acerca de la organización del conocimiento extraido y de las estrategias para solucionar el problema.

- **Diseño**: Toda la información obtenida hasta el momento debe guiar al ingeniero del conocimiento para desarrollar nuevas estrategias y conceptos que sirvan para conseguir conocimiento adicional.

Además del ingeniero del conocimiento y del experto, existen otros roles que participan en el desarrollo de un sistema experto, más allá de la adquisición del conocimiento [50] (Figura 3.5). El programador será el encargado de la programación del sistema, valga la redundancia,

describiendo el conocimiento adquirido en términos que pueda entender una máquinas; el manager del proyecto cumple el mismo papel que en otros desarrollos software, que es el de asegurar que las metas se vayan cumpliendo e interacturar con el resto de roles; y por último, el usuario final será quien use el sistema y por el que se deben cumplir las metas previstas en el desarrollo. Este usuario será quien determine la aceptación final del proyecto en función de su satisfacción.

3.3. Búsqueda de Soluciones

Los algoritmos de búsqueda de soluciones son una de las tareas tradicionales de la rama de la inteligencia artificial. En estos algoritmos el problema se convierte en un espacio de estados a los que se llega a partir de ciertas acciones, de forma que el estado final al que lleguemos será considerado la solución de nuestro problema. Según Banzhaf [69], *la representación del problema define el espacio de posibles soluciones que el sistema puede encontrar para ese determinado problema*.

Lo que se busca en estos algoritmos de búsqueda es, pues, una secuencia de Estados/Acciones que nos lleve desde el momento inicial del problema a su solución. Por tanto, un problema como un espacio de estados debe definir claramente los siguientes elementos:

- Una **situación inicial** de la que parte el problema.
- Un **objetivo final**, que será el estado considerado como solución.
- Los diferentes **estados** por los que podemos pasar.
- Las **acciones** que se deben llevar a cabo en un estado para pasar a otro diferente.

El clásico ejemplo del 8-Puzzle nos servirá para ilustrar cada uno de esos componentes. En este problema tenemos un tablero de 9 piezas, 8 de ellas numeradas más un hueco vacío. El reto consiste en mover el espacio vacío al numero colindante y poner este en el anterior hueco vació hasta ordenar el puzzle numéricamente, como se ve en la Figura 3.6.

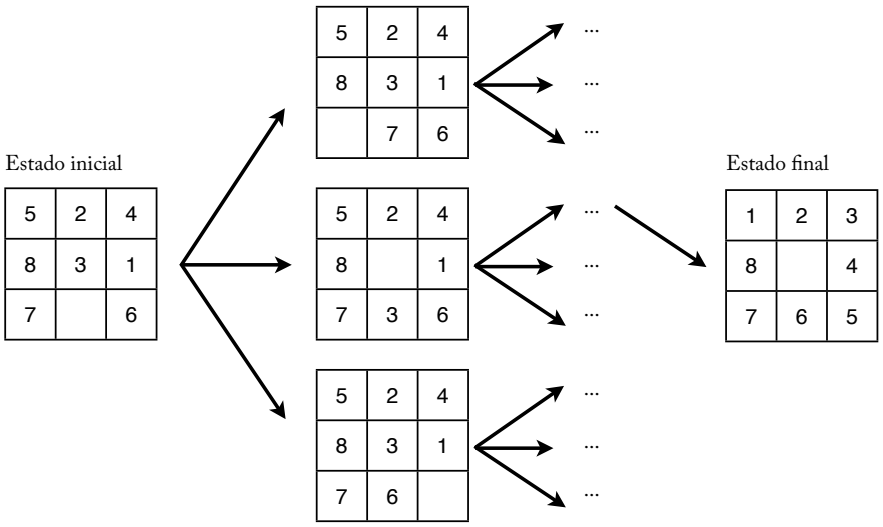

Figura 3.6: Ejemplo de espacio de estados para el problema del 8-Puzzle.

Como se ve, el espacio de estados puede considerarse como un árbol, y la frontera será el conjunto de las hojas del árbol (los estados que no han pasado por ninguna acción todavía). En este ejemplo el papel de cada componente es muy claro:

- El puzzle desordenado será la situación inicial.
- El puzzle ordenado será el estado final.
- El estado estará representado por la posición exacta de cada uno de las 9 piezas.
- Las acciones que marcan el paso de estados serán cuatro, dependiendo del desplazamiento del hueco: arriba, abajo, derecha o izquierda.

Después de representar el problema como espacio de estados se aplica un algoritmo de búsqueda de soluciones para encontrar la solución. Dependiendo del algoritmo, este se puede clasificar en uno de los tres tipos diferentes de métodos de búsqueda [58]:

- **Búsqueda no informada**

  Los algoritmos de este tipo solo utilizan la información de la definición del problema, y su tarea es buscar el estado final entre todo el espacio de estados. Por tanto, estos

algoritmos tratarán siempre de abrir lo máximo el árbol de búsqueda para buscar todos los resultados posibles. Es decir, siempre seleccionará el nodo de profundidad menor en la frontera para seguir expandiéndose.

Este tipo de algoritmos siempre son capaces de encontrar la solución (si es que existe una), pero su gran desventaja es su poca escalabilidad. Es decir, si la complejidad del problema aumenta en excesos, el tiempo necesario para encontrar una solución crecerá enormemente. La complejidad de un problema viene marcada por el número de estados posibles (en algunos problemas el conjunto puede no ser finito) y el número máximo de acciones necesario para alcanzar una solución satisfactoria.

Además del tiempo, la memoria es el otro gran problema de este tipo de algoritmos, puesto que necesitan *recordar* toda la estructura del árbol, con sus estados y acciones.

Por último, este tipo de algoritmos no siempre nos proporciona una solución óptima. Es decir, si el problema cuenta con diferentes soluciones, el algoritmo nos devuelve la primera que encuentra.

- **Búsqueda informada**

  En la búsqueda informada no solo se tiene en cuenta la definición del problema, sino el conocimiento que da el propio estado del problema. Este conocimiento se denomina Heurística, y junto con el coste del estado serán los componentes de la función de evaluación (Fórmula 3.4).

$$f(n) = c(n) + H(n) \tag{3.4}$$

  El coste ($c(n)$) será cuánto ha costado llegar hasta ese estado, mientras que la heurística ($H(n)$) expresará cómo de bueno es ese estado. Si antes se expandían los hijos que tuvieran menor profundidad del árbol, con la búsqueda informada expandiremos los hijos que mejor valor obtengan en la función de evaluación.

  La función de heurística variará con el problema, y en su planteamiento residirá el éxito o el fracaso de este método de búsqueda. Una heurística óptima ayudará a encontrar los mejores valores y de forma más rápida.

Los beneficios de este método respecto al anterior son evidentes. Se gana en eficiencia, puesto que solo buscamos en estados que son prometedores, y especialmente en escalabilidad, ya que no hace falta expandir todo el árbol.

- **Búsqueda local**

  Este tipo de búsquedas es completamente diferente a los anteriores. En los métodos de búsqueda local no se busca una solución al problema ni se parte de un estado inicial, sino que se busca una solución optima al problema y se parte de una solución inicial. Son por tanto algoritmos que buscar optimizar soluciones, donde no se trabaja con espacios de estados sino con espacios de soluciones.

  En este proceso de búsqueda se va guardando la mejor solución encontrada hasta el momento y se busca a partir de ella. Por tanto, este tipo de búsquedas también cuentan con una función de evaluación que valora la bondad

  La búsqueda de soluciones se reparte en dos procesos: intensificación y exploración. El primero es una búsqueda de soluciones cercanas a la solución de la que se parte; el segundo, sin embargo, consiste en buscar por otros estados más lejanos de los que no se conoce todavía su valor.

  Normalmente, un algoritmo de búsqueda comienza con un proceso de exploración, tratando de buscar soluciones prometedores por todo el espacio de soluciones, y acaba con el proceso de intensificación cuando ya tiene una solución muy buena y quiere mejorarla sin cambiarla demasiado.

Ya que no existe una única solución, sino soluciones mejores o peores para cada diseño, en Gaudii se han implementado algoritmos basados en búsquedas locales. Vamos a profundizar con más detalle en varios de estos algoritmos en las siguientes secciones.

3.3.1. Ascensión de Colinas

La Ascensión de Colinas es el método de búsqueda local más simple de los existentes. Este algoritmo consiste en un bucle que va evaluando continuamente soluciones cercanas a la

mejor solución parcial. Es decir, es una búsqueda local que no tiene proceso de exploración, sino solo de intensificación. El algoritmo de esta búsqueda puede verse en el Algoritmo 1.

Algoritmo 1 Ascensión de Colinas

```
1: solucionActual = crearSolucionInicial()
2: repetir
3:     vecino = buscarMejorVecino(solucionActual)
4:     si valor(vecino) <= valor(solucionActual) entonces
5:         devolver solucionActual
6:     fin si
7:     solucionActual = vecino
8: fin repetir
```

Este algoritmo no mantiene un árbol de búsqueda, por lo que no se sabe nada del problema que esté más allá de los vecinos más cercanos. Es por eso que también se le conoce como *búsqueda local avara*, puesto que solo coge el mejor vecino sin pensar donde ir después. Esto puede ser un problema en determinadas zonas del espacio de soluciones (Figura 3.7), donde el algoritmo puede quedarse atascado. Estas son las zonas peligrosas para el algoritmo:

- **Máximo local**, un estado mejor que los vecinos colindantes pero peor que otros más alejados.
- **Meseta**, una zona donde los vecinos tienen el mismo valor que la solución actual.
- **Risco**, un tipo especial de máximo local que es imposible de sobrepasar con un movimiento simple.

Pese a que existen diversas formas de salvar esas zonas, no existe garantía de que lo vaya a hacer, por lo que el gran problema de este algoritmo es que depende en exceso de la forma que tenga el espacio de soluciones. A pesar de esto, y de su aparente simpleza, este método de búsqueda ofrece unos resultados más que aceptables, al menos para encontrar buenos máximos locales.

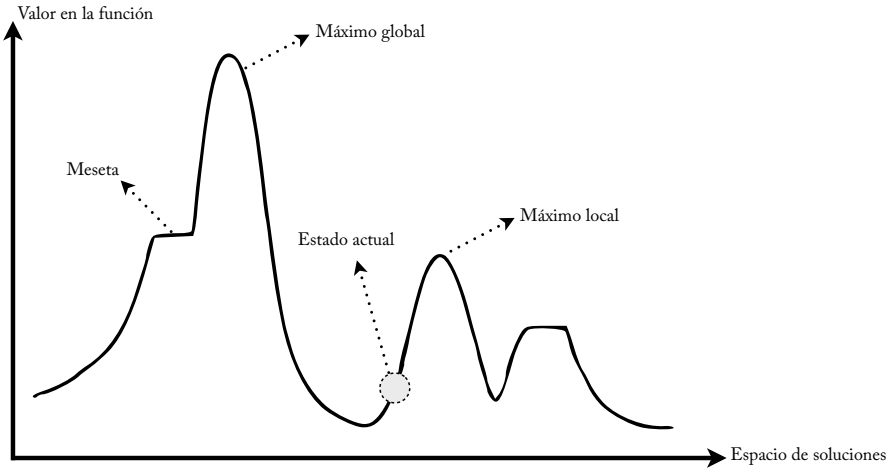

Figura 3.7: Ejemplo de espacio de soluciones.

3.3.2. Haz Local

El algoritmo de Haz Local (*Local Beam*) trata de solucionar el problema de amnesia del algoritmo de Ascensión de Colinas. Es decir, en lugar de guardar un único estado, este método almacena *k* estados generados aleatoriamente al inicio del algoritmo. A cada paso, se generan otros *k* sucesores para cada solución inicial. Si alguno de estos sucesores cumple los requisitos para ser considerado solución, el algoritmo se detiene. En caso contrario, se seleccionan las mejores *k* soluciones de toda la lista completa y se continua con el bucle.

A primera vista puede parecer que este algoritmo es una versión especial de la Ascensión de Colinas en el que se mantienen *k* soluciones en paralelo. Esto no es así, puesto que en cada iteración del bucle se seleccionan *k* soluciones de todas la lista completa. Es decir, si una solución ha generados dos soluciones muy buenas, estas dos aparecerán en la lista de la siguiente iteración; al contrario, si una solución no ha generado buenas soluciones, esta y sus sucesores se eliminan de la búsqueda.

Lo bueno de este algoritmo es que las zonas menos prometedoras se abandonan rápidamente, pero el problema es que al final las *k* soluciones tienden a agruparse en la misma zona, normalmente un máximo local, por lo que acaba careciendo de la suficiente diversidad.

3.3.3. Enfriamiento Simulado

Hasta ahora, los algoritmos estudiados adolecen de un problema claramente identificable: la falta de diversidad. Es decir, ambos acaban estancados en una zona de máximos locales de manera que les impide alcanzar el máximo local.

El Enfriamiento Simulado (Algoritmo 2) se basa en la idea de permitir ciertos movimientos malos que permita al algoritmo escapar de los máximos locales y acercarse al máximo global.

Algoritmo 2 Enfriamiento Simulado

```
1:  T = T0
2:  Sactual = GenerarSolucionActual
3:  Smejor = Sactual
4:  mientras T>=Tf hacer
5:      para i=1 hasta L(T) hacer
6:          Scandidata = seleccionaSolucionVecina(Sactual, T)
7:          δ = coste(Scandidata) - coste(Sactual)
8:          si U(0,1)<e^(-δ/T) ∨ δ<0 entonces
9:              Sactual = Scandidata
10:         fin si
11:         si coste(Sactual)<coste(Smejor) entonces
12:             Smejor = Sactual
13:         fin si
14:     fin para
15:     T = α(T)
16: fin mientras
17: devolver Smejor
```

Estos errores intencionados son frecuentes al principio, como parte del proceso de exploración, pero esta frecuencia disminuye conforme avanza el algoritmo en su proceso. Para implementar esta característica se utilizan dos bucles anidados. El bucle de fuera marca un temperatura que va disminuyendo, y el bucle interior va dando saltos por el grupo de solucio-

nes.

El bucle interior se asemeja al algoritmo de Ascension de Colinas, pero en lugar de obtener el mejor resultado, la selección es totalmente aleatoria y la búsqueda se realiza en saltos más grandes. Estos saltos vienen marcados por la temperatura: cuanto más alta, mayor distancia de exploración; al tiempo que disminuye, la búsqueda será más local.

El bucle interior se repite un determinado número de veces, marcado por la función *L(T)*, que también puede, o no, depender de la temperatura (es decir, se puede implementar para que busque menos veces al final del algoritmo). En cada iteración realiza un salto aleatorio y comprueba si es mejor que la solución actual.

El resto es similar al resto de algoritmos de búsqueda local: se mantiene una variable con la mejor solución encontrada y se busca a partir de ella. La mayor diferencia, y su característica más representativa, es que fuerza al proceso a cometer fallos mediante búsquedas aleatorias con el propósito de acercarse a algún máximo global.

El fin de búsqueda del algoritmo viene marcado por su temperatura final (*Tf*), y los resultados dependen en gran medida de los valores de temperatura inicial y final. Si se le da el tiempo suficiente, el algoritmo de enfriamiento simulado es capaz de encontrar la mejor solución para un determinado problema.

3.3.4. Algoritmos genéticos

Según los biólogos, el mundo que observamos hoy en día es el resultado de miles de años de selección natural, donde solo aquellos seres que se revelan como los más aptos para su entorno consiguen sobrevivir a lo largo de generaciones, mientras los menos aptos se quedan por el camino evolutivo. Podría decirse, pues, que la generación actual de seres vivos que poblan el planeta es el resultado más óptimo al problema de adaptarse en el planeta Tierra.

El último de los algoritmos de búsqueda local que explicaremos está basados en algoritmos genéticos. Este tipo de algoritmos es, según Marcsyk[46], una técnica de programación utilizada para resolver problemas complejos y basada en la reproducción biológica. Tiene el mismo funcionamiento que los anteriores: un conjunto de soluciones potenciales de ese problema y una función de aptitud que permite evaluar cuantitativamente a cada candidata.

Esta función de aptitud evalúa a cada candidata y realiza una de estas dos acciones: desecharla si está entre los peores resultados o seleccionarla si es de las más prometedoras. Se conservan estas soluciones prometedoras y se les permite reproducirse, introduciendo en el camino pequeña alteraciones con una probabilidad muy baja. Estos resultados vuelven a evaluarse y se repite el ciclo hasta encontrar la generación de soluciones óptimas al problema.

Según Poli[54], el funcionamiento de un algoritmo genético sigue la estructura mostrada en el Algoritmo 3. A partir de un conjunto de soluciones, repite un bucle de búsqueda, selección y mejora hasta encontrar una solución posible o darse una condición de parada (por ejemplo, haber alcanzado una determinada población de individuos).

Algoritmo 3 Algoritmo genético

1: Crear una población inicial de soluciones de forma aleatoria
2: **repetir**
3: *Evaluar* cada solución con la función de aptitud.
4: *Seleccionar* una o dos soluciones de la población según su aptitud.
5: *Crear* nuevas soluciones usando operadores genéticos.
6: **hasta que** se encuentra una solución aceptable o se cumple alguna condición de parada.
7: **devolver** Mejor solución encontrada

La parte más importante es la de crear nuevas soluciones mediante operadores genéticos (Algoritmo 3, Línea 5). Para comprender como funcionan es imprescindible saber cómo vamos a representar la información en nuestro sistema, que es al fin y al cabo la mayor diferencia de este método respecto a los visto anteriormente.

Cada individuo de la población (es decir, cada posible solución del problema) será un *cromosoma* formado por una cadena de *genes*. Cada uno de estos genes controla la herencia de una característica determinada. Los genes no tienen porqué ser bits binarios, sino que se puede utilizar cualquier número entero si con eso facilita la representación del problema. Este tipo de representación se utiliza para facilitar la reproducción entre soluciones.

Estos algoritmos trabajan con tres funciones claramente definidas: generación, evaluación y cruzamiento. Las dos primeras son similares a las vistas en anteriores algoritmos, mientras que la de cruzamiento es el principal atractivo de este método.

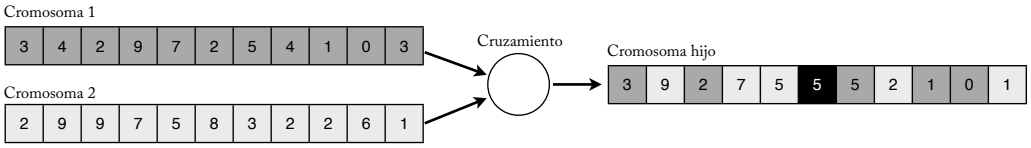

Figura 3.8: Ejemplo de cruzamiento entre cromosoma.

La función de cruzamiento representa la reproducción entre cromosomas (Figura 3.8). En esta se determina gen a gen si el valor se obtendrá de la *madre* o del *padre*.

Además, existe una posibilidad muy baja de que haya mutación. Es decir, que uno de los bits no pertenezca ni a la madre ni al padre, sino que se genere aleatoriamente. Esta mutación representa la propia mutación biológica, y por lo general acarrea resultados más dañinos que beneficiosos al resultado. A pesar de ello tiene una función muy importante, que es la de evitar que la búsqueda acabe estancándose en máximos locales, un problema que, como ya se ha visto, se da frecuentemente en otros algoritmos de búsqueda de soluciones [50].

El principal problema de este algoritmo es la dificultad de representar ciertos problemas, pero de conseguirse puede aportar resultados óptimos e incluso inesperados. Por ejemplo, en 1999 una persona patentó el diseño de una antena que en realidad no había sido diseñada por él, sino por un algoritmo genético que había ideado e implementado [42]. El resultado final fue extraño a simple vista, casi ilógico, pero debido al criterio de selección desarrollado fue efectivamente la solución más optima.

3.4. Diseño Gráfico

La disciplina del diseño gráfico es una variante artística y, como tal, es difícil de modelar puesto que entra en juego la creatividad e inspiración de un artista, conceptos que están lejos de ser una ciencia exacta.

A pesar de ello, en todas las disciplinas artísticas existen ciertas reglas y normas que ayudan en la creación de obras. Por ejemplo, en la fotografía existen numerosas reglas de composición para ordenar los elementos. De la misma manera, en el mundo de la música existen proyectos, como el desarrollado por Bruce [35] utilizando algoritmos genéticos, que ayuda

a los usuarios en composiciones musicales y que de paso demuestra que puede modelarse el conocimiento de un área artística.

Al igual que en estas disciplinas, en el diseño gráfico existen numerosas reglas que pueden considerarse universales para la creación de obras gráfica, desde donde colocar los elementos hasta qué colores usar. En este capítulo se explicarán con mas detalle estas reglas.

3.4.1. Principios básicos de diseño: Reglas C.R.A.P.

Según Williams [68], existen cuatro principios básicos que se deben seguir para que un diseño gráfico sea visualmente atractivo. Estos cuatro principios son Proximidad, Alineamiento, Repetición y Contraste.

- **Proximidad**

  En los diseños de principiantes es común encontrarnos con toda la información desperdigada por el espacio de diseño, sin ninguna conexión visual entre ellos; como si existiera miedo por dejar espacios en blanco. El principio de Proximidad nos indica que debemos agrupar los elementos que tengan alguna relación entre sí de forma que estos se vean como un conjunto cohesionado de elementos. Por el contrario, la información que no está relacionada no debería estar próxima.

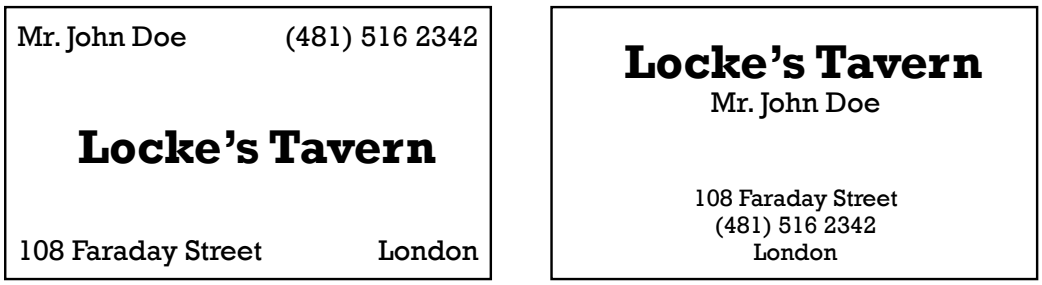

Figura 3.9: Mejora de un diseño utilizando el principio de Proximidad

De esta forma conseguimos que en nuestro diseño existan pocas unidades visuales. Por ejemplo, en la primera tarjeta de visita de la Figura 3.9 existen hasta cinco unidades visuales, y no existe una dirección visual clara que dicte al lector en la linea de lectura a seguir. En la imagen de la derecha, sin embargo, se ha agrupado la información

en dos bloques claros: quién y qué se publicita en la tarjeta en la parte superior, y la información de contacto en la parte inferior; ambas partes están claramente separadas.

El objetivo básico de este principio es el de organizar la información en bloques comunes. Es más fácil que alguien lea, y recuerde, un diseño si la información está correctamente organizada. Además, de esta forma se fomenta el espacio blanco en los diseños, necesarios para que estos puedan *respirar*.

- **Alineamiento**

  Este principio establece que ningún elemento debería estar colocado arbitrariamente en la página, sino que cada uno de ellos debería tener una conexión visual con otro elemento del mismo espacio. Cuando varios elementos están alineados entre sí se crea una conexión entre ellos mediante una linea invisible. Esta conexión visual es más débil que la proximidad, pero también indica que los elementos tienen algo en común.

Figura 3.10: Ejemplo de uso del alineamiento.

Este principio también persigue organizar la información, pero a su vez busca transmitir sensación de unidad en el diseño. Que dos elementos no estén cercanos en el mismo espacio no significa que no puedan tener una conexión.

- **Repetición**

  Se ha visto que la unidad es una parte muy importante en un diseño. Con el principio de Repetición reafirmaremos esa unidad y ganaremos cohesión en el documento. El objetivo es repetir cualquier aspecto del diseño a lo largo de toda la obra. Cualquier

aspecto significa *cualquier* aspecto: peso de las fuentes, imágenes, colores, tamaños, relaciones espaciales, etc.

Esta continua repetición añade consistencia a la obra, pero se debe tener cuidado de no repetir en exceso el elemento repetido o el diseño puede pecar de poco contraste, como se observará en el cuarto principio.

- **Contraste**

  El contraste es una de las principales maneras de añadir interés visual a la obra, amén de ayudar a organizarla jerárquicamente. Este principio establece que si dos cosas son diferentes, entonces deben parecer realmente diferentes. Si son diferentes pero no acaba de parecerlo, entonces tendremos una situación de **conflicto**.

  Cualquier elemento es susceptible de utilizarse para enfatizar el contraste: cambios de fuentes, cambios de colores, cambios de orientación del contenido, etc. Pero el contraste se debe de hacer de forma exagerada, para que no dé lugar a equivocaciones; no debe parecer que ha sido un error. Por ejemplo, si tenemos una zona marrón no podemos hacer contraste con un color verde, o crear contraste entre una fuente *regular* y una *semibold*.

  En la Figura 3.11 se puede observar como mejora un diseño solo con crear contraste entre sus elementos.

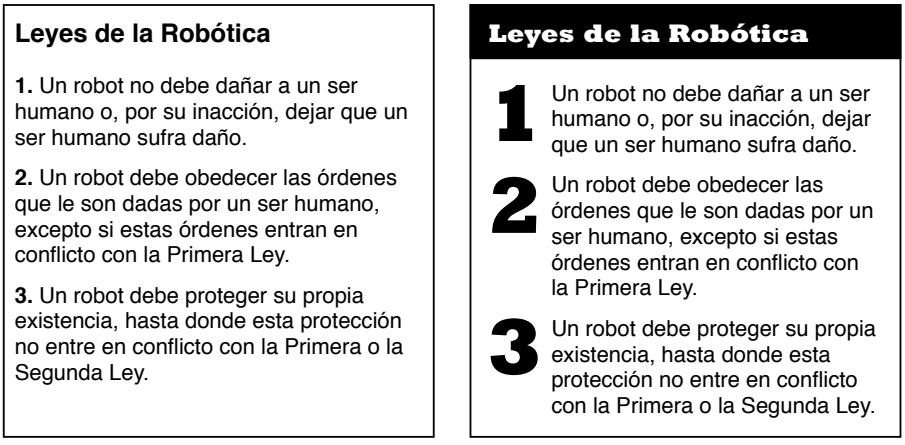

Figura 3.11: Ejemplo de uso del contraste.

3.4.2. Uso de tipografías

La tipografía es un conjunto de técnicas para el manejo, ordenamiento y diseños de tipos o fuentes. El diseñador de tipos, Hermann Zapf, la definió así:

> *La tipografía es arquitectura en dos dimensiones, basada en experiencia e imaginación y guiada por reglas y legibilidad. Y este es el propósito de la tipografía: el ordenamiento de los elementos del diseño dentro de una estructura deberían permitir al lector concentrarse fácilmente en el mensaje sin aminorar su velocidad de lectura.*

En la búsqueda de la tipografía más adecuada para un diseño deben tenerse en cuenta diversos factores. Según Williams [68], pueden existir tres tipos de relaciones entre las fuentes tipográficas que usemos en nuestros diseños:

- Una relación **concordante** se da cuando solo usamos fuentes de la misma familia sin mucha variación en peso, estilo o tamaño. Con este tipo de relaciones es sencillo mantener la armonía y la repetición, pues todos los textos mantienen el mismo tipo de letra. El problema es qu se corre el riesgo de crear un diseño demasiado apagado.

- Las relaciones **conflictivas** ocurren cuando se utilizan distintos tipos de letra que son similares entre sí. Es decir, su estilo es el mismo sin llegar a ser el mismo tipo de letra, creando confusión en el lector. Se debe de evitar este tipo de situaciones.

- Por último, las relaciones de **contraste** se dan cuando se utilizan distintos tipos de letra que son claramente distintos entre sí, logrando un efecto visualmente atractivo gracias al contraste que desprenden.

En la Figura 3.12 se ilustra un ejemplo de estas relaciones. Como se ve, en la relación de conflicto se crea confusión de lector al usar una fuente distinta que no llega a ser claramente diferenciable. El lector puede llegar a preguntarse si no se trata de un error en lugar de algo intencionado. Se debe tratar, por tanto, de o bien mantener una misma identidad visual, o bien crear contraste claramente con los tipos de letra.

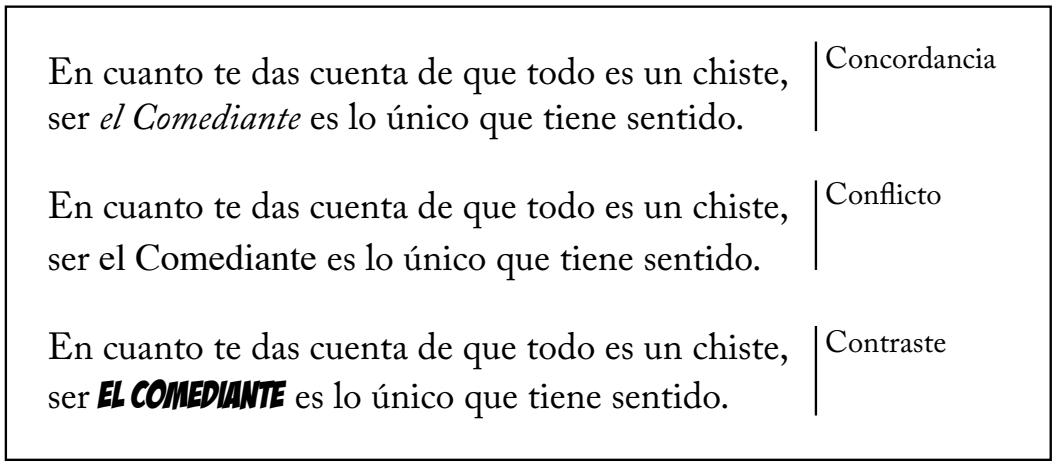

Figura 3.12: Ejemplos de las relaciones entre las tipografías.

Para entender alguna de las formas de crear contraste es imprescindible saber clasificar las fuentes según su tipo. No existe un acuerdo unánime sobre qué grupos son los adecuados; ni siquiera cuantos deben ser. Una taxonomía cercana al consenso común es la que proponen Williams [59] o, en gran parte, Ambrose et al. [17] (Figura 3.13).

- **Old Style** Este tipo de fuentes pertenecen a la familia Serif. Es decir, las letras suelen acabar con un remate, o gracia, sus extremos; esto se hace para facilitar la lectura, ya que guía al ojo a seguir el texto. Las fuentes Old Style están basadas en los textos a mano de los escribanos. Por eso, este tipo de fuentes suele tener una moderada transición de grosor en el trazo, como imitando el trazo de una estilográfica.

  Las fuentes Old Style son muy recomendables para grandes cantidades de texto, ya que la fuente no llama la atención por sí misma y deja que sea el texto el que acapare el protagonismo.

- **Modern**

  Al igual que las Old Style, las fuentes Modern son también Serif, pero ahora el remate es más fino y su orientación es completamente horizontal – contraria a la orientación inclinada en el remate de las fuentes Oldstyle. En esta ocasión no se persigue una imitación del trazo de una estilográfica, sino que el trazo es más recto y su transición es

mucho más radical que en las fuentes Old Style. Vistas desde lejos, estas fuentes parecen estar formadas únicamente por lineas verticales.

- **Slab Serif**

  Nacidas en plena revolución industrial, las Slab Serif se caracterizan por engordar el grosor de la tipografía y por remarcar radicalmente el remate de las letras. Además, estas tienen poca o ninguna transición de grosor en el trazo.

  Este tipo de fuentes son muy legibles y pueden ser utilizadas en textos largos, aunque al hacerlo la página dará una imagen más oscura debido al aumento de grosor con respecto a las fuentes Old Style.

- **Sans Serif**

  La palabra "sans" significa "sin" en francés, así que este tipo de fuentes prescinde por completo del Serif, o remate, al final de los trazos de las letras, lo que se convierte en su rasgo más distintivo. En las fuentes Sans no suele existir transición de grosor; es decir, todo su trazo mantiene el mismo ancho.

- **Script**

  La categoría Scripts abarca todas esas fuentes que simulan la escritura manual, ya sea de forma tradicional, infantil, etc. Este tipo de fuentes deben usarse con moderación o de lo contrario recargarán en demasía el diseño. Por supuesto no se recomienda su uso en textos largos.

- **Graphic**

  Probablemente la clase más fácil de identificar. Son aquellas fuentes de aire divertido y fáciles de usar; existe más de una para cada extravagancia que quieras expresar. Al ser tan personales e inconfundibles, su uso potencial está muy limitado.

Como se ha comentado, en los diseños se debe tratar de crear relaciones **Concordantes** o de **Contraste** entre las familias de fuentes tipográficas. La primera de ellas es fácil, pues se trata únicamente de utilizar el mismo tipo de letra en todo el diseño; para crear relaciones de

Figura 3.13: Los diferentes tipos de fuentes.

contraste que sean visualmente atractivas se debe recurrir a las siguientes características de las fuentes [68]:

- **Tamaño**

  Este tipo de contraste es el más evidente: fuentes grandes y fuentes pequeñas. Para crear contraste se deben utilizar fuentes de tamaño muy distinto; no se crea contraste con una fuente de 12pt y otra de 16pt. Se deben de hacer contrastes muy evidentes o la gente puede pensar que se trata de un error. En el ejemplo de la cabecera del periódico (Figura 3.14) queda claro cual es el elemento importante. El resto de elementos siguen ahí, pero no son en realidad importantes para el público. Si alguien quiere saber el *Volumen* o el *Número* aun puede leerlo, pero no interfiere con lo realmente importante en el diseño de la cabecera.

Figura 3.14: Ejemplo del uso de contraste por tamaño.

- **Peso**

  Cuando hablamos del peso de una fuente lo hacemos del grosor de sus lineas. Un mismo tipo de fuente posee toda una familia de grosores que abarcan desde el más delgado (conocido como *Light*) hasta el más grueso (*Black* o *ExtraBold*), como se ve en la

Figura 3.15. Al igual que ocurría con el tamaño, el contraste debe ser claro y evidente. Por otro lado, el contraste entre pesos es muy útil a la hora de organizar la información en grupos.

Rockwell Light
Rockwell Regular
Rockwell Bold
Rockwell ExtraBold

Figura 3.15: Variedad de grosores para la misma fuente.

- **Estructura**

  Cada uno de los tipos de fuente expuestos anteriormente representa una estructura diferente. Ya que el ojo inexperto – es decir, prácticamente todo el mundo – es incapaz de distinguir entre muchos de ellos, la forma más fácil de crear contraste es utilizar una fuente tipo Serif (Old Style, Slab Serif o Modern) con una Sans Serif. En ocasiones esto no será suficiente, pero se puede recurrir al contraste de tamaño y/o peso para acentuar la diferencia.

3.4.3. Teoría del color

Johann Goethe – el poeta y científico alemán, entre otras muchas cosas – se preguntó una vez qué reglas existían para poder combinar colores armónicamente. Sabía de buena mano, por sus amigos músicos, que las reglas para alcanzar la armonía en la música eran de sobras conocidas; sin embargo, al hablar con sus amigos pintores descubrió descorazonado que no existían unas reglas claras que definieran esa consonancia con los colores. Goethe se embarcó en un estudio sobre los colores que culminaría en su obra *Farbenlehre*[3]. Sus amigos artistas le dijeron que utilizar reglas de armonía era restrictivo y poco creativo; Goethe les dijo que había que conocer las reglas, aunque solo fuera por el privilegio de romperlas.

[3]En alemán, *Teoría de colores*

El color tiene tres atributos intrínsecos: Tono, Saturación y Valor. El primero de ellos hace referencia a la frecuencia del color en el espectro visible. Cuando observamos una manzana roja, en realidad no estamos viendo una manzana *roja*, sino unas determinadas ondas de luz reflejadas en el objeto mientras que otras son absorbidas. El **Tono** se refiere al color que entendemos como tal: rojo, verde, naranja, etc. La **Saturación** de un color determina su intensidad. Un color muy vivo se dice que está saturado, mientras que uno apagado toma el término de desaturado. Por su parte, el **Valor** de un color es su luminosidad u oscuridad intrínseca, aunque esta es relativa; por ejemplo, mientras que entendemos el amarillo como un color luminoso, este podría parecer un color oscuro junto al blanco. En la Figura 3.16 se puede observar el cambi ode un mismo Tono de color dependiendo de su Saturación y Valor.

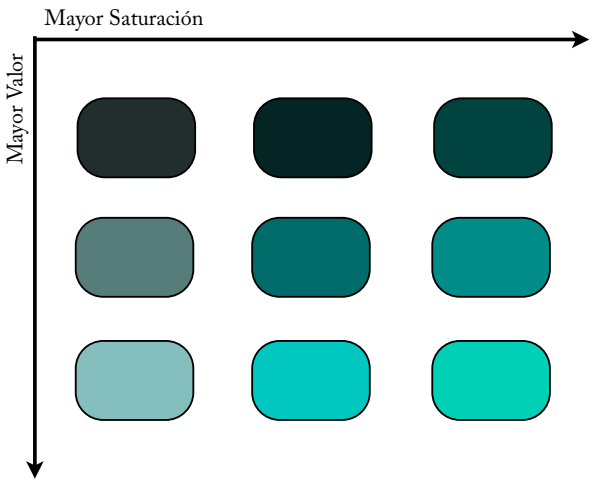

Figura 3.16: Cambios de Valor y Saturación a un mismo Tono.

Al igual que Goethe, o Newton antes que él, muchos han sido los científicos y artistas que han tratado de organizar los colores en modelos visuales. El más común es la rueda de Albert Munsell [59], científico y pintor británico, representada en la Figura 3.17. La rueda muestra los diferentes Tonos con los mismos valores de Saturación y Valor.

A partir de la rueda de Munsell podemos obtener relaciones de colores que formarán nuestros **esquemas de color**. Estos esquemas representen dos o más colores que funcionan bien juntos en una determinada situación. Los esquemas (Figura 3.18) que podemos obtener de la rueda de color de Munsell son los siguientes:

Figura 3.17: Rueda de colores de Munsell

- **Colores análogos**: Son aquellos colores adyacentes en la rueda de colores, como pueden ser el rojo, naranja y naranja-amarillo. En este tipo de esquemas. Suelen combinar bien juntos para crear diseños serenos y agradables al ojo humano. Hay que tener cuidado con el contraste en este tipo de esquemas puesto que los colores suele ser muy similares.

- **Colores complementarios**: Son aquellos colores que se encuentran en posiciones opuestas de la rueda, como el rojo y el verde o el violeta y el amarillo. Son perfectos para destacar elementos, puesto que el contraste entre ambos colores es muy elevado – especialmente si los colores están muy saturados. Por contra, jamás deben utilizarse para representar texto.

- **Triada**: Un esquema de Triada usa tres colores equidistantes de la rueda de Munsell. El resultado suele ser una mezcla intensa, incluso cuando los colores están desaturados.

Existiendo formas de crear esquemas de colores que casen entre sí, el objetivo se centra en elegir un color adecuado a partir del cual desarrollar el esquema. Esta elección es, evidentemente, relativa al contexto del propio diseño. El factor principal para elegir un color u otro es la sensación que queramos transmitir con nuestro diseño: modernidad, pasión, alegría, calma, etc. El Cuadro 3.1 muestra la asociación de colores con las sensaciones que estos transmiten según Heller [34].

Figura 3.18: Esquemas de colores

3.4.4. Regla de los tercios

La composición de elementos en una imagen cuenta con numerosas reglas provenientes, en su mayoría, del mundo fotográfico. Este tipo de reglas no aseguran siempre un buen resultado, pero son un buen punto de partida para la ordenación de elementos. La más conocida de estas reglas es la Regla de los Tercios, que asegura una composición visual fuerte y atractiva siguiendo unas pautas muy sencillas y claras.

Se denominan tercios de una imagen a las cuatro intersecciones resultantes de las rectas que dividen una imagen en tres parte iguales, tanto horizontal como verticalmente. Según la regla de los tercios, la composición será visualmente más atractiva si el elemento principal de la imagen se encuentra en alguno de los tercios.

Esta regla se considera como un medio simple de aproximación a la proporción áurea. Esta proporción, que se representa por la letra griega ϕ (Fi), es el número irracional 1.618. Las proporciones que dan como resultado este número se han utilizado en la arquitectura desde la antigüedad, así como en el diseño de numerosos objetos cotidianos (por ejemplo, en las dimensiones de las tarjetas de crédito), debido a que esta proporción resulta muy armoniosa a los ojos. Además de objetos creados por el hombre, la naturaleza está plagada de ejemplos del número dorado, como también se le conoce.

Como se ve en la Figura 3.19, los tercios son los puntos marcados con un círculo (justo en la zona de la intersecciones). Tal y como está tomada la fotografía, la composición se considera visualmente más fuerte que si el gato estuviera completamente en medio.

Sensacion	Color Base	Color Acentuación
Agrado	Verde, Rosa	Azul, Blanco, Amarillo
Calma	Verde, Azul	Rosa, Blanco
Amistad	Azul, Blanco	Rojo, Naranja, Marrón
Tecnología	Plateado, Gris	Violeta, Magenta
Alegría	Rojo, Amarillo	Verde, Azul, Rosa
Lujo	Dorado, Violeta	Plateado, Negro, Rojo
Diversión	Rojo, Naranja	Rosa, Blanco, Azul
Feminidad	Rosa, Rojo	Azul, Naranja, Amarillo
Maculinidad	Azul, Negro	Marrón, Rojo, Plateado
Esplendor	Dorado, Rojo	Violeta, Gris
Calidez	Rojo, Naranja	Amarillo, Marrón
Tradición	Beige, Marrón	Dorado, Verde

Cuadro 3.1: Asociación de colores según Heller.

De la misma manera, es común utilizar las rectas horizontales para colocar elementos en ese tercio. Por ejemplo, la mirada de una persona o la línea del horizonte. De hecho, esto se conoce comúnmente como la Regla del Horizonte; es especialmente socorrida por fotógrafos para tomas de paisajes, de forma que este se sitúe en uno de los tercios pero nunca en el medio (Figura 3.20).

3.5. Visión por computador

La visión por computador es el conjunto de las tecnologías que permite a las máquinas *ver*. Es decir, la principal meta de esta disciplina es que las máquinas sean capaces de capturar imágenes, procesar la información que puedan tener y tomar decisiones útiles a partir de éstas.

Según Bradski et al. [20], la visión por computador es *la transformación de datos desde una imagen o una videocámara a bien una decisión o a una nueva representación*.

Estos datos de entrada pueden provenir tanto de una fotografía, una secuencia de vídeo o

Figura 3.19: La cara del gato se encuentra justo en uno de los tercios de la imagen.

datos multi-dimensionales de los escáneres médicos. Además, éstos suelen incluir información referente al contexto de su captura; por ejemplo, saber si la entrada es de una cámara de seguridad o de si se trata de un escáner cerebral.

Por su parte, una nueva representación puede consistir en darle color a una imagen, darle valor de umbral o eliminar el movimiento de la cámara en una secuencia de imagen.

La visión por computador aun se encuentra en una etapa primitiva, pero en los últimos años se ha avanzado mucho y sus aplicaciones presentes, y futuras, son numerosas:

- **Visión por computador médica**

  Quizá el campo más beneficiado por la visión por computador es el de la medicina, donde se pueden utilizar técnicas para extraer información de imágenes de escáneres para posteriormente presentar un diagnóstico sobre el paciente. Algunos ejemplos del tipo de información que puede extraerse de una imagen es la detección tumores, la arteriosclerosis, el aumento de de tamaño de los órganos o el flujo de sangre.

- **Robótica**

  La aplicaciones más populares dentro del campo de la robótica consisten en el guiado de robots industriales o la navegación de robots móviles por nuestras casas y oficinas,

Figura 3.20: El horizonte está cerca del la linea horizontal superior, mientras que la silla intenta colocarse cerca de uno de los tercios inferiores.

pero existen otras líneas de investigación como la visión humanoide, que es *el sistema de visión que se centra en la emulación de los seres humanos por parte del robot* [40].

- **Militar**

  La aplicación militar más evidente es la guía de misiles a lugares u objetivos específicos, pero se usa además como reconocimiento de zonas de combate que puedan proporcionar importante información estratégica sobre la situación.

- **Vehículos autónomos**

  Aunque aun están lejos los días en los que los coches viajen solos, los vehículos autónomos tienen hoy diversos campos destacados. La agricultura, por ejemplo, o en la exploración espacial de otros planetas, como ha hecho la NASA en diversas ocasiones, son dos campos en los que la visión por computador puede servir de gran ayuda.

- **Vigilancia**

  Es una de las aplicaciones actuales más prometedoras. Va desde la vigilancia en edificios hasta la detección de explosivos, entre otros trabajos más actuales como la detec-

ción de comportamiento incorrecto del tráfico en las vías urbanas [65].

3.5.1. OpenCV

OpenCV es un conjunto de librerías escritas en C/C++ destinadas a la visión por computador. Es multiplataforma y existe un desarrollo activo para crear interfaces para otros lenguajes como Python, Ruby o Matlab [20]. OpenCV fue diseñado para ser muy eficiente, puesto que está especialmente destinado a aplicaciones en tiempo real.

3.5.1.1. Detección de rostros

La detección facial no debe confundirse con el reconocimiento facial; esta última detecta un rostro y además reconoce de quien es ese rostro, mientras que la detección facial simplemente detecta los rostros en la imagen.

Por tanto, la detección de rostros puede verse como un caso específico de detección de objetos de cualquier tipo. En OpenCV se utiliza un clasificador Haar previamente entrenado para reconocer rostros, pero éste puede entrenarse para el reconocimiento de cualquier objeto.

La técnica utilizada se basa en el trabajo de Viola y Jones [66]; estos clasificadores utiliza el umbral dejado por las diferencias y sumas de una serie de áreas rectangulares como las de la Figura 3.21.

Cada una de estas áreas rectangulares representa alguna zona de contraste entre regiones de la imagen, de forma que un grupo de éstas puede ser usada para representar el contraste mostrado en una cara y sus relaciones espaciales (dos ojos, una nariz, etc.).

Se entrena al clasificador con cientos de imágenes de ejemplos positivos de un determinado objeto (en este caso, de caras), y lo mismo con otras tantas que representen ejemplos negativos. Todos los ejemplos deben ser el mismo tamaño (por ejemplo, de 20x20).

El clasificador busca en una imagen de entrada por regiones de este tamaño, devolviendo 1 si ha encontrado el objeto o 0 en caso contrario, pero además está diseñado de forma que estas regiones son fácilmente redimensionables; así puede buscar el objeto concreto sea cual sea el tamaño, y además es mucho más eficiente que redimensionar la propia imagen.

Por último, se dice que el clasificador es en cascada, puesto que el clasificador en sí mismo

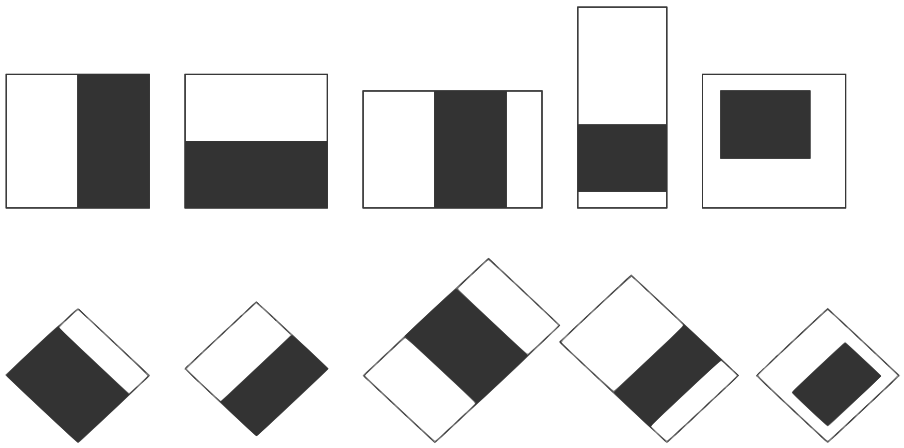

Figura 3.21: Areas rectangulares usadas en los clasificadores. Imagen sacada de la documentación de Intel sobre IPP.

está formado por varios clasificadores simples. Estos se van aplicando sobre la imagen hasta que uno de los clasificadores la rechace o ninguno lo haga.

3.6. Desarrollo web

Un sitio web es un conjunto de páginas web enlazadas entre sí mediante hiperenlaces. Estas páginas web son documentos HTML a los que se puede acceder desde un navegador web.

Existen dos tipos de páginas [15]:

- **Web estática:** Este tipo de páginas son aquellas que devuelven el mismo contenido para todas las peticiones de los usuarios. Es decir, están generados por archivos HTML fijos, por lo que para modificar la web es necesario acceder al archivo en el servidor y modificarlo.
- **Web dinámica:** Como contrapartida, la web dinámica genera contenido distinto a cada usuario dependiendo del contexto o de las condiciones del sitio web. El contenido de estas páginas, por tanto, puede modificarse desde el propio sitio.

Por su parte, una página web dinámica puede construirse combinando estos dos tipos de

lenguajes:

- **Lenguajes del lado del servidor**

  Existe numerosos lenguajes para desarrollar aplicaciones web dinámicas, como pueden ser Ruby, PHP, JSP o ASP. En todos los casos, el cliente envía una petición en uno de estos lenguajes, el servidor entonces interpreta el código y genera un HTML que devuelve al lado del cliente (Figura 3.22).

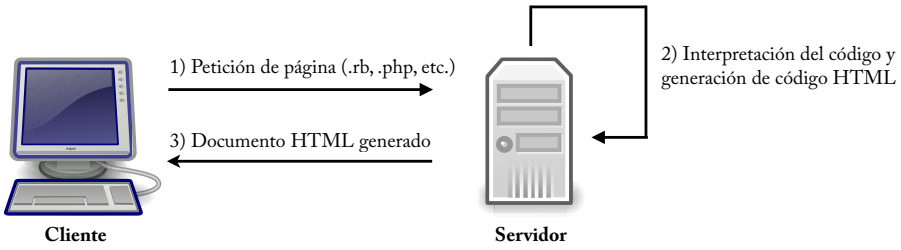

Figura 3.22: Petición de página con un lenguaje del lado del servidor.

  Algunas de las condiciones que pueden modificar el comportamiento de la pagina son formularios web, ciertos parámetros en la URL o el tipo de navegador usado en el momento en el que se accede a la web.

- **Lenguajes del lado del cliente**

  Lenguajes script como JavaScript o ActionScript se utilizan para modificar el comportamiento de la interfaz de la página web cuando se dan ciertos eventos por el teclado o el ratón.

Actualmente, la gran mayoría de los proyectos web que se llevan a cabo se realizan con páginas web dinámicas. Esto es debido a dos causas muy claras. Por un lado, por el movimiento de la Web 2.0 [52], donde se anima a los usuarios a generar su propio contenido y compartirlo con los demás desde la propia web. Por otro lado está el auge de las aplicaciones web, que no son sino páginas web dinámicas que mimetizan el comportamiento de las aplicaciones de escritorio de forma que un usuario podría tener todas las aplicaciones, incluso el propio sistema operativo, dentro del navegador.

Una de las técnicas más utilizadas en el desarrollo de páginas web dinámicas es Ajax, que se detallará a continuación.

3.6.1. Ajax

Ajax es el acrónimo de *Asyncronous JavaScript and Xml* (JavaSript Asíncrono y XML), y es una técnicamente que utiliza el objeto `XMLHHttpRequest` de JavaScript para realizar peticiones asíncronas al servidor. O lo que es lo mismo, peticiones que no requieren refrescar la página por completo [19].

Existen varias diferencias entre el modelo tradicional y el modelo Ajax [56]. En el modelo tradicional, el navegador solicita algún elemento al servidor, como puede ser un documento HTML, y este devuelve el documento completo. Por ejemplo, en un documento HTML que cuente con una cabecera, un pie y un área de contenido, cada nueva carga de página implicará cargar toda la página por completo aunque algunas zonas, como la cabecera y el pie, no hayan cambiado nada.

Sin embargo, en el modelo Ajax la parte del cliente se divide en dos capas lógicas: capa de interfaz y capa Ajax. Esta capa Ajax actúa como intermediaria entre las peticiones del cliente y el servidor. Por ejemplo, en un formulario enviado por el cliente, la capa Ajax maneja esa entrada de datos, interactúa con el servidor y luego actualiza la capa de la interfaz de usuario. Hay que hacer notar que este diagrama no tiene porque darse siempre: puede que la capa de Ajax no tenga que solicitar nada al servidor (por ejemplo, al validar los campos de un formulario se haría todo en el lado del cliente) o que ni siquiera actualice la interfaz (simplemente informe al servidor de las acciones del usuario). Las diferencias entre ambos modelos pueden observarse en la Figura 3.23.

El uso de Ajax ha cambiado la forma de crear y utilizar los sitios web. Gracias a la carga asíncrona y las librerías JavaScript, las aplicaciones webs pueden aspirar a comportarse como aplicaciones de escritorio.

En Gaudii se han utilizado dos de las librerías más populares de JavaScript. La primera de ellas, Prototype [9], es un framework orientado al desarrollo de aplicaciones webs, y su principal objetivo es facilitar al desarrollador la implementación de JavaScript/Ajax. La

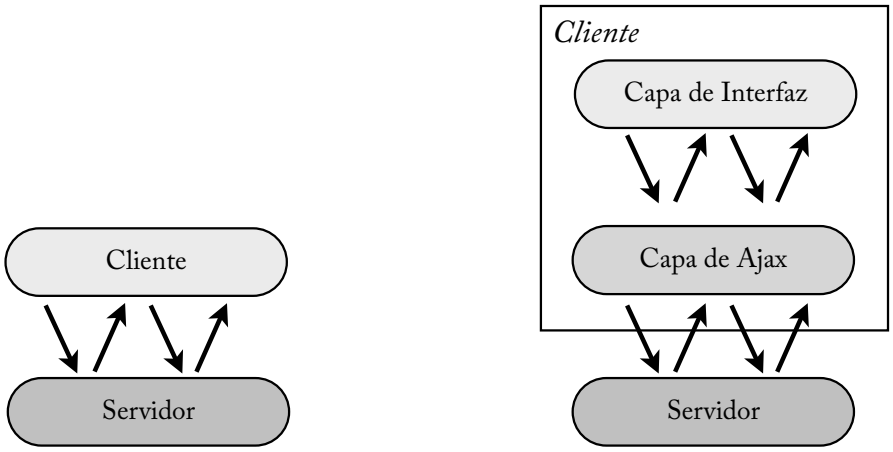

Figura 3.23: Modelo tradicional de peticiones al servidor y modelo Ajax.

segunda librería es script.aculo.us [14], un conjunto de librerías desarrolladas con Prototype para proveer efectos visuales dinámicos y otros controles Ajax (como el *drag&drop*). Ambas librerías están incluidas en el framework de Ruby On Rails.

3.6.2. Ruby On Rails

Ruby On Rails (también conocido como RoR) es un framework de código abierto orientado al desarrollo de aplicaciones web y basado en el lenguaje de programación Ruby. Sigue el modelo de arquitectura MVC (Modelo Vista Controlador) y su filosofía es la de crear aplicaciones web utilizando el mínimo código posible y con el mínimo trabajo en configuraciones.

La primera versión de Ruby On Rails fue lanzada en el año 2000, y desde entonces ha ido creciendo en popularidad. Muchas compañías lo han adoptado por su sencillez y simplicidad, pero especialmente por la velocidad con la que puedes crear prototipos funcionales. Algunas webs conocidas que utilizan este framework son Twitter, Github, Scribd o La Coctelera.

Actualmente, un 71,42 % de los sitios web dinámicos se crean utilizando un framework debido a las facilidades que aporta al desarrollador sin que por ella deba renunciar a la flexibilidad [12].

Como se ha comentado, las aplicaciones de Ruby On Rails se dividen en tres tipos de

componentes [57]:

- **Modelo:** Representa las clases de la aplicación. En RoR, los modelos pueden ser migrados automáticamente a tablas de bases de datos utilizando la clase ActiveRecord.
- **Vista:** En una aplicación de escritorio representaría la interfaz de usuario; en Ruby son normalmente los documentos HTML con código Ruby embebido.
- **Controlador:** Los controladores son los encargados de gestionar la aplicación- Estos reciben datos de entrada, interactúa con el modelo y muestra la vista apropiada al usuario.

La Figura 3.24 muestra una visión general de la arquitectura MVC. Primero (1) el navegador envía una petición, posteriormente (2) el controlador interactúa con el modelo e (3) invoca a la vista con el resultado para que, por último, (4) la vista renderiza la siguiente página.

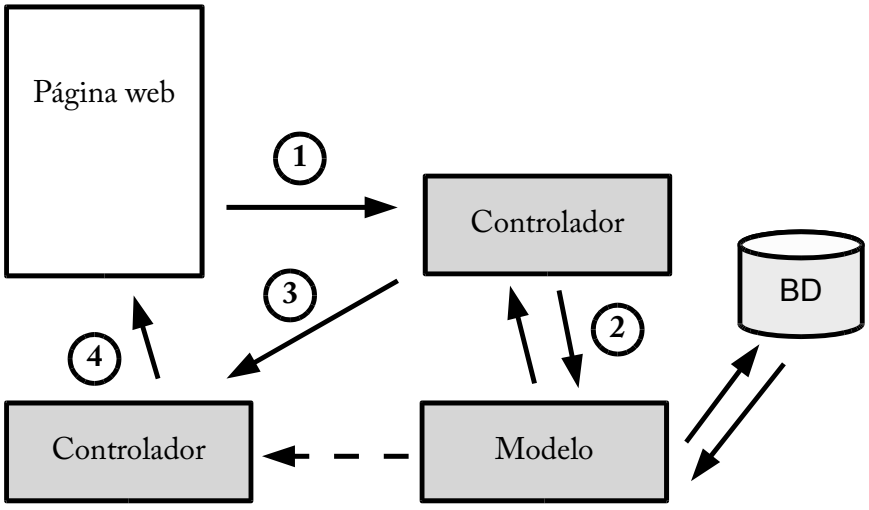

Figura 3.24: La arquitectura Modelo-Vista-Controlador en Rails.

Ruby On Rails fuerza al usuario a utilizar esta arquitectura como parte del paradigma *Convention over Configuration* (Convención sobre Configuración), que busca disminuir el trabajo del desarrollador en configuraciones para ganar en simplicidad.

Ruby On Rails soporta el gestor de paquetes de Ruby, RubyGems, que permite instalar plugins en forma de paquetes (Gemas). Además de las librerías JavaScript mencionadas en

la anterior sección, RoR ofrece soporte para la creación de servicios web con REST y un sistema de *scaffolding* para construir aplicaciones basadas en bases de datos.

Capítulo 4

Metodología de trabajo

La arquitectura del sistema, en el más alto nivel, es la que podemos ver en la Figura 4.1. El sistema se divide en varias etapas independientes entre sí:

- **Etapa de Análisis y Preprocesamiento:** Durante esta etapa se recogen los valores de entrada del diseño, texto e imágenes, así como las preferencias del usuarios con el diseño. Esta imagen da como salida los elementos de diseño crudos, sin ningún tipo de procesamiento todavía, y las preferencias del usuario.

- **Etapa de Producción Genética:** En esta etapa se generan los cromosomas para cada diseño

- **Etapa del Uso del Modelo Experto:** En esta etapa se infieren los resultados de cada cromosoma y se generan los elementos del diseño crudos. Al finalizar esta etapa, los elementos del diseño ya han sido procesados.

- **Etapa de Composición:** Por último, en esta etapa se componen los elementos de diseño ya formados y se devuelve al usuario.

A lo largo de este capítulo explicaremos con más detalle cada una de estas etapas y los módulos que la forman.

Para el desarrollo de GAUDII se ha seguido la metodología de Modelo de Prototipos, donde se ha ido construyendo versiones parciales del sistema que cubrieran una parte de los

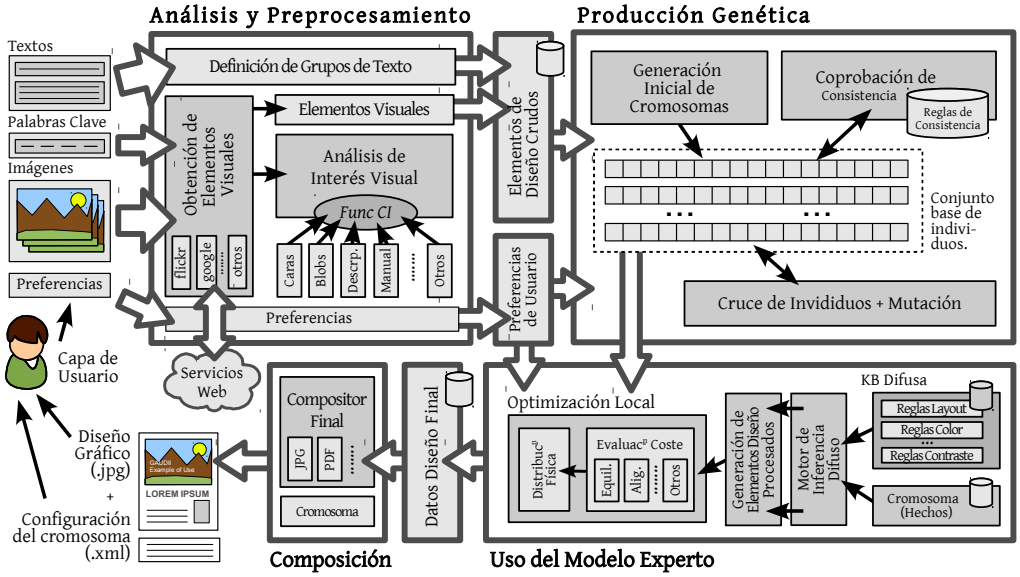

Figura 4.1: Visión general de la arquitectura del sistema.

requisitos para, posteriormente, ir aprendiendo el resto e incorporarlos paulatinamente a la plataforma. Se ha seguido esta metodología por la naturaleza del proyecto, donde muchas veces se ha tenido que probar diferentes vías de acción para un mismo problema; el Modelo de Prototipos permite evolucionar paulatinamente la plataforma con las partes más claras y obtener resultados visibles casi inmediatamente para poder probar el resto de módulos en desarrollo.

Las tecnologías utilizadas en el desarrollo del sistema son las siguientes:

- **Ruby/Ruby On Rails:** Se ha utilizado el lenguaje Ruby [13], y el framework Ruby On Rails [11], para la construcción del sistema y de la interfaz web. Se ha optado por este lenguaje por su sencillez a la hora de crear primeros prototipos funcionales del sistema, lo que nos ha permitido probar diferentes soluciones en ciertos módulos y optar por la más adecuada, su variedad de librerías y la facilidad para interaccionar con software escrito en otros lenguajes. Además, Ruby y Ruby On Rails cuentan con una licencia GPL. Por último, se han utilizado diversas librerías de tecnología asíncrona, como son script.aculo.us [14] y Prototype [9], que permitían dotar a la aplicación con una interactividad similar a la de las aplicaciones de escritorio.

- **C:** Por razones de eficiencia, un modulo del sistema dedicado a un problema de búsqueda se ha desarrollado utilizando este lenguaje.

- **ImageMagick/RMagick:** Para la creación de los diseños y el tratamiento de imágenes se ha utilizado la librería ImageMagick [4] y la interfaz para RMagick [10] para su utilización con Ruby, ambas bajo una licencia compatible la GPL. En unas primeras versiones se utilizaba la librería Processing [8] para estas tareas, pero por razones de eficiencia se optó finalmente por ImageMagick.

- **OpenCV:** Para la implementación de módulos de reconocimiento de imágenes se ha utilizado esta librería de visión por computador [7] que cuenta con una licencia BSD compatible con la GPL. Estos módulos se han escrito en el lenguaje C++. Además, se ha utilizado la librería cvBlob para la detección de objetos en imágenes.

4.1. Etapa de Análisis y Preprocesamiento

Durante esta etapa se recogen los datos de entrada, texto e imágenes, que introduzca el usuario para su diseño, así como sus preferencias. Los objetivos de esta etapa son:

- Almacenar las preferencias que introduzca el usuario referentes a su diseño.

- Obtener los elementos visuales del usuarios.

- Analizar el interés visual de la imagen principal del documento.

- Organizar en grupos los textos introducidos por el usuario para su diseño.

Para acometer algunas de estas tareas se utilizan diversos módulos de soporte. Los módulos utilizados en esta etapa son los siguientes:

- **Módulo de obtención de elementos visuales:** Este módulo se utilizará para obtener la imagen principal del diseño.

- **Modulo de Análisis de Interés Visual:** Este módulo recopila diversos submódulos que procesan la imagen principal en busca de puntos visuales de alto interés.

En las siguientes subsecciones estudiaremos estos módulos con más detalle.

En esta etapa, los datos introducidos por el usuario se almacenan en un único objeto `Design`. Este objeto se crea al inicio del proceso, y actua como *cookie* para guardar todo el proceso creativo. A su vez, se crea un identificador para el usuario que está creando el diseño basado en su IP, fecha y hora del inicio de la creación. Este identificador también se guarda en el objeto `Design`, y lo que suba o cree el usuario (diseños, imágenes, archivos XML, etc.) se almacena en el sistema con este identificador único.

Un diseño de Gaudii tiene al menos una imagen principal y un título. Los textos, incluido el título, están agrupados en grupos de textos. De esta forma, se cumple el principio de proximidad de las reglas C.R.A.P. Es decir, si el cartel anuncia algún evento, los detalles de la fecha y la hora estarán dentro del mismo grupo de texto, pues estos están relacionados entre sí.

Existen cinco tipos de texto: título, subtitulo, texto destacado, texto normal y notas. El orden de colocación de los textos dentro de un mismo grupo será el mismo que seguirá el sistema para ordenarlos, pero no tiene porqué ser así con el orden de los grupos.

Una vez añadido un grupo, el usuario puede subir imágenes y asociarlas al grupo. Por ejemplo, en un cartel que anuncie una conferencia, la foto del ponente puede ser subida al lado del grupo donde aparece su nombre y su cargo. La imagen se guarda como archivo `DesignImage` dentro de un objeto asociado al grupo.

4.1.1. Módulo de Obtención de Elementos Visuales

En este módulo se obtienen los elementos visuales que se utilizarán en el diseño. Para ello recurrirá a servicios web externos utilizando palabras clave introducidas por el usuario. El servicio devolverá imágenes relacionadas con esas palabras clave para que el usuario escoja una.

En la versión final de Gaudii se ha implementado una búsqueda por la página Flickr [3], que recoge fotografías subidas por usuarios amateurs y profesionales. Se ha elegido esta web por dos motivos. El primero es que cuenta con una potente APi abierta a desarrolladores, y el segundo es que permite buscar fotos teniendo en cuenta la licencia de la imagen, con lo que

solo buscamos fotografías con licencias libres.

Para la búsqueda de imágenes en Flickr se ha utilizado el plugin de Ruby *flickr.rb*. Las imágenes se muestran en pantalla ordenadas por Interestingness [5] y teniendo en cuenta su licencia.

Este módulo podría ser ampliable a otros motores de búsqueda de imágenes. Por ejemplo, tanto Google[1] como Yahoo[2] cuenta con una API para su utilización. El problema de estas es que no tienen en cuenta las licencias de las imágenes obtenidas por el buscador.

Además de obtener resultados mediante servicios web, el usuario puede optar por subir sus propias imágenes al sistema.

4.1.2. Módulo de Análisis de Interés Visual

A lo largo de este módulo se ejecutan diversos submódulos que tienen como función encontrar el punto de interés de una imagen. El punto de interés de la imagen será una zona rectangular donde resida toda la importancia de la imagen. Con esta información, Gaudii tratará de, por un lado, sacar partido a esa zona colocando otros elementos de interés cerca de ella y de, por otro lado, evitar colocar cualquier elemento encima de esa zona.

En la versión final de Gaudii se han implementado dos submódulos escritos en C++ que utilizan la librería OpenCV:

- **Detección de rostros**

  El primero de estos módulos busca posibles caras en la imagen ya que, de existir alguna, ésta sería el punto de interés de la imagen. Utiliza un clasificador Haar que proporciona OpenCV, donde ya se ha entrenado al sistema en la detección de rostros. Tras esta búsqueda se devuelve un array de punteros a estructuras donde se almacena la información de los rostros. El sistema escogerá el rostro de mayor tamaño. Se pueden ver ejemplos de este submódulo en la Figura 4.2.

- **Detección de objetos**

  El segundo módulo busca objetos que estén claramente aislados en la imagen. Se utiliza la librería cvBlob, diseñada para buscar objetos en imágenes. Ésta librería está más

Figura 4.2: Ejemplos de detección de rostros.

pensada para la detección de forma muy claras y evidentes, como manchas blancas sobre fondos oscuros, así que primero se somete a la imagen a un método de valor del umbral (*thresholding*). Esto convierte los colores de la imagen a prácticamente blanco y negro, por lo que facilita la tarea de búsqueda.

En la Figura 4.3 se puede observar el proceso con diversos ejemplos. El primero de ellos es el más claro, con un elemento claramente diferenciable en la imagen. En la segunda imagen existen, además de la flor, los claros de las esquinas superiores. Para solucionar esto, de entre todos los blobs encontrados solo se eligen a los objetos que superen un determinado tamaño (entre un 4 % y un 25 % del tamaño total de la imagen), y de entre estos se elige al que esté mas cerca de alguno de los cuatro tercios de la imagen. Por último, el tercer ejemplo muestra como en imágenes complejas el algoritmo es incapaz de encontrar elementos destacados.

Estos submódulos son ejecutados sobre la imagen y cada uno devuelve una región encontrada. Esta región es mostrada al usuario, que puede optar por elegir la zona encontrada por el sistema o definir la suya propia. En Gaudii se da preferencia a la elección manual del usuario, seguida por la detección de rostros y, por ultimo, por la detección de objetos.

Otro módulo cuya implementación se ha estudiado ha sido uno basado en descriptores de imagen. Esta posible aportación se comenta con más profundidad en el Capítulo 6, en la sección dedicada a las líneas de investigación futuras.

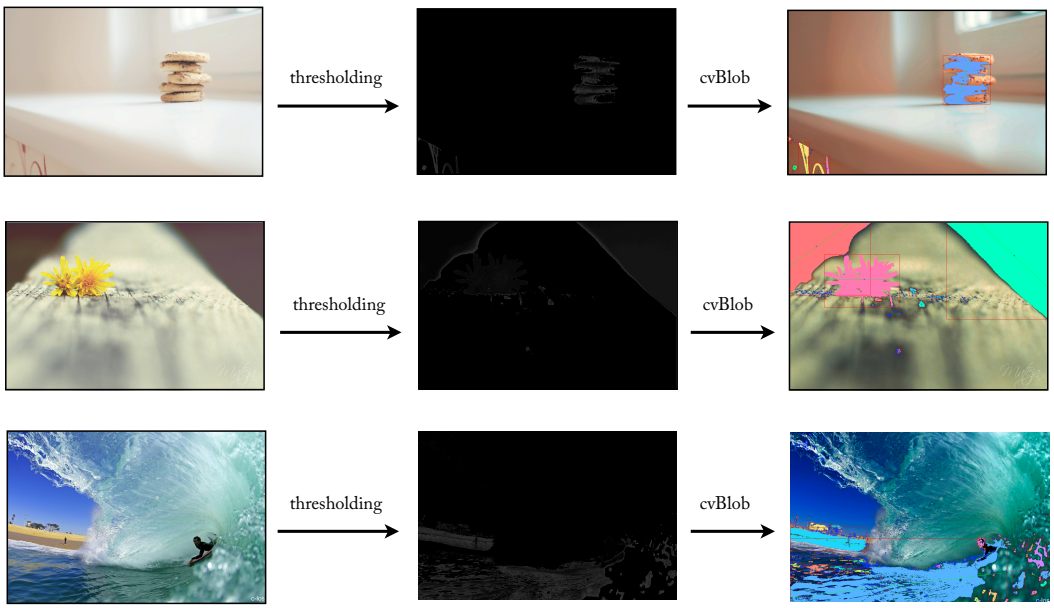

Figura 4.3: Ejemplos de detección de objetos.

4.2. Etapa de Producción Genética

En esta etapa se generan aleatoriamente las cadenas de genes, los cromosomas, que definirán a cada individuo; es decir, a cada diseño. Los objetivos de esta etapa son:

- Crear cromosomas para cada diseño de forma aleatoria para preservar la heterogeneidad.
- Crear unas reglas de consistencia para los cromosomas.
- Crear un sistema de cruzamiento entre cromosomas.

Cada gen del cromosoma representa una característica determinada del diseño. Estos genes están marcados por un número entero entre el 0 y el 9, indicando, en la mayoría de los casos, el nivel con el que se cumplirá esa característica. Estos genes están agrupados en grupos dependiendo de a qué característica general del diseño afecte: generales del documento, características para el contraste, características para la repetición, valores de entrada y características de la composición.

A continuación se explica en detalle cada gen del cromosoma.

Grupo del gen	Descripción del gen
Documento	Este gen marca la proporción del cartel. En sus valores más bajos, la proporción será 1:1 (es decir, un diseño cuadrado), mientras que en sus valores más altos alcanzará proporciones panorámicas.
Documento	Marca si el cartel debe tener un orientación vertical u horizontal.
Documento	Determina el estilo que seguirá a fuente principal. Bien una fuente más gráfica (tipo Script o Graphic) o bien un estilo más serio (el resto: Sans Serif, OldStyle, etc.)
Documento	Determina la cantidad de colores que aparecerán en el cartel. En su nivel más bajo el diseño será en blanco y negro, mientras que en el nivel más alto se utilizarán muchos colores.
Documento	Marca de donde se obtienen los colores, si de la imagen principal o aleatoriamente.
Documento	Este gen marca si la imagen principal se usa como fondo completo del diseño o si solo ocupa una parte de este.
Documento	Determina si se utilizarán fondos oscuros o claros para el diseño.
Contraste	Todos los genes del grupo de Contraste determinan qué elementos del diseño utilizar para que exista contraste. Este gen determina si utilizar el tamaño de las fuentes para crear contraste. En su nivel más alto la diferencia entre el titulo y resto del texto será muy alta, mientras que en el nivel más bajo el tamaño será similar para no crear contraste.
Contraste	Este gen marca si han de usarse colores muy contrastados entre las fuentes. Este gen marcará, por tanto, si el tipo de esquema de colores lo forman colores completamente opuestos (esquema de complementarios o de triada).
Contraste	Dada la forma principal que tenga la imagen (curva o recta), este gen determinará si las fuentes deben crear mucho o poco contraste utilizando estilos muy rectos (fuentes Modern y OldStyle) o más curvos (fuentes Sans y Slab Serif).

Contraste	Este gen implica si ha de usarse algún tipo de decoración con los textos (fondo de color o borde) para crear contraste.
Contraste	Este gen determina si usar o no el grosor de las fuentes para crear contraste. En los niveles más altos, el titulo tendrá un estilo Negrita o Black, mientras que el resto de fuentes utilizarán estilos Normal o Light.
Contraste	Este gen marca si el estilo de fuente secundario debe ser como la fuente principal u otro para crear contraste.
Contraste	Este gen habilita la posibilidad de que el título y el subtitulo aparezcan en mayúsculas para aumentar el contraste.
Repetición	Al contrario que los genes de Contraste, el grupo de Repetición determina si una característica debe repetirse más o menos entre ellos. Este gen determina si se debe usar el tamaño de las fuentes como elemento repetitivo.
Repetición	Este gen marca si se debe usar un esquema de colores similares, como puede ser el esquema de análogos.
Repetición	Este gen marca si se debe repetir la forma de la imagen en la forma de las fuentes.
Repetición	Este gen determina si el grosor de las fuentes debe ser el mismo o similar.
Repetición	Este gen marca si la fuente secundaria debe ser la misma que la principal.
Entrada	Este gen es el único que no se genera aleatoriamente, sino que determina la forma que predomina en la imagen principal (curva o recta).
Composición	En diseños donde la imagen no ocupa el fondo por completo, este gen marca si debe situarse en una posición tradicional (arriba en diseños verticales, izquierda en horizontales) o hacerlo en una alternativa (abajo y derecha, respectivamente).
Composición	En diseños donde la imagen no ocupa el fondo por completo, este gen marca los bordes que deben tener la imagen y los textos al colocarse.

Composición	En diseños donde la imagen no ocupa el fondo por completo, este gen marca si se debe recortar la imagen alrededor del punto de interés.
Composición	Este gen determina que tipo de decoración van a tener los textos (en caso de que el gen de Contraste por decoración este activo). Puede ser un fondo detrás del texto o un borde.

Está cadena de genes es un objeto de la clase `Genes`, que a su vez es herencia de la clase `Array`. La vista de esta clase puede verse en la Figura 4.4.

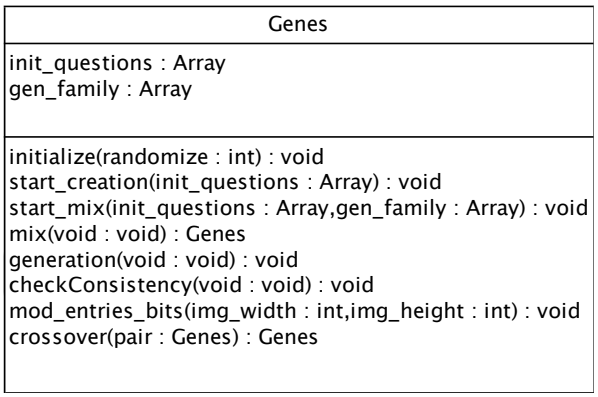

Figura 4.4: Clase `Genes`.

Gaudii implementa ciertas reglas de consistencia para evitar contradicciones en los cromosomas. Como se ha visto, existen cromosomas de los grupos de Repetición y Contraste que tratan sobre el mismo aspecto, pero es imposible que, por ejemplo, un diseño tenga un alto contraste entre los colores y a la vez los repita. Por esta razón se implementan una serie de reglas que comprueban los genes conflictivos y modifican su valor si es necesario.

A pesar de que la generación genética es aleatoria, existen otras reglas de comprobación para asegurar que ciertos genes del cromosoma cumplen los especificado en las preferencias iniciales introducidas por el usuario.

4.2.1. Cruce de individuos

Una de las principales razones para optar por los algoritmos genéticos para generar con ellos los diseños es porque estos permiten la posibilidad de mezclarse y crear objetos hijos que hereden las características de los objetos padres. En Gaudii la utilidad es evidente: después de generar diseños desde cero, se pueden elegir los más atractivos visualmente y mezclarlos para obtener más diseños similares a ellos.

La mezcla genética dará como resultado un único array genético a usar en uno de los diseños. El primer paso de la mezcla genética es mezclar todos los padres, los diseños previamente elegidos, entre sí. Esto es un coeficiente binomial $\binom{c}{k}$, siendo c el número de padres y k igual a 2 (el número de ellos que participará en cada combinación), que dará lugar a $\frac{c!}{k!(c-k)!}$ hijos. El array de los padres se guardará en la primera posición del array `gen_family`, mientras que el array de hijos lo hará en la segunda posición (Figura 4.5).

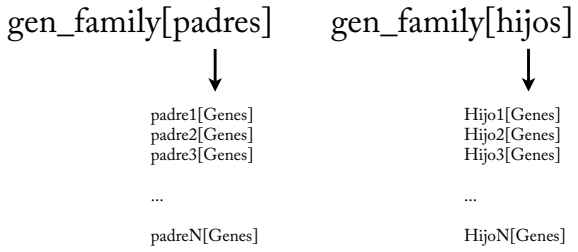

Figura 4.5: Array `Gen_family`.

La mezcla genética para dar con los hijos está implementada en la clase `Genes` en la función `crossover`. Uno de los Genes padre llama a la función pasándole como argumento otro de los padres. Ya dentro de la función, un algoritmo aleatorio decide el valor que tendrá cada gen del cromosoma. Existe un 47,5 % para cada padre para que el hijo herede uno de sus genes, y un 5 % restante de posibilidades destinadas a mutaciones aleatorias.

Después de esto, la generación de diseños sigue los mismos pasos que en una generación normal, salvo que ahora en lugar de una creación genética aleatoria, se pasa a la clase `Genes` el array con los padres y los hijos para usar estos en la mezcla final. Esta mezcla genética final crea dos nuevos hijos, y estos se mezclan, durante un bucle de 20 iteraciones, con alguno de los padres o hijos aleatoriamente seleccionados. Al final, estos dos nuevos hijos se mezclan

entre sí y dan como resultado la cadena genética que será definitiva para uno de los diseños.

Esta cadena de genes definitiva se someterá también a las reglas de consistencia.

4.3. Etapa de Uso del Modelo Experto

Durante esta etapa se procesan los elementos de diseño crudos y las cadenas de genes, de forma que al final del proceso se devuelven los elementos del diseño ya formateados. Desde una visión global, se reciben diseños (objetos `Design`) con una cadena genética asociada. Esta cadena es primero evaluada y procesada para obtener los consecuentes; posteriormente estos consecuentes son procesados para dale formato a los elementos de texto e imágenes del documento; por último, estos elementos ya formateados, y con un tamaño definido, se envían al módulo de optimización local para que los sitúe en el espacio libre que exista en el documento.

Estos son los objetivos de esta etapa:

- Inferir los consecuentes del cromosoma del diseño.
- Generar los elementos del diseño a partir de los consecuentes obtenidos.
- Ordenar los elementos en el espacio del diseño.

En esta etapa se han utilizado diversos módulos de soporte:

- **Motor de Inferencia Difusa:** Este módulo recoge el cromosoma de cada diseño, cuyos genes actuarán como hechos en la inferencia, y obtiene los consecuentes.
- **Módulo de Generador de Elementos:** En este módulo se evalúan cada uno de los consecuentes y se da formato a los elementos de diseño crudo.
- **Módulo de Optimización Local:** Una vez generados los elementos, estos se deben colocar en el espacio de diseño disponible. Este módulo será el responsable de decidir la localización de los elementos.

En las siguientes subsecciones se explicará con más detalle estos módulos.

Variable	Tipo	Etiqueta	Cjto. Var. Ling.
Gen de contraste por tamaño	Entrada	CS	{L, M, H}
Gen de repetición por tamaño	Entrada	RS	{L, H}
Tamaño Título	Salida	H0	{VB, Hu}

Cuadro 4.2: Algunas variables del sistema.

4.3.1. Motor de Inferencia Difusa

Este módulo se encargan de recibir por entrada los valores del cromosoma, inferirá los consecuentes y remitirá los valores de salida a cada diseño. Es decir, los hechos de este motor de inferencia será los genes del cromosoma.

El motor de inferencia implementado soporta cualquier número de entradas, pero tan solo una salida que es la que añadirá en la cadena de valores obtenidos. Las funciones de pertenencia pueden tener cualquier forma definida mediante dos puntos y dos pendientes (trapecios, triángulos, rectángulos, etc.). Para calcular la salida de cada regla se utiliza el mínimo (T-norma estándar); para la conclusión se usa el máximo (T-conorma estándar); por último, en la etapa de deborrosificación se aplica el centro de gravedad.

Se han utilizado un total de 128 reglas agrupadas en tareas de Documento, Repetición/-Contraste y Composición. Dada la extensión de las reglas y sus variables, a continuación solo se mostrará una pequeña lista del total de variables (Cuadro 4.2) y reglas utilizadas (Cuadro 4.3). Las variables lingüísticas que aparecen hacen referencia a Low (L), High (H), Medium (M), Very Big (VB) y Huge (Hu). En el Anexo A del presente documento se pueden consultar el resto de reglas utilizadas en el sistema.

En la versión final de Gaudii, estas reglas se almacenan en un fichero XML.

4.3.1.1. Implementación del MID

Para la implementación del Motor de Inferencia Difuso se han utilizado seis clases siguiendo la estructura lógica de estos sistemas: el motor de inferencia cuenta con *n* sistemas, cada uno con su conjunto de reglas y variables lingüísticas. El diagrama de clases puede verse en la Figura 4.6.

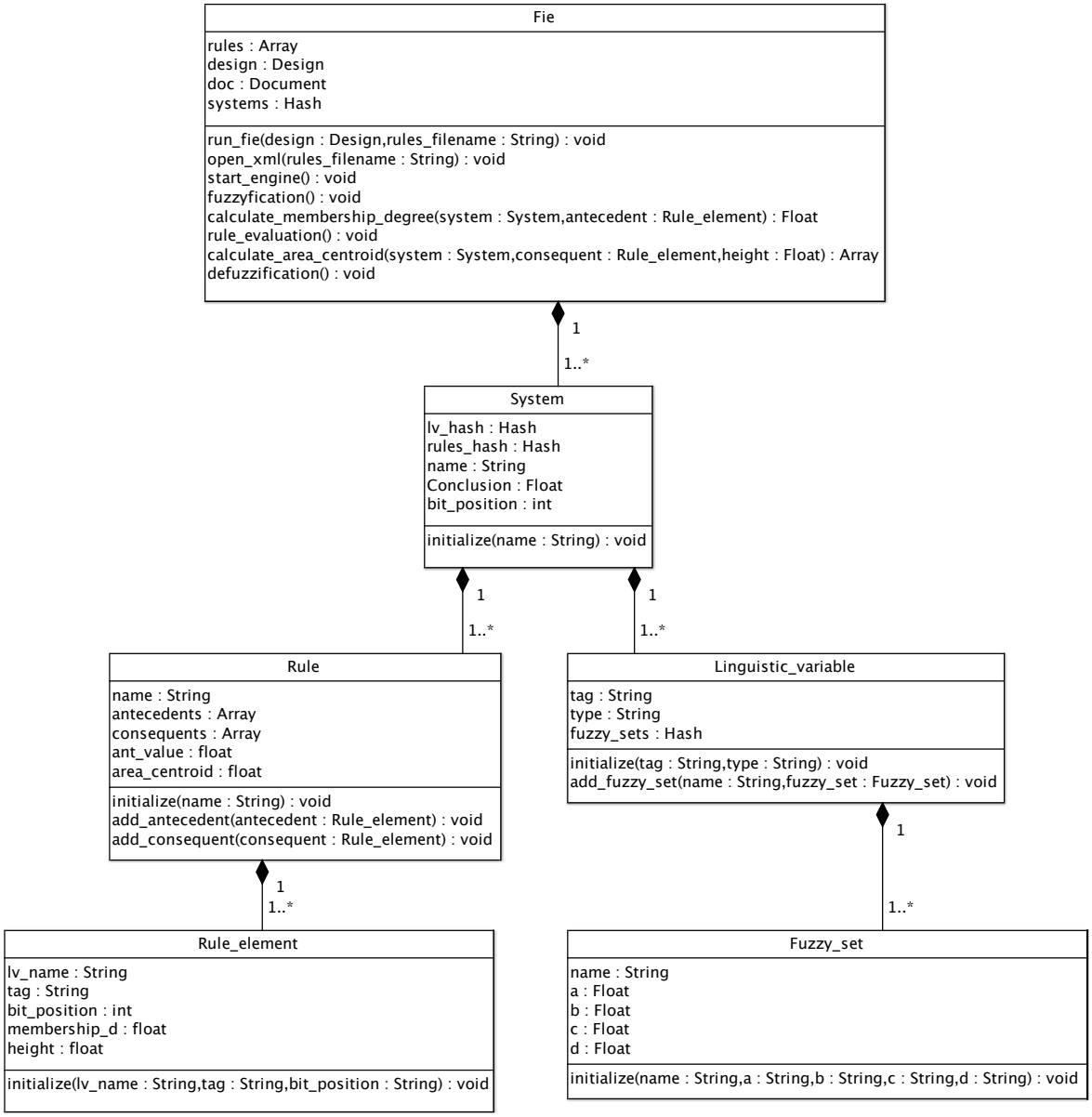

Figura 4.6: Clase `Fie` (*Fuzzy Inference Engine*, Motor de Inferencia Difusa).

R	Antecedente	Consecuente
R_{21}:	If CS is L $\wedge$ RS is L	$\Rightarrow$ H0 is VB
R_{22}:	If CS is L $\wedge$ RS is H	$\Rightarrow$ H0 is VB
R_{23}:	If CS is M $\wedge$ RS is L	$\Rightarrow$ H0 is VB
R_{24}:	If CS is M $\wedge$ RS is H	$\Rightarrow$ H0 is VB
R_{25}:	If CS is H $\wedge$ RS is L	$\Rightarrow$ H0 is Hu

Cuadro 4.3: Reglas para la elección del tamaño del título.

El pseudocódigo del Motor de Inferencia Difusa es el que se ve en el Algoritmo 4.

Algoritmo 4 Motor de Inferencia Difusa

```
1: arrancar_motor()
2: para cada Sistema Experto hacer
3:     borrosificación()
4:     evaluar_reglas();
5:     deborrosificación()
6: fin para
```

La división del proceso es, como se ve en el pseudocódigo, la seguida en el Método de Mandami. A continuación explicaremos en más detalle cada paso de la implementación:

- **Paso 0. Arrancar motor**

  En este paso se preparan las estructuras de datos del sistema. Para ello se recorre el XML que almacena las reglas y se crean e inicializan los objetos correspondientes a lo sistemas, reglas y variables lingüísticas.

- **Paso 1. Borrosificación**

  En este paso se calcula el grado de pertenencia de cada antecedente para el sistema que corresponda. Primero se obtiene la forma de cada función de pertenencia y posteriormente se devuelve el grado de pertenencia dependiendo del correspondiente gen.

- **Paso 2. Evaluar las reglas**

En este paso se evalúa el antecedente de cada regla para obtener su valor mínimo (en Gaudii, todas las reglas utilizan el operador lógico AND). Después de esto calcula el área y el centroide, o centro de gravedad, para cada regla.

- **Paso 3. Deborrosificación**

  Este último paso en realidad agrupa los dos últimos pasos del método de Mamdani: agregación de los consecuentes y deborrosificación.

  Con el área y el centroide de cada regla calcula el centroide utilizando la Fórmula 3.3. El proceso puede verse en el Algoritmo 5:

Algoritmo 5 Proceso de Deborrosificación

```
1: sumaProductos = 0
2: sumaAreas = 0
3: para cada Regla hacer
4:    sumaProductos += Regla.area + Regla.centroide
5:    sumaAreas += Regla.area
6: fin para
7: conclusion = sumaProductos / sumaAreas
```

Por último, cada conclusión se añade al correspondiente bit del objeto Diseño para luego generar alguna de las características del diseño.

Se ha optado por utilizar diversas tablas Hash (o arrays asociativos) para manejar tanto las variables lingüísticas como las reglas, ya que ambas tienen una etiqueta que las define. Con esta elección el sistema gana en claridad, puesto que en todo momento se referencia a los nombres de cada componente.

4.3.2. Módulo de Generador de Elementos

En este módulo se utilizan los consecuentes obtenidos en el Motor de Inferencia Difusa. Esos consecuentes se procesan y de su evaluación se obtiene la información necesaria para formatear los elementos del diseño. Este módulo envía, posteriormente, los elementos formateados al módulo de búsqueda local para que los distribuya en el espacio del documento.

La estructura de datos de este módulo puede verse en la Figura 4.7. En total se utilizan siete clases; la principal es `Design`, que está compuesta, en gran parte, por objetos del resto de las clases. Este módulo se divide en distintas funciones dependiendo de las características que procesen:

- **Funciones del Documento:** Este grupo de funciones generan propiedades relativas al documento en sí.
- **Funciones de Colores:** Estas funciones se encargan de crear el esquema de colores del diseño.
- **Funciones de Fuentes:** Se encargan de formatear correctamente el texto del diseño.
- **Funciones de Precomposición:** Antes de enviar los datos al módulo de búsqueda local se realizan diversas funciones para preparar los elementos.

Todas estas funciones tienen un funcionamiento similar: primero observan el resultado de un determinado bit en el array `values_bits`, y posteriormente generan un elemento basándose en ese resultado. En ocasiones, como en el tamaño de las fuentes, ese mismo bit tendrá el resultado ya generado. Estos grupos de funciones se explicarán con más detalle a continuación.

4.3.2.1. Funciones del Documento

Como se ha comentado, estas funciones son relativas a la generación de características generales del cartel:

`generate_ratio` genera el tipo de proporción que tendrá el cartel. Desde ratio 1:1, que dará lugar a un diseño cuadrado, hasta la proporción 16:9 que genera un diseño panorámico. Esta generación es orientativa, puesto que no podemos saber con antelación cuanto espacio necesitaremos para los elementos. Debido a esto existen funciones posteriores que ajustan el diseño en caso de que sobre o falte espacio. Esto solo es para los casos en los que la imagen no cubre completamente el fondo del diseño, evidentemente. Si la imagen cubre el fondo, la proporción del cartel será la de la imagen.

Figura 4.7: Estructuras de datos para la generación de elementos de diseño.

`generate_orientation` determina si el diseño será vertical u horizontal. Con esto ocurre igual que con la función anterior: si la imagen es todo el fondo no se podrá generar un diseño con una orientación contraria a ésta.

La función `generate_sizes` termina lo empezado por las anteriores funciones. Aquí, dependiendo del ratio y la orientación, se generan el ancho y el largo del documento en píxeles.

Por último, la función `generate_margin` genera los márgenes del documento para los textos y la imagen a partir del correspondiente gen (Figura 4.8). Estos márgenes están discretizados para, como veremos más adelante, facilitar la tarea de alinear elementos.

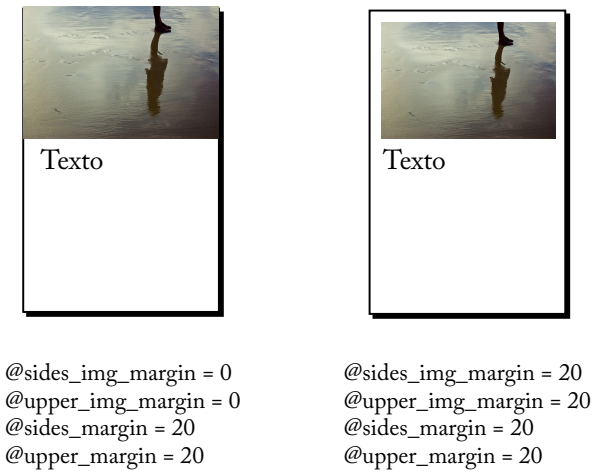

Figura 4.8: Diferentes tipos de márgenes.

`modify_image` es una función que, dependiendo del correspondiente bit, hará que la imagen principal sea versión recortada de la misma. Esta versión será el recorte explicado en el puto de Obtención de Datos.

El siguiente paso es calcular el tamaño final que tendrá la imagen. Esto se realizará en la función `generate_image_sizes`, y dependerá de todo lo calculado anteriormente, pues se ajustará al tamaño del documento, a su orientación y a los márgenes. En caso de que la imagen ocupe todo el fondo del documento, el tamaño de este será el de la imagen.

Por último, `generate_predominant_shape` genera la forma que predominará en el diseño (las fuentes curvas o rectas) dependiendo del correspondiente bit.

4.3.2.2. Funciones de Colores

Todas estas funciones son relativas a la creación y uso del esquema de colores en el diseño. La función donde se realiza esta tarea es `generate_color_scheme`, y hace uso exhaustivo de la clase `Colour` y sus funciones.

Por tanto, en este apartado se crea un esquema de colores para el diseño. Este se almacena en un array que está distribuido como aparece en la Figura 4.9.

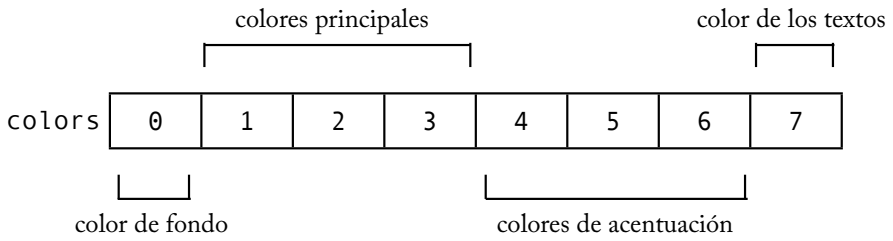

Figura 4.9: El array `colors`.

La generación de colores depende de varios de los resultados inferidos. Primero se obtiene el color principal del diseño, que podrá ser obtenido desde la imagen principal o de forma aleatoria. Los otros dos colores principales son versiones claras y oscuras del mismo color.

Para obtener los colores de la imagen, primero la reducimos de tamaño. De esta manera, con la propia reducción se eliminan muchos colores a base de hacer la media entre ellos. Posteriormente usamos la función `quantize` de ImageMagick; ésta analiza los colores de la imagen y elige un determinado set de colores que representa a la imagen.

Con esa función se obtienen cinco colores representativos de la imagen. Posteriormente alteramos su tono y saturación ligeramente. Esto es debido a que los colores de los resultados se muestran más apagados que en la imagen original por la mezcla a la que han sido sometidos.

Por último, obtenemos el histograma de color de la imagen. Este histograma nos dará una tabla hash con los colores finales y el número de pixeles de cada uno en la imagen.

El color más usado se tomará como el color de fondo de la imagen; del resto obtenemos, por orden de número de pixeles en la imagen, otros dos colores siempre que cumplan ciertos

requisitos de saturación y luminosidad, para evitar que el color principal acabe siendo algún tono de gris.

Volviendo al esquema de colores del diseño, los colores de acentuación serán colores que consigan crear un esquema visualmente atractivo con el color principal ya obtenido. Como elegir unos colores u otros depende del contraste, o la repetición, definidas por el array genético (colores análogos, complementarios, de su triada, etc.).

El color de fondo del diseño, en caso de que no sea la imagen principal el fondo completo, dependerá del bit de fondo, pudiendo ser blanco/negro o una versión clara/oscura del color principal.

Lo último de la función `generate_color_scheme` es la generación del color de los textos, que se hará teniendo en cuenta que haga contraste con el color de fondo plano o, en caso de que haya una imagen, el color más usado en esta (que se calculó en su momento).

Otra función de generación de colores es `generate_text_backgrounds`; a ésta solo se le llama en caso de que el bit correspondiente se haya activado. Se recorre el array de colores y se elige un color, blanco o negro dependiendo de su luminosidad, para cada color. El objetivo de esta función es llenar el array `bg_colors`, que se utilizará para la decoración de los textos. En el caso de que la decoración sea el fondo de un texto, el color original pasa a ser el fondo, mientras que el color generado aquí rellena las letras. Idéntico caso en caso de que la decoración sea añadir un borde a las fuentes (Figura 4.10).

Figura 4.10: Decoración de los textos.

Por último, la función `improve_legibility` recorre el array de colores y comprueba

que los colores elegidos tienen el suficiente contraste con el color de fondo. Para esto se hace uso de la función `is_contrasted_enough`, de la clase `Colour`; esta clase, y sus funciones, se explicarán a continuación.

La clase Colour y la obtención de colores

En esta clase se realizan las operaciones con los colores de la clase `Design`. Tiene cuatro atributos para definir el color: un entero para cada componente RGB (rojo, verde y azul) y una cadena que representa el valor hexadecimal de estos valores. Alternativamente, también tiene funciones para transformar estos valores RGB al espacio HSL (Tono, Saturación y Brillo). A continuación explicaremos las funciones más relevantes de esta clase.

La función `is_contrasted_enough` revisa si entre otro color y el que hace la llamada existe el suficiente contraste como para que sean texto y fondo, respectivamente. Se ha utilizado una adaptación de la formula promovida por el W3C para establecer criterios universales respecto a la accesibilidad en sitios web. Esta fórmula especifica que la diferencia del brillo del color (Fórmula 4.1) de ambos colores debe ser superior a 125, y que a su vez la diferencia de los colores (Fórmula 4.2) debe ser mayor que 500. Estos dos factores se calculan de la siguiente forma:

$$BrilloDelColor = \frac{(Valor_{rojo} * 299 + Valor_{verde} * 587 + Valor_{azul} * 114)}{1000} \tag{4.1}$$

$$DiferenciaDeColores = Diferencia_{rojo} + Diferencia_{verde} + Diferencia_{azul} \tag{4.2}$$

$$Diferencia_{componente} = max(Comp_1, Comp_2) + min(Comp_1, Comp_2) \tag{4.3}$$

En el caso de Gaudii, las restricciones son algo más suaves puesto que se trata de diseño impreso y no de accesibilidad web. Así, en la implementación final los valores se han fijado en 100, en la diferencia de brillo, y 300, en la diferencia de colores, para que exista buen contraste entre ellos.

Siguiendo con las funciones de `Colour`, existen diversas funciones que modificar el brillo y la saturación de los colores según las necesidades del sistema. Todas estas funciones

funcionan de manera similar: transforman el color del espacio RGB al HSL, modifican el valor (normalmente se reduce o aumenta por un determinado factor) y lo devuelven en formato RGB en forma de objeto `Colour`.

Las funciones más importantes de esta clase son las referidas a la búsqueda de colores armónicos entre sí. Esta búsqueda tiene que ver con mover el color en el espacio HSL unos determinados grados. Por ejemplo, para obtener el color complementario se suma 180º, o para obtener los dos colores restantes de la triada se suma y se resta 120º.

Por tanto, esta búsqueda de colores armónicos necesita una función de salto. En Gaudii se ha implementado con el nombre de `jump_to_color`. A esta función se le pasa el salto y devuelve el color correspondiente. Teniendo en cuenta lo dicho en el anterior párrafo, en un principio se desarrolló transformando el color al espacio HSL y modificando el valor del Tono, pero no funcionó porque en el estudio de la armonía de colores se había hecho pensando en el espacio de color RBY (rojo, azul y amarillo), mientras que los ordenadores trabajan en el espacio RGB.

Por ejemplo, para obtener el color complementario se debe hacer un salto de 180º en la escala de color; mientras que el resultado debería ser el verde, al ser el espacio RGB el complementario que nos devuelve el salto es el azul celeste (Figura 4.11).

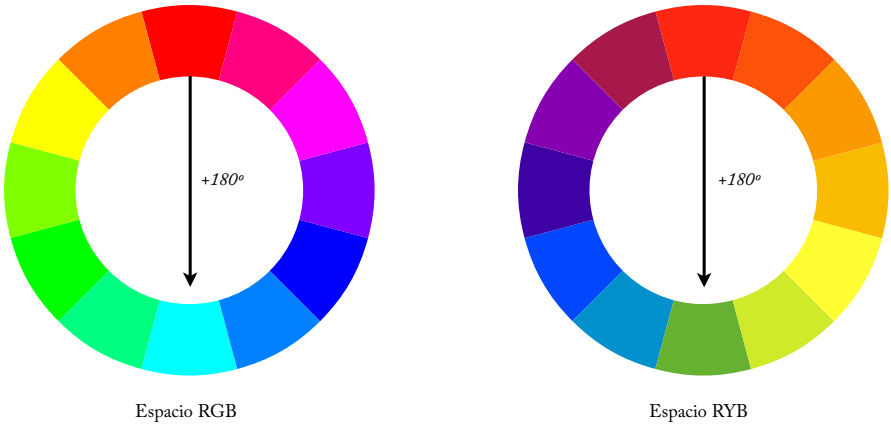

Figura 4.11: Diferencias entre espacios de colores.

Por tanto, no se podía utilizar el simple salto de tono en HSL. En su lugar se ha desarrollado modelando la rueda RYB en 12 arrays, dividiendo así la rueda en 12 sectores de 30º

(claramente diferenciables si se observa el espacio de color en la Figura 4.11). De esta forma, un salto de, por ejemplo, 180º se modelará como un salto de 6 sectores.

Pero el salto de sector no basta puesto que cada color tiene una posición determinada en el sector. Para solucionar este problema, Gaudii calcula la posición dentro de cada sector calculando la diferencia de posición del primer color respecto a su Tono. Es decir, cada sector que modela el sistema almacena de la información del tono saturación y brillo máximos y mínimos. Lo que se hace es calcular, mediante la diferencia de tonos, la posición dentro del sector; posteriormente, con el máximo y el mínimo de cada componente HSL se calcula el avance dentro de cada sector.

El Tono y la Saturación se calculan de forma similar, pero además luego se modifican debido a que cada sector tiene unos valores de Brillo y Saturación diferentes. Se calcula primero cual es la diferencia que había en el primero color con respecto a la saturación y brillo medio del sector, y luego ese factor de diferencia se aplica al brillo y saturación del nuevo color.

Por último, se calcula el valor RGB y se devuelve un nuevo color generado a partir de ese RGB.

4.3.2.3. Funciones de Fuentes

Las funciones de este grupo trabajan en el formateado del textos del diseño. Es decir, en determinar las fuentes que utilizarán (tipografía, grosor, tamaño, etc.). Para crear estas fuentes, primero se escogen las familias de fuentes que se utilizarán. Siguiendo uno de los consejos de Samara [59] sobre la creación de diseños[1], en Gaudii se crean una o dos familias de fuentes para el cartel.

El uso de cada fuente dependerá de cierto gen del array, pero generalmente la fuente principal se usa en los títulos o textos destacados, mientras que la secundaria se usa en el resto. Es por eso que la principal puede ser un estilo de fuente serio (Sans, Slab Serif, etc.) o uno más gráfico (Script o Graphic), mientras que la secundaria siempre utilizará un estilo más serio. En esta función se elige primero el estilo de fuente para cada familia, primaria y secundaria, y posteriormente se llama a la función `get_font_name` junto al tipo de forma

[1] Una de las reglas básicas de Diseño Gráfico para este autor es *Usa como mucho dos familias de fuentes*

que queremos para la fuente.

El tipo de forma es importante, puesto que actúa en repetición, o en contraste, con el de la imagen principal. En Gaudii se ha clasificado cada tipo de letra dependiendo de su curvatura o rectitud:

- Modern: Completamente rectas.
- Old Style: Ligeramente rectas.
- Slab Serif: Ligeramente curvas.
- Sans Serif: Completamente curvas.
- Script: Depende de la fuente (normalmente curvas).
- Graphic: Depende de la fuente.

La función `get_font_name` recorre un fichero XML con todas las fuentes disponibles en Gaudii. Cada entrada del archivo incluye el tipo de forma, el tipo de fuente (*others* se refiere a las fuentes serias) y la ruta del archivo. La función elige una del tipo buscado de forma aleatoria y devuelve su ruta. El siguiente fragmento muestra una entrada de ejemplo del archivo XML:

```
1 <font name="Comiquita Sans" type="graphic" shape="curved" filename="public
    /fonts/ComiquitaSans.ttf"/>
```

Una vez se ha elegido las dos familias de fuentes que utilizará el diseño, el sistema llama a la función `generate_font` para todos los textos del sistema. Esta función llama a su vez a diversas funciones, todas pertenecientes a la clase `Letters`, que establecen los atributos de cada texto: tamaño, peso, ruta del archivo y color. La elección de cada una depende de los bits de repetición y contraste.

4.3.2.4. Funciones de Precomposición

Estas funciones están destinadas a preparar cierta información del documento que necesita el modulo de búsqueda local, principalmente información relativa a la posición y el tamaño del espacio libre del documento.

La primera de las funciones que realiza esta tarea, `composite_main_image_position`, se encarga de distribuir la imagen en el documento. Como todo, depende del gen; puede que tenga que ocupar completamente el diseño o que ocupe solo una parte y deje el resto a los textos. En este último caso, también se decide si toma una posición normal (izquierda en carteles horizontales; arriba en carteles verticales) o alternativa (derecha y abajo, respectivamente). En cualquier caso, se coloca la imagen teniendo en cuenta los márgenes generados anteriormente y se actualizan las variables del objeto `DesignImage` con las posiciones x e y.

Una vez se sabe la posición de la imagen, se calcula el espacio libre para la composición en la función `generate_white_spaces`. En el caso de la imagen como fondo total, el espacio será todo el documento salvo el punto de interés de ésta; en el otro caso, será el espacio dejado por la imagen y los márgenes. La información sobre el espacio en blanco se almacena en las variables del objeto `Design`.

Después de generar el espacio libre, se comprueba que el ancho del título y el subtítulo caben dentro de ese espacio con el tamaño de letra asignado. De no caber, se reduce el tamaño hasta que quepa. La diferencia de tamaño respecto a la reducción se aplica también al resto de fuentes para mantener el contraste o la repetición que se asignó al principio.

La función `initialize_composition_values` calcula el tamaño de todos los grupos; cada grupo tiene una función para calcular su tamaño, llamada `calculate_sizes`, que recorre sus elementos y calcula el ancho y alto que alcanzan, teniendo en cuenta en tamaño de las fuentes y el de la imagen asociada al grupo, de existir ésta. El alto de cada elemento se almacene en un array del propio grupo para luego procesar la posición exacta de cada elemento. Es decir, el sistema sabrá a que altura colocar el tercer elemento si se conoce la altura de los dos anteriores.

Por otro lado, los tamaños de los grupos son múltiplos de 20 para facilitar la colocación posterior; de forma que si no llega a múltiplo de 20 se suma lo necesario para llegar. Aunque sobre espacio, cada grupo se alineara por uno de los dos lados con otros grupos que también se alinearan al mismo lado, por lo que no importará que el tamaño de cada grupo sea un poco mayor.

Por su parte, `adapt_design_size` comprueba el tamaño del espacio libre y el de la suma de los grupos. Si la diferencia es muy alta a favor del espacio libre, reduce este para

facilitar la posterior distribución.

Por último, `calculate_interest_point` genera un punto de interés aleatoriamente. La mayor probabilidad es que sea el punto de interés marcado por el usuario; las otras posibilidades son que e punto de interés sea una de las zonas prefijadas por Gaudii, teniendo en cuenta el punto de interés marcado por el usuario y la cantidad de área de este punto en las zonas.

4.3.3. Módulo de Optimización Local

En este módulo se organizan los grupos de elementos en el espacio de diseño. La parte más importante reside en un submódulo donde se utiliza un algoritmo de búsqueda local, el algoritmo de Enfriamiento Simulado (Algoritmo 2); concretamente una versión ligeramente modificada que puede verse en el Algoritmo 6.

Este algoritmo se ha escrito en el lenguaje C debido a que su rendimiento era muy superior a la versión Ruby del mismo. A esta aplicación en C se le llama pasándole toda la información por la linea de argumentos del programa. La sintaxis de llamada es la siguiente:

```
$ ./simulated_annealing infoEspacioBlanco infoPuntoInteres
tipoColocacionImagen orientacionDiseño infoGrupo1 ... infoGrupoN
```

Siendo `infoEspacioBlanco` e `infoPuntoInteres` un conjunto de cuatro argumentos que especifican la posición de la esquina superior izquierda y su dimensiones respectivas. Los argumentos `infoGrupoN` también pasan esos cuatro argumentos, además de la posición de su punto medio y de si es un grupo normal. La posición inicial de cada grupo se calcula en Ruby antes de lanzar el modulo escrito en C. Dentro ya del citado módulo se almacena toda la información de los argumentos en estructuras para facilitar la comprensión y el procesado de la información.

El resto del proceso del algoritmo se explicará a continuación.

4.3.3.1. Función de comprobación

Para ganar en flexibilidad, Gaudii implementa diversas funciones para ajustar el tamaño del documento en función de la cantidad de texto que haya en él. Una de estas funciones se

Algoritmo 6 Enfriamiento Simulado en Gaudii

```
1: T = T0
2: Sactual = GenerarSolucionActual
3: Smejor = Sactual
4: mientras T>=Tf hacer
5:    si T==TemperaturaAdecuada ∧ Comprobacion==TRUE entonces
6:       Comprobacion = comprobarEspacio(Smejor, espacioBlanco)
7:    fin si
8:    para i=1 hasta L(T) hacer
9:       Scandidata = seleccionaSolucionVecina(Sactual, T)
10:      δ = coste(Scandidata) - coste(Sactual)
11:      si U(0,1)<e^(-δ/T) ∨ δ<0 entonces
12:         Sactual = Scandidata
13:      fin si
14:      si coste(Sactual) < coste(Smejor) entonces
15:         Smejor = Sactual
16:      fin si
17:   fin para
18:   T = α(T)
19: fin mientras
20: devolver Smejor
```

encuentra dentro del Algoritmo de Enfriamiento Simulado (concretamente las líneas 5, 6 y 7 del Algoritmo 6).

La función comprobarEspacio (línea 6) se llama a determinadas temperaturas, cuando el algoritmo ya ha encontrado alguna solución viable. Esta función comprueba cuantos grupos de textos se ha salido del espacio blanco asignado. En caso de que ninguno se haya salido, significará que hay espacio suficiente para los grupos de texto y no se volverá a llamar a la función. En caso contrario, y dependiendo del número de grupos que se hayan salido, se hará crecer el tamaño del diseño.

El crecimiento que añadimos al espacio en blanco original se devuelve posteriormente en la cadena final, de forma que el sistema reconoce qué parte ha crecido y cuanto para actualizar el tamaño de los diseños. Esta función solo es válida en los casos que la imagen no ocupa todo el fondo del diseño, sino solo una parte.

4.3.3.2. Seleccionar vecino

En el algoritmo de Enfriamiento Simulado se parte de una solución inicial y se van buscando soluciones vecinas a esta. Tal y como está definida, una solución es un array de estructuras de grupos, cada una de estas definidas por su posición y tamaño. Una solución vecina a otra es un movimiento de todos los grupos de texto en ambos ejes. La distancia del movimiento de los grupos es completamente aleatoria, pues el algoritmo busca forzar las malas soluciones para salir de los máximos locales y alcanzar el máximo global.

Debido a que el espacio del diseño se puede llegar a dividir en miles de píxeles, en Gaudii se ha optado por discretizar este espacio. Al discretizar el espacio se fuerza a que sea más probable que los elementos se alineen entre sí, algo que se tendrá en cuenta en la función de coste. En concreto, en la versión final se ha dividido el espacio de forma que simula una rejilla con espacios de 20 píxeles (Figura 4.12).

Como se ha dicho, el salto que realiza cada elemento se hace de forma aleatoria, y este depende de la temperatura actual en el algoritmo. De esta manera hacemos que un proceso de exploración al principio del algoritmo, para luego centrarnos en un proceso de intensificación al final. Es decir, al principio se buscan soluciones muy lejanas para ver cual es la más optima, y al final se buscan soluciones cercanas a la mejor que se encontró en el proceso de

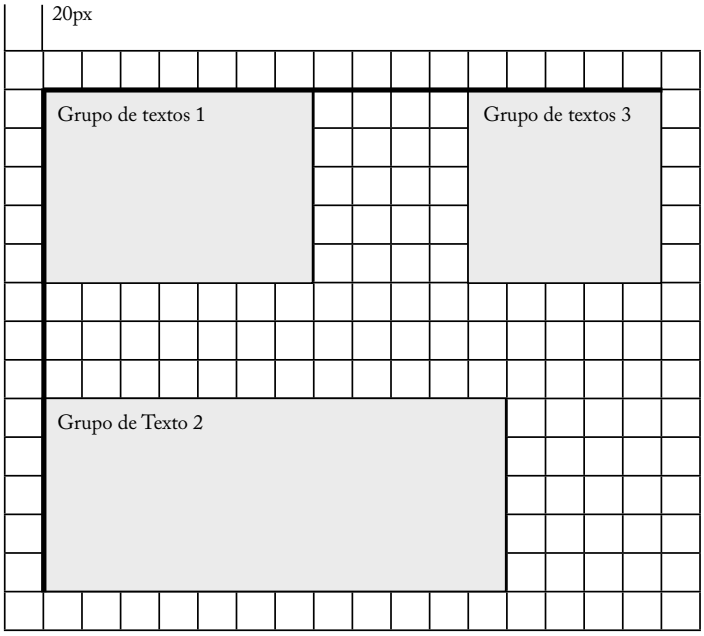

Figura 4.12: Elementos colocados en saltos de 20 píxeles.

exploración. Esta alteración del salto se implementa según la Fórmula 4.4.

$$Salto_{final} = Salto_{inicial} * \frac{Temperatura_{actual}}{Temperatura_{maxima}} \tag{4.4}$$

4.3.3.3. Evaluación del Coste

En esta parte del algoritmo se implementan diversas funciones de coste para evaluar cada solución encontrada. La distribución de los grupos de textos en el diseño no se debe hacer con una búsqueda normal de aprovechamiento del espacio, sino que debemos encontrar la forma en que además sea visualmente atractivo según las reglas de composición gráfica.

En este módulo se pueden implementar diversas funciones de coste para ordenar los elementos según esas directrices. En la versión final de Gaudii se han implementado tres funciones para calcular el coste: función de solapamiento, función de equilibrio y función de alineación. Las dos últimas funciones están normalizadas (es decir, devuelven un float entre 0 y 1 dependiendo del coste) mientras que la función de solapamiento actúa como discriminador de soluciones.

Es decir, si las soluciones no superan la función de solapamiento no se calculan las otras funciones y se devuelve un número muy grande (Algoritmo 7). Esta función de solapamiento comprueba si alguno de los grupos esta encima del otro, ya que en la búsqueda de soluciones vecinas no se tiene en cuenta esa eventualidad. De esta forma, además, evitamos que se calcule el coste a soluciones que se sabe que no van a funcionar.

Algoritmo 7 Evaluación del coste de la solución

```
1: si costeSolapamiento == numeroMuyGrande entonces
2:     devolver numeroMuyGrande
3: si no
4:     devolver (costeEquilibrio * factorEquilibrio) + (costeAlineacion * factorAlineacion)
5: fin si
```

En la versión final de Gaudii se ha otorgado un 70 % de importancia al coste por alineación y un 30 % al coste por equilibrio. A continuación explicaremos con mayor detalle estas dos funciones.

Coste de equilibrio

Para calcular el equilibrio de un diseño, al que también podemos referirnos como *equilibrio visual*, consideramos que cada elemento en el plano del diseño tiene un determinado peso visual, y lo que se pretende es distribuir los pesos de los elementos de forma que el centroide resultante del polígono de los puntos medios de los grupos esté lo más cercano posible al centro del diseño (Figura 4.13). La distancia de este centroide respeto al centro del espacio en blanco se normalizará con la máxima distancia posible en el espacio libre. Cuanto más cerca esté el centroide del centro, mejor será la solución.

Para conocer el valor del centroide utilizamos la fórmula de calcular éstos en geometría, cambiando las *x* en el caso de querer calcular las ordenadas.

$$C = \frac{\sum x_1 * p_1 + x_2 * p_2 + x_3 * p_3 + ... + x_k * p_k}{\sum p_1 + p_2 + p_3 + ... + p_k}$$

El peso visual de cada grupo, p_x, se calcula dependiendo de su tamaño y de los elementos dentro de él. El punto de interés de la imagen es el elemento con más peso visual, seguido de

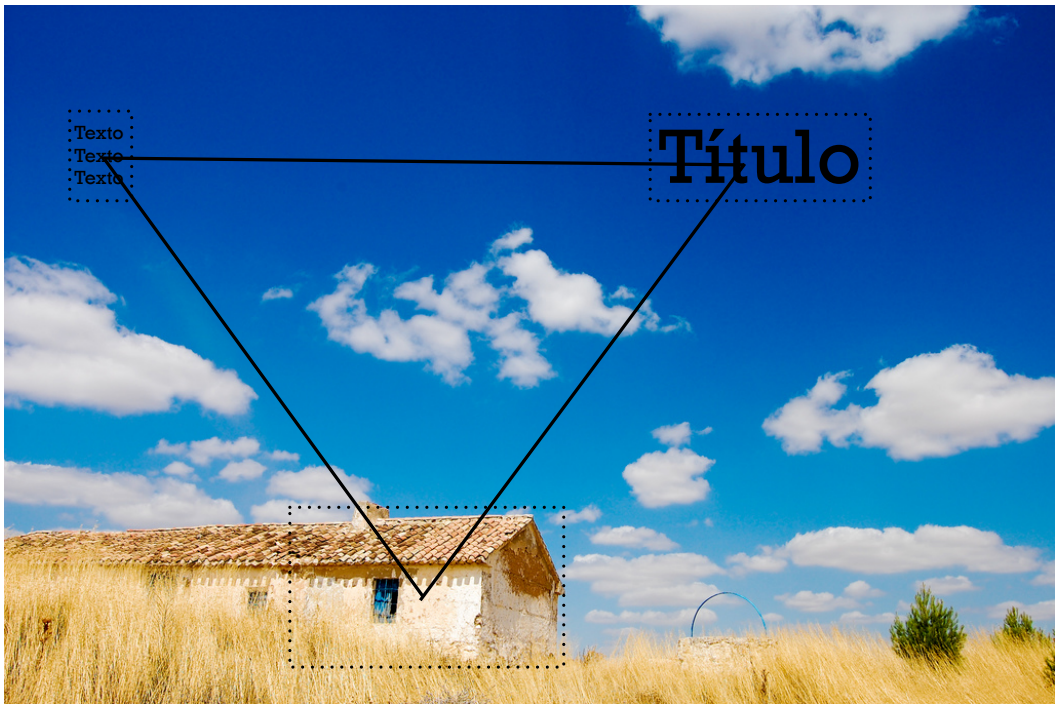

Figura 4.13: Ejemplo de calculo del centroide en función del peso visual de cada elemento.

cerca por el título del cartel. El resto de elementos (subtitulo, texto destacado, texto normal y notas) le siguen de forma decreciente.

Esta función del coste de equilibrio es muy importante para que los elementos consigan no solo colocarse, sino que además lo hagan consiguiendo una composición armónica desde el punto de vista gráfico.

Coste de alineación

Utilizando únicamente la función de coste de equilibrio, los elementos se colocan de una forma visualmente atractiva, pero sin ningún tipo de orden visual. La función de coste de alineación suple esta carencia, puesto que obliga a los elementos a colocarse respecto a ciertos ejes viuales para obtener un mejor coste.

Esta función se basa en uno de los principios C.R.A.P., concretamente en el de Alineamiento, que especificaba que hay que tratar de colocar los elementos en la misma alineación para conseguir una conexión visual entre ellos.

Para obtener el coste de alineación se calcula la cantidad de grupos que están alineados

respecto a ciertas lineas de alineamiento, y se le da un valor dependiendo de como de buena sea esta alineación. Las lineas de alineamiento son las propias de cada grupo, los márgenes generados y las del punto de interés de la imagen.

Cada grupo de textos comienza con un determinado coste, y este se va restando en función de lo bueno que sea su alineamiento. Las mejores posiciones que puede tener un grupo de textos son, de mejor a peor:

1. **Alignment Complete Great:** En este alineamiento, el grupo de textos está alineado, tanto en el eje X como en el Y, con los márgenes y con al menos otro grupo de textos.

2. **Alignment Complete Good:** Este indica que el grupo se ha alineado correctamente con los márgenes del cartel, tanto eje X como Y, pero no con ningún otro objeto.

3. **Alignment Complete Poor:** El grupo de textos se ha alineado con algún otro objeto, pero ambos están lejos de los márgenes del documento.

4. **Alignment Halfway:** Aquí indica que el grupo está alineado con algún margen o grupo, pero solo en alguno de los ejes. La resta al coste es menor en este caso, pero al acercarse a una solución ideal se le premia.

Además de estas posible soluciones, estar alineado con los márgenes o con el punto de interés de la imagen todavía sigue restando más al coste inicial de cada grupo. Al final, todos los costes individuales se suman y se divide entre el máximo posible, de forma que se obtiene el coste normalizado.

4.3.3.4. Otras funciones de ordenación de elementos

Se ha visto que en el algoritmo de Enfriamiento Simulado que existe una función destinada a aumentar el tamaño del lienzo en caso de que los textos no quepan en el espacio libre asignado. Esto solo funciona, por razones evidentes, en los casos que la imagen principal no ocupaba el fondo del cartel en su totalidad. Para estos últimos casos no se puede ampliar el tamaño del lienzo, por lo que se ha optado por crear una función que compruebe la disposición de los elementos y que reduzca el tamaño de las fuentes si es necesario.

El submódulo en C no dispone de la información necesaria para acometer esta tarea, puesto que, por un lado, la función que calcula las dimensiones de los textos pertenece a RMagick, y por otro, a ese módulo no se le pasa toda esa cantidad de información por la línea de argumentos. La solución es que el módulo de C devuelva la cadena no solo con la posición de los elementos, sino también con el coste final de esa solución. De ser mayor que 1.0, se entiende que no ha encontrado el espacio porque no había, por lo que se reduce el tamaño de las fuentes, se recalculan los tamaños y se vuelve a llamar al módulo de distribución de elementos. Esto se hace hasta tres veces; en caso de que tras tres intentos los textos no quepan, se devolverá la última solución encontrada.

Antes de procesar la posición de los grupos de textos que nos ha devuelto el módulo, la función `cut_white_space_edges` comprueba la posición de los grupos y elimina el posible espacio sobrante en alguno de los bordes (Figura 4.14)

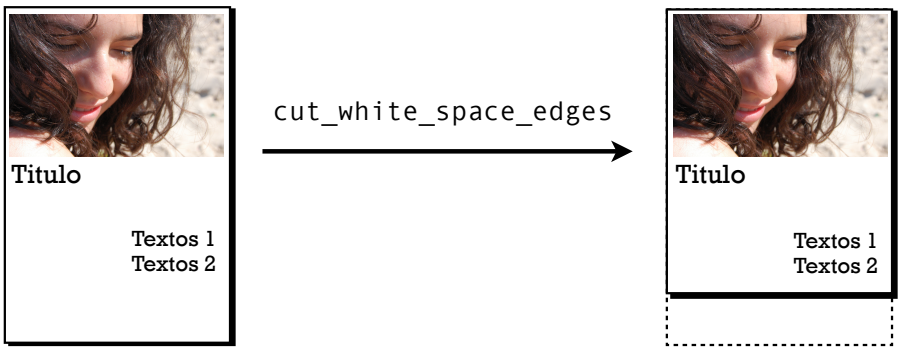

Figura 4.14: Esta función elimina el espacio blanco restante en los bordes.

Por último, un bucle recorre los grupos del diseño para generar la posición de los items de cada grupo. La función a la que llama cada grupo es `process_texts_position`; la posición x de cada item será la misma que la del grupo, mientras que la del eje y irá variando respecto a la altura de cada item. Como se vio en el calculó del tamaño de los grupos, cada grupo guarda un array con la altura de todos sus elementos, por lo que simplemente va sumándolas.

4.4. Etapa de Composición

Durante esta etapa se recogen todos los objetos generados en la anterior etapa y se procesan para crear el archivo de salida al usuario. Por tanto, el único objetivo de esta etapa es procesar los datos creados durante las anteriores etapas, crear un archivo de salida para el usuario y devolver al proceso principal la ruta de este archivo.

Los objetos generados anteriormente tienen la suficiente información como para poder ser enviados a distintos compositores, por lo que esta etapa del proceso es totalmente independiente y pueden usarse distintas herramientas para la generación de los carteles.

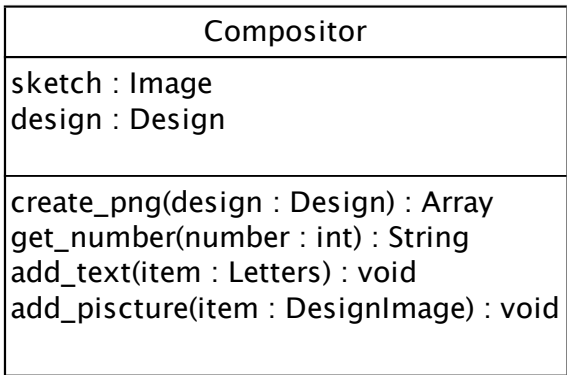

Figura 4.15: Clase `Compositor`.

En una primera versión de Gaudii, el compositor se implementó utilizando Processing, un lenguaje de programación de código abierto basado en Java. Debido a problemas de eficiencia, en la versión final de Gaudii se ha implementado utilizando ImageMagick y su interfaz para Ruby, RMagick. En cualquier caso, y como se ha dicho ya, cualquier compositor podría utilizarse.

El método principal de la clase `Compositor` (Figura 4.15) es `create_png`, que recibe el objeto `Design` y procesa la información para crear el archivo de imagen del diseño. El pseudocódigo del proceso completo puede verse en el Algoritmo 8.

Primero se crea un objeto de la clase Image, de la librería de ImageMagick, especificándole las dimensiones que se establecieron en la generación de los elementos del diseño. En este objeto, la variable `sketch`, insertaremos las imágenes y textos generados. El primero de es-

Algoritmo 8 Compositor

```
1: sketch = nuevaImagen(Design.imagenPrincipal)
2: insertarImagenPrincipal
3: para grupo en Grupos hacer
4:     para item en grupo hacer
5:         si item es Texto entonces
6:             insertarImagen(item)
7:         fin si
8:     fin para
9:     si grupo tiene imagen entonces
10:        insertarImagen(grupo.imagen)
11:    fin si
12: fin para
```

tos elementos será la imagen principal, usando el método `add_picture` que recibe el objeto `DesignImage` en el que se almacenan las dimensiones finales de la imagen, su posición y la ruta.

Después, una serie de bucles recorren todos los grupos de textos y, a su vez, todos los elementos de texto e imagen restantes. Las imágenes usan el mismo método que la imagen principal, mientras que los textos llaman al método `add_text`. Este método funciona de manera similar al de la imagen, salvo que la creación de la fuente es algo más compleja debido a que cuenta con un mayor número de variables que la definen: grosor, color, fondo, borde y alineación, además de la posición y el tamaño. Todas se crean con la clase `Draw` y el método `annotate` de ImageMagick.

Después de insertar todos los elementos en el diseño, se crea un objeto de salida. ImageMagick tiene un amplio soporte de formatos de imagen. En la versión final de Gaudii se ha optado por el formato PNG en detrimento del más común JPEG por diversas razones:

- El formato PNG es, al contrario que el JPEG, un estándar libre.
- La compresión del PNG se produce sin perdida de calidad en el resultado, mientras que la compresión del JPEG es mayor y produce cierta perdida en las zonas de contraste

alto entre colores. En Gaudii, muchos diseños tienen un fondo de un color plano sobre el que se colocan los textos, por lo que en JPEG los bordes sufrirían perdida de calidad, mientras que en PNG no.

- El formato PNG tiene soporte para el canal Alfa de transparencias. Si bien es cierto que Gaudii no crea actualmente ningún elemento con zonas de transparencia, podría implementarse en un futuro.

Por último, se crean dos versiones del archivo: uno a alta resolución para la descarga por parte del usuario y otro más pequeño para mostrarlo en la interfaz web junto al resto de diseños creados. Como se ha explicado, en esta etapa se acaba devolviendo al controlador un array que contiene dos cadenas de texto con las rutas de estos ficheros.

4.5. Uso posterior de las estructuras de los diseños

El módulo de creación de elementos no entra dentro de las etapas principales del diseño. El objetivo principal de este módulo es el de permitir al usuario obtener estructuras del diseño que sean de su gusto para poder usarlas en futuros diseños.

La estructura del diseño se descarga en un archivo, en XML, que contiene l información sobre el cromosoma, las fuentes utilizadas y los colores. Este archivo puede posteriormente ser subido en la pantalla de generación de diseños para crear otros diseños con el mismo estilo del que nos descargamos. Es decir, utilizar los mismos colores, fuentes y demás características de un cartel en otro completamente distinto.

En el Anexo del presente documento puede encontrarse un ejemplo de la estructura de un diseño en formato XML.

Ademas del cromosoma, se recoge la información de las fuentes y de los colores porque estos pudieron crearse respecto al diseño original, pero se desconoce si éstos funcionaran con el nuevo diseño. Esto es principalmente debido a que los colores se obtienen, por lo general, de la imagen del diseño, al igual que la forma de las letras puede depender de la forma predominante en la imagen.

Por eso mismo, Gaudii crea tres diseños con el archivo XML. En los tres se usa el mismo

array genético, pero solo en uno se utilizan los mismos colores y la misma fuente. En los otros dos, en uno se utiliza los mismos colores y en otro se repite la fuente. De esta forma no se limita al nuevo diseño con patrones creados para el original.

Estos nuevos diseños también pueden ser utilizados en las mezclas genéticas.

Capítulo 5

Resultados obtenidos

Este capítulo es una muestra de los resultados obtenidos por Gaudii en la generación y mezcla de diseños. En las pruebas se han utilizado distintos parámetros de entrada de texto para ver el desempeño del sistema con distintos números de elementos.

Las secciones en las que se divide este capítulo son:

- **Generación de diseños:** Se muestran ejemplos de varios resultados generados a partir de distintas cantidades de textos e imágenes. Estos resultados son generados por la plataforma desde cero.
- **Mezcla de diseños:** Se muestran resultados de Gaudii en lo referente a la mezcla de diseños. Se presentarán los diseños previos a la mezcla y el resultado final de ésta.
- **Tiempos por Etapas:** En esta sección se muestran los tiempo para cada una de las etapas del proceso del sistema.
- **Tiempos de ejecución de las versiones Ruby y C:** Por último, en esta sección se muestran los tiempos del proceso de Enfriamiento Simulado en C y Ruby, así como las razones que motivaron el cambio al lenguaje C.

Todas las pruebas se han realizado utilizando un servidor Mongrel, en un equipo con un procesador Intel Core 2 Duo a 2GHz y 4GB de RAM. Se han utilizado además las versiones 1.8.7 de Ruby y la 2.3.2 de Ruby On Rails.

5.1. Generación de diseños

Ejemplo 1

El primer ejemplo es el anuncio de un curso de cocina (Figura 5.1). Este cartel tiene tres grupos de textos y una imagen de fondo, cuyo punto de interés es el montón de galletas.

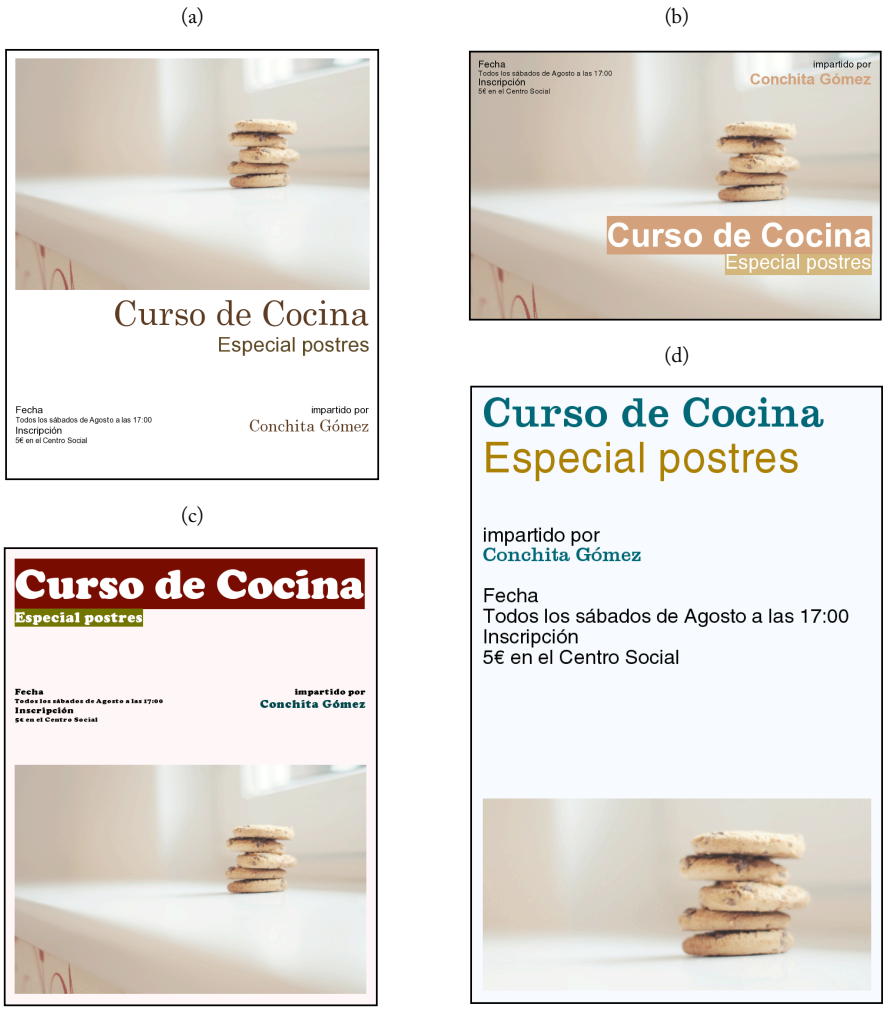

Figura 5.1: Ejemplo 1 de generación de diseños.

La distribución de los textos ha sido la siguiente:

- **Grupo 1**
  - *Curso de cocina* - Título

- *Especial postres* - Subtítulo

- **Grupo 2**
  - *impartido por* - Texto normal
  - *Conchita Gómez* - Texto Destacado

- **Grupo 3**
  - *Fecha* - Texto normal
  - *Todos los sábados de Agosto a las 17:00* - Notas
  - *Inscripción* - Texto normal
  - *5e en el Centro Social* - Notas

Se puede observar cómo cada ejemplo toma diferentes tonos de colores dependiendo del bit genético. Los diseños *(a)* y *(b)* siguen unos colores que no contrastan con la imagen, dando una sensación de repetición y uniformidad; por contra, los diseños *(c)* y *(d)* ofrecen colores que contrastan más, ofreciendo imágenes con mayor impacto visual.

Los diseños *(b)* y *(c)* muestran el tipo de decoración en los textos en el que se añade un fondo de color.

La alineación de los elementos ha tendido siempre a colocarse en alguno de los márgenes del diseño, en lo referente al eje X; en los ejemplos *(a)*, *(b)* y *(c)* parte de los textos se ha alineado entre sí respecto al eje Y.

Ejemplo 2

Este segundo ejemplo anuncia un simposio. El diseño presenta solo dos grupos de textos, uno de ellos con una imagen adjunta. En la imagen de fondo, el punto de interés es la vaca.

La distribución de los textos ha sido la siguiente:

- **Grupo 1**
  - *La población vacuna* - Título
  - *¿en peligro de extinción?* - Subtítulo

Figura 5.2: Ejemplo 2 de generación de diseños.

- **Grupo 2**
  - *Simposio a cargo del Dr. Stevenson* - Texto normal
  - *prestigioso biólogo* - Texto Destacado
  - Imagen asociada.

En este grupo de ejemplos se muestra como se pueden añadir imágenes asociadas a un determinado grupo de texto. En este caso, la imagen del doctor responsable del simposio. Esta se coloca a un lado del texto (dependiendo de la alineación).

El diseño *(a)* muestra un ejemplo del zoom a la imagen original; de esta forma se ha hecho hincapié en el punto de interés de la imagen.

Por último, en el diseño *(c)* se ve el otro tipo de decoración de textos, el borde coloreado, en el título del diseño.

5.2. Mezcla de diseños

La Figura 5.3 muestra un ejemplo de la mezcla de diseños. Se ha utilizado un caso con dos grupos de textos; el punto de interés de la imagen ha sido marcado en el hocico del cerdo.

Algunos detalles a destacar de este ejemplo:

- El padre *(c)* y el hijo *(i)* son prácticamente idénticos en tipografía y tamaño de documento; simplemente difieren en la decoración de los textos y, ligeramente, en el esquema de colores. En el caso del diseño *(g)*, es practicamente idéntico salvo en el tipo de letra.

- Los 3 padres tiene orientación vertical y de proporciones cercanas al 4:3. Todos los hijos se mantienen vertical y con unas proporciones similares (salvo en el caso del hijo *(d)*, que se aleja un poco de esa proporción).

- El padre *(b)* utiliza un contraste muy alto entre tamaños de fuentes, algo que ha heredado el hijo *(e)*.

- Los esquemas de colores en los padres eran prácticamente de repetición, con ligeros contrastes en los diseños *(a)* y *(b)*. La mayoría de los hijos ha seguido este patrón, salvo en los casos de los hijos *(d)* y *(h)*, en los que el esquema de colores es diferente (probablemente por la mutación).

En general, se puede observar como los hijos heredan características de uno u otro padre, salvo algunos casos por causa de la mutación.

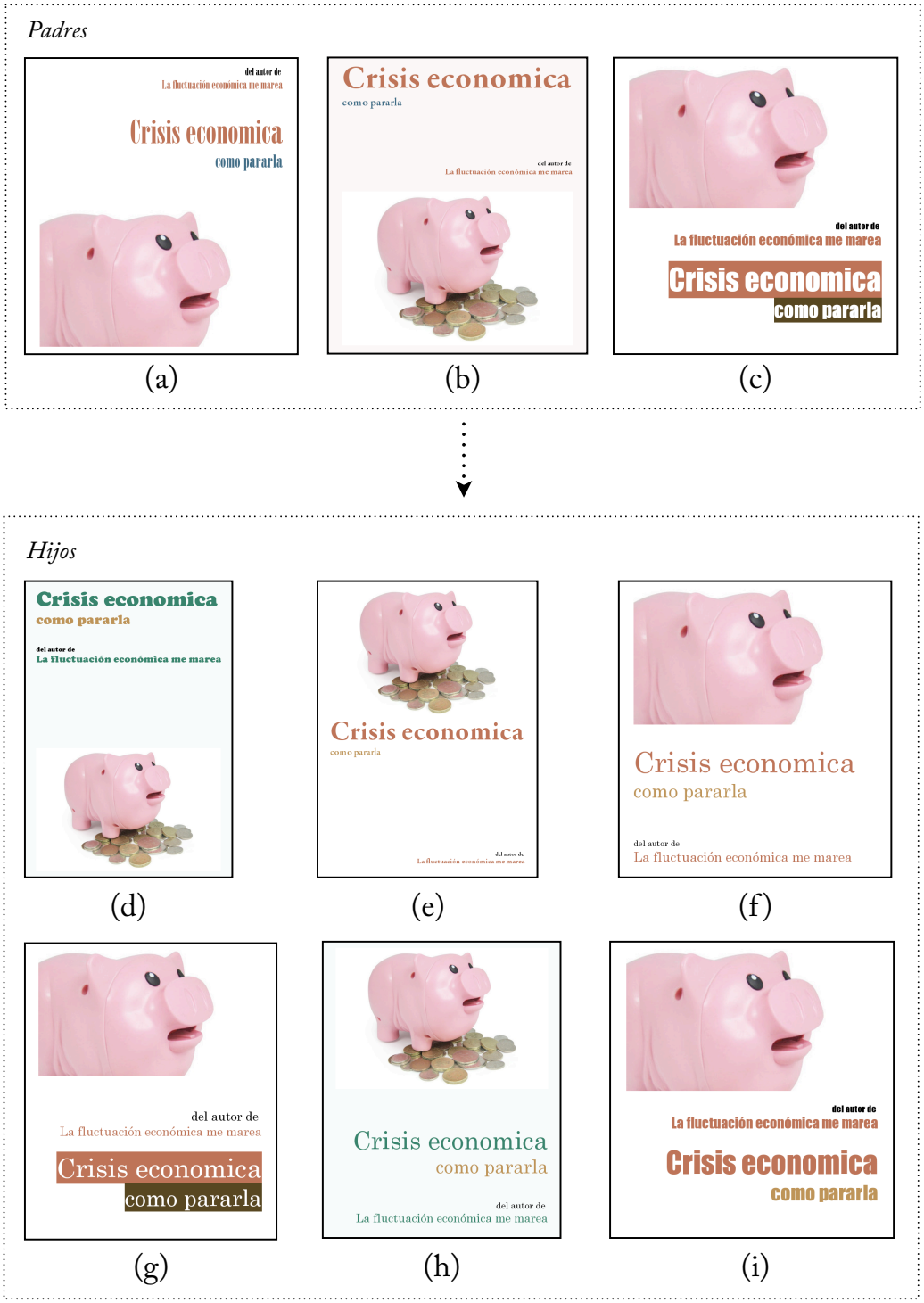

Figura 5.3: Ejemplo de mezcla de diseños.

5.3. Tiempos por etapas

En esta sección mostraremos algunos ejemplos de los tiempos que dedica el sistema a cada etapa de procesamiento.

Ejemplo 1

Este ejemplo se realiza con 2 grupos de elementos, con un total de 5 elementos de texto.

- Etapa de Producción Genética: 0.000345 s.
- Etapa de Uso del Modelo Experto: 2.01782 s.
- Etapa de Composición: 0.901639 s.

Ejemplo 2

Este ejemplo se realiza con 3 grupos de elementos, con un total de 7 elementos de texto.

- Etapa de Producción Genética: 0.000447 s.
- Etapa de Uso del Modelo Experto: 2.328836 s.
- Etapa de Composición: 0.902303 s.

Ejemplo 3

Este ejemplo se realiza con 4 grupos de elementos, con un total de 10 elementos de texto y 1 imagen asociada a uno de los grupos.

- Etapa de Producción Genética: 0.000337 s.
- Etapa de Uso del Modelo Experto: 2.708623 s.
- Etapa de Composición: 1.213649 s.

5.4. Tiempos de ejecución de las versiones Ruby y C

La primera versión del algoritmo de Enfriamiento Simulado dedicado a la ordenación de elementos se desarrolló en Ruby. Debido a que es un lenguaje interpretado (como Python, Perl

o Lisp), Ruby no puede ofrecer el mismo rendimiento que los lenguajes compilados como C. Por este motivo, esta primera versión del algoritmo ofrecía un rendimiento pobre.

La versión en Ruby no contaba con la función de comprobar el espacio disponible a ciertas temperaturas (`checkSpace`) que luego se implementaría en la versión en C, la temperatura iba desde 100.0º hasta 60.0º, y el bucle interior (determinado por la función `L`) alcanzaba un máximo de 40 iteraciones.

En la versión final en C, la temperatura comienza en 300.0º y termina en 10.0º, con un máximo de 100 iteraciones en el bucle interior. Lo siguiente son tiempos de ejecución de Gaudii para crear nueve diseños, tanto con la versión en Ruby como en C:

- **Diseño con 2 grupos de textos**
  - Ruby: 57,51 segundos (6,39 segundos por diseño)
  - C: 28,28 segundos (3,14 segundos por diseño)
- **Diseño con 3 grupos de textos**
  - Ruby: 72,42 segundos (8,04 segundos por diseño)
  - C: 34,22 segundos (3,83 segundos por diseño)
- **Diseño con 4 grupos de textos**
  - Ruby: 96,63 segundos (10,73 segundos por diseño)
  - C: 35,62 segundos (3,95 segundos por diseño)

Como se puede apreciar, el código en C ofrece mayor rendimiento y, especialmente, mejor escalabilidad, todo pese a realizar más iteraciones y contar con funciones no implementadas en la prematura versión de Ruby.

Capítulo 6

Conclusiones y propuestas

6.1. Conclusiones

En esta sección se comenta en que medida se han alcanzado los objetivos propuesto al inicio del desarrollo:

- **Modelado de reglas de composición gráfica. La colocación de los elementos en el diseño dará como resultado una creación artística, y por tanto debe realizarse empleando reglas de composición gráfica. El sistema deberá permitir el trabajo con altos niveles de incertidumbre en el modelado de cualquier conocimiento experto en disciplinas artísticas.** La base de Gaudii es una motor de inferencia difusa y un conjunto de reglas que evalúan los bits genéticos de cada diseño. Este sistema de reglas puede ampliarse fácilmente para tratar más aspecto que no se han incorporado a Gaudii, y permite trabajar con altos niveles de incertidumbre. Se han utilizado numerosas fuentes bibliográficas para la adquisición del conocimiento experto, de forma que pueda asegurarse en gran medida la universalidad de las reglas creadas para composición gráfica, así como su fidelidad a los conceptos básicos con los que trabajan los diseñadores gráficos.

- **Automatización en el proceso de creación. Se debe minimizar la intervención humana en el proceso de generación de diseños. El usuario proporcionará textos y alguna imagen que quiera utilizar para el diseño. El sistema deberá ser capaz de**

combinar esta información utilizando las reglas de composición gráfica definidas anteriormente. El sistema de generación y composición de diseños es completamente automático y transparente al usuario; éste solo debe introducir los textos e imágenes. Además, Gaudii incorpora varios módulos de visión por computador para detectar puntos de interés en las imágenes y automatizar lo máximo posible este proceso. Aun así, el usuario puede especificar por sí mismo cual es el punto de interés que quiere.

- **Adaptabilidad al gusto particular de cada usuario. La plataforma debe permitir al usuario guiar el proceso de generación de obras, adaptándose a sus gustos generando obras de acuerdo a sus preferencias. Esta información podrá ser usada por el sistema para obtener futuras obras para un mismo usuario.** El sistema permite al usuario organizar sus textos e imágenes en los grupos que más considere convenientes. Además, al principio se le pregunta varias cuestiones con el fin de generar un diseño más acorde a sus gustos (qué tipo de colores usar, qué tipo de fuentes usar, etc.). Por último, en la mezcla genética podrá guiar al sistema en el tipo de obras que más le han gustado.

- **Diversidad y heterogeneidad. El sistema debe ser capaz de generar distintas obras que serán presentadas al usuario. Éste podrá elegir entre las alternativas que se le presentan o guiar al sistema en la creación de otras parecidas a un subconjunto de éstas.** Todos los diseños se crean a partir de una cadena genética generada aleatoriamente, y esta representa, en cada bit, características propias de este diseño. Dado que se genera aleatoriamente, y que el número de bits representativos es considerable, se puede afirmar que el sistema es capaz de generar diseños muy diversos sobre una misma obra. De la misma manera, el sistema permite al usuario mezclar genéticamente varios diseños a su gusto; de esta forma el usuario puede ir combinando diseños de su interés hasta obtener el que más se ajuste a sus gustos. Por último, el sistema permite descargar un XML con la información genética del proyecto que podrá utilizarse en posteriores diseños.

- **Interfaz multidispositivo. El sistema podrá ser usado mediante un entorno web sin la necesidad de ningún tipo de instalación en el ordenador o dispositivo del**

usuario. Se emplearán mecanismos de interacción avanzados para evitar la recarga del interfaz en primer plano mediante tecnologías de petición asíncrona. Gaudii está completamente desarrollado en un entorno web; utiliza además diversas tecnologías asíncronas para asemejar la experiencia web con la de una aplicación de escritorio, como entornos *drag&drop* o peticiones remotas de zonas aisladas de la página sin necesidad de recargar esta por completo.

- **Sistema multiplataforma. El sistema debe desarrollarse tal que pueda ser utilizado en distintos sistemas operativos y distinto hardware. Para conseguir este aspecto se desarrollará, como se ha explicando anteriormente, sobre una interfaz web.** El sistema se ha desarrollado en su totalidad en una plataforma web, por lo que se asegura su multiplataformidad.

- **Desarrollar del sistema empleando estándares y software libre. Para facilitar su explotación y posterior ampliación y mantenimiento, el sistema se desarrollará empleando estándares y software libre.** Todo el software desarrollado durante el proyecto esta amparado bajo una licencia GPL. Así mismo, toda la tecnología utilizada está publicada bajo alguna licencia de código abierto reconocida por la *Open Source Iniciative* [6].

6.2. Lineas de investigación abiertas

Debido a que Gaudii es la primera aproximación a la generación automática de diseños con base a reglas de composición gráfica, hay muchos aspectos del sistema que podrían ser investigados con mayor profundidad. Aquí se presentan algunas de las ideas que podrían mejorar el sistema y dotarlo de una mayor profundidad.

6.2.1. Sistema de usuarios

La implementación de un sistema de usuarios dotaría al sistema de un grado más de *inteligencia*, puesto que podría ser capaz de reconocer a los usuarios de pasadas creaciones y crear diseños basados en sus gustos.

En concreto, el sistema podría implementar un sistema en el que el usuario puntúe los diseños en un determinado baremo. Estas puntuaciones se almacenarían en una base de datos de usuarios; Gaudii accedería a ésta para un determinado usuario y evaluaría los gustos del usuarios basándose en las cadenas puntuadas.

Por ejemplo, se podría evaluar como un usuario prefiere en numerosos diseños que exista alto contraste entre el tamaño de las fuentes, pero como, de la misma manera, rechaza repetidamente diseños en los que se usan colores demasiado llamativos. La evaluación sería bit a bit para cada uno de los diseños almacenados en el sistema, ponderando la cantidad de veces que aparece ese bit con la respectiva nota del diseño.

Alternativamente, esta gran base de datos de puntuaciones de los usuarios podrían usarse para hacer que Gaudii aprendiese de los gustos más populares. De esta forma, cualquier usuario podría solicitar a Gaudii que crease un diseño basándose en lo que ha aprendido del resto de usuarios.

Estas dos opciones se mostrarían al usuario al principio de la creación del diseño, de forma que pueda crear desde cero con las preguntas ya implementadas, basándose en los gustos propios o utilizando los globales. Este sistema dotaría a Gaudii de mayor profundidad y versatilidad.

6.2.2. Uso de descriptores visuales

Actualmente, Gaudii solo tiene en cuenta el punto de interés de la imagen a la hora de componer los elementos alrededor de ella. Es un método efectivo, pero que cuenta con ciertas limitaciones en imágenes en las que el punto de interés no está del todo claro.

Los descriptores visuales se utilizan para describir las características visuales de un determinado video o imagen. Por ejemplo, la forma, el color o la textura predominante en una imagen.

Estos descriptores se clasifican en dos tipos:

- **Información general:** Estos son descriptores de bajo nivel y proporcionan información sobre aspectos generales como el color, la forma, la textura o el movimiento.

- **Dominio específico:** Estos descriptores ofrecen información más detallada de los objetos y eventos que aparecen en la escena. Por ejemplo, podría utilizarse para diferenciar si una fotografía esta tomada en la naturaleza o en la ciudad.

En el caso de Gaudii, la información sobre la forma de la imagen introducida por el usuario sería de gran utilidad. Se podría entrenar al sistema con una serie de imágenes con formas bien definidas, como se puede ver en la Figura 6.1.

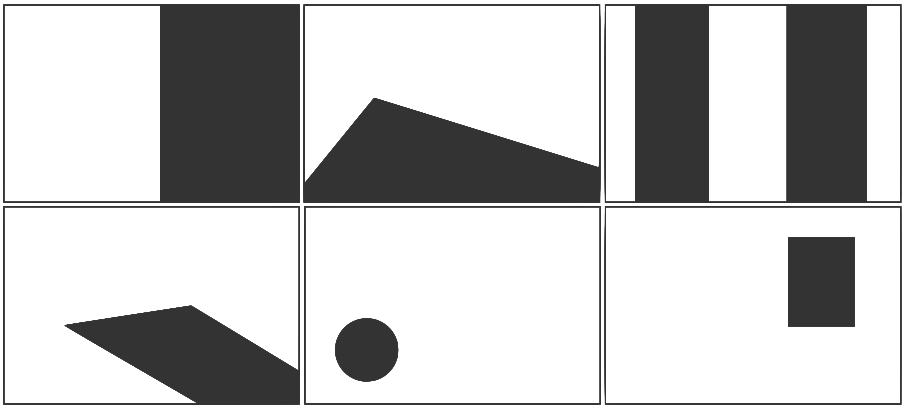

Figura 6.1: Ejemplos de descriptores de imagen

Luego, cuando el usuario introdujera una imagen, esta se evaluaría en función de todos los descriptores (Figura 6.2). Esta evaluación nos daría la forma predominante en la imagen, y de esta manera el sistema ganaría en flexibilidad en la parte de la composición. Por ejemplo, actualmente no se aprovecha si alguna de las formas en la imagen es diagonal para colocar el texto en paralelo a esta; en la imagen de la montaña (Figura 6.2), el título podría hacerlo.

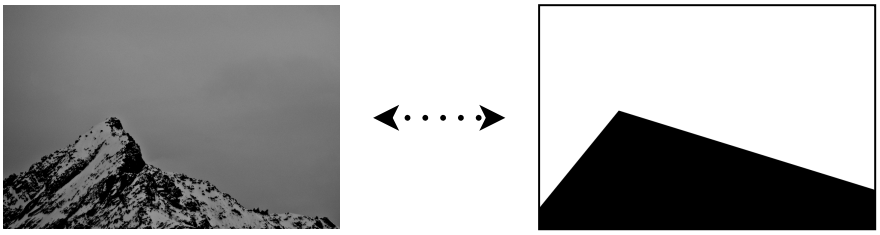

Figura 6.2: Imagen asociada a un descriptor de imagen

6.2.3. Mayor intervención en la creación

Uno de los objetivos de Gaudii era evitar, en la medida de lo posible, la intervención del usuario en el proceso de generación y composición de elementos. Este propósito tiene en cuenta que son muchos los usuarios que no saben o no quieren participar en esta tarea, pero también es cierto que hay otros que necesitarían aportar su parte.

La más idea básica sería ampliar las preguntas iniciales hacia un control más avanzado del diseño, donde se puedan elegir otros aspectos importante como la colocación de la imagen, el esquema de colores o el tamaño máximo de las fuentes. Esta elección solo supondría insertar ciertos elementos en el objeto `Design` y en el cromosoma, en lugar de generarlos u obtenerlos aleatoriamente.

Partiendo de esa idea básica, otros controles mas avanzados podrían permitir al usuario especificar las dimensiones y colocación de ciertos elementos en el diseño del cartel. Por ejemplo, muchas empresas tienen una imagen corporativa clara y específica. No solo basta con elegir ciertos colores, sino que además los elementos deben colocarse en una determinada posición, con una determinada fuente y delante de un determinado color de fondo.

Esto no vendría a sustituir a las herramientas actuales de composición, puesto que solo una parte se especificaría por el usuario y el resto sería siendo una composición generada por la máquina, y además sería útil para otros ámbitos más allá del usuario casual.

Apéndice A

Código fuente

Dada la extensión del código, más de 5200 líneas, en este anexo solo se muestran los módulos más relevantes.

A.1. Conjunto de Reglas

```
<?xml version="1.0"?>
<FIE>
 <system name="doc_shape">
  <linguisticvariable name="shape_bit" type="input">
         <fuzzyset tag="low" a="0" b="0" c="2" d="4" />
         <fuzzyset tag="medium" a="1" b="3" c="6" d="8" />
         <fuzzyset tag="high" a="5" b="7" c="9" d="9" />
     </linguisticvariable>
  <linguisticvariable name="design_shape" type="output">
   <fuzzyset tag="squared" a="0" b="0" c="2" d="4" />
   <fuzzyset tag="standard" a="1" b="3" c="6" d="8" />
   <fuzzyset tag="wide" a="4" b="7" c="9" d="9" />
     </linguisticvariable>
  <rule name="R1">
         <antecedent varname="shape_bit" bit_position="0" tag="low" />
         <consequent varname="design_shape" bit_position="0" tag="squared"
             />
     </rule>
  <rule name="R2">
         <antecedent varname="shape_bit" bit_position="0" tag="medium" />
         <consequent varname="design_shape" bit_position="0" tag="standard
            " />
     </rule>
  <rule name="R3">
         <antecedent varname="shape_bit" bit_position="0" tag="high" />
         <consequent varname="design_shape" bit_position="0" tag="wide" />
     </rule>
 </system>
 <system name="doc_orientation">
  <linguisticvariable name="orientation_bit" type="input">
```

```
29          <fuzzyset tag="low" a="0" b="0" c="4" d="6" />
30          <fuzzyset tag="high" a="3" b="5" c="9" d="9" />
31       </linguisticvariable>
32    <linguisticvariable name="design_orientation" type="output">
33     <fuzzyset tag="horizontal" a="0" b="0" c="4" d="5" />
34     <fuzzyset tag="vertical" a="5" b="6" c="9" d="9" />
35       </linguisticvariable>
36    <rule name="R4">
37          <antecedent varname="orientation_bit" bit_position="1" tag="low"
               />
38     <consequent varname="design_orientation" bit_position="1" tag="
          horizontal" />
39       </rule>
40    <rule name="R5">
41          <antecedent varname="orientation_bit" bit_position="1" tag="high"
                />
42          <consequent varname="design_orientation" bit_position="1" tag="
               vertical" />
43       </rule>
44   </system>
45   <system name="doc_font_face_type">
46    <linguisticvariable name="font_face_type_bit" type="input">
47          <fuzzyset tag="low" a="0" b="0" c="3" d="5" />
48          <fuzzyset tag="medium" a="2" b="4" c="5" d="7" />
49          <fuzzyset tag="high" a="4" b="6" c="7" d="9" />
50       </linguisticvariable>
51    <linguisticvariable name="design_font_face_type" type="output">
52     <fuzzyset tag="graphic" a="0" b="0" c="2" d="4" />
53     <fuzzyset tag="script" a="1" b="3" c="6" d="8" />
54     <fuzzyset tag="others" a="5" b="7" c="9" d="9" />
55       </linguisticvariable>
56    <rule name="R11">
57          <antecedent varname="font_face_type_bit" bit_position="3" tag="
               low" />
58          <consequent varname="design_font_face_type" bit_position="3" tag=
               "graphic" />
59       </rule>
60    <rule name="R12">
61          <antecedent varname="font_face_type_bit" bit_position="3" tag="
               medium" />
62          <consequent varname="design_font_face_type" bit_position="3" tag=
               "script" />
63       </rule>
64    <rule name="R13">
65          <antecedent varname="font_face_type_bit" bit_position="3" tag="
               high" />
66          <consequent varname="design_font_face_type" bit_position="3" tag=
               "others" />
67       </rule>
68   </system>
69   <system name="doc_use_of_colors">
70    <linguisticvariable name="use_of_colors_bit" type="input">
71     <fuzzyset tag="low" a="0" b="0" c="1" d="3" />
72          <fuzzyset tag="medium" a="1" b="2" c="7" d="8" />
73          <fuzzyset tag="high" a="7" b="8" c="9" d="9" />
74       </linguisticvariable>
75    <linguisticvariable name="design_design_use_of_colors" type="output">
76     <fuzzyset tag="no_colors" a="0" b="0" c="1" d="2" />
77     <fuzzyset tag="normal_use" a="2" b="3" c="7" d="8" />
78     <fuzzyset tag="high_use" a="7" b="8" c="9" d="9" />
```

```
79        </linguisticvariable>
80     <rule name="R14">
81            <antecedent varname="use_of_colors_bit" bit_position="21" tag="
                 low" />
82            <consequent varname="design_design_use_of_colors" bit_position="
                 21" tag="no_colors" />
83        </rule>
84     <rule name="R15">
85            <antecedent varname="use_of_colors_bit" bit_position="21" tag="
                 medium" />
86            <consequent varname="design_design_use_of_colors" bit_position="
                 21" tag="normal_use" />
87        </rule>
88     <rule name="R16">
89            <antecedent varname="use_of_colors_bit" bit_position="21" tag="
                 high" />
90            <consequent varname="design_design_use_of_colors" bit_position="
                 21" tag="high_use" />
91        </rule>
92    </system>
93    <system name="doc_darkness">
94     <linguisticvariable name="darkness_bit" type="input">
95      <fuzzyset tag="low" a="0" b="0" c="4" d="6" />
96            <fuzzyset tag="high" a="3" b="5" c="9" d="9" />
97        </linguisticvariable>
98     <linguisticvariable name="design_darkness" type="output">
99      <fuzzyset tag="low" a="0" b="0" c="4" d="6" />
100           <fuzzyset tag="high" a="3" b="5" c="9" d="9" />
101       </linguisticvariable>
102    <rule name="R17">
103           <antecedent varname="darkness_bit" bit_position="24" tag="low" />
104           <consequent varname="design_darkness" bit_position="25" tag="low"
                 />
105       </rule>
106    <rule name="R18">
107           <antecedent varname="darkness_bit" bit_position="24" tag="high" /
                 >
108           <consequent varname="design_darkness" bit_position="25" tag="high
                 " />
109       </rule>
110   </system>
111   <system name="doc_bg">
112    <linguisticvariable name="picture_as_background" type="input">
113           <fuzzyset tag="yes" a="0" b="0" c="4" d="6" />
114     <fuzzyset tag="no" a="3" b="5" c="9" d="9" />
115       </linguisticvariable>
116    <linguisticvariable name="design_bg" type="output">
117     <fuzzyset tag="picture" a="0" b="0" c="4" d="6" />
118           <fuzzyset tag="plain" a="3" b="5" c="9" d="9" />
119       </linguisticvariable>
120    <rule name="R19">
121           <antecedent varname="picture_as_background" bit_position="23" tag
                 ="yes" />
122           <consequent varname="design_bg" bit_position="22" tag="picture" /
                 >
123       </rule>
124    <rule name="R20">
125           <antecedent varname="picture_as_background" bit_position="23" tag
                 ="no" />
126           <consequent varname="design_bg" bit_position="22" tag="plain" />
```

```
127       </rule>
128    </system>
129    <system name="contrast_repetition_height0">
130     <linguisticvariable name="contrast_size" type="input">
131            <fuzzyset tag="low" a="0" b="0" c="2" d="4" />
132            <fuzzyset tag="medium" a="1" b="3" c="6" d="8" />
133            <fuzzyset tag="high" a="5" b="7" c="9" d="9" />
134       </linguisticvariable>
135     <linguisticvariable name="repetition_size" type="input">
136            <fuzzyset tag="low" a="0" b="0" c="4" d="6" />
137            <fuzzyset tag="high" a="3" b="5" c="9" d="9" />
138       </linguisticvariable>
139     <linguisticvariable name="design_font_height0" type="output">
140      <fuzzyset tag="very_big"  a="64" b="72" c="80" d="82" />
141      <fuzzyset tag="huge"   a="90" b="102" c="108" d="120" />
142       </linguisticvariable>
143     <rule name="R21">
144            <antecedent varname="contrast_size" bit_position="4" tag="low" />
145      <antecedent varname="repetition_size" bit_position="12" tag="low" />
146            <consequent varname="design_font_height0" bit_position="4" tag="
                  very_big" />
147       </rule>
148     <rule name="R22">
149            <antecedent varname="contrast_size" bit_position="4" tag="low" />
150      <antecedent varname="repetition_size" bit_position="12" tag="high" />
151            <consequent varname="design_font_height0" bit_position="4" tag="
                  very_big" />
152       </rule>
153     <rule name="R23">
154            <antecedent varname="contrast_size" bit_position="4" tag="medium"
                   />
155      <antecedent varname="repetition_size" bit_position="12" tag="low" />
156            <consequent varname="design_font_height0" bit_position="4" tag="
                  very_big" />
157       </rule>
158     <rule name="R24">
159            <antecedent varname="contrast_size" bit_position="4" tag="medium"
                   />
160      <antecedent varname="repetition_size" bit_position="12" tag="high" />
161            <consequent varname="design_font_height0" bit_position="4" tag="
                  very_big" />
162       </rule>
163     <rule name="R25">
164            <antecedent varname="contrast_size" bit_position="4" tag="high" /
                  >
165      <antecedent varname="repetition_size" bit_position="12" tag="low" />
166            <consequent varname="design_font_height0" bit_position="4" tag="
                  huge" />
167       </rule>
168    </system>
169    <system name="contrast_repetition_height1">
170     <linguisticvariable name="contrast_size" type="input">
171            <fuzzyset tag="low" a="0" b="0" c="2" d="4" />
172            <fuzzyset tag="medium" a="1" b="3" c="6" d="8" />
173            <fuzzyset tag="high" a="5" b="7" c="9" d="9" />
174       </linguisticvariable>
175     <linguisticvariable name="repetition_size" type="input">
176            <fuzzyset tag="low" a="0" b="0" c="4" d="6" />
177            <fuzzyset tag="high" a="3" b="5" c="9" d="9" />
178       </linguisticvariable>
```

```
179    <linguisticvariable name="design_font_height1" type="output">
180     <fuzzyset tag="normal"   a="26"  b="32"  c="36"  d="42" />
181     <fuzzyset tag="big"   a="40" b="46" c="56" d="68" />
182     <fuzzyset tag="very_big"  a="64" b="72" c="80" d="82" />
183       </linguisticvariable>
184    <rule name="R26">
185           <antecedent varname="contrast_size" bit_position="4" tag="low" />
186     <antecedent varname="repetition_size" bit_position="12" tag="low" />
187           <consequent varname="design_font_height1" bit_position="5" tag="
                 big" />
188       </rule>
189    <rule name="R27">
190           <antecedent varname="contrast_size" bit_position="4" tag="low" />
191     <antecedent varname="repetition_size" bit_position="12" tag="high" />
192           <consequent varname="design_font_height1" bit_position="5" tag="
                 very_big" />
193       </rule>
194    <rule name="R28">
195           <antecedent varname="contrast_size" bit_position="4" tag="medium"
                  />
196     <antecedent varname="repetition_size" bit_position="12" tag="low" />
197           <consequent varname="design_font_height1" bit_position="5" tag="
                 normal" />
198       </rule>
199    <rule name="R29">
200           <antecedent varname="contrast_size" bit_position="4" tag="medium"
                  />
201     <antecedent varname="repetition_size" bit_position="12" tag="high" />
202           <consequent varname="design_font_height1" bit_position="5" tag="
                 big" />
203       </rule>
204    <rule name="R30">
205           <antecedent varname="contrast_size" bit_position="4" tag="high" /
                 >
206     <antecedent varname="repetition_size" bit_position="12" tag="low" />
207           <consequent varname="design_font_height1" bit_position="5" tag="
                 normal" />
208       </rule>
209   </system>
210   <system name="contrast_repetition_height2">
211    <linguisticvariable name="contrast_size" type="input">
212           <fuzzyset tag="low" a="0" b="0" c="2" d="4" />
213           <fuzzyset tag="medium" a="1" b="3" c="6" d="8" />
214           <fuzzyset tag="high" a="5" b="7" c="9" d="9" />
215       </linguisticvariable>
216    <linguisticvariable name="repetition_size" type="input">
217           <fuzzyset tag="low" a="0" b="0" c="4" d="6" />
218           <fuzzyset tag="high" a="3" b="5" c="9" d="9" />
219       </linguisticvariable>
220    <linguisticvariable name="design_font_height2" type="output">
221     <fuzzyset tag="very_little" a="12"  b="12"  c="16"  d="20" />
222     <fuzzyset tag="little"   a="14"  b="18"  c="24"  d="28" />
223     <fuzzyset tag="normal"   a="26"  b="32"  c="36"  d="42" />
224       </linguisticvariable>
225    <rule name="R31">
226           <antecedent varname="contrast_size" bit_position="4" tag="low" />
227     <antecedent varname="repetition_size" bit_position="12" tag="low" />
228           <consequent varname="design_font_height2" bit_position="6" tag="
                 normal" />
229       </rule>
```

```
230     <rule name="R32">
231           <antecedent varname="contrast_size" bit_position="4" tag="low" />
232      <antecedent varname="repetition_size" bit_position="12" tag="high" />
233           <consequent varname="design_font_height2" bit_position="6" tag="
                 normal" />
234        </rule>
235     <rule name="R33">
236           <antecedent varname="contrast_size" bit_position="4" tag="medium"
                  />
237      <antecedent varname="repetition_size" bit_position="12" tag="low" />
238           <consequent varname="design_font_height2" bit_position="6" tag="
                 normal" />
239        </rule>
240     <rule name="R34">
241           <antecedent varname="contrast_size" bit_position="4" tag="medium"
                  />
242      <antecedent varname="repetition_size" bit_position="12" tag="high" />
243           <consequent varname="design_font_height2" bit_position="6" tag="
                 normal" />
244        </rule>
245     <rule name="R35">
246           <antecedent varname="contrast_size" bit_position="4" tag="high" /
                 >
247      <antecedent varname="repetition_size" bit_position="12" tag="low" />
248           <consequent varname="design_font_height2" bit_position="6" tag="
                 little" />
249        </rule>
250   </system>
251   <system name="contrast_repetition_height3">
252    <linguisticvariable name="contrast_size" type="input">
253           <fuzzyset tag="low" a="0" b="0" c="2" d="4" />
254           <fuzzyset tag="medium" a="1" b="3" c="6" d="8" />
255           <fuzzyset tag="high" a="5" b="7" c="9" d="9" />
256        </linguisticvariable>
257    <linguisticvariable name="repetition_size" type="input">
258           <fuzzyset tag="low" a="0" b="0" c="4" d="6" />
259           <fuzzyset tag="high" a="3" b="5" c="9" d="9" />
260        </linguisticvariable>
261    <linguisticvariable name="design_font_height3" type="output">
262     <fuzzyset tag="very_little" a="12"  b="12"  c="16"  d="20" />
263     <fuzzyset tag="little"   a="14"  b="18"  c="24"  d="28" />
264     <fuzzyset tag="normal"   a="26"  b="32"  c="36"  d="42" />
265        </linguisticvariable>
266    <rule name="R36">
267           <antecedent varname="contrast_size" bit_position="4" tag="low" />
268     <antecedent varname="repetition_size" bit_position="12" tag="low" />
269           <consequent varname="design_font_height3" bit_position="7" tag="
                 little" />
270        </rule>
271    <rule name="R37">
272           <antecedent varname="contrast_size" bit_position="4" tag="low" />
273     <antecedent varname="repetition_size" bit_position="12" tag="high" />
274           <consequent varname="design_font_height3" bit_position="7" tag="
                 normal" />
275        </rule>
276    <rule name="R38">
277           <antecedent varname="contrast_size" bit_position="4" tag="medium"
                  />
278     <antecedent varname="repetition_size" bit_position="12" tag="low" />
```

```
279           <consequent varname="design_font_height3" bit_position="7" tag="
                 little" />
280       </rule>
281    <rule name="R39">
282           <antecedent varname="contrast_size" bit_position="4" tag="medium"
                  />
283     <antecedent varname="repetition_size" bit_position="12" tag="high" />
284           <consequent varname="design_font_height3" bit_position="7" tag="
                 normal" />
285       </rule>
286    <rule name="R40">
287           <antecedent varname="contrast_size" bit_position="4" tag="high" /
                 >
288     <antecedent varname="repetition_size" bit_position="12" tag="low" />
289           <consequent varname="design_font_height3" bit_position="7" tag="
                 little" />
290       </rule>
291   </system>
292   <system name="contrast_repetition_height4">
293    <linguisticvariable name="contrast_size" type="input">
294           <fuzzyset tag="low" a="0" b="0" c="2" d="4" />
295           <fuzzyset tag="medium" a="1" b="3" c="6" d="8" />
296           <fuzzyset tag="high" a="5" b="7" c="9" d="9" />
297       </linguisticvariable>
298    <linguisticvariable name="repetition_size" type="input">
299           <fuzzyset tag="low" a="0" b="0" c="4" d="6" />
300           <fuzzyset tag="high" a="3" b="5" c="9" d="9" />
301       </linguisticvariable>
302    <linguisticvariable name="design_font_height4" type="output">
303     <fuzzyset tag="very_little" a="12"  b="12"  c="16"  d="20" />
304     <fuzzyset tag="little"   a="14"  b="18"  c="24"  d="28" />
305     <fuzzyset tag="normal"   a="26"  b="32"  c="36"  d="42" />
306       </linguisticvariable>
307    <rule name="R41">
308           <antecedent varname="contrast_size" bit_position="4" tag="low" />
309     <antecedent varname="repetition_size" bit_position="12" tag="low" />
310           <consequent varname="design_font_height4" bit_position="8" tag="
                 very_little" />
311       </rule>
312    <rule name="R42">
313           <antecedent varname="contrast_size" bit_position="4" tag="low" />
314     <antecedent varname="repetition_size" bit_position="12" tag="high" />
315           <consequent varname="design_font_height4" bit_position="8" tag="
                 normal" />
316       </rule>
317    <rule name="R43">
318           <antecedent varname="contrast_size" bit_position="4" tag="medium"
                  />
319     <antecedent varname="repetition_size" bit_position="12" tag="low" />
320           <consequent varname="design_font_height4" bit_position="8" tag="
                 very_little" />
321       </rule>
322    <rule name="R44">
323           <antecedent varname="contrast_size" bit_position="4" tag="medium"
                  />
324     <antecedent varname="repetition_size" bit_position="12" tag="high" />
325           <consequent varname="design_font_height4" bit_position="8" tag="
                 little" />
326       </rule>
327    <rule name="R45">
```

```
328           <antecedent varname="contrast_size" bit_position="4" tag="high" /
                 >
329      <antecedent varname="repetition_size" bit_position="12" tag="low" />
330           <consequent varname="design_font_height4" bit_position="8" tag="
                 very_little" />
331       </rule>
332    </system>
333    <system name="contrast_repetition_color_main">
334     <linguisticvariable name="contrast_colors" type="input">
335           <fuzzyset tag="low"  a="0" b="0" c="4" d="6" />
336           <fuzzyset tag="high"  a="3" b="5" c="9" d="9" />
337       </linguisticvariable>
338     <linguisticvariable name="repetition_colors" type="input">
339           <fuzzyset tag="low"  a="0" b="0" c="4" d="6" />
340           <fuzzyset tag="high"  a="3" b="5" c="9" d="9" />
341       </linguisticvariable>
342     <linguisticvariable name="source_of_color" type="input">
343           <fuzzyset tag="picture" a="0" b="0" c="2" d="4" />
344      <fuzzyset tag="some_of_the_picture" a="1" b="3" c="6" d="8" />
345           <fuzzyset tag="random"  a="5" b="7" c="9" d="9" />
346       </linguisticvariable>
347     <linguisticvariable name="picture_as_background" type="input">
348           <fuzzyset tag="yes" a="0" b="0" c="4" d="6" />
349      <fuzzyset tag="no" a="3" b="5" c="9" d="9" />
350       </linguisticvariable>
351     <linguisticvariable name="main_color" type="output">
352      <fuzzyset tag="comp_main_pic"   a="0"  b="0"  c="1"  d="3" />
353           <fuzzyset tag="contrast_main_pic"  a="1"  b="2"  c="4"  d="5" />
354      <fuzzyset tag="main_pic"    a="4"  b="5"  c="7"  d="8" />
355           <fuzzyset tag="random"    a="6"  b="8"  c="9"  d="9" />
356       </linguisticvariable>
357     <rule name="R46">
358           <antecedent varname="contrast_colors" bit_position="5" tag="high"
                  />
359      <antecedent varname="repetition_colors" bit_position="13" tag="low" />
360      <antecedent varname="source_of_color" bit_position="22" tag="picture" /
            >
361      <antecedent varname="picture_as_background" bit_position="23" tag="no"
            />
362           <consequent varname="main_color" bit_position="9" tag="
                 comp_main_pic" />
363       </rule>
364     <rule name="R47">
365           <antecedent varname="contrast_colors" bit_position="5" tag="low"
                  />
366      <antecedent varname="repetition_colors" bit_position="13" tag="high" />
367      <antecedent varname="source_of_color" bit_position="22" tag="picture" /
            >
368      <antecedent varname="picture_as_background" bit_position="23" tag="no"
            />
369           <consequent varname="main_color" bit_position="9" tag="main_pic"
                  />
370       </rule>
371     <rule name="R48">
372           <antecedent varname="contrast_colors" bit_position="5" tag="high"
                  />
373      <antecedent varname="repetition_colors" bit_position="13" tag="low" />
374      <antecedent varname="source_of_color" bit_position="22" tag="picture" /
            >
```

```
375     <antecedent varname="picture_as_background" bit_position="23" tag="yes"
            />
376           <consequent varname="main_color" bit_position="9" tag="
                contrast_main_pic" />
377       </rule>
378    <rule name="R49">
379           <antecedent varname="contrast_colors" bit_position="5" tag="low"
                />
380     <antecedent varname="repetition_colors" bit_position="13" tag="high" />
381     <antecedent varname="source_of_color" bit_position="22" tag="picture" /
           >
382     <antecedent varname="picture_as_background" bit_position="23" tag="yes"
            />
383           <consequent varname="main_color" bit_position="9" tag="main_pic"
                />
384       </rule>
385    <rule name="R50">
386           <antecedent varname="contrast_colors" bit_position="5" tag="high"
                />
387     <antecedent varname="repetition_colors" bit_position="13" tag="low" />
388     <antecedent varname="source_of_color" bit_position="22" tag="random" />
389     <antecedent varname="picture_as_background" bit_position="23" tag="no"
            />
390           <consequent varname="main_color" bit_position="9" tag="random" />
391       </rule>
392    <rule name="R51">
393     <antecedent varname="repetition_colors" bit_position="13" tag="low" />
394     <antecedent varname="repetition_colors" bit_position="13" tag="high" />
395     <antecedent varname="source_of_color" bit_position="22" tag="random" />
396     <antecedent varname="picture_as_background" bit_position="23" tag="no"
            />
397           <consequent varname="main_color" bit_position="9" tag="random" />
398       </rule>
399    <rule name="R52">
400           <antecedent varname="contrast_colors" bit_position="5" tag="high"
                />
401     <antecedent varname="repetition_colors" bit_position="13" tag="low" />
402     <antecedent varname="source_of_color" bit_position="22" tag="
            some_of_the_picture" />
403     <antecedent varname="picture_as_background" bit_position="23" tag="no"
            />
404           <consequent varname="main_color" bit_position="9" tag="main_pic"
                />
405       </rule>
406    <rule name="R53">
407           <antecedent varname="contrast_colors" bit_position="5" tag="low"
                />
408     <antecedent varname="repetition_colors" bit_position="13" tag="high" />
409     <antecedent varname="source_of_color" bit_position="22" tag="
            some_of_the_picture" />
410     <antecedent varname="picture_as_background" bit_position="23" tag="no"
            />
411           <consequent varname="main_color" bit_position="9" tag="main_pic"
                />
412       </rule>
413    <rule name="R54">
414           <antecedent varname="contrast_colors" bit_position="5" tag="high"
                />
415     <antecedent varname="repetition_colors" bit_position="13" tag="low" />
```

```
416     <antecedent varname="source_of_color" bit_position="22" tag="
            some_of_the_picture" />
417     <antecedent varname="picture_as_background" bit_position="23" tag="yes"
            />
418           <consequent varname="main_color" bit_position="9" tag="
                contrast_main_pic" />
419       </rule>
420     <rule name="R55">
421           <antecedent varname="contrast_colors" bit_position="5" tag="low"
                />
422     <antecedent varname="repetition_colors" bit_position="13" tag="high" />
423     <antecedent varname="source_of_color" bit_position="22" tag="
            some_of_the_picture" />
424     <antecedent varname="picture_as_background" bit_position="23" tag="yes"
            />
425           <consequent varname="main_color" bit_position="9" tag="main_pic"
                />
426       </rule>
427   </system>
428   <system name="contrast_repetition_color_accents">
429    <linguisticvariable name="contrast_colors" type="input">
430           <fuzzyset tag="low"  a="0" b="0" c="4" d="6" />
431           <fuzzyset tag="high"  a="3" b="5" c="9" d="9" />
432       </linguisticvariable>
433    <linguisticvariable name="repetition_colors" type="input">
434           <fuzzyset tag="low"  a="0" b="0" c="4" d="6" />
435           <fuzzyset tag="high"  a="3" b="5" c="9" d="9" />
436       </linguisticvariable>
437    <linguisticvariable name="source_of_color" type="input">
438           <fuzzyset tag="picture" a="0" b="0" c="2" d="4" />
439     <fuzzyset tag="some_of_the_picture" a="1" b="3" c="6" d="7" />
440           <fuzzyset tag="random"  a="6" b="7" c="9" d="9" />
441       </linguisticvariable>
442    <linguisticvariable name="picture_as_background" type="input">
443           <fuzzyset tag="yes" a="0" b="0" c="4" d="6" />
444     <fuzzyset tag="no" a="3" b="5" c="9" d="9" />
445       </linguisticvariable>
446    <linguisticvariable name="accent_colors" type="output">
447     <fuzzyset tag="secondary_pic"   a="0"  b="0"  c="1"  d="3" />
448           <fuzzyset tag="triadic"    a="1"  b="2"  c="4"  d="5" />
449     <fuzzyset tag="analogous"    a="4"  b="5"  c="7"  d="8" />
450           <fuzzyset tag="complementary"  a="6"  b="8"  c="9"  d="9" />
451       </linguisticvariable>
452    <rule name="R56">
453           <antecedent varname="contrast_colors" bit_position="5" tag="high"
                />
454     <antecedent varname="repetition_colors" bit_position="13" tag="low" />
455     <antecedent varname="source_of_color" bit_position="22" tag="picture" /
            >
456     <antecedent varname="picture_as_background" bit_position="23" tag="no"
            />
457           <consequent varname="accent_colors" bit_position="23" tag="
                triadic" />
458       </rule>
459    <rule name="R57">
460           <antecedent varname="contrast_colors" bit_position="5" tag="low"
                />
461     <antecedent varname="repetition_colors" bit_position="13" tag="high" />
462     <antecedent varname="source_of_color" bit_position="22" tag="picture" /
            >
```

```
463     <antecedent varname="picture_as_background" bit_position="23" tag="no"
            />
464           <consequent varname="accent_colors" bit_position="23" tag="
                secondary_pic" />
465       </rule>
466     <rule name="R58">
467           <antecedent varname="contrast_colors" bit_position="5" tag="high"
                 />
468      <antecedent varname="repetition_colors" bit_position="13" tag="low" />
469      <antecedent varname="source_of_color" bit_position="22" tag="picture" /
            >
470      <antecedent varname="picture_as_background" bit_position="23" tag="yes"
             />
471           <consequent varname="accent_colors" bit_position="23" tag="
                triadic" />
472       </rule>
473     <rule name="R59">
474           <antecedent varname="contrast_colors" bit_position="5" tag="low"
                />
475      <antecedent varname="repetition_colors" bit_position="13" tag="high" />
476      <antecedent varname="source_of_color" bit_position="22" tag="picture" /
            >
477      <antecedent varname="picture_as_background" bit_position="23" tag="yes"
             />
478           <consequent varname="accent_colors" bit_position="23" tag="
                analogous" />
479       </rule>
480     <rule name="R60">
481           <antecedent varname="contrast_colors" bit_position="5" tag="high"
                 />
482      <antecedent varname="repetition_colors" bit_position="13" tag="low" />
483      <antecedent varname="source_of_color" bit_position="22" tag="random" />
484      <antecedent varname="picture_as_background" bit_position="23" tag="no"
            />
485           <consequent varname="accent_colors" bit_position="23" tag="
                complementary" />
486       </rule>
487     <rule name="R61">
488      <antecedent varname="contrast_colors" bit_position="5" tag="low" />
489      <antecedent varname="repetition_colors" bit_position="13" tag="high" />
490      <antecedent varname="source_of_color" bit_position="22" tag="random" />
491      <antecedent varname="picture_as_background" bit_position="23" tag="no"
            />
492           <consequent varname="accent_colors" bit_position="23" tag="
                analogous" />
493       </rule>
494     <rule name="R62">
495           <antecedent varname="contrast_colors" bit_position="5" tag="high"
                 />
496      <antecedent varname="repetition_colors" bit_position="13" tag="low" />
497      <antecedent varname="source_of_color" bit_position="22" tag="
            some_of_the_picture" />
498      <antecedent varname="picture_as_background" bit_position="23" tag="no"
            />
499           <consequent varname="accent_colors" bit_position="23" tag="
                triadic" />
500       </rule>
501     <rule name="R63">
502           <antecedent varname="contrast_colors" bit_position="5" tag="low"
                />
```

```
503     <antecedent varname="repetition_colors" bit_position="13" tag="high" />
504     <antecedent varname="source_of_color" bit_position="22" tag="
           some_of_the_picture" />
505     <antecedent varname="picture_as_background" bit_position="23" tag="no"
           />
506           <consequent varname="accent_colors" bit_position="23" tag="
              analogous" />
507       </rule>
508    <rule name="R64">
509           <antecedent varname="contrast_colors" bit_position="5" tag="high"
                />
510     <antecedent varname="repetition_colors" bit_position="13" tag="low" />
511     <antecedent varname="source_of_color" bit_position="22" tag="
           some_of_the_picture" />
512     <antecedent varname="picture_as_background" bit_position="23" tag="yes"
            />
513           <consequent varname="accent_colors" bit_position="23" tag="
              triadic" />
514       </rule>
515    <rule name="R65">
516           <antecedent varname="contrast_colors" bit_position="5" tag="low"
                />
517     <antecedent varname="repetition_colors" bit_position="13" tag="high" />
518     <antecedent varname="source_of_color" bit_position="22" tag="
           some_of_the_picture" />
519     <antecedent varname="picture_as_background" bit_position="23" tag="yes"
            />
520           <consequent varname="accent_colors" bit_position="23" tag="
              analogous" />
521       </rule>
522   </system>
523   <system name="contrast_repetition_color_bg">
524    <linguisticvariable name="contrast_colors" type="input">
525           <fuzzyset tag="low"  a="0" b="0" c="4" d="6" />
526           <fuzzyset tag="high"  a="3" b="5" c="9" d="9" />
527       </linguisticvariable>
528    <linguisticvariable name="repetition_colors" type="input">
529           <fuzzyset tag="low"  a="0" b="0" c="4" d="6" />
530           <fuzzyset tag="high"  a="3" b="5" c="9" d="9" />
531       </linguisticvariable>
532    <linguisticvariable name="picture_as_background" type="input">
533           <fuzzyset tag="yes" a="0" b="0" c="4" d="6" />
534     <fuzzyset tag="no" a="3" b="5" c="9" d="9" />
535       </linguisticvariable>
536    <linguisticvariable name="background_color" type="output">
537     <fuzzyset tag="bw"    a="0"  b="0"  c="3"  d="4" />
538           <fuzzyset tag="pale"  a="2"  b="3"  c="6"  d="8" />
539     <fuzzyset tag="nevermind" a="5"  b="7"  c="9"  d="9" />
540       </linguisticvariable>
541    <rule name="R66">
542           <antecedent varname="contrast_colors" bit_position="5" tag="high"
                />
543     <antecedent varname="repetition_colors" bit_position="13" tag="low" />
544     <antecedent varname="picture_as_background" bit_position="23" tag="no"
           />
545           <consequent varname="background_color" bit_position="24" tag="bw"
                />
546       </rule>
547    <rule name="R67">
```

```
548           <antecedent varname="contrast_colors" bit_position="5" tag="low"
                 />
549      <antecedent varname="repetition_colors" bit_position="13" tag="high" />
550      <antecedent varname="picture_as_background" bit_position="23" tag="no"
            />
551           <consequent varname="background_color" bit_position="24" tag="
                 pale" />
552        </rule>
553     <rule name="R68">
554      <antecedent varname="picture_as_background" bit_position="23" tag="yes"
            />
555           <consequent varname="background_color" bit_position="24" tag="
                 nevermind" />
556        </rule>
557    </system>
558    <system name="contrast_repetition_shape">
559     <linguisticvariable name="contrast_shape" type="input">
560           <fuzzyset tag="low"  a="0" b="0" c="4" d="6" />
561           <fuzzyset tag="high"  a="3" b="5" c="9" d="9" />
562        </linguisticvariable>
563     <linguisticvariable name="repetition_shape" type="input">
564           <fuzzyset tag="low"  a="0" b="0" c="4" d="6" />
565           <fuzzyset tag="high"  a="3" b="5" c="9" d="9" />
566        </linguisticvariable>
567     <linguisticvariable name="input_shape" type="input">
568      <fuzzyset tag="curved"     a="0" b="0" c="2" d="3" />
569           <fuzzyset tag="slightly_curved"  a="2" b="3" c="4" d="5" />
570           <fuzzyset tag="slightly_straight"  a="4" b="5" c="6" d="7" />
571      <fuzzyset tag="straight"    a="6" b="7" c="9" d="9" />
572        </linguisticvariable>
573     <linguisticvariable name="design_elements_shape" type="output">
574      <fuzzyset tag="curved"     a="0" b="0" c="2" d="3" />
575           <fuzzyset tag="slightly_curved"  a="2" b="3" c="4" d="5" />
576           <fuzzyset tag="slightly_straight"  a="4" b="5" c="6" d="7" />
577      <fuzzyset tag="straight"    a="6" b="7" c="9" d="9" />
578        </linguisticvariable>
579     <rule name="R69">
580           <antecedent varname="contrast_shape" bit_position="6" tag="low" /
                 >
581      <antecedent varname="repetition_shape" bit_position="14" tag="high" />
582      <antecedent varname="input_shape" bit_position="20" tag="
            slightly_curved" />
583           <consequent varname="design_elements_shape" bit_position="10" tag
                 ="slightly_curved" />
584        </rule>
585     <rule name="R70">
586           <antecedent varname="contrast_shape" bit_position="6" tag="high"
                 />
587      <antecedent varname="repetition_shape" bit_position="14" tag="low" />
588      <antecedent varname="input_shape" bit_position="20" tag="
            slightly_curved" />
589           <consequent varname="design_elements_shape" bit_position="10" tag
                 ="straight" />
590        </rule>
591     <rule name="R71">
592           <antecedent varname="contrast_shape" bit_position="6" tag="low" /
                 >
593      <antecedent varname="repetition_shape" bit_position="14" tag="high" />
594      <antecedent varname="input_shape" bit_position="20" tag="curved" />
```

```
595           <consequent varname="design_elements_shape" bit_position="10" tag
                  ="curved" />
596       </rule>
597     <rule name="R72">
598           <antecedent varname="contrast_shape" bit_position="6" tag="high"
                  />
599      <antecedent varname="repetition_shape" bit_position="14" tag="low" />
600      <antecedent varname="input_shape" bit_position="20" tag="curved" />
601           <consequent varname="design_elements_shape" bit_position="10" tag
                  ="straight" />
602       </rule>
603     <rule name="R73">
604           <antecedent varname="contrast_shape" bit_position="6" tag="low" /
                  >
605      <antecedent varname="repetition_shape" bit_position="14" tag="high" />
606      <antecedent varname="input_shape" bit_position="20" tag="
            slightly_straight" />
607           <consequent varname="design_elements_shape" bit_position="10" tag
                  ="slightly_straight" />
608       </rule>
609     <rule name="R74">
610           <antecedent varname="contrast_shape" bit_position="6" tag="high"
                  />
611      <antecedent varname="repetition_shape" bit_position="14" tag="low" />
612      <antecedent varname="input_shape" bit_position="20" tag="
            slightly_straight" />
613           <consequent varname="design_elements_shape" bit_position="10" tag
                  ="curved" />
614       </rule>
615     <rule name="R55">
616           <antecedent varname="contrast_shape" bit_position="6" tag="low" /
                  >
617      <antecedent varname="repetition_shape" bit_position="14" tag="high" />
618      <antecedent varname="input_shape" bit_position="20" tag="straight" />
619           <consequent varname="design_elements_shape" bit_position="10" tag
                  ="straight" />
620       </rule>
621     <rule name="R76">
622           <antecedent varname="contrast_shape" bit_position="6" tag="high"
                  />
623      <antecedent varname="repetition_shape" bit_position="14" tag="low" />
624      <antecedent varname="input_shape" bit_position="20" tag="straight" />
625           <consequent varname="design_elements_shape" bit_position="10" tag
                  ="curved" />
626       </rule>
627   </system>
628   <system name="contrast_background">
629    <linguisticvariable name="contrast_background" type="input">
630           <fuzzyset tag="low"  a="0" b="0" c="2" d="3" />
631     <fuzzyset tag="medium"  a="2" b="3" c="6" d="7" />
632           <fuzzyset tag="high"  a="6" b="7" c="9" d="9" />
633       </linguisticvariable>
634     <linguisticvariable name="text_background" type="output">
635      <fuzzyset tag="no_use"  a="0"  b="0"  c="2"  d="3" />
636           <fuzzyset tag="medium_use"  a="2"  b="3"  c="6"  d="7" />
637      <fuzzyset tag="high_use"  a="6"  b="7"  c="9"  d="9" />
638       </linguisticvariable>
639     <rule name="R79">
640           <antecedent varname="contrast_background" bit_position="8" tag="
                  low" />
```

```
641           <consequent varname="text_background" bit_position="12" tag="
                 no_use" />
642       </rule>
643    <rule name="R80">
644           <antecedent varname="contrast_background" bit_position="8" tag="
                 medium" />
645           <consequent varname="text_background" bit_position="12" tag="
                 medium_use" />
646       </rule>
647    <rule name="R121">
648           <antecedent varname="contrast_background" bit_position="8" tag="
                 high" />
649           <consequent varname="text_background" bit_position="12" tag="
                 high_use" />
650       </rule>
651   </system>
652   <system name="contrast_background_decoration">
653    <linguisticvariable name="decoration_bit" type="input">
654           <fuzzyset tag="low" a="0" b="0" c="4" d="6" />
655     <fuzzyset tag="high" a="3" b="5" c="9" d="9" />
656       </linguisticvariable>
657    <linguisticvariable name="decoration" type="output">
658     <fuzzyset tag="undercolor" a="0" b="0" c="4" d="6" />
659     <fuzzyset tag="border" a="3" b="5" c="9" d="9" />
660       </linguisticvariable>
661    <rule name="R127">
662           <antecedent varname="decoration_bit" bit_position="31" tag="low"
                 />
663           <consequent varname="decoration" bit_position="30" tag="
                 undercolor" />
664       </rule>
665    <rule name="R128">
666           <antecedent varname="decoration_bit" bit_position="31" tag="high"
                  />
667           <consequent varname="decoration" bit_position="30" tag="border" /
                 >
668       </rule>
669   </system>
670   <system name="contrast_repetition_weight_h0">
671    <linguisticvariable name="contrast_weight" type="input">
672           <fuzzyset tag="low" a="0" b="0" c="2" d="4" />
673           <fuzzyset tag="medium" a="1" b="3" c="6" d="8" />
674           <fuzzyset tag="high" a="5" b="7" c="9" d="9" />
675       </linguisticvariable>
676    <linguisticvariable name="repetition_weight" type="input">
677           <fuzzyset tag="low"  a="0" b="0" c="4" d="6" />
678           <fuzzyset tag="high"  a="5" b="7" c="9" d="9" />
679       </linguisticvariable>
680    <linguisticvariable name="font_weight_h0" type="output">
681     <fuzzyset tag="semibold" a="3" b="4" c="5" d="6" />
682     <fuzzyset tag="bold" a="5" b="6" c="7" d="8" />
683     <fuzzyset tag="black" a="7" b="8" c="9" d="9" />
684       </linguisticvariable>
685    <rule name="R84">
686           <antecedent varname="contrast_weight" bit_position="10" tag="low"
                  />
687     <antecedent varname="repetition_weight" bit_position="17" tag="low" />
688           <consequent varname="font_weight_h0" bit_position="14" tag="
                 semibold" />
689       </rule>
```

```
690     <rule name="R85">
691           <antecedent varname="contrast_weight" bit_position="10" tag="low"
                 />
692      <antecedent varname="repetition_weight" bit_position="17" tag="high" />
693           <consequent varname="font_weight_h0" bit_position="14" tag="
                 semibold" />
694        </rule>
695     <rule name="R86">
696           <antecedent varname="contrast_weight" bit_position="10" tag="
                 medium" />
697      <antecedent varname="repetition_weight" bit_position="17" tag="low" />
698           <consequent varname="font_weight_h0" bit_position="14" tag="bold"
                 />
699        </rule>
700     <rule name="R87">
701           <antecedent varname="contrast_weight" bit_position="10" tag="
                 medium" />
702      <antecedent varname="repetition_weight" bit_position="17" tag="high" />
703           <consequent varname="font_weight_h0" bit_position="14" tag="bold"
                 />
704        </rule>
705     <rule name="R88">
706           <antecedent varname="contrast_weight" bit_position="10" tag="high
                 " />
707           <consequent varname="font_weight_h0" bit_position="14" tag="bold"
                 />
708        </rule>
709   </system>
710   <system name="contrast_repetition_weight_h1">
711    <linguisticvariable name="contrast_weight" type="input">
712           <fuzzyset tag="low" a="0" b="0" c="2" d="4" />
713           <fuzzyset tag="medium" a="1" b="3" c="6" d="8" />
714           <fuzzyset tag="high" a="5" b="7" c="9" d="9" />
715        </linguisticvariable>
716    <linguisticvariable name="repetition_weight" type="input">
717           <fuzzyset tag="low"  a="0" b="0" c="4" d="6" />
718           <fuzzyset tag="high"  a="5" b="7" c="9" d="9" />
719        </linguisticvariable>
720    <linguisticvariable name="font_weight_h1" type="output">
721     <fuzzyset tag="semibold" a="3" b="4" c="5" d="6" />
722     <fuzzyset tag="light" a="0" b="0" c="1" d="2" />
723     <fuzzyset tag="normal" a="1" b="2" c="3" d="4" />
724        </linguisticvariable>
725    <rule name="R89">
726           <antecedent varname="contrast_weight" bit_position="10" tag="low"
                 />
727      <antecedent varname="repetition_weight" bit_position="17" tag="low" />
728           <consequent varname="font_weight_h1" bit_position="15" tag="
                 normal" />
729        </rule>
730    <rule name="R90">
731           <antecedent varname="contrast_weight" bit_position="10" tag="low"
                 />
732      <antecedent varname="repetition_weight" bit_position="17" tag="high" />
733           <consequent varname="font_weight_h1" bit_position="15" tag="
                 semibold" />
734        </rule>
735    <rule name="R91">
736           <antecedent varname="contrast_weight" bit_position="10" tag="
                 medium" />
```

```
737     <antecedent varname="repetition_weight" bit_position="17" tag="low" />
738           <consequent varname="font_weight_h1" bit_position="15" tag="light
                " />
739       </rule>
740    <rule name="R92">
741           <antecedent varname="contrast_weight" bit_position="10" tag="
                medium" />
742     <antecedent varname="repetition_weight" bit_position="17" tag="high" />
743           <consequent varname="font_weight_h1" bit_position="15" tag="light
                " />
744       </rule>
745    <rule name="R93">
746           <antecedent varname="contrast_weight" bit_position="10" tag="high
                " />
747           <consequent varname="font_weight_h1" bit_position="15" tag="light
                " />
748       </rule>
749   </system>
750   <system name="contrast_repetition_weight_h2">
751    <linguisticvariable name="contrast_weight" type="input">
752           <fuzzyset tag="low" a="0" b="0" c="2" d="4" />
753           <fuzzyset tag="medium" a="1" b="3" c="6" d="8" />
754           <fuzzyset tag="high" a="5" b="7" c="9" d="9" />
755       </linguisticvariable>
756    <linguisticvariable name="repetition_weight" type="input">
757           <fuzzyset tag="low"  a="0" b="0" c="4" d="6" />
758           <fuzzyset tag="high"  a="5" b="7" c="9" d="9" />
759       </linguisticvariable>
760    <linguisticvariable name="font_weight_h2" type="output">
761     <fuzzyset tag="semibold" a="3" b="4" c="5" d="6" />
762     <fuzzyset tag="bold" a="5" b="6" c="7" d="8" />
763     <fuzzyset tag="black" a="7" b="8" c="9" d="9" />
764     <fuzzyset tag="normal" a="1" b="2" c="3" d="4" />
765       </linguisticvariable>
766    <rule name="R94">
767           <antecedent varname="contrast_weight" bit_position="10" tag="low"
                />
768     <antecedent varname="repetition_weight" bit_position="17" tag="low" />
769           <consequent varname="font_weight_h2" bit_position="16" tag="
                semibold" />
770       </rule>
771    <rule name="R95">
772           <antecedent varname="contrast_weight" bit_position="10" tag="low"
                />
773     <antecedent varname="repetition_weight" bit_position="17" tag="high" />
774           <consequent varname="font_weight_h2" bit_position="16" tag="
                normal" />
775       </rule>
776    <rule name="R96">
777           <antecedent varname="contrast_weight" bit_position="10" tag="
                medium" />
778     <antecedent varname="repetition_weight" bit_position="17" tag="low" />
779           <consequent varname="font_weight_h2" bit_position="16" tag="bold"
                />
780       </rule>
781    <rule name="R97">
782           <antecedent varname="contrast_weight" bit_position="10" tag="
                medium" />
783     <antecedent varname="repetition_weight" bit_position="17" tag="high" />
```

```
784           <consequent varname="font_weight_h2" bit_position="16" tag="
                  normal" />
785       </rule>
786     <rule name="R98">
787           <antecedent varname="contrast_weight" bit_position="10" tag="high
                  " />
788           <consequent varname="font_weight_h2" bit_position="16" tag="black
                  " />
789       </rule>
790   </system>
791   <system name="contrast_repetition_weight_h3">
792     <linguisticvariable name="contrast_weight" type="input">
793           <fuzzyset tag="low" a="0" b="0" c="2" d="4" />
794           <fuzzyset tag="medium" a="1" b="3" c="6" d="8" />
795           <fuzzyset tag="high" a="5" b="7" c="9" d="9" />
796       </linguisticvariable>
797     <linguisticvariable name="repetition_weight" type="input">
798           <fuzzyset tag="low"  a="0" b="0" c="4" d="6" />
799           <fuzzyset tag="high"  a="5" b="7" c="9" d="9" />
800       </linguisticvariable>
801     <linguisticvariable name="font_weight_h3" type="output">
802       <fuzzyset tag="light" a="0" b="0" c="1" d="2" />
803       <fuzzyset tag="normal" a="1" b="2" c="3" d="4" />
804       </linguisticvariable>
805     <rule name="R99">
806           <antecedent varname="contrast_weight" bit_position="10" tag="low"
                  />
807       <antecedent varname="repetition_weight" bit_position="17" tag="low" />
808           <consequent varname="font_weight_h3" bit_position="17" tag="
                  normal" />
809       </rule>
810     <rule name="R100">
811           <antecedent varname="contrast_weight" bit_position="10" tag="low"
                  />
812       <antecedent varname="repetition_weight" bit_position="17" tag="high" />
813           <consequent varname="font_weight_h3" bit_position="17" tag="
                  normal" />
814       </rule>
815     <rule name="R101">
816           <antecedent varname="contrast_weight" bit_position="10" tag="
                  medium" />
817       <antecedent varname="repetition_weight" bit_position="17" tag="low" />
818           <consequent varname="font_weight_h3" bit_position="17" tag="
                  normal" />
819       </rule>
820     <rule name="R102">
821           <antecedent varname="contrast_weight" bit_position="10" tag="
                  medium" />
822       <antecedent varname="repetition_weight" bit_position="17" tag="high" />
823           <consequent varname="font_weight_h3" bit_position="17" tag="
                  normal" />
824       </rule>
825     <rule name="R103">
826           <antecedent varname="contrast_weight" bit_position="10" tag="high
                  " />
827           <consequent varname="font_weight_h3" bit_position="17" tag="light
                  " />
828       </rule>
829   </system>
830   <system name="contrast_repetition_weight_h4">
```

```
831    <linguisticvariable name="contrast_weight" type="input">
832           <fuzzyset tag="low" a="0" b="0" c="2" d="4" />
833           <fuzzyset tag="medium" a="1" b="3" c="6" d="8" />
834           <fuzzyset tag="high" a="5" b="7" c="9" d="9" />
835       </linguisticvariable>
836    <linguisticvariable name="repetition_weight" type="input">
837           <fuzzyset tag="low"  a="0" b="0" c="4" d="6" />
838           <fuzzyset tag="high"  a="5" b="7" c="9" d="9" />
839       </linguisticvariable>
840    <linguisticvariable name="font_weight_h4" type="output">
841     <fuzzyset tag="light" a="0" b="0" c="1" d="2" />
842     <fuzzyset tag="normal" a="1" b="2" c="3" d="4" />
843       </linguisticvariable>
844    <rule name="R104">
845           <antecedent varname="contrast_weight" bit_position="10" tag="low"
                 />
846     <antecedent varname="repetition_weight" bit_position="17" tag="low" />
847           <consequent varname="font_weight_h4" bit_position="18" tag="
                 normal" />
848       </rule>
849    <rule name="R105">
850           <antecedent varname="contrast_weight" bit_position="10" tag="low"
                 />
851     <antecedent varname="repetition_weight" bit_position="17" tag="high" />
852           <consequent varname="font_weight_h4" bit_position="18" tag="
                 normal" />
853       </rule>
854    <rule name="R106">
855           <antecedent varname="contrast_weight" bit_position="10" tag="
                 medium" />
856     <antecedent varname="repetition_weight" bit_position="17" tag="low" />
857           <consequent varname="font_weight_h4" bit_position="18" tag="
                 normal" />
858       </rule>
859    <rule name="R107">
860           <antecedent varname="contrast_weight" bit_position="10" tag="
                 medium" />
861     <antecedent varname="repetition_weight" bit_position="17" tag="high" />
862           <consequent varname="font_weight_h4" bit_position="18" tag="
                 normal" />
863       </rule>
864    <rule name="R108">
865           <antecedent varname="contrast_weight" bit_position="10" tag="high
                 " />
866           <consequent varname="font_weight_h4" bit_position="18" tag="light
                 " />
867       </rule>
868   </system>
869   <system name="contrast_repetition_font_face">
870    <linguisticvariable name="contrast_font_face" type="input">
871           <fuzzyset tag="low" a="0" b="0" c="4" d="6" />
872           <fuzzyset tag="high" a="5" b="6" c="9" d="9" />
873       </linguisticvariable>
874    <linguisticvariable name="repetition_font_face" type="input">
875     <fuzzyset tag="low" a="0" b="0" c="4" d="6" />
876           <fuzzyset tag="high" a="5" b="7" c="9" d="9" />
877       </linguisticvariable>
878    <linguisticvariable name="design_font_face_2" type="output">
879     <fuzzyset tag="same" a="0" b="0" c="2" d="4" />
880     <fuzzyset tag="different" a="5" b="7" c="9" d="9" />
```

```
881       </linguisticvariable>
882     <rule name="R109">
883           <antecedent varname="contrast_font_face" bit_position="11" tag="
                 low" />
884      <antecedent varname="repetition_font_face" bit_position="18" tag="low"
             />
885           <consequent varname="design_font_face_2" bit_position="19" tag="
                 same" />
886       </rule>
887     <rule name="R110">
888           <antecedent varname="contrast_font_face" bit_position="11" tag="
                 low" />
889      <antecedent varname="repetition_font_face" bit_position="18" tag="high"
              />
890           <consequent varname="design_font_face_2" bit_position="19" tag="
                 same" />
891       </rule>
892     <rule name="R111">
893           <antecedent varname="contrast_font_face" bit_position="11" tag="
                 high" />
894      <antecedent varname="repetition_font_face" bit_position="18" tag="low"
             />
895           <consequent varname="design_font_face_2" bit_position="19" tag="
                 different" />
896       </rule>
897   </system>
898   <system name="composition_doc_margins">
899    <linguisticvariable name="doc_margins_bit" type="input">
900           <fuzzyset tag="no_margin" a="0" b="0" c="1" d="2" />
901     <fuzzyset tag="low" a="1" b="2" c="4" d="5" />
902     <fuzzyset tag="medium" a="4" b="5" c="7" d="8" />
903           <fuzzyset tag="high" a="7" b="8" c="9" d="9" />
904       </linguisticvariable>
905     <linguisticvariable name="margin" type="output">
906      <fuzzyset tag="no_margin" a="0" b="0" c="1" d="2" />
907      <fuzzyset tag="low" a="1" b="2" c="4" d="5" />
908      <fuzzyset tag="medium" a="4" b="5" c="7" d="8" />
909           <fuzzyset tag="high" a="7" b="8" c="9" d="9" />
910       </linguisticvariable>
911     <rule name="R115">
912           <antecedent varname="doc_margins_bit" bit_position="28" tag="
                 no_margin" />
913           <consequent varname="margin" bit_position="26" tag="no_margin" />
914       </rule>
915     <rule name="R116">
916           <antecedent varname="doc_margins_bit" bit_position="28" tag="low"
                  />
917           <consequent varname="margin" bit_position="26" tag="low" />
918       </rule>
919     <rule name="R117">
920           <antecedent varname="doc_margins_bit" bit_position="28" tag="
                 medium" />
921           <consequent varname="margin" bit_position="26" tag="medium" />
922       </rule>
923     <rule name="R118">
924           <antecedent varname="doc_margins_bit" bit_position="28" tag="high
                 " />
925           <consequent varname="margin" bit_position="26" tag="high" />
926       </rule>
927   </system>
```

```
928   <system name="composition_main_image_position">
929    <linguisticvariable name="main_image_position_bit" type="input">
930           <fuzzyset tag="normal" a="0" b="0" c="4" d="6" />
931     <fuzzyset tag="alternative" a="3" b="5" c="9" d="9" />
932       </linguisticvariable>
933    <linguisticvariable name="main_image_position" type="output">
934     <fuzzyset tag="normal" a="0" b="0" c="4" d="6" />
935     <fuzzyset tag="alternative" a="3" b="5" c="9" d="9" />
936       </linguisticvariable>
937    <rule name="R119">
938           <antecedent varname="main_image_position_bit" bit_position="27"
                 tag="normal" />
939           <consequent varname="main_image_position" bit_position="27" tag="
                 normal" />
940       </rule>
941    <rule name="R120">
942           <antecedent varname="main_image_position_bit" bit_position="27"
                 tag="alternative" />
943           <consequent varname="main_image_position" bit_position="27" tag="
                 alternative" />
944       </rule>
945   </system>
946   <system name="contrast_all_caps">
947    <linguisticvariable name="all_caps_bit" type="input">
948           <fuzzyset tag="low"  a="0" b="0" c="4" d="5" />
949     <fuzzyset tag="medium"  a="3" b="5" c="6" d="7" />
950           <fuzzyset tag="high"  a="6" b="8" c="9" d="9" />
951       </linguisticvariable>
952    <linguisticvariable name="all_caps" type="output">
953     <fuzzyset tag="none"    a="0" b="0" c="3" d="5" />
954     <fuzzyset tag="title"    a="2" b="4" c="5" d="7" />
955     <fuzzyset tag="title_subtitle"  a="6" b="8" c="9" d="9" />
956       </linguisticvariable>
957    <rule name="R122">
958           <antecedent varname="all_caps_bit" bit_position="29" tag="low" />
959           <consequent varname="all_caps" bit_position="28" tag="none" />
960       </rule>
961    <rule name="R123">
962           <antecedent varname="all_caps_bit" bit_position="29" tag="medium"
                  />
963           <consequent varname="all_caps" bit_position="28" tag="title" />
964       </rule>
965    <rule name="R124">
966           <antecedent varname="all_caps_bit" bit_position="29" tag="high" /
                 >
967           <consequent varname="all_caps" bit_position="28" tag="
                 title_subtitle" />
968       </rule>
969   </system>
970   <system name="composition_zooming">
971    <linguisticvariable name="zooming_bit" type="input">
972           <fuzzyset tag="low" a="0" b="0" c="4" d="6" />
973     <fuzzyset tag="high" a="3" b="5" c="9" d="9" />
974       </linguisticvariable>
975    <linguisticvariable name="zooming" type="output">
976     <fuzzyset tag="no" a="0" b="0" c="4" d="6" />
977     <fuzzyset tag="yes" a="3" b="5" c="9" d="9" />
978       </linguisticvariable>
979    <rule name="R125">
980           <antecedent varname="zooming_bit" bit_position="30" tag="low" />
```

```
981           <consequent varname="zooming" bit_position="29" tag="no" />
982       </rule>
983     <rule name="R126">
984           <antecedent varname="zooming_bit" bit_position="30" tag="high" />
985           <consequent varname="zooming" bit_position="29" tag="yes" />
986       </rule>
987    </system>
988  </FIE>
```

A.2. Motor de Inferencia Difusa

```
 1  class Fie
 2    attr_accessor :rules, :design, :doc, :systems
 3
 4    def initialize()
 5
 6    end
 7
 8    #Executes all the process
 9    def run_fie(design, rules_filename="public/xml/rules.xml")
10      @design=design
11      @systems = {}
12      open_xml(rules_filename)
13      start_engine
14      fuzzyfication
15      rule_evaluation
16      defuzzification
17    end
18
19    #It opens an XML file passed by argument. "rules.xml" will be
20    #loaded by default if there is n oargument.
21    def open_xml(rules_filename)
22      @doc = Document.new(File.new(rules_filename))
23    end
24
25    #This function reads the XML and add the data to the
26    #data structure of the FIE.
27    def start_engine
28      root=doc.root
29      #Systems checking
30      root.elements.each do |system|#For each System do...
31        system_name=system.attributes["name"]
32        systems[system_name]=System.new(system_name)
33        #Linguistic variables and Rules checking
34        system.elements.each do |lvr| #For each rule or linguistic
             variable do...
35          if lvr.name==’linguisticvariable’ #It is a linguistic
               variable
36            lv_name=lvr.attributes["name"]
```

```
37            systems[system_name].lv_hash[lv_name]=Linguistic_variable
                 .new(lv_name, lvr.attributes["type"])
38            #Fuzzy Sets checking
39            lvr.elements.each do |fs|#For each Fuzzy Set do...
40              systems[system_name].lv_hash[lv_name].add_fuzzy_set(fs.
                   attributes["tag"], Fuzzy_set.new(fs.attributes["tag"
                   ], fs.attributes["a"], fs.attributes["b"], fs.
                   attributes["c"], fs.attributes["d"]))
41            end
42          else#It is a rule
43            rule_name=lvr.attributes["name"]
44            systems[system_name].rules_hash[rule_name] = Rule.new(
                 rule_name)
45            #Antecedents and Consequents checking
46            lvr.elements.each do |anco|#For each antecedent or
                 consequent do...
47              if anco.name=='antecedent' #It is an antecedent
48                systems[system_name].rules_hash[rule_name].
                     add_antecedent(Rule_element.new(anco.attributes["
                     varname"],anco.attributes["tag"], anco.attributes[
                     "bit_position"]))
49              else #It is a consequent
50                 systems[system_name].rules_hash[rule_name].
                       add_consequent(Rule_element.new(anco.attributes["
                       varname"],anco.attributes["tag"], anco.attributes
                       ["bit_position"]))
51                 systems[system_name].bit_position=anco.attributes["
                       bit_position"].to_i
52              end
53            end #if antecedent/consequent
54          end #if linguisticvariables/rules
55        end #each Linguistic Variables and rules
56      end #each Systems
57    end#def Start_engine
58
59    #This function checks every antecedent in every
60    #rule and calculate its membership degree by
61    #calling that function.
62    def fuzzyfication
63      systems.each do |key_s,s|#for each System
64        s.rules_hash.each do |key_r,r|#for each Rule
65          r.antecedents.each do |a|#for each Antecedent
66            a.membership_d=calculate_membership_degree(s,a)
67          end #each antecedents
68        end #each rules
69      end #each systems
70    end #def fuzzyfication
71
72    #Given a system and an antecedent, this calculates its
73    #membership degree
74    def calculate_membership_degree(system,antecedent)
75      a = system.lv_hash[antecedent.lv_name].fuzzy_sets[antecedent.
           tag].a
```

```
76      b = system.lv_hash[antecedent.lv_name].fuzzy_sets[antecedent.
           tag].b
77      c = system.lv_hash[antecedent.lv_name].fuzzy_sets[antecedent.
           tag].c
78      d = system.lv_hash[antecedent.lv_name].fuzzy_sets[antecedent.
           tag].d
79      x= design.gen_bits[antecedent.bit_position]
80
81      if x>=a and x<b#between a and b
82        if x == a
83          0
84        else
85          (x-a)/(b-a)
86        end
87      elsif x>=b and x<=c#between b and c
88        1
89      elsif x>c and x<=d #between c and d
90        if x == d
91          0
92        else
93          (d-x)/(d-c)
94        end
95      else#out of the function
96        0
97      end#if point checking
98
99    end#def calculate_membership_degree
100
101   #Given a system, a consequent and the maximum hegith
102   #obtained from the antecedents, this calculates the area
103   #and the centroid for each consequent.
104   def calculate_area_centroid(system, consequent, height)
105     fuzzy_set = systems[system.name].lv_hash[consequent.lv_name].
           fuzzy_sets[consequent.tag]
106
107     a= fuzzy_set.a
108
109     if fuzzy_set.a==fuzzy_set.b
110       b=fuzzy_set.b
111     else
112       b= height*(fuzzy_set.b - a) + a
113     end
114
115     d= fuzzy_set.d
116
117     if fuzzy_set.c==fuzzy_set.d
118       c=fuzzy_set.c
119     else
120       c= d - height*(fuzzy_set.d - fuzzy_set.c)
121     end
122
123     big_base = d - a
124     small_base = big_base -((b-a)+(d-c))
```

```
125
126       #Trapezoid area= height*(big base + small base)/2
127       area= height*(big_base+small_base)/2
128       centroid = a + (d-a)/2
129
130       return [area, centroid]
131
132     end#def calculate_area_centroid
133
134     #This evaluates the antecedents in every rule and
135     #gets the minimun value. After that, it calculates the
136     #area and centroid for every rule by calling
137     #the calculate_area_centroid fuction
138     def rule_evaluation
139       systems.each do |key_s, s|#Systems
140         s.rules_hash.each do |key_r, r|#Rules
141           r.antecedents.each do |a| #Antecedents
142             r.ant_value = [r.ant_value, a.membership_d].min #Having
                    minimun value between all the antecedents for that
                    rule
143           end#each Antecedents
144           r.consequents.each do |c| #Consequents
145             r.area_centroid = calculate_area_centroid(s, c, r.
                    ant_value)
146           end#each Consequents
147         end#each Rules
148       end#each Systems
149     end#def rule_evaluation
150
151     #Given all the areas for each consequents in the system's rules,
152     #it calculates the final consequents.
153     def defuzzification
154       products_sum=0
155       areas_sum=0
156       systems.each do |key_s, s|#Systems
157         s.rules_hash.each do |key_r, r|#Rules
158           products_sum += r.area_centroid[0]*r.area_centroid[1] #Area
                  * Centroid
159           areas_sum += r.area_centroid[0] #Area
160         end#each Rules
161         s.conclusion=products_sum/areas_sum
162         products_sum=0
163         areas_sum=0
164         design.values_bits[systems[s.name].bit_position]=s.conclusion
              #We store the result in an array in the Design object
165       end#each Systems
166     end#def defuzzification
167
168   end #Fie class
```

A.3. Clase Design

```
1  require 'rexml/document'
2  include REXML
3
4  class Design
5    attr_accessor :groups, :width, :height, :vertical, :background, :
       gen_bits, :values_bits, :ratio, :image_colors
6    attr_accessor :primary_font, :secondary_font, :image_shape, :
       predominant_shape, :colors, :bg_colors
7    attr_accessor :sides_img_margin, :sides_margin, :upper_img_margin
       , :upper_margin
8    attr_accessor :white_space_w, :white_space_h, :white_space_x, :
       white_space_y, :init_questions, :main_image
9    attr_accessor :interest_point, :interest_box, :visitor_id, :
       current_number
10
11   def initialize()
12     @image_shape='curved'
13     @groups = Array.new
14     @values_bits = []
15     @gen_bits = Genes.new
16     @colors = []
17     @bg_colors = []
18     @image_colors = []
19     @init_questions = {}
20     @current_number = 0
21     @interest_point = []
22   end
23
24   def add_group(group)
25     @groups << group
26   end
27
28   def add_init_questions(symbol, value)
29     @init_questions[symbol] = value
30   end
31
32   def add_main_image(image_path)
33     @main_image = DesignImage.new(image_path,Constants::Main_image)
34
35     face_analysis = `./app/models/openCV/face/face #{@main_image.
         source}`
36     blob_analysis = `./app/models/openCV/blob/blob #{@main_image.
         source}`
37
38     face_array = face_analysis.split(' ')
39     blob_array = blob_analysis.split(' ')
40
41     if (face_array[2].to_i !=0 and face_array[3].to_i!=0) or (
         blob_array[2].to_i !=0 and blob_array[3].to_i!=0)
```

```
42        if face_array[2].to_i !=0 and face_array[3].to_i!=0
43           @main_image.analysis = face_analysis
44        else
45          @main_image.analysis = blob_analysis
46        end
47
48        metrics = @main_image.analysis.split(' ')
49
50        picture = Image.read(@main_image.source).first
51        gc = Draw.new
52        gc.stroke('black')
53        gc.stroke_width(3)
54        gc.fill('none')
55        gc.rectangle(metrics[0].to_i, metrics[1].to_i, metrics[0].
             to_i+metrics[2].to_i,metrics[1].to_i+metrics[3].to_i)
56
57        gc2 = Draw.new
58        gc2.stroke('white')
59        gc2.stroke_width(1)
60        gc2.fill('none')
61        gc2.rectangle(metrics[0].to_i, metrics[1].to_i, metrics[0].
             to_i+metrics[2].to_i,metrics[1].to_i+metrics[3].to_i)
62
63        gc.draw(picture)
64        gc2.draw(picture)
65        picture.write('public/images/uploaded_pictures/'+@visitor_id+
             '_tmp.png')
66      else
67        @main_image.analysis = nil
68      end
69
70    end
71
72    def mod_gen_bits
73      gen_bits.mod_entries_bits(@main_image.width, @main_image.height
          )
74    end
75
76    #This is the main function of the class
77    #here is where all the functions are called
78    def generate_design_values(xml_colors=false, xml_fonts=false)
79      #DOCUMENT
80      generate_ratio #Generates the ratio of the document
81      generate_orientation #Generates if the design has horizontal or
            vertical orientation
82      generate_sizes #Generates the sizes based on the ratio
83      generate_margin
84      if values_bits[Constants::V_Zooming]>4.5 and values_bits[
           Constants::V_Type_of_BG] > 4.5 #Plain BG and zooming
           activated
85        modify_image
86      end
```

```
87      generate_images_sizes #Adjust the sizes based on the design
            sizes
88      generate_predominant_shape #Straight, curved, etc.
89      #COLORS
90      if not xml_colors
91        generate_color_scheme
92      end
93      if values_bits[Constants::V_Texts_BG]>3.5 and not xml_colors
94        generate_text_backgrounds
95      end
96      if values_bits[Constants::V_Type_of_BG]>4.5
97        improve_legibility
98      end
99      #FONTS
100     if not xml_fonts
101       generate_family_fonts
102     end
103
104     groups.each do |g| #Groups
105       g.items.each do |i| #Items
106         if i.instance_of? Letters
107           generate_font(i)
108         end
109       end
110     end
111
112     #####PRECOMPOSITION
113     composite_main_image_position
114     generate_white_spaces
115
116     new_ratio = @groups[0].items[0].font_size.to_f
117     @groups[0].items[0].font_size = Letters.new.adjust_font_size(
            @groups[0].items[0], @white_space_w) #We check the length
            for the title
118     if @groups[0].items.length>1
119       @groups[0].items[1].font_size = Letters.new.adjust_font_size(
              @groups[0].items[1], @white_space_w)
120     end
121     new_ratio = @groups[0].items[0].font_size.to_f/new_ratio.to_f <
            0.60 ? 0.60 : @groups[0].items[0].font_size.to_f/new_ratio.
            to_f
122     if new_ratio != 1
123       groups.each do |g| #Groups
124         g.items.each do |i| #Items
125           if i.instance_of? Letters and i.type != 0
126             i.adapt_font_size(new_ratio)
127           end
128         end
129       end
130     end
131
132     #COMPOSITION
133     initialize_composition_values
```

```
134       if values_bits[Constants::V_Type_of_BG] > 4.5#Plain bg
135         adapt_design_size
136       end
137       calculate_interest_point
138       start_composition
139     end
140
141     def generate_margin
142       if values_bits[Constants::V_Type_of_BG] > 4.5 #Plain bg
143         if values_bits[Constants::V_Margins]<1.5 #no sides_img_margin
144           @sides_img_margin=0
145           @upper_img_margin=0
146           @sides_margin=20
147           @upper_margin=20
148         elsif values_bits[Constants::V_Margins]>=1.5 and values_bits[
              Constants::V_Margins]<4.5 #low
149           @sides_img_margin=20
150           @upper_img_margin=20
151           @sides_margin=20
152           @upper_margin=20
153         elsif values_bits[Constants::V_Margins]>=4.5 and values_bits[
              Constants::V_Margins]<8 #medium
154           @sides_img_margin=40
155           @upper_img_margin=20
156           @sides_margin=40
157           @upper_margin=20
158         else #high
159           @sides_img_margin=40
160           @upper_img_margin=40
161           @sides_margin=40
162           @upper_margin=20
163         end
164
165         if not @vertical
166           @sides_margin= 20
167         end
168
169       else #Picture as background
170         @sides_img_margin=0
171         @upper_img_margin=0
172         @sides_margin=20
173         @upper_margin=20
174       end
175     end
176
177     #Zooming to the image
178     def modify_image
179       if @vertical
180         @main_image = @main_image.zooming_v
181       else
182         @main_image = @main_image.zooming_h
183       end
184     end
```

```
185
186   def generate_images_sizes
187     previous_width = @main_image.width
188     if values_bits[Constants::V_Type_of_BG] > 4.5 #Plain background
189       if @width>@height || (@width==@height && @main_image.height >
              @main_image.width)
190         img_ratio= @main_image.height.to_f/(@height -
              upper_img_margin*2).to_f
191         @main_image.height = @height - upper_img_margin*2
192         @main_image.width = @main_image.width.to_f / img_ratio
193       elsif @width<@height || (@width==@height && @main_image.width
              > @main_image.height)
194         img_ratio= @main_image.width.to_f/(@width -
              sides_img_margin*2).to_f
195         @main_image.width = @width - sides_img_margin*2
196         @main_image.height = @main_image.height.to_f / img_ratio
197       elsif @width==@height and @main_image.width == @main_image.
              height
198         modify_image
199         generate_images_sizes
200       end
201     else
202       @main_image.height = @height
203       @main_image.width = @width
204     end
205
206     @main_image.readapt_interest_box(@width.to_f/previous_width.
          to_f)
207   end
208
209   def generate_sizes
210     if values_bits[Constants::V_Type_of_BG] > 4.5 #Plain background
211       if ratio == 1.0
212         @width = 800
213         @height = 800
214       elsif ratio == 1.15
215         @width=920
216         @height=800
217       elsif ratio == 1.33
218         @width=930
219         @height=700
220       elsif ratio == 1.50
221         @width=1050
222         @height=700
223       else
224         @width=1200
225         @height=700
226       end
227
228       if @vertical #if vertical is true, then the design is
              vertical oriented
229         aux=@width
230         @width=@height
```

```
231           @height=aux
232         end
233       else #Picture as background
234         if @main_image.vertical
235           @width = 900/(@main_image.height.to_f/@main_image.width.
                to_f)
236           @height = 900
237         else
238           @width = 900
239           @height = 900/(@main_image.width.to_f/@main_image.height.
                to_f)
240         end
241       end
242     end
243
244     def generate_orientation
245       if values_bits[Constants::V_Design_orientation]<4.5
246         @vertical=false #Horizontal
247       else
248         @vertical=true #Vertial
249       end
250     end
251
252     def generate_ratio
253       if values_bits[Constants::V_Design_ratio]==2
254         @ratio=1
255       elsif values_bits[Constants::V_Design_ratio]>2 and values_bits[
             Constants::V_Design_ratio]<4.5
256         @ratio=1.15
257       elsif values_bits[Constants::V_Design_ratio]==4.5
258         @ratio=1.33
259       elsif values_bits[Constants::V_Design_ratio]>4.5 and
              values_bits[Constants::V_Design_ratio]<6.5
260         @ratio=1.50
261       elsif values_bits[Constants::V_Design_ratio]==6.5
262         @ratio=1.77
263       end
264
265     end
266
267     #Are colors suitable?
268     def improve_legibility
269       @colors.length.times do |n|
270         if n!=0 and not colors[0].is_contrasted_enough(colors[n])
271           if not colors[0].is_contrasted_enough(colors[n]) and
                  values_bits[Constants::V_Darkness]<=4.5
272             colors[n]=colors[n].get_dark(20)
273           elsif not colors[0].is_contrasted_enough(colors[n]) and
                  values_bits[Constants::V_Darkness]>4.5
274             colors[n]=colors[n].get_pale(90)
275           end #if not end
276       end#If end
277       end#color.length do end
```

```
278     end
279
280     #Here is where it gets the main colors
281     #from the main image
282     def obtain_main_image_colors
283       pic = ImageList.new(@main_image.source)
284       pic.resize!(0.20)
285       #Reducing picture colors
286       pic = pic.quantize(5, Magick::RGBColorspace, 0, 0, false)
287       #By modulating them we get better colors
288       pic = pic.modulate(1.2, 1.8, 1.0) #L S H
289       #Background color will be the most used color in the picture
290       color_pixels = pic.color_histogram
291
292       max=0
293       background_px = 0
294
295       color_pixels.each do |k,v|
296         if v >max
297           max=v
298           background_px = k
299         end
300       end
301
302       @image_colors << Colour.new(background_px.red, background_px.
              green, background_px.blue) #image_colors[0] is the
              background
303
304       color_pixels.each do |k,v|
305         hsl = k.to_hsla
306         if hsl[2]>=35 and hsl[2]<=75
307           @image_colors << Colour.new(k.red, k.green, k.blue)
308         end
309       end
310
311       @image_colors[1] = @image_colors[1]==nil ? @image_colors[0] :
              @image_colors[1]
312       @image_colors[2] = @image_colors[2]==nil ? @image_colors[0] :
              @image_colors[2]
313
314     end
315
316     #The main Color function
317     def generate_color_scheme
318       if values_bits[Constants::V_Use_of_colors]<=2 #No colors
319         if values_bits[Constants::V_Darkness]>4.5 #Dark design
320           if values_bits[Constants::V_Darkness]>4.5 #BW
321             @colors[0]=Colour.new
322           else #Dark
323             @colors[0]=Colour.new.get_random_color.get_dark.
                    get_desaturated
324           end
325         else #Light design
```

```
326           if values_bits[Constants::V_Darkness]>4.5 #BW
327             @colors[0]=Colour.new(255,255,255)
328           else #Pale
329             @colors[0]=Colour.new.get_random_color.get_pale.
                  get_desaturated
330           end
331         end
332         if values_bits[Constants::V_Darkness]<=4.5 #Dark
333           @colors[1]=Colour.new
334           @colors[2]=Colour.new
335           @colors[3]=Colour.new
336           @colors[4]=Colour.new(40,40,40)
337           @colors[5]=Colour.new(60,60,60)
338           @colors[6]=Colour.new(80,80,80)
339           @colors[7]=Colour.new
340         elsif values_bits[Constants::V_Darkness]>=4.5 #Light
341           @colors[1]=Colour.new(255,255,255)
342           @colors[2]=Colour.new(255,255,255)
343           @colors[3]=Colour.new(255,255,255)
344           @colors[4]=Colour.new(230,230,230)
345           @colors[5]=Colour.new(220,220,220)
346           @colors[6]=Colour.new(210,210,210)
347           @colors[7]=Colour.new(255,255,255)
348         end
349
350       else #Colors
351         #Main color (colors[1] - colors[3])
352         if values_bits[Constants::V_Main_color]<=2 #Complementary
              main pic
353           @colors[1] = @image_colors[1].get_complementary
354         elsif values_bits[Constants::V_Main_color]>2 and values_bits[
              Constants::V_Main_color]<=5 #Contrast main pic
355           @colors[1] = @image_colors[1].get_contrasted
356         elsif values_bits[Constants::V_Main_color]>5 and values_bits[
              Constants::V_Main_color]<=8 #main_pic
357           @colors[1] = @image_colors[1]
358         else #random
359           @colors[1] = Colour.new.get_random_color()
360         end
361         @colors[2] = colors[1].get_lighter
362         @colors[3] = colors[1].get_darker
363
364         #Accent colors (colors[4] - colors[6])
365         if values_bits[Constants::V_Accent_colors]<=2 #secondary
366           accent_colors = @image_colors[2].get_analogous
367           @colors[4]=@image_colors[2]
368           2.times do |n|
369             @colors[n+5] = accent_colors[n]
370           end
371         elsif values_bits[Constants::V_Accent_colors]>2 and
              values_bits[Constants::V_Accent_colors]<=4.5 #triadic
372           accent_colors = colors[1].get_triadic
373           3.times do |n|
```

```
374           @colors[n+4] = accent_colors[n]
375         end
376       elsif values_bits[Constants::V_Accent_colors]>4.5 and
             values_bits[Constants::V_Accent_colors]<=8 #analogous
377         accent_colors = colors[1].get_analogous
378         3.times do |n|
379           @colors[n+4] = accent_colors[n]
380         end
381       else #complementary
382         @colors[4] = colors[1].get_complementary
383         accent_colors = colors[4].get_analogous
384         2.times do |n|
385           @colors[n+5] = accent_colors[n]
386         end
387       end
388
389       #Background color (colors[0])
390       if values_bits[Constants::V_BG_color]<7 #Plain background
391         if values_bits[Constants::V_Darkness]>4.5 #Dark design
392           if values_bits[Constants::V_BG_color]>4.5 #BW
393             @colors[0]=Colour.new
394           else #Dark
395             @colors[0]=colors[1].get_dark
396           end
397         else #Light design
398           if values_bits[Constants::V_BG_color]>4.5 #BW
399             @colors[0]=Colour.new(255,255,255)
400           else #Pale
401             @colors[0]=colors[1].get_pale
402           end
403         end
404       else
405         @colors[0]=Colour.new
406       end
407
408       #Text color (colors[7])
409       if values_bits[Constants::V_Darkness]>=4.5 #Contrast with
             plain background
410         if values_bits[Constants::V_Type_of_BG]<4.5 #Light bg
411           @colors[7]=Colour.new
412         else #Dark bg
413           @colors[7]=Colour.new(255,255,255)
414         end
415       else #Contrast with picture as background
416         if @image_colors[0].get_lightness > Constants::
              Min_lightness
417           @colors[7]=Colour.new
418         else
419           @colors[7]=Colour.new(255,255,255)
420         end
421       end
422     end
423
```

```
424       @background = @colors[0]
425     end
426
427     def generate_text_backgrounds
428       @colors.length.times do |n|
429         if @colors[n].get_lightness< Constants::Min_lightness
430           @bg_colors[n] = Colour.new(255,255,255)
431         else
432           @bg_colors[n] = Colour.new
433         end
434       end
435     end
436
437     def generate_predominant_shape
438       if values_bits[Constants::V_Predominant_shape]<2
439         @predominant_shape='curved'
440       elsif values_bits[Constants::V_Predominant_shape]>2 and
             values_bits[Constants::V_Predominant_shape]<=4.5
441         @predominant_shape='slightly_curved'
442       elsif values_bits[Constants::V_Predominant_shape]>4.5 and
             values_bits[Constants::V_Predominant_shape]<=6.5
443         @predominant_shape='slightly_straight'
444       else
445         @predominant_shape='straight'
446       end
447     end
448
449     def generate_family_fonts
450       if values_bits[Constants::V_Primary_font_style]<2
451         primary_font_style='graphic'
452       elsif values_bits[Constants::V_Primary_font_style]>2 and
             values_bits[Constants::V_Primary_font_style]<6.5
453         primary_font_style='script'
454       else
455         primary_font_style='others'
456       end
457
458       secondary_font_style='others'
459
460       @primary_font=get_font_name(primary_font_style,
             predominant_shape)   #Gets the name of the font
461       if values_bits[Constants::V_Secondary_font_style]<4.5 #same
462         @secondary_font=get_font_name(secondary_font_style,
              predominant_shape)
463       else #different
464         @secondary_font=get_font_name(secondary_font_style,
              image_shape)
465       end
466     end
467
468     def get_font_name(font_style, shape)
469       possible_fonts = []
470       doc = Document.new(File.new('public/xml/fonts.xml'))
```

```
471       root=doc.root
472
473       root.elements.each do |font|
474         if font.attributes["type"]==font_style and font.attributes["
              shape"]==shape
475           possible_fonts << font.attributes["name"]
476         elsif font.attributes["type"]==font_style and (font_style=='
              graphic' or font_style=='script')
477           possible_fonts << font.attributes["name"]
478         end
479       end
480
481       return possible_fonts[rand(possible_fonts.length)]
482     end
483
484     #Formatting the text...
485     def generate_font(item)
486       item.generate_font_size(values_bits)       #Size
487       item.generate_font_weight(values_bits)     #Weight
488       item.generate_font_filename(values_bits, primary_font,
            secondary_font)    #Font Face
489       item.generate_font_color(values_bits[Constants::V_Use_of_colors
            ], values_bits[Constants::V_Texts_BG], colors, bg_colors,
            values_bits[Constants::V_type_of_BG_decoration])
490     end
491
492     def composite_main_image_position
493       if values_bits[Constants::V_Type_of_BG] > 4.5 #Plain background
494         if values_bits[Constants::V_Image_position]< 4.5 #Normal
              composition (Left and Up)
495           @main_image.x_pos = sides_img_margin
496           @main_image.y_pos = upper_img_margin
497         else #Alternative composition (Right and Down)
498           if @vertical #Horizontal
499             @main_image.x_pos = sides_img_margin
500             @main_image.y_pos = @height - (upper_img_margin +
                  @main_image.height)
501           else #Vertical
502             @main_image.x_pos = @width - (sides_img_margin +
                  @main_image.width)
503             @main_image.y_pos = upper_img_margin
504           end
505         end
506       else # Picture as background
507         @main_image.x_pos = 0
508         @main_image.y_pos = 0
509       end
510     end
511
512     def initialize_composition_values
513       @groups.each do |g|
514         g.calculate_sizes
515       end
```

```
516
517       #Interest box
518       @interest_box = Group.new
519       @interest_box.x_pos = @main_image.interest_box_x.to_i
520       @interest_box.y_pos = @main_image.interest_box_y.to_i
521       @interest_box.x_average_pos = @main_image.
             interest_box_average_x
522       @interest_box.y_average_pos = @main_image.
             interest_box_average_y
523       @interest_box.width = @main_image.interest_box_width.to_i
524       @interest_box.height = @main_image.interest_box_height.to_i
525       @interest_box.weight = @main_image.interest_box_width.to_i
526       @interest_box.normal_group = false
527     end
528
529     #This returns the difference between
530     #free space and groups space
531     def space_factor
532       groups_area = values_bits[Constants::V_Type_of_BG] > 4.5 ? 0 :
             @interest_box.width * @interest_box.height
533       @groups.each do |g|
534         groups_area += g.width * g.height
535       end
536       white_space_area = @white_space_w * @white_space_h
537
538       return groups_area/white_space_area
539     end
540
541     #Just in case there's too much white space...
542     def adapt_design_size
543       while space_factor < Constants::Min_allowed_factor
544         if @vertical
545           @height /= Constants::Shrink_factor
546           @height += @height%20 == 0 ? 0 : 20-@height%20
547         elsif not @vertical
548           @width /= Constants::Shrink_factor
549           @width += @width%20 == 0 ? 0 : 20-@width%20
550         end
551         composite_main_image_position
552         generate_white_spaces
553         white_space_area = white_space_w * white_space_h
554       end
555     end
556
557     def enlarge_white_space(width_grow, height_grow)
558       if height_grow!=0
559         @height += height_grow
560         @height += @height%20 == 0 ? 0 : 20-@height%20
561       elsif width_grow!=0
562         @width += width_grow
563         @width += @width%20 == 0 ? 0 : 20-@width%20
564       end
565
```

```
566       composite_main_image_position
567       generate_white_spaces
568     end
569
570     def reduce_font_size
571       @groups.each do |g|
572         g.items.each do |i|
573           if i.instance_of? Letters
574             if i.type<=1 #Title and subtitle
575               i.font_size *= Constants::Shrink_font_factor_big
576             else #Highlighted, Normal and Notes
577               i.font_size *= Constants::Shrink_font_factor_small
578             end
579           end
580         end
581         g.calculate_sizes
582       end
583     end
584
585     def calculate_interest_point
586       randomization = Kernel.rand
587       found = false
588       n=0
589
590       while not found
591         if randomization <= @main_image.rndm_interest[n][0]
592           found = true
593           @interest_point[0] = @main_image.rndm_interest[n][1]
594           @interest_point[1] = @main_image.rndm_interest[n][2]
595         end
596         n+=1
597       end
598     end
599
600     def create_arguments(current_solution)
601       args = ''
602       args += @white_space_x.to_i.to_s+' '+@white_space_y.to_i.to_s+'
              '+@white_space_w.to_i.to_s+' '+@white_space_h.to_i.to_s+' '
603       args += @interest_box.x_pos.to_i.to_s+' '+@interest_box.y_pos.
             to_i.to_s+' '+@interest_box.width.to_i.to_s+' '+
             @interest_box.height.to_i.to_s+' '
604       args += values_bits[Constants::V_Type_of_BG] > 4.5 ? '1 ' : '0
             ' #1, plain bg. 0, image as bg
605       args += @vertical ? '1 ' : '0 ' #1, Vertical; 0, horizontal
606
607       current_solution.each do |g|
608         args += g.weight.to_i.to_s+' '+g.x_pos.to_i.to_s+' '+g.y_pos.
               to_i.to_s+' '+g.width.to_i.to_s+' '+g.height.to_i.to_s+' '
609         args += g.x_average_pos.to_i.to_s+' '+g.y_average_pos.to_i.
               to_s+' '
610         args += g.normal_group ? '1 ' : '0 '
611       end
612
```

```
613       return args
614     end
615
616     #This is where the C application is executed
617     def start_simulated_annealing(args)
618       #The app's output is saved in "result" variable
619       result = `./app/models/C/simulated_annealing #{args}`
620
621       result_array = result.split(' ')
622
623       return result_array
624     end
625
626     #This is where the composition begins
627     def start_composition
628       current_solution = generate_initial_solution
629       calculate_weight(current_solution)
630       args = create_arguments(current_solution)
631       result_array = start_simulated_annealing(args)
632
633       solution_cost = result_array[result_array.length-3].to_f
634       width_grow = result_array[result_array.length-2].to_i
635       height_grow = result_array[result_array.length-1].to_i
636       if(width_grow!=0 or height_grow !=0)
637         enlarge_white_space(width_grow, height_grow)
638       end
639       times_to_try = 3
640
641       while solution_cost > 1.0 and times_to_try>0
642         if values_bits[Constants::V_Type_of_BG]<=4.5
643           reduce_font_size
644         end
645
646         current_solution = generate_initial_solution
647         calculate_weight(current_solution)
648
649         args = create_arguments(current_solution)
650         result_array = start_simulated_annealing(args)
651
652         solution_cost = result_array[result_array.length-3 ].to_f
653
654         times_to_try -= 1
655       end
656
657       loop_n = (result_array.length - 2)/3
658
659       loop_n.times do |n|
660         @groups[n].x_pos = result_array[0+n*3].to_i;
661         @groups[n].y_pos = result_array[1+n*3].to_i;
662         @groups[n].alignment = result_array[2+n*3].to_i==0 ? 'right'
              : 'left';
663       end
664
```

```
665       if values_bits[Constants::V_Type_of_BG] > 4.5
666         cut_white_space_edges
667       end
668
669       @groups.length.times do |n|
670         @groups[n].process_texts_position(@width)
671       end
672
673     end
674
675     #Removes the free space near the edges
676     def cut_white_space_edges
677       x1_array = Array.new
678       x2_array = Array.new
679       y1_array = Array.new
680       y2_array = Array.new
681
682       @groups.each do |g|
683         x1_array << g.x_pos
684         x2_array << g.x_pos + g.width
685         y1_array << g.y_pos
686         y2_array << g.y_pos + g.height
687       end
688
689       if @vertical
690         #Normal position of the image (up)
691         if values_bits[Constants::V_Image_position]< 4.5
692           @height = y2_array.max + @upper_margin
693         #Alternative position(down)
694         else
695           new_height = @height - (y1_array.min-@upper_margin)
696
697           @groups.each do |g|
698             g.y_pos -= (@height - new_height)
699           end
700           @main_image.y_pos -= (@height - new_height)
701
702           @height = new_height
703         end
704       else
705         #Normal position of the image (left)
706         if values_bits[Constants::V_Image_position]< 4.5
707           @width = x2_array.max + @sides_margin
708         #Alternative position of the image (right)
709         else
710           new_width = @width - (x1_array.min-@sides_margin)
711
712           @groups.each do |g|
713             g.x_pos -= (@width - new_width)
714           end
715           @main_image.x_pos -= (@width - new_width)
716
717           @width = new_width
```

```
718         end
719       end
720     end
721
722     #This just puts the groups one after another
723     #and this becomes the initial solution
724     def generate_initial_solution
725       aux_groups = @groups.groups_deep_clone
726       x=@white_space_x
727       y=@white_space_y
728
729       aux_groups.each do |g|
730         g.allocate_this(x,y)
731         y+=g.height
732       end
733
734       if values_bits[Constants::V_Type_of_BG] <= 4.5 #Image as
             background
735         aux_groups << @interest_box
736       end
737
738       return aux_groups
739     end
740
741
742     def calculate_weight(groups_array)
743       groups_array.each do |g|
744         if g.normal_group
745           g.calculate_weight
746         end
747       end
748     end
749
750     #Depending on the margin, image size and
751     #document size, this generates the
752     #free space of the document
753     def generate_white_spaces
754       #Plain background
755       if values_bits[Constants::V_Type_of_BG] > 4.5
756         if @vertical
757           @white_space_w = @width - @sides_margin*2
758           @white_space_h = @height - (@main_image.height +
                @upper_img_margin + @upper_margin*2)
759           @white_space_x = @sides_margin
760           @white_space_y = values_bits[Constants::V_Image_position] <
                4.5 ? @main_image.height+@upper_margin+
                @upper_img_margin : @upper_margin
761         else
762           @white_space_w = @width - (@main_image.width +
                @sides_img_margin + @sides_margin*2)
763           @white_space_h = @height - @upper_margin*2
764           @white_space_x = values_bits[Constants::V_Image_position] <
                4.5 ? @main_image.width+2*@sides_margin : @sides_margin
```

```
765           @white_space_y = @upper_margin
766         end
767       #Image as background
768       else
769         @white_space_w= @width - @sides_margin*2
770         @white_space_h= @height - @upper_margin*2
771         @white_space_x= @sides_margin
772         @white_space_y= @upper_margin
773       end
774     end
775
776     #Here the XML is created
777     def get_xml
778       header = "<DESIGN></DESIGN>"
779       path = 'public/users_xml/'+@visitor_id+'.xml'
780       f = File.new(path, 'w+')
781       f.write header
782       f.close
783
784       doc = Document.new File.new(path)
785       root = doc.root
786
787       ###Genes
788       gen_bits_string = ''
789       gen_bits.each do |g|
790         gen_bits_string += g.to_s
791       end
792
793       genes = Element.new "genes"
794       genes.attributes["string"] = gen_bits_string
795
796       root.add_element genes
797
798       ###Colors
799       foreground = Element.new "foreground"
800       @colors.length.times do |n|
801         color = Element.new "fg_color"
802         color.attributes["red"]= @colors[n].red.to_s
803         color.attributes["green"]= @colors[n].green.to_s
804         color.attributes["blue"]= @colors[n].blue.to_s
805         color.attributes["n"]=n.to_s
806
807         foreground.add_element color
808       end
809       root.add_element foreground
810
811       background = Element.new "background"
812       @bg_colors.length.times do |n|
813         color = Element.new "bg_color"
814         color.attributes["red"]= @bg_colors[n].red.to_s
815         color.attributes["green"]= @bg_colors[n].green.to_s
816         color.attributes["blue"]= @bg_colors[n].blue.to_s
817         color.attributes["n"]=n.to_s
```

```
818
819         background.add_element color
820       end
821       root.add_element background
822
823       ###Fonts
824       fonts = Element.new "fonts"
825
826       p = Element.new "font"
827       p.attributes["type"] = "primary"
828       p.attributes["font"] = @primary_font
829       fonts.add_element p
830       s = Element.new "font"
831       s.attributes["type"] = "secondary"
832       s.attributes["font"] = @secondary_font
833       fonts.add_element s
834
835       root.add_element fonts
836
837       File.open(path, 'w') do |f|
838          f.puts root
839       end
840
841       return path
842
843     end
844
845   end #Class end
```

A.4. Módulo de búsqueda local: Enfriamiento Simulado

```
 1  #include <stdio.h>
 2  #include <string.h>
 3  #include <stdlib.h>
 4  #include <math.h>
 5
 6  #define VBN 10000
 7  #define RIGHT 0
 8  #define LEFT 1
 9  #define ALIGNMENT_COST_PER_GROUP 200
10  #define ALIGNMENT_COMPLETE_POOR 90
11  #define ALIGNMENT_COMPLETE_GOOD 120
12  #define ALIGNMENT_COMPLETE_GREAT 150
13  #define ALIGNMENT_HALFWAY 45
14  #define ALIGNMENT_MARGINS 20
15  #define ALIGNMENT_INTEREST 10
16  #define ALIGNMENT_PERCENTAGE 0.7
17  #define BALANCE_PERCENTAGE 0.3
18  #define MAX 100
```

```
19 #define PLAIN   1.0
20 #define BACKGROUND  0.0
21 #define FALSE  0
22 #define TRUE  1
23 #define VERTICAL  1
24 #define HORIZONTAL  0
25 #define SMALL_GROWTH 0.20
26 #define MEDIUM_GROWTH 0.25
27 #define BIG_GROWTH 0.30
28
29 /*Structs*/
30 struct group{
31  int weight;
32  int x_pos;
33  int y_pos;
34  int width;
35  int height;
36  int normalGroup;
37  int x_avg;
38  int y_avg;
39  int alignment;
40 };
41
42 struct space{
43  int x;
44  int y;
45  int w;
46  int h;
47 };
48
49 int l(int t);
50 void alpha(float *t);
51 void chooseNeighbour(struct group *candidate, struct space space,
      int size, float t);
52 float cost(struct group *candidate, struct space space, struct
      space interestBox, int size, int designType);
53 int checkBorders(struct group g, struct space *space, int size);
54 int checkSpace(struct group *candidate, struct space *space, int
      orientation, int size);
55 int getOverlapCost(struct group *candidate, int size);
56 float getBalanceCost(struct group *candidate, struct space space,
      int size);
57 float getAlignmentCost(struct group *candidate, struct space space,
       struct space interestBox, int size, int designType);
58 int isOverlapping(struct group one, struct group two);
59 char * itoa (int i);
60
61 int main (int argc, char const *argv[]){
62  struct group currentSolution[(argc-11)/8];
63  struct group bestSolution[(argc-11)/8];
64  struct group candidateSolution[(argc-11)/8];
65  struct space whiteSpace,originalWhiteSpace,interestBox;
66  int nGroups=(argc-11)/8;
```

```
67   int i, designType, orientation, keepChecking=TRUE;
68   float tEnd= 10.0, t= 300.0, delta, calc, random;
69   char result[MAX];
70
71   whiteSpace.x=atoi(argv[1]); whiteSpace.y=atoi(argv[2]);
72   whiteSpace.w=atoi(argv[3]); whiteSpace.h=atoi(argv[4]);
73
74   interestBox.x=atoi(argv[5]); interestBox.y=atoi(argv[6]);
75   interestBox.w=atoi(argv[7]); interestBox.h=atoi(argv[8]);
76
77   designType = atoi(argv[9]);
78   orientation = atoi(argv[10]);
79
80   srand(getpid());
81
82   for(i=0;i<nGroups;i++){
83    currentSolution[i].weight =   atoi(argv[11 + i*8]);
84    currentSolution[i].x_pos =    atoi(argv[12 + i*8]);
85    currentSolution[i].y_pos =    atoi(argv[13 + i*8]);
86    currentSolution[i].width =    atoi(argv[14 + i*8]);
87    currentSolution[i].height =   atoi(argv[15 + i*8]);
88    currentSolution[i].x_avg =    atoi(argv[16 + i*8]);
89    currentSolution[i].y_avg =    atoi(argv[17 + i*8]);
90    currentSolution[i].normalGroup = atoi(argv[18 + i*8]);
91    currentSolution[i].alignment = 1;
92   }
93
94   originalWhiteSpace = whiteSpace;
95
96   /*****Simulated Annealing*****/
97   memcpy(&bestSolution, &currentSolution, sizeof(currentSolution));
98
99   while(t >= tEnd){
100   if((int)t%75 == 0 && t!=300.0 && designType==PLAIN &&
        keepChecking==TRUE){
101    keepChecking = checkSpace(bestSolution, &whiteSpace, orientation
        , nGroups);
102   }
103
104   for(i = 0; i < l(t); i++){
105    memcpy(&candidateSolution, &currentSolution, sizeof(
        currentSolution));
106    chooseNeighbour(candidateSolution, whiteSpace, nGroups, t);
107    delta = cost(candidateSolution, whiteSpace, interestBox, nGroups
        , designType) - cost(currentSolution, whiteSpace, interestBox
        , nGroups, designType);
108    calc = exp(-(delta)/t);
109    random = drand48();
110    if(random<calc || delta <0)
111     memcpy(&currentSolution, &candidateSolution, sizeof(
         currentSolution));
112    if(cost(currentSolution, whiteSpace, interestBox, nGroups,
        designType) < cost(bestSolution, whiteSpace, interestBox,
```

```
        nGroups, designType))
113       memcpy(&bestSolution, &currentSolution, sizeof(currentSolution)
            );
114     }
115     alpha(&t);
116   }
117
118   /*Printing the output. This will be get by the main process*/
119
120   for(i=0;i<nGroups;i++){
121     if(bestSolution[i].normalGroup == 1){
122       printf("%i ", bestSolution[i].x_pos);
123       printf("%i ", bestSolution[i].y_pos);
124       printf("%i ", bestSolution[i].alignment);
125     }
126   }
127
128   /*Cost*/
129   printf("%f ", cost(bestSolution, whiteSpace, interestBox, nGroups,
          designType));
130   /*Size modifications*/
131   printf("%i ", whiteSpace.w - originalWhiteSpace.w);  /*Width*/
132   printf("%i ", whiteSpace.h - originalWhiteSpace.h);  /*Height*/
133
134   return 0;
135 }
136
137 /*Cheking function*/
138 int checkSpace(struct group *solution, struct space *space, int
      orientation, int size){
139   int i=0, isOut[size], nOut=0;
140   float factor, grow=0;
141
142   for(i = 0; i < size; i++){
143     if(checkBorders(solution[i], space, size)==TRUE){
144       nOut++;
145       isOut[i]=TRUE;
146     }
147     else
148       isOut[i]=FALSE;
149   }
150
151   if(nOut == 1)
152     factor = BIG_GROWTH;
153   else if(nOut == 2)
154     factor = MEDIUM_GROWTH;
155   else if(nOut >= 3)
156     factor = SMALL_GROWTH;
157
158   for(i=0;i<size;i++)
159     if(isOut[i]==TRUE)
160       grow+= orientation==VERTICAL? solution[i].height*factor :
            solution[i].width*factor;
```

```
161
162   grow += (int)grow%20==0?0: 20-(int)grow%20;
163
164   if(orientation==VERTICAL)
165    space->h+=grow;
166   else
167    space->w+=grow;
168
169   if(nOut!=0){
170    return TRUE;
171   }
172   else
173    return FALSE;
174  }
175
176  int checkBorders(struct group g, struct space *space, int size){
177   int i, isOut=FALSE;
178
179   if(g.x_pos < space->x ||
180    g.x_pos + g.width > space->x + space->w ||
181    g.y_pos < space->y ||
182    g.y_pos + g.height > space->y + space->h ){
183     isOut=TRUE;
184   }
185
186   return isOut;
187  }
188
189  float cost(struct group *candidate, struct space space, struct
       space interestBox, int size, int designType){
190   int overlapCost;
191   float balanceCost, alignmentCost;
192
193   /*Overlap Cost*/
194   overlapCost = getOverlapCost(candidate, size);
195
196   /*If it's overlapping*/
197   if(overlapCost == 1){
198    return VBN;
199   }
200   /*If not...*/
201   else{
202    /*Balance Cost*/
203    balanceCost=getBalanceCost(candidate, space, size);
204    /*Alignment Cost*/
205    alignmentCost = getAlignmentCost(candidate, space, interestBox,
         size, designType);
206
207    return ((balanceCost*BALANCE_PERCENTAGE)+(alignmentCost*
         ALIGNMENT_PERCENTAGE));
208
209   }
210
```

```
211 }
212
213 float getAlignmentCost(struct group *candidate, struct space space,
        struct space interestBox, int size, int designType){
214  int xAligns[size], yAligns[size], x2Aligns[size], y2Aligns[size];
215  int i, j;
216  float totalCost, individualCost, normCost;
217  int xCheck, x2Check, yCheck, y2Check, marginLeft, marginRight,
        marginBottom, marginTop;
218  int interestPointLeft, interestPointRight, interestPointUp,
        interestPointDown;
219
220  for(i=0;i<size;i++){
221   xAligns[i]= candidate[i].x_pos;
222   yAligns[i]= candidate[i].y_pos;
223   x2Aligns[i]= candidate[i].x_pos+candidate[i].width;
224   y2Aligns[i]= candidate[i].y_pos+candidate[i].height;
225  }
226
227  for(i=0;i<size;i++){
228   xCheck=0; x2Check=0; yCheck=0; y2Check=0;
229   marginLeft=0; marginRight=0; marginBottom=0; marginTop=0;
230   interestPointLeft=0; interestPointRight=0; interestPointDown=0;
        interestPointUp=0;
231
232   for(j=0;j<size;j++){
233    if(i!=j && candidate[i].normalGroup==1 && candidate[j].
         normalGroup==1){
234     /*X2 Y2 check*/
235     if(x2Aligns[i]==x2Aligns[j]){
236      x2Check=1;
237      candidate[i].alignment= 0;
238     }
239     else if(y2Aligns[i]==y2Aligns[j])
240      y2Check=1;
241
242     /*X Y check*/
243     if(xAligns[i]==xAligns[j]){
244      xCheck=1;
245      candidate[i].alignment= 1;
246     }
247     else if(yAligns[i]==yAligns[j])
248      yCheck=1;
249
250     /*Margins check */
251     if(xAligns[i]==space.x){
252      marginLeft=1;
253      candidate[i].alignment= 1;
254     }
255     else if(x2Aligns[i]==space.x+space.w){
256      marginRight=1;
257      candidate[i].alignment= 0;
258     }
```

```
259      else if(yAligns[i]==space.y){
260       marginTop=1;
261      }
262      else if(y2Aligns[i]==space.y+space.h){
263       marginBottom=1;
264      }
265      /*Interest points check */
266      if(xAligns[i]==interestBox.x && designType==BACKGROUND){
267       interestPointLeft=1;
268       candidate[i].alignment= 1;
269      }
270      else if(x2Aligns[i]==interestBox.x+interestBox.w  && designType
            ==BACKGROUND){
271       interestPointRight=1;
272       candidate[i].alignment= 0;
273      }
274      else if(yAligns[i]==interestBox.y && designType==BACKGROUND)
275       interestPointUp=1;
276      else if(y2Aligns[i]==interestBox.y+interestBox.h  && designType
            ==BACKGROUND)
277       interestPointDown=1;
278
279     }//if
280    }//for
281
282    individualCost = ALIGNMENT_COST_PER_GROUP;
283
284    if(((xCheck || x2Check) && (marginLeft || marginRight)) && ((
          yCheck || y2Check) && (marginTop || marginBottom)))
285     individualCost -= ALIGNMENT_COMPLETE_GREAT;
286    else if( (marginLeft || marginRight) && (marginTop ||
          marginBottom) )
287     individualCost -= ALIGNMENT_COMPLETE_GOOD;
288    else if( (xCheck || x2Check) && (yCheck || y2Check) )
289     individualCost -= ALIGNMENT_COMPLETE_POOR;
290    else if( (xCheck || x2Check || marginLeft || marginRight) || (
          yCheck || y2Check || marginTop || marginBottom) )
291     individualCost -= ALIGNMENT_HALFWAY;
292
293    if(marginLeft || marginRight)
294     individualCost -= ALIGNMENT_MARGINS;
295    else if(interestPointDown || interestPointUp ||
          interestPointRight || interestPointLeft)
296     individualCost -= ALIGNMENT_INTEREST;
297
298    totalCost += individualCost>0? individualCost:0;
299   }//for
300
301   normCost = totalCost/(ALIGNMENT_COST_PER_GROUP*size);
302
303   return normCost;
304
305  }
```

```
306
307 float getBalanceCost(struct group *candidate, struct space space,
       int size){
308  int i, centroidX, centroidY, centerX, centerY;
309  float weights=0, xWeights=0, yWeights=0, maxDistance, distance;
310  for(i=0;i<size;i++){
311   weights+=candidate[i].weight;
312   xWeights+= candidate[i].weight+candidate[i].x_avg;
313   yWeights+= candidate[i].weight+candidate[i].y_avg;
314  }
315
316  centroidX = (int) (xWeights/weights);
317  centroidY = (int) (yWeights/weights);
318
319  centerX = space.w/2;
320  centerY = space.h/2;
321
322  distance = sqrt(pow(centroidX-centerX, 2) + pow(centroidY-centerY,
       2));
323  maxDistance = sqrt(pow(space.w, 2) + pow(space.h, 2));
324
325  return distance/maxDistance;
326 }
327
328 int getOverlapCost(struct group *candidate, int size){
329  int i, j;
330  int somethingWrong=0;
331
332  for(i = 0; i < size && somethingWrong == 0; i++)
333   for(j = 0; j < size && somethingWrong == 0; j++)
334    if(i!=j)
335     somethingWrong = isOverlapping(candidate[i], candidate[j]);
336
337  return somethingWrong;
338 }
339
340 int isOverlapping(struct group one, struct group two){
341  int xCheck, xCheck1, xCheck2, xCheck3, xCheck4;
342  int yCheck, yCheck1, yCheck2, yCheck3, yCheck4;
343  int final;
344
345  /*X checks*/
346  xCheck1 = (one.x_pos >= two.x_pos && one.x_pos <= two.x_pos + two.
       width)?1:0;
347  xCheck2 = (one.x_pos+one.width >= two.x_pos && one.x_pos+one.width
       <= two.x_pos + two.width)?1:0;
348  xCheck3 = (two.x_pos >= one.x_pos && two.x_pos<=one.x_pos+one.
       width)?1:0;
349  xCheck4 = (two.x_pos+two.width >= one.x_pos && two.x_pos+two.width
       <=one.x_pos+one.width)?1:0;
350
351  xCheck = xCheck1 + xCheck2 + xCheck3 + xCheck4 > 0?1:0;
352
```

```
353   /*Y checks*/
354   yCheck1 = (one.y_pos >= two.y_pos && one.y_pos <= two.y_pos + two.
          height)?1:0;
355   yCheck2 = (one.y_pos+one.height >= two.y_pos && one.y_pos+one.
          height <= two.y_pos + two.height)?1:0;
356   yCheck3 = (two.y_pos >= one.y_pos && two.y_pos<=one.y_pos+one.
          height)?1:0;
357   yCheck4 = (two.y_pos+two.height >= one.y_pos && two.y_pos+two.
          height<=one.y_pos+one.height)?1:0;
358
359   yCheck = yCheck1 + yCheck2 + yCheck3 + yCheck4 > 0?1:0;
360
361   final = xCheck + yCheck > 1?1:0;
362
363   return final;
364  }
365
366  void chooseNeighbour(struct group *candidate, struct space space,
         int size, float t){
367   int i, widthSteps, heightSteps, xJump, yJump;
368   int x_pos2, y_pos2, test;
369   for(i=0;i<size;i++){
370    if(candidate[i].normalGroup == 1){
371     widthSteps = (space.w - candidate[i].width)/20;
372     heightSteps = (space.h - candidate[i].height)/20;
373
374     xJump = widthSteps==0?0:(rand()%widthSteps)*20;
375     yJump = heightSteps==0?0:(rand()%heightSteps)*20;
376
377     /*Long distances in case Temperature is high; low distances in
            case it isnt*/
378     xJump = (int) (xJump*(t/300.0));
379     yJump = (int) (xJump*(t/300.0));
380
381     /*Checking the jump is multiple of 20*/
382     xJump -= xJump%20!=0?xJump%20:0;
383     yJump -= yJump%20!=0?yJump%20:0;
384
385     /*Random negative jumps*/
386     xJump -= rand()%2==0?0:xJump*2;
387     yJump -= rand()%2==0?0:yJump*2;
388
389
390     /*Now we move the group with that jumping distance*/
391     candidate[i].x_pos += xJump;
392     candidate[i].y_pos += yJump;
393
394     /*We run some checks on the calculations*/
395     x_pos2 = candidate[i].x_pos + candidate[i].width;
396     y_pos2 = candidate[i].y_pos + candidate[i].height;
397
398     candidate[i].x_pos = candidate[i].x_pos<space.x?space.x:
          candidate[i].x_pos;
```

```
399     candidate[i].y_pos = candidate[i].y_pos<space.y?space.y:
           candidate[i].y_pos;
400
401     candidate[i].x_pos = x_pos2 > space.x + space.w?(space.x+space.w
           )-candidate[i].width:candidate[i].x_pos;
402     candidate[i].y_pos = y_pos2 > space.y + space.h?(space.y+space.h
           )-candidate[i].height:candidate[i].y_pos;
403
404     candidate[i].x_pos -= candidate[i].x_pos%20!=0?candidate[i].
           x_pos%20:0;
405     candidate[i].y_pos -= candidate[i].y_pos%20!=0?candidate[i].
           y_pos%20:0;
406
407     /*We too calculate the average point*/
408     candidate[i].x_avg = candidate[i].x_pos + candidate[i].width/2;
409     candidate[i].y_avg = candidate[i].y_pos + candidate[i].height/2;
410
411    }
412   }
413  }
414
415  void alpha(float *t){
416   *t=*t-1;
417  }
418
419  int l(int t){
420   int k= 100, times;
421   times = k - (((k - t)/10)*5);
422   return times;
423  }
424
425  char * itoa (int i){
426   char str_val [MAX] ;
427
428   snprintf (str_val, sizeof (str_val), "%d", i) ;
429
430   return strdup (str_val) ;
431  }
```

A.5. Estructura XML de un Diseño

```
 1  <DESIGN>
 2   <genes string='18224321814478577674516585883870'/>
 3   <foreground>
 4    <fg_color n='0' blue='0' green='0' red='0'/>
 5    <fg_color n='1' blue='255' green='255' red='255'/>
 6    <fg_color n='2' blue='255' green='255' red='255'/>
 7    <fg_color n='3' blue='255' green='255' red='255'/>
 8    <fg_color n='4' blue='230' green='230' red='230'/>
 9    <fg_color n='5' blue='220' green='220' red='220'/>
10    <fg_color n='6' blue='210' green='210' red='210'/>
```

```
11      <fg_color n='7' blue='255' green='255' red='255'/>
12     </foreground>
13     <background>
14      <bg_color n='0' blue='255' green='255' red='255'/>
15      <bg_color n='1' blue='0' green='0' red='0'/>
16      <bg_color n='2' blue='0' green='0' red='0'/>
17      <bg_color n='3' blue='0' green='0' red='0'/>
18      <bg_color n='4' blue='0' green='0' red='0'/>
19      <bg_color n='5' blue='0' green='0' red='0'/>
20      <bg_color n='6' blue='0' green='0' red='0'/>
21      <bg_color n='7' blue='0' green='0' red='0'/>
22     </background>
23     <fonts>
24      <font font='Garamond' type='primary'/>
25      <font font='CenturySchoolbook' type='secondary'/>
26     </fonts>
27    </DESIGN>
```

Apéndice B

Manual de usuario

En este anexo se explicará como utilizar el sistema en una máquina conectada a internet.

La página inicial de **Gaudii** es una breve descripción del sistema; para comenzar el proceso de creación de diseños hay que pulsar el único botón de la pagina, *Comenzar nuevo diseño*.

Paso 1. Preferencias del Diseño. Al usuario se le presentan las preferencias sobre su diseño. Esto influirá en el tipo de diseños generados (colores, oscuridad, etc.). También se le cuestiona acerca de donde obtener la imagen para el diseño. El usuario puede bien subirla él mismo o buscar en Flickr a través de la aplicación (Figura B.1).

Paso 2.a. Subir una imagen. En este apartado el usuario solo debe elegir un archivo de imagen de su ordenador. Al subirlo se le confirmará que se ha subido con éxito(Figura B.2).

Paso 2.b. Elegir una imagen de Flickr. La otra opción es buscar en FLickr. Esta búsqueda se hace mediante etiquetas. De las imágenes que aparecen, el usuario puede seleccionar cual ver a mayor tamaño y después elegirla para su diseño (Figura B.3).

Paso 3. Análisis de la imagen. Gaudii tratará de buscar el punto de interés en la imagen. Primero mostrará el punto de interés encontrado, y el usuario puede elegir entre quedarse con él o marcar él uno. Si elige marcar uno solo tiene que trazar un rectángulo sobre la imagen para elegirlo (Figura B.4).

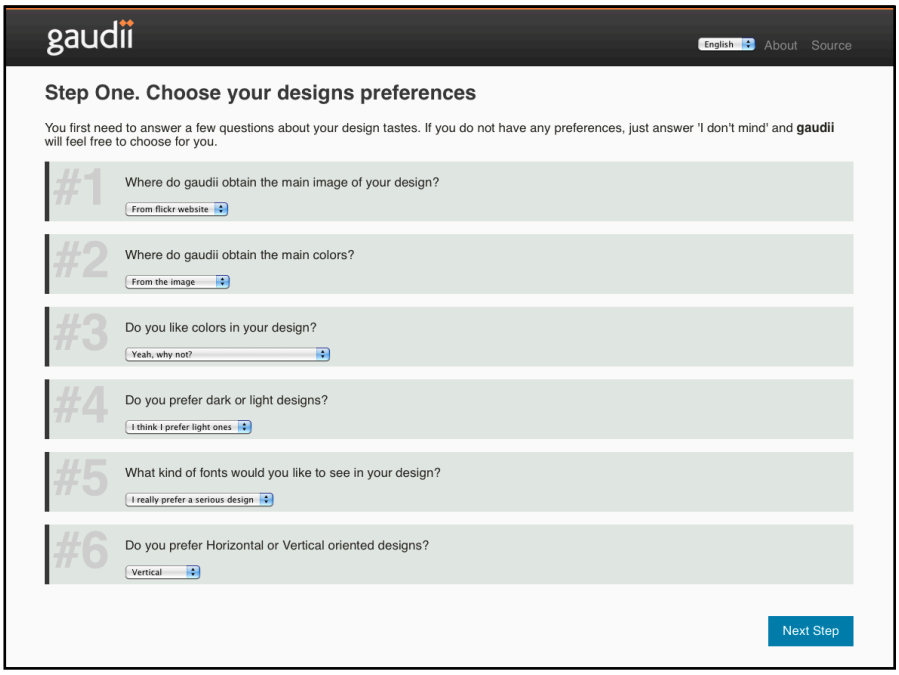

Figura B.1: Paso 1. Elegir las preferencias del diseño.

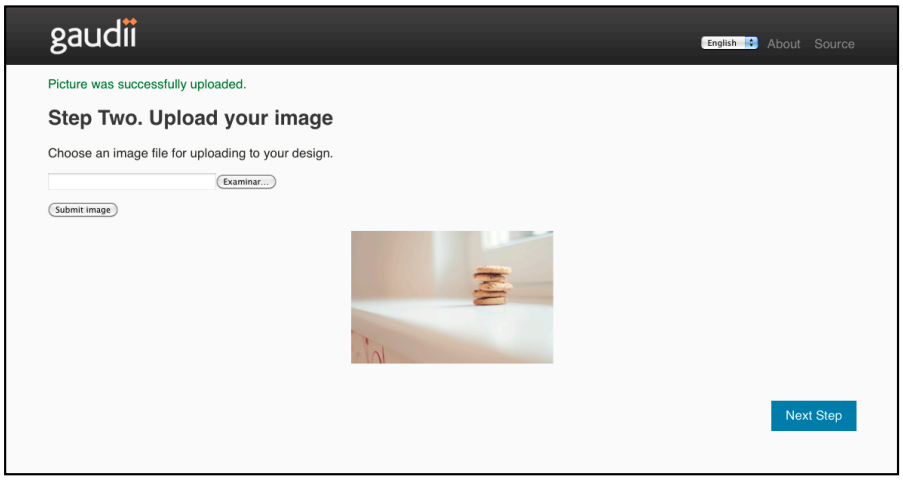

Figura B.2: Paso 2. Subir la imagen

Paso 4. Elementos de texto. Primero, el usuario debe introducir un título (obligatorio) y un subtitulo (opcional) para el proyecto. Una vez añadidos aparecerá un menú para añadir grupos de textos e imágenes. Cada elemento de texto puede ser Texto Destacado, Texto Normal y Notas, siendo el orden decreciente en importancia. Además, se puede elegir si relacionar una

Figura B.3: Paso 2.b. Elegir una imagen de Flickr

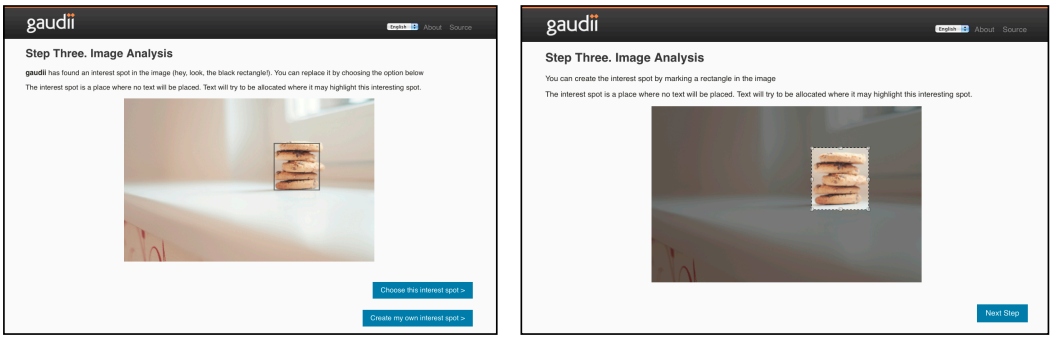

Figura B.4: Paso 3. Análisis de la imagen

imagen a algún grupo de texto (ver Figura B.5 con el proceso paso a paso).

Paso 5. Creación y Mezcla de Diseños. En la pantalla de creación hay que pulsar el botón de *Obtener Diseños* para que Gaudii genere 9 nuevos diseños. Cuando aparecen los 9 diseños tenemos diferentes opciones:

- **Elegir diseños.** Seleccionando la opción de Descargar, podremos ver la imagen a tamaño completo para poder descargarla (Figura B.6).
- **Descargar XML.** Pulsando el botón de cada diseño podemos descargarnos su XML asociado. Este XML se podrá utilizar posteriormente para crear diseños con la misma estructura que el elegido (Figura B.7).
- **Mezclar diseños.** El usuario puede arrastrar diseños a la zona de la derecha para añadirlos al mezclador. Cuando haya dos o más aparecerá la opción de mezclar estos diseños

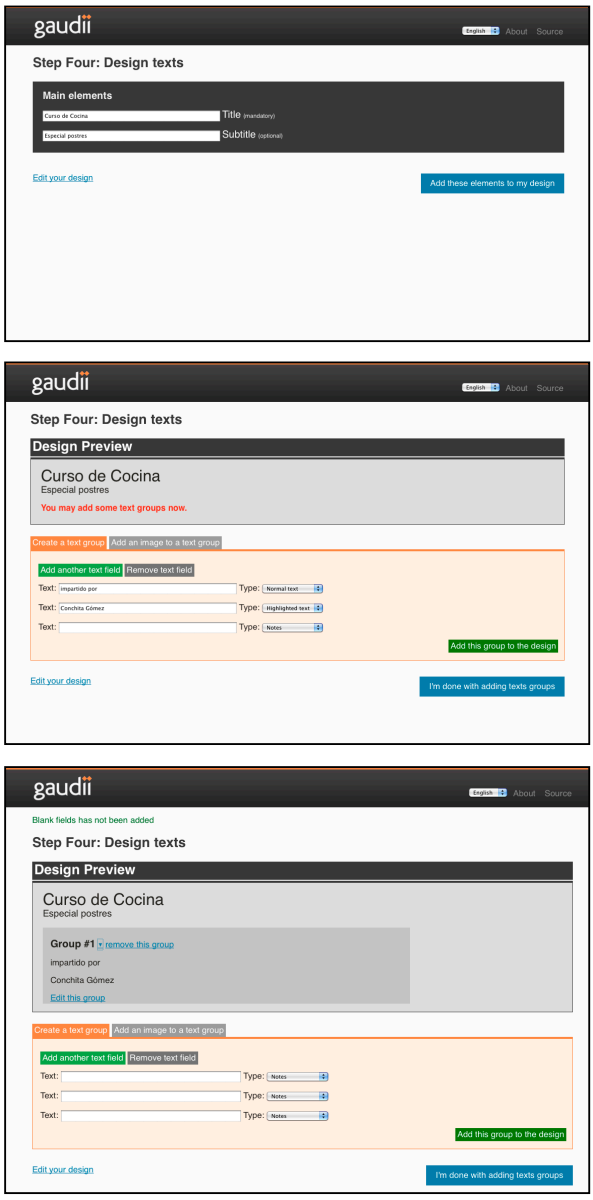

Figura B.5: Proceso de añadir textos.

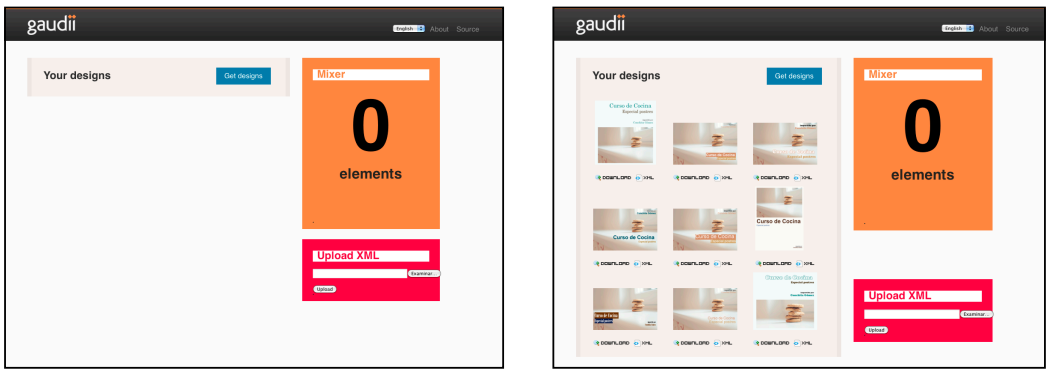

Figura B.6: Vista de la zona de diseños

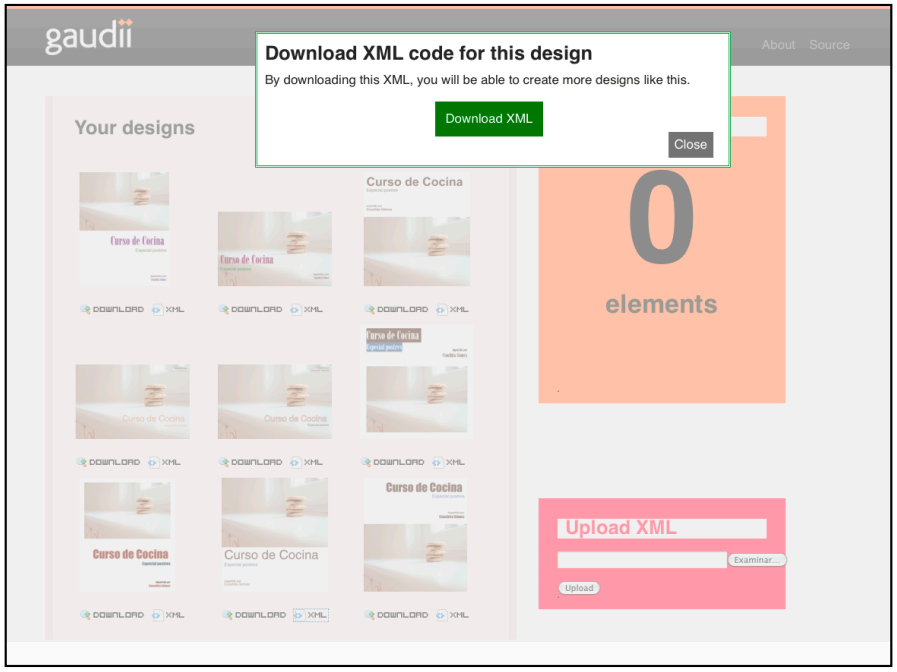

Figura B.7: Descarga del XML

(al pulsarlo, el icono de una cadena genética indicará el proceso). De esta mezcla aparecerán otros 9 carteles en la zona de diseños (Figura B.8).

Paso Opcional. Creación desde XML. Se puede utilizar archivos XML de los diseños generados anteriormente para crear nuevos diseños. Tan solo hay que subir el archivo y Gaudii generará 3 diseños con la misma estructura, colores y tipografías del original. Estos diseños

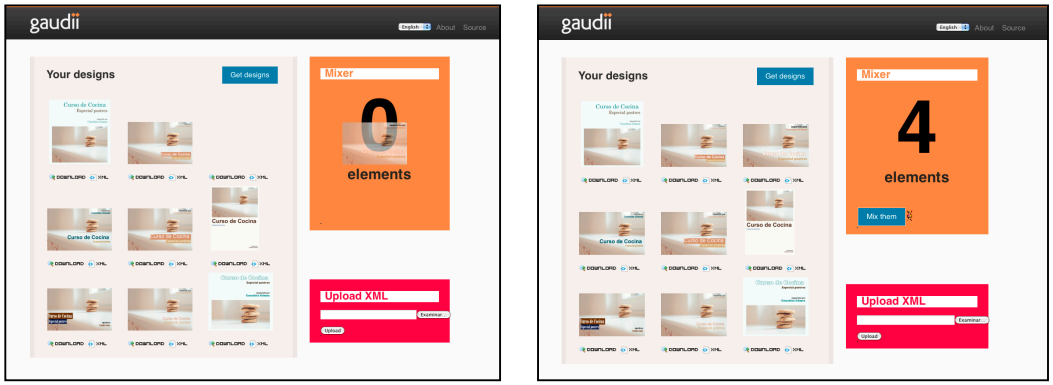

Figura B.8: Mezcla de diseños

pueden usarse, al igual que el resto, para la mezcla de nuevas cadenas geneticas.

Por último, los elementos añadidos en la mezcla genética no se eliminan si se generan nuevos, de forma que podemos generar varios grupos de diseños e ir añadiéndolos a la mezcla. Los diseños de la mezcla solo desaparecerán al crear la combinación.

Apéndice C

Manual de Instalación

En este anexo se explicarán los pasos necesarios para arrancar el sistema en un servidor local. Para esto se necesitan los siguientes paquetes:

- **Ruby:** Las librerías de Ruby pueden descargarse en su sitio oficial:

  `http://www.ruby-lang.org/es/downloads/`

- **Ruby On Rails:** El framework puede descargarse de su sitio oficial. Se recomienda instalar también RubyGems para instalar los paquetes de otras librerías utilizadas. El paquete de RoR incluye el servidor para lanzar el sistema y las librerías de JavaScript.

  `http://www.rubyonrails.org.es/descargas.html`

- **ImageMagick y RMagick:** La librería gráfica y el interfaz para usarlo en Ruby. ImageMagick puede descargarse desde su web oficial:

  `http://www.imagemagick.org/script/index.php`

  Mientras que RMagick se puede descargar la página del proyecto:

  `http://rubyforge.org/projects/rmagick/`

- **OpenCV:** La librería de visión por computador puede descargarse desde la página el proyecto:

  `http://sourceforge.net/projects/opencvlibrary/`

El resto de plugins utilizados (*RedBox* y *flickr.rb*) ya se incluyen en los archivos fuentes del proyecto.

Antes de lanzar el servidor deben compilarse los archivos fuentes de C y C++. Los archivos de C pueden encontrarse en la siguiente ruta:

```
/app/models/C/
```

Mientras que los de C++ se encuentran en esta otra ruta:

```
/app/models/openCV
```

Una vez compilados estos archivos fuente, la aplicación puede lanzarse desde la consola de comandos. Dentro de la carpeta del proyecto se debe escribir:

```
gaudii$ script/server
```

Ahora para acceder a la aplicación se escribe en el navegador la dirección que devuelva el servidor:

```
=>Booting Mongrel
=>Rails 2.3.2 application starting on http://0.0.0.0:3000
=>Call with -d to detach
=>Ctrl-C to shutdown server
```

Bibliografía

[1] Api ajax de google para búsquedas. http://code.google.com/intl/es/apis/ajaxsearch.

[2] Api de yahoo para búsquedas de imágenes. http://developer.yahoo.com/.

[3] Flickr. http://www.flickr.com/.

[4] Imagemagick's offical site. http://www.imagemagick.org/.

[5] Interestingness en flickr. http://www.flickr.com/explore/interesting/.

[6] Open source initiative's open source definition. http://www.opensource.org/docs/osd.

[7] Opencv library. http://opencvlibrary.sourceforge.net/.

[8] Processing's offical site. http://processing.org/.

[9] Prototype's offical site. http://www.prototypejs.org/.

[10] Rmagick's offical site. http://rmagick.rubyforge.org/.

[11] Ruby on rails' offical site. http://rubyonrails.org.

[12] Ruby on rails usage statistics. http://trends.builtwith.com/framework/Ruby-on-Rails.

[13] Ruby's offical site. http://www.ruby-lang.org/es/.

[14] script.aculo.us' offical site. http://script.aculo.us/.

[15] Web page - wikipedia, the free encyclopedia.

[16] J. S. Aikins. *A Representation Scheme Using Both Frames and Rules*, pages 424–440. Addison-Wesley, 1984.

[17] G. Ambrose and P. Harris. *The Fundamentals of Typography*. AVA Publishing, 2006.

[18] C. Anderson, D. Buchsbaum, J. Potter, and E. Bonabeau. Making interactive evolutionary graphic design practical. In *Evolutionary Computation in Practice*, pages 125–141. 2008.

[19] L. Babin. *Introducción a Ajax con PHP*. Anaya Multimedia, 2007.

[20] G. Bradski and A. Kaehler. *Learning OpenCV*. O'Reilly, 2008.

[21] Y. M. Chae. *Expert Systems in Medicine*. CRC Press, 1997.

[22] G. Chen and T. T. Pham. *Introduction to fuzzy sets, fuzzy logic, and fuzzy control systems*. CRC Press, 2001.

[23] K. Chiba, H. Ohwada, and F. Mizoguchi. Acquiring graphic design knowledge with nonmonotonic inductive learning. In *ILP '99: Proceedings of the 9th International Workshop on Inductive Logic Programming*, pages 56–67, London, UK, 1999. Springer-Verlag.

[24] R. Davis, H. Schrobe, and P. Szolovits. What is knowledge representation? *AI Magazine*, 14(1):17–33, 1993.

[25] D. Decoste. The future of chess-playing technologies and the significance of kasparov versus deep blue. In *Papers from the 1997 AAAI Workshop*, 1997.

[26] S. R. DiPaola and L. Gabora. Incorporating characteristics of human creativity into an evolutionary art algorithm. In *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, pages 2450–2456, New York, NY, USA, 2007. ACM.

[27] J. Durkin. Dustpro: A distributed expert system for coal mine dust control. In *ESD/SMI Second Annual Expert Systems Conference, Detroit*, pages 377,385, 1988.

[28] J. Durkin. *Expert Systems: Design and development*. Prentice Hall, 1994.

[29] J. Durkin. *History and Applications*. Academic Press, 2001.

[30] J. Durkin. *Tools and Applications*. Academic Press, 2001.

[31] P. K. F. Hayes-Roth and D. J. Mostow. *Knowledge acquisition, knowledge programming, and knowledge refinement*. Rand Publication Series, Santa Monica, CA, 1980.

[32] J. M. Gutierrez.

[33] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall International, 1994.

[34] E. Heller. *Psicología del color*. Gustavo GIli, 2004.

[35] B. L. Jacob. Composing with genetic algorithms. In *International Computer Music Association*, pages 452–455, 1995.

[36] M. I. Jordan and C. M. Bishop. Neural networks. In *CRC Handbook of Computer Science*, pages 536–556. CRC Press, 1996.

[37] A. Kandel. *Fuzzy Expert Systems*. CRC Press, 1991.

[38] M. Karabatak and M. C. Ince. An expert system for detection of breast cancer based on association rules and neural network. *Expert Syst. Appl.*, 36(2):3465–3469, 2009.

[39] P. D. Karp, K. L. Myers, and T. Gruber. The genetic frame protocol. In *In Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, Montreal, Quebec, Canada, 1995.

[40] K. Kato, M. Shamoto, and K. Yamamoto. An implementation of humanoid vision - analysis of eye movement and implementation to robot. In *SICE, 2007 Annual Conference*, pages 744–747, 2007.

[41] C. Krishnamoorthy and S. Rajeev. *Artificial Intelligence and Expert Systems for Engineers*. CRC Press, 1996.

[42] D. S. Linden. *Innovative antenna design using genetic algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[43] Y. I. Liou. *Expert System Technology: Knowledge Acquisition*. CRC Press, 1997.

[44] J. Mackinlay. Automating the design of graphical presentations of relational information. pages 66–82, 1999.

[45] E. H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Human-Computer Studies*, 1975.

[46] A. Marczyk. Algoritmos geneticos y computacion evolutiva. Encontrado en Internet en http://the-geek.org/docs/algen/ [Accedido en Mayo del 2009].

[47] K. Mehrotra, C. K. Mohan, and S. Ranka. *Elements of Articial Neural Networks*. Bradford Books, 1996.

[48] M. Misky. A framework for representing knowledge. Technical report, Cambridge, MA, USA, 1974.

[49] M. Misky. Minsky's frame system theory. In *TINLAP '75: Proceedings of the 1975 workshop on Theoretical issues in natural language processing*, pages 104–116, Morristown, NJ, USA, 1975. Association for Computational Linguistics.

[50] M. Negnevitsky. *Artificial Intelligence: A Guide to Intelligent Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[51] U. B. of Labor Statistics. *Occupational Outlook Handbook 2010-2011 Edition*. Penguin Group, 2010. http://data.bls.gov/cgi-bin/print.pl/oco/ocos090.htm.

[52] T. O'Reilly. O'reilly network: What is web 2.0, 2005.

[53] M. Petrik. Knowledge representation for expert systems. In *International Conference for Undergraduate and Graduate Students of Applied Mathematics*, Slovak University of Technology, Bratislava, 2004.

[54] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008.

[55] C. Raphael. A probabilistic expert system for automatic musical accompaniment. *Journal of Computational and Graphical Statistics*, 10, 1999.

[56] S. Raymond. *Ajax on Rails*. O'Reilly Media, Inc., 2007.

[57] S. Ruby, D. Thomas, and D. Hansson. *Agile Web Development with Rails, Third Edition*. Pragmatic Bookshelf, 2009.

[58] S. J. Russell and Norvig. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall, 2003.

[59] T. Samara. *Design Elements. A Graphic Style Manual*. Rockport, 2007.

[60] E. H. Shortliffe. *Mycin: a rule-based computer program for advising physicians regarding antimicrobial therapy selection.* PhD thesis, Stanford, CA, USA, 1975.

[61] J. N. Siddall. *Expert Systems for Engineers*. CRC Press, 1990.

[62] W. Siler and J. J. Buckley. *Fuzzy Expert Systems and Fuzzy Reasoning*. John Wiley Sons, Inc., 2005.

[63] S. Slade. Case-based reasoning: A research paradigm. *AI Magazine*, 12(1), 1991.

[64] H. Tang, K. C. Tan, and Z. Yi. *Neural Networks: Computational Models and Applications*. Springer, 2007.

[65] D. Vallejo, J. Albusac, L. Jiménez, C. Gonzalez-Morcillo, and J. M. García. A cognitive surveillance system for detecting incorrect traffic behaviors. *Expert Syst. Appl.*, 36(7):10503–10511, 2009.

[66] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. pages 511–518, 2001.

[67] D. Waterman. *A Guide To Expert Systems*. Addison-Wesley, Reading, MA, 1986.

[68] R. Williams. *The Non-Designer's Design Book: Design and Typographic Principles for the Visual Novice*. Pearson Education, 1994.

[69] R. E. K. Wolfgang Banzhaf, Peter Nordin and F. D. Francone. *Genetic Programming. An Introduction : On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers, 1997.

[70] J.-D. Wu and C.-H. Liu. An expert system for fault diagnosis in internal combustion engines using wavelet packet transform and neural network. *Expert Syst. Appl.*, 36(3):4278–4286, 2009.

[71] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.