Capítulo 4

Prueba de Adaptabilidad

Capítulo 4. Prueba de Adaptabilidad

Como se mencionó en el capítulo 2 actualmente no es válido que el software únicamente funcione bien y resuelva el problema que le corresponde, idealmente el sistema debe poder crecer y modificarse para solucionar algún otro problema ligeramente diferente. A esta característica se le llama Adaptabilidad.

4.1 Introducción

Propiamente dicho, el concepto de Adaptabilidad consiste en la habilidad de modificar un sistema para que funcione con conceptos de dominios de aplicaciones diferentes [Bruegge, Dutoit, 2004]. Para medir qué tan adaptable puede ser un sistema, se plantean las siguientes preguntas [McCall, Cavano, 1978]:

- ¿Se podrá usar el software en otra máquina? (Portabilidad)
- ¿Se podrá reutilizar alguna parte del programa? (Reusabilidad)
- ¿Podrá el sistema interactuar con otro programa? (Interoperabilidad)

Las características planteadas en forma de pregunta se explican a continuación, además se expone la forma en que se evaluó CASSIEL.

4.2 Portabilidad

La portabilidad, aunque tiene cierta relación con la reusabilidad, es importante mencionar que no es lo mismo, ya que se refiere a la facilidad con que un sistema o

componente del mismo puede ser transferido a diferentes ambientes de hardware o software [Bruegge, Dutoit, 2004].

A pesar de que las metodologías de desarrollo no incorporan alguna estrategia para lograr la portabilidad, esta característica es considerada como una meta para cualquier clase de software.

Existen dos tipos de adaptabilidad [Mooney, 1990]:

- *Portabilidad Binaria*: Se refiere a portar la forma ejecutable, la cual ofrece varias ventajas, pero sólo es posible llevarla a cabo a través de ambientes muy similares.
- *Portabilidad de Código Fuente*: Como su nombre lo indica, asume la disponibilidad del código fuente, pero provee la oportunidad de adaptar la unidad de software a una amplia variedad de ambientes.

El proceso de portabilidad, se lleva a cabo mediante dos componentes principales, los cuales son denominados *Transportación* y *Adaptación* [Mooney, 1990]. La *transportación* es el movimiento físico. La *Adaptación* es cualquier modificación llevada a cabo a la versión original.

En sí no existe una manera de calcular la portabilidad de un sistema, sin embargo existen algunos procesos y métricas que emplean las técnicas de estimación de costos para obtener indicadores aproximados. El costo de redesarrollar el software y el costo de portar el software (el cual implica analizar el match entre las interfaces de la

unidad de software y aquellas del objetivo), se puede obtener mediante una sencilla función llamada *Grado de portabilidad (GP).*

GP = 1 – (costo de portar el software / costo de redesarrollar el software)

Donde si el valor obtenido es mayor a cero, se concluye que la portabilidad es más efectiva que el redesarrollar.

En el caso de CASSIEL, debido a que el código del sistema se encontraba "*hardcoded*", se llevaron a cabo algunas modificaciones tal y como lo indica el capítulo 3, que permitieron que éste aumentara su nivel de portabilidad.

Para llevar a cabo el cálculo de los costos, se empleó el cálculo del número de líneas de código. Para lograr que CASSIEL se ejecute en otra máquina, en su versión "*hardcoded*", se tendrían que modificar aproximadamente 10 líneas de 15 clases que manejaban alguna conexión a la base de datos, datos de e-mail, etc., lo cual da un total de aproximadamente 150 líneas de código.

En la versión portable, sólo se debe llevar a cabo la modificación del archivo propiedades (*Properties),* donde se manejan las variables empleadas en las conexiones ya sea a la base de datos o al mail; dicho archivo cuenta con 38 líneas que deberán ser modificadas para la adaptación.

El cálculo del grado de portabilidad de CASSIEL en la versión portable, sería el siguiente:

$$GP = 1 – (38 / 150) = 1 - (0.2533) = 0.7466$$

Por lo tanto, el grado de portabilidad de CASSIEL es de 0.74. Lo cual indica que actualmente es más fácil portar el sistema a distintos ambientes.

4.3 Reusabilidad

El desarrollo de un sistema es una tarea muy cara, sin embargo hay una forma de hacer que el costo baje, esto es mediante la reusabilidad, ya que se utilizan componentes elaborados previamente y que son fácilmente empleados en una variedad de sistemas, por lo tanto, se reduce el costo de diseño, desarrollo (tiempo y dinero), y de mantenimiento debido a que estas partes ya están consolidadas y fueron ampliamente probadas.

La reusabilidad es la capacidad de reutilización de un sistema o partes de él, es decir, hasta qué punto se puede volver a emplear un programa en otras aplicaciones, en relación al empaquetamiento y alcance de las funciones que realiza. [Pressman, 2002].

Para medir la reusabilidad de un sistema se deben tomar en cuenta el nivel de abstracción, las interfaces entre los módulos, cohesividad entre clases y el acoplamiento entre los objetos o clases. El mantenimiento de un diseño es mejorado cuando los cambios pueden ser hechos fácilmente sin propagarse a otras partes del sistema. [Chatterjee, 2003].

Para lograr que un software sea reutilizable, el acoplamiento debe ser reducido, lo cual implica que el número de relaciones entre todas las clases del sistema sean mínimas; y la cohesión debe ser grande, ya que ayuda a flexibilizar la estructura entre clases.

Una forma importante de reducir de la complejidad de un programa es mediante el incremento de la modularización, la cual puede ser medida mediante la cohesión y el acoplamiento:

- Acoplamiento: Grado en que las clases de un sistema son dependientes una de otra, existen diversos tipos de acoplamiento como se puede observar en la tabla 4.1:

Acoplamiento de Contenido	Cuando un componente modifica datos que son internos a otros componentes.
Acoplamiento Común	Cuando se usan variables globales.
Acoplamiento de Control	Cuando un procedimiento llama a otro usando una "bandera" o un "comando" que controla explícitamente lo que hace un segundo procedimiento.
Acoplamiento de Estampa	Cuando una de las clases de aplicación es declarada como "tipo" de un argumento de un método.
Acoplamiento de Datos	Cuando los tipos de los argumentos de los métodos son primitivos o librerías.
Acoplamiento de llamado de Rutinas	Cuando una rutina llama a otra.
Acoplamiento de uso de Tipos	Cuando un método usa un tipo de dato definido en otro módulo.
Inclusión	Cuando un componente importa un paquete.
Acoplamiento externo	Cuando un módulo tiene dependencia en cosas tales como el sistema operativo, librerías compartidas o hardware.

Tabla 4.1: Tipos de Acoplamiento

En el caso de CASSIEL, se presentan dos tipos de acoplamiento:

- ✓ Inclusión, ya que los distintos paquetes que conforman el sistema hacen uso de las clases contenidas en otros, aislando así tareas específicas en cada módulo y a su vez facilitando la integración de los servicios

- ✓ Llamado de Rutinas, ya que existen servicios remotos (servidor de recursos y servidor de mensajes de CASSIEL) que inclusive, no tienen que estar necesariamente en el mismo servidor ya que las llamadas se ejecutan remotamente a manera de una comunicación cliente-servidor

- ✓ Acoplamiento de los Datos, afortunadamente cada modulo maneja sus propios datos (por ejemplo, el UMS maneja una base de datos y CASSIEL la suya) permitiendo así el aislamiento de las funciones de manipulación de información, pero también haciendo posible la integración de ésta en un solo sistema.

- Cohesión: Grado en que una entidad soporta un propósito singular en el sistema, la tabla 4.2 muestra los diversos niveles de cohesión:

Cohesión funcional	Cuando todo el código que computa un resultado en particular, están juntos y lo demás esta separado.
Cohesión de capa	Cuando todas las facilidades para dar o acceder a un conjunto de servicios relacionados están juntos, y lo demás esta separado.
Cohesión Comunicativa	Cuando todos los módulos que accedan o manipulan ciertos datos están juntos, y lo demás esta separado.
Cohesión Secuencial	Cuando los procedimientos, donde un procedimiento da entradas para el siguiente procedimiento, juntos y lo demás esta separado.
Cohesión Procedural	Juntar procedimientos que son usados uno tras otro.
Cohesión Temporal	Cuando las operaciones que son llevadas a cabo durante la misma fase de la ejecución del programa está junto, y lo demás esta separado.
Cohesión de utilidad	Cuando utilidades relativas que no pueden ser lógicamente colocadas en otras unidades de cohesión están juntas.

Tabla 4.2: Niveles de Cohesión

Por el orden en el que se encuentra el código de CASSIEL, se le puede ubicar en el caso de *Cohesión de Capa,* ya que, cada modulo del sistema (ver figura 4.1) forma parte de un paquete independiente al cual a su vez separa sus componentes de acuerdo a la filosofía del MVC (Model–View–Controller) de manera que es muy fácil integrar esos módulos para crear la aplicación y reutilizarlos para varios propósitos

Se puede ver que hay diferentes paquetes comunes usados por CASSIEL y el resto de las aplicaciones, de manera que se puede usar la reutilización de diferentes módulos.

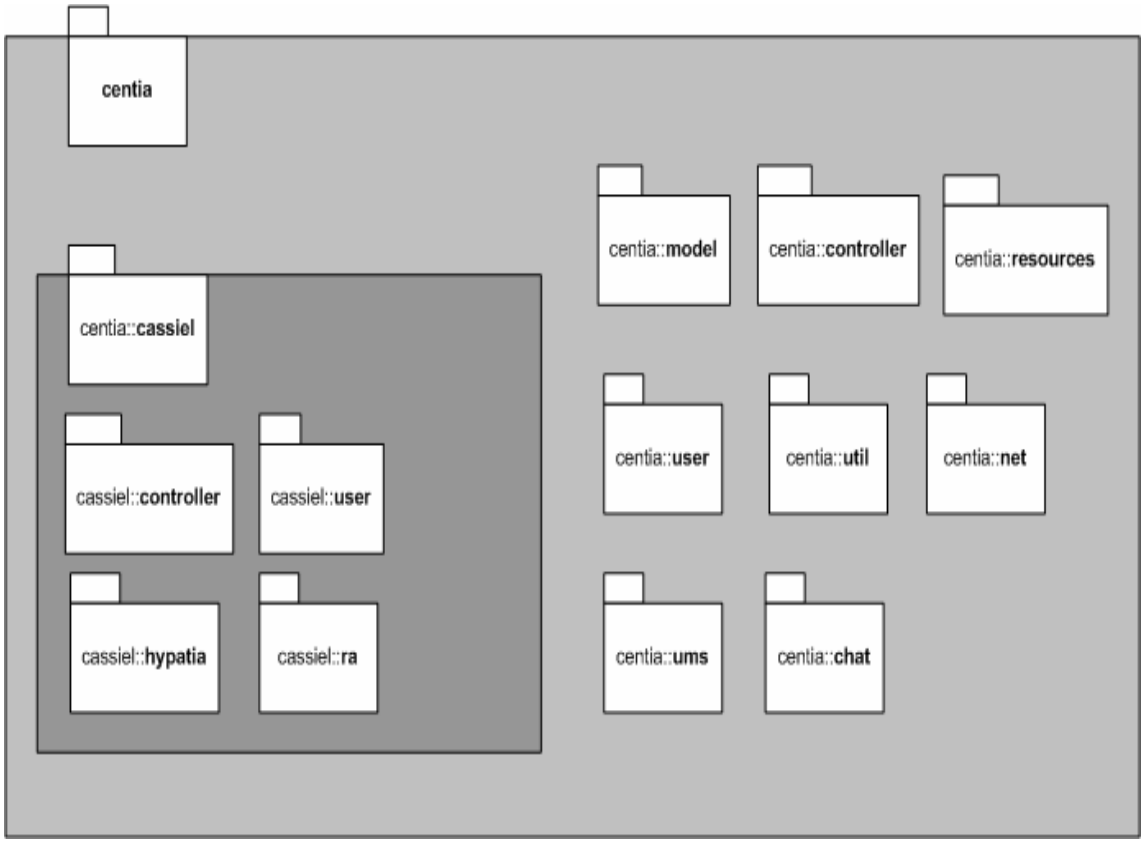

Fig. 4.1 Diagrama de Paquetes en CASSIEL

Existen tres clases de componentes de software que forman parte de una típica aplicación de software dependiendo de su relación y dependencia con el dominio de acción. A continuación se presenta una tabla (tabla 4.3) que muestra dicha clasificación, y otra (tabla 4.4) que muestra el caso específico de CASSIEL.

Tipo	Porcentaje Idóneo	Descripción
Dominio-Independiente	20%	Incluye ADTs (Abstract Data Types), rutinas de utilidad, librerías, que son útiles en un amplio rango de problemas.
Dominio-Específica	65%	Este es para software, que solo es útil dentro del dominio específico.
Aplicación-Específica	15%	Incluye software que solo implementa el único detalle de requerimientos de una aplicación

Tabla 4.3 Tipos de componentes en una aplicación

Tipo	Porcentaje Idóneo	Observaciones
Dominio-Independiente	30%	Ya que incluye los módulos que manipulan los perfiles de usuario
Dominio-Específica	50%	Todos los módulos dedicados a mantener el plan de aprendizaje
Aplicación-Específica	20%	Debido a la cantidad de clases necesarias para el control y presentación

Tabla 4.4 Tipos de componentes en CASSIEL

4.4 Interoperatividad

La interoperatividad es la habilidad de dos o más sistemas o componentes para intercambiar y compartir información, para usarla de manera adecuada [Sanders, Hamilton Jr., 2003]. Para efectos de unificación y análisis de interoperabilidad en una arquitectura, se ha creado un diagrama [Chatfield, 1998] (ver figura 4.2) que describe tres vistas de una arquitectura (operacional, de sistemas y técnica) junto con su relación de funcionamiento. Esto permite especificar criterios técnicos para implementar sistemas.

Fig. 4.2 Las tres vistas de interoperabilidad en una arquitectura [Chatfield, 1998]

La medición de interoperabilidad de un sistema no es tarea sencilla y se debe hacer a varios niveles, en el caso específico de CASSIEL se empleó el modelo LISI, para poder realizar un análisis de este aspecto de la adaptabilidad.

LISI (Levels of Information Systems Interoperability), es una disciplina y un proceso para definir, determinar y certificar el grado de interoperatividad requerido o logrado entre organizaciones o sistemas. [Chatfield, 1998], tratando a la interoperatividad como simples conexiones entre sistemas. El usar LISI identifica el nivel de interoperatividad requerido por la vista operacional de nodo a nodo.

El *modelo de Madurez* de Interoperabilidad clasifica 5 niveles de "la naturaleza general de la interoperabilidad", como *Aislado, Conectado, Funcional, Dominio y Empresa* [Sanders, Hamilton, 2003], cada uno con características representativas (ver tabla 4.5).

Nivel	*Intercambio de Información*
4: Universal Manipulación interactiva Datos compartidos y aplicaciones	• Información de dominios diferentes y colaboración compartida • Colaboración avanzada
3: Integrado Datos compartidos Aplicaciones separadas	• Bases de datos compartidas • Colaboración sofisticada
2: Distribuido Funciones comunes mínimas Aplicaciones y datos separados	• Intercambio heterogéneo de productos • Colaboración Básica
1: Conectado Conexiones electrónicas, aplicaciones y datos Aplicaciones y datos separados	• Intercambio homogéneo de productos
0: Aislado No conectado	• Enlace Manual

Tabla 4.5: Modelo de Madurez LISI

Las vistas del sistema responden identificando y determinando características en términos de cuatro atributos: Procedimientos, Aplicaciones, Infraestructura y Datos (PAID) los cuales representan las capacidades del sistema. Esto integra un *Modelo de Referencia* que toma los 5 niveles de interoperatividad describiéndolos con los atributos ya mencionados [Sanders, Hamilton, 2003] (Figura 4.3).

Naturaleza de la Interacción Operacional de Información	*Ambiente de Computación Correspondiente*	*Código de Nivel*	*Implicaciones* P	A	I	D
Manipulación interactiva de Dominios diferentes	Universal	4	Nivel Empresa	Interactivo	Topologías Múltiples	Modelo Empresa
Aplicaciones y Bases de Datos Compartidas	Integrado	3	Niel Dominio	Grupal	Redes Mundiales	Modelo Dominio
Intercambio Complejo de Medios	Distribuido	2	Nivel Programa	Automatización	Redes Locales	Modelo del Programa
Intercambio Electrónico Simple	Conectado	1	Nivel Sitio / Local	Manejadores del Sistema	Conexión Simple	Local
Entrada Manual	Aislado	0	Control de Acceso	N / A	Independiente	Privado

Fig. 4.3 El Modelo de Referencia LISI [Sanders, Hamilton, 2003]

En el caso de CASSIEL y tomando en cuenta el modelo LISI, se podría decir que como un todo cae en el *nivel 1: Conectado*, ya que separa aplicaciones y datos. Sin embargo, al verlo hacia adentro, CASSIEL incorpora subsistemas, por lo que cae en el *nivel 3: Integrado* ya que cuenta con aplicaciones compartidas y trabaja con varias bases de datos.

Para hacer de CASSIEL un sistema con nivel de madurez Universal, habría que abrir sus servicios, probablemente mediante el uso de Servicios Web (*Web Services*)

para que otras aplicaciones utilicen su modelo de usuario y demás componentes del sistema y de esta manera puedan enriquecerse con el plan de aprendizaje y recursos disponibles en el sistema de aprendizaje colaborativo.

4.5 Comentarios Finales

En este capítulo se presentó un análisis extenso de la adaptabilidad de CASSIEL, se mencionaron las mejoras tanto en portabilidad, reusabilidad e interoperabilidad alcanzadas en el estado actual del sistema, después de haber realizado algunos cambios.

A pesar de que es difícil afirmar que CASSIEL es 100% adaptable, el software posee un grado de adaptabilidad que permite al sistema:

- Ser implantado en diversos ambientes, bajo distintas condiciones y modificando ciertas variables de su entorno
- Ser ampliamente reutilizable por otras aplicaciones que necesiten emplear el mismo modelo de aprendizaje implementado
- Estar en comunicación con otros módulos o sistemas, algo que quizás no ha sido explorado aún y que se podría mejorar desarrollando servicios en base a los componentes ya elaborados.