

Universidad Carlos III de Madrid  
Escuela Politécnica Superior



Herramienta CASE de desarrollo de asistentes  
de navegación Web basada en los diagramas  
MSC

Proyecto Fin de Carrera  
Ingeniería Técnica Sistemas de Telecomunicación

**Autor:** Daniel García Jones  
**Tutor:** Dr. Vicente Luque Centeno  
Noviembre 2004



**Título:** CREACIÓN DE UNA HERRAMIENTA CASE DE DESARROLLO DE SISTEMAS DE NAVEGACIÓN AUTOMÁTICA DEL WEB BASADO EN LOS DIAGRAMAS DE SECUENCIAS DE MENSAJES Y EL LENGUAJE WEBL.

**Autor:** Daniel García Jones

**Director:** Vicente Luque Centeno

## EL TRIBUNAL

Presidente:

-----

Vocal:

-----

Secretario:

-----

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día

--

de

-----

de 2004 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

PRESIDENTE SECRETARIO VOCAL



*"The good things come for those who wait..."* - Wycleaf Jean



# Agradecimientos

A mi familia, y en especial a mi madre, que a su particular modo siempre ha estado ahí para apoyarme, y aconsejarme. Desde luego que sin vosotros no hubiera sido posible.

A la gente que me ha ayudado "técnicamente" a lo largo del Proyecto: Pablo, que me enseñó lo genial y sencillo que es Python, Gorka que me recomendó ya hace mucho DIA y a la gente del G.U.L. y en especial a Pay por los innumerables y valiosos consejos sobre Linux.

A los "más grandes": Niño, Edu, Mili y Fur, por todas aquellas locuras y buenos momentos que, a pesar de mi mala memoria, no olvidaré nunca. A Blanca, por su paciencia al aguantar mis rollos y sus buenos consejos (aunque a veces no quisiera escucharlos). A Raul, mi Abuelo favorito, que también es uno de los grandes aunque no salga en la foto. A Chico, por tantos cafés, charlas, bocatas de la casa y confesiones, y por ser tan tú mismo. A Laura, Julia, MJ y Sergio, mis compañer@s de clase en Sistemas, y de muchas otras cosas. Y al resto de mi Peña Superior (y no me refiero sólo al título): Gabi, Ramón, Molo, Raquel, Cano, Sonia, Nacho, Iván, Sara y Willy, Elisota, Patricia, Lau, Sergut, Sergi (salvador Granadino), la gente de Asociaciones y del GSAI, Carolina, Sacha, Juanvi y a los que probablemente olvide. No puedo decir que todo haya sido de color de rosa en mi paso por la Universidad, pero desde luego ha merecido la pena aunque sólo sea por haber compartido con vosotros todos estos años.

A Felix que sabe mejor que nadie que las cosas buenas llegan después de una espera. A Nacho y a Emi, mi opositor favorito: yota. A Rosa, por todo. A Kike y a Espi que también han tenido Proyecto para rato.

¡Muchas gracias a todos!





# Resumen

Cada vez es más común el uso de aplicaciones accesibles desde la Web para un número cada vez mayor de propósitos. A medida que el número y la variedad de aplicaciones crece, el tiempo de procesamiento manual de la información en la WWW toma especial relevancia en tareas que sería muchas veces posible realizar de forma autónoma por un ordenador mediante la creación de sistemas de navegación automática, incrementando la productividad y minimizando los errores propios de la naturaleza humana.

Sin embargo, la creación de estos programas es una tarea compleja. Los costes de desarrollo, implantación y mantenimiento son muy altos, debido principalmente a las peculiares características y la constante evolución de la Web. Es de interés desarrollar nuevos métodos de programación Web que deberán además estar basados en estándares y tener un adecuado nivel de abstracción para favorecer la comprensión por un mayor número de personas y garantizar unas cotas de robustez y funcionalidad en un medio tan dinámico y sujeto a fallos como es Internet.

Este proyecto plantea cómo los Diagramas de Secuencias de Mensajes, definidos por el ITU-T en la recomendación z.120, pueden ser aplicados a la especificación de requisitos de sistemas diseñados para automatizar tareas en la Web, de tal forma que disminuyan los costes de desarrollo y mantenimiento de estos sistemas.

Para demostrar la validez del modelo planteado, se ha programado una herramienta CASE de desarrollo basado en los diagramas MSC. Este programa incluye un editor MSC de forma que los diagramas se construyan de forma sencilla, empleando internamente la sintaxis XML definida para describir este tipo de sistemas. La aplicación incluye así mismo los mecanismos de transformación de los diagramas MSC a código ejecutable en el lenguaje de programación WebL. Esta aplicación ha sido probada mediante la confección de una colección de ejemplos sencillos que abarcan desde consultas simples a Google o descargas de las cabeceras RSS de un conocido sitio Web, a la consulta del buzón de correo de una aplicación WebMail.



# Abstract

The use of remote applications on the WWW for many purposes is getting common. At the same time the number and variety of applications increases, also does the time of manual processing of data on the Web which becomes an important factor in tasks that could be done in an autonomous way by computers with the creation of automatic Web browsing systems increasing human productivity, minimising errors due to human nature.

But creating this kind of software is far from trivial. The development, installation and maintenance costs are high, mainly due to the peculiar aspects and constant evolution of the Web. It's of interest to develop new programming methodologies based on standards and with a correct grade of abstraction to make the comprehension easier for a greater audience and assure robustness and functionality in a medium so dynamic and limited by errors such as the WWW.

This project studies the way in which Message Sequence Charts, defined by the ITU-T in Z.120 recommendation, can be used to specificate systems to automate tasks on Internet so developing and maintenance costs decrease. To probe that, we have created a CASE tool for Web developing based on MSC.

This software includes an MSC editor to make easier the construction of the charts, employing internally an XML syntax defined to describe this kind of systems. The application also includes tools for transforming MSC to executable code in WebL programming language and visualise the charts. It has been tested developing a set of examples to solve simple tasks as searching in Google or reading the inbox of a WebMail application.



# Índice general

<b>Agradecimientos</b>	<b>v</b>
<b>Resumen</b>	<b>vii</b>
<b>Abstract</b>	<b>ix</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Breve historia de Internet	1
1.2. Motivación	1
1.3. Enfoque	2
1.4. Objetivos del proyecto	3
1.5. Fases del proyecto	3
1.6. Contenido de la memoria	4
<b>2. Estado del arte</b>	<b>5</b>
2.1. Análisis de las tareas en el Web	5
2.1.1. Navegación automática	6
2.1.1.1. Diferencias entre navegación manual y navegación automática	6
2.1.1.1.1. Esfuerzo	6
2.1.1.1.2. Propensión a errores	6
2.1.1.1.3. Tiempo de respuesta	6
2.1.1.1.4. Requisitos hardware y software	6
2.1.1.1.5. Adecuación de repositorios	7
2.1.1.1.6. Procesamiento	7
2.1.1.1.7. Coste de implantación y adaptabilidad	7
2.1.2. Características de los datos de el Web	7
2.1.2.1. Orientación a la visualización	7
2.1.2.2. Accesibilidad Web	7
2.1.2.3. Distribución de la información	8
2.1.2.4. Ausencia de semántica en el marcado	8
2.1.2.5. Voluminosidad	8
2.1.2.6. Heterogeneidad	11
2.1.2.7. Regularidad estructural	11
2.1.2.8. Varios lenguajes de marcado	11
2.1.3. Tipos de programas de navegación automatizada	12
2.1.4. Modelo de computación sobre el Web	13
2.1.5. Costes de la navegación automatizada	13
2.1.5.1. Coste de desarrollo	14
2.1.5.2. Coste de mantenimiento	14
2.2. XML y tecnologías derivadas	14
2.2.1. eXtensible Markup Language	14
2.2.2. Definición de lenguajes XML	15
2.2.2.1. DTD's	15
2.2.2.2. XML Schema	16
2.2.3. XPath	17
2.2.3.1. Secuencias	17
2.2.3.2. Variables	17
2.2.3.3. Operadores aritmético-lógicos y de comparación	17
2.2.3.4. Ejes de navegación	17

2.2.3.5.	Predicados . . . . .	18
2.2.3.6.	Funciones . . . . .	18
2.2.4.	XSLT . . . . .	18
2.3.	Combinadores de servicio . . . . .	19
2.4.	Los Diagramas de Secuencias de Mensajes . . . . .	20
2.4.1.	Diagramas MSC básicos . . . . .	21
2.4.1.1.	Entidades . . . . .	21
2.4.1.2.	Mensajes . . . . .	21
2.4.1.3.	Acciones . . . . .	21
2.4.1.4.	Temporizadores . . . . .	22
2.4.1.5.	Condiciones . . . . .	23
2.4.1.6.	Expresiones inline . . . . .	23
2.4.1.7.	Descomposición modular: Referencias . . . . .	24
2.4.1.8.	Corregiones . . . . .	24
2.4.1.9.	Creación y destrucción dinámica de entidades . . . . .	24
2.4.2.	Documentos MSC . . . . .	25
2.4.3.	Documentos MSC de alto nivel (hMSC) . . . . .	26
2.4.4.	MSC para especificación de requisitos y diseño de clientes Web . . . . .	27
2.4.4.1.	XPlore . . . . .	28
2.5.	El lenguaje WebL . . . . .	28
2.5.1.	Características generales . . . . .	29
2.5.2.	Núcleo del lenguaje . . . . .	29
2.5.2.1.	Expresiones, constantes y operadores . . . . .	29
2.5.2.2.	Tipos de datos . . . . .	29
2.5.2.3.	Variables y Contextos . . . . .	29
2.5.2.4.	Sentencias . . . . .	29
2.5.2.5.	Constructores . . . . .	30
2.5.2.6.	Funciones construidas en WebL . . . . .	30
2.5.2.7.	Módulos . . . . .	30
2.5.2.8.	Comentarios . . . . .	30
2.5.3.	Combinadores de Servicios . . . . .	30
2.5.3.1.	Servicio . . . . .	31
2.5.3.2.	Ejecución secuencial $S \text{ ? } T$ . . . . .	31
2.5.3.3.	Ejecución concurrente $S \mid T$ . . . . .	31
2.5.3.4.	Time-out $\text{Timeout}(t, S)$ . . . . .	31
2.5.3.5.	Repetición $\text{Retry}(S)$ . . . . .	31
2.5.3.6.	Non-termination $\text{Stall}()$ . . . . .	31
2.5.4.	Álgebra de marcado . . . . .	32
2.5.4.1.	Páginas, etiquetas, partes y conjuntos de partes . . . . .	32
2.5.4.1.1.	Páginas . . . . .	32
2.5.4.1.2.	Etiquetas . . . . .	32
2.5.4.1.3.	Partes . . . . .	32
2.5.4.1.4.	Conjunto de Partes . . . . .	32
2.5.4.2.	Funciones de búsqueda . . . . .	32
2.5.4.2.1.	Búsqueda de Elementos . . . . .	33
2.5.4.2.2.	Búsqueda de Patrones . . . . .	33
2.5.4.3.	Funciones Misceláneas . . . . .	33
2.5.4.4.	Operadores de conjuntos de partes y funciones . . . . .	33
2.5.4.4.1.	Operadores básicos . . . . .	33
2.5.4.4.2.	Operadores posicionales . . . . .	33
2.5.4.4.3.	Operadores jerárquicos . . . . .	34
2.5.4.4.4.	Operadores regionales . . . . .	34
2.5.4.4.5.	Funciones misceláneas . . . . .	34
2.5.4.5.	Modificación de páginas . . . . .	34
2.5.4.5.1.	Crear partes . . . . .	34
2.5.4.5.2.	Insertar partes . . . . .	34
2.5.4.5.3.	Borrar partes . . . . .	34
2.5.4.5.4.	Sustituir partes . . . . .	34
2.5.5.	Elección de WebL . . . . .	34
2.5.6.	Ejemplos WebL . . . . .	35
2.5.6.1.	Ejemplo 1: Consulta de enlaces rotos . . . . .	35

2.5.6.2.	Ejemplo 2: Descarga de una página Web para lectura off-line . . . . .	36
2.5.6.3.	Ejemplo 3: Uso de la biblioteca Forms.webl . . . . .	36
2.6.	Conclusiones sobre el estado del arte . . . . .	36
<b>3.</b>	<b>Herramienta CASE de desarrollo Web basado en los MSC y el lenguaje WebL</b>	<b>39</b>
3.1.	MSC para representación de tareas en la World Wide Web . . . . .	39
3.1.1.	Elementos de la norma necesarios . . . . .	40
3.1.1.1.	Documentos MSC . . . . .	40
3.1.1.2.	Descripción de tareas Web mediante diagramas MSC . . . . .	40
3.1.1.2.1.	Entidades . . . . .	41
3.1.1.2.2.	Mensajes . . . . .	41
3.1.1.2.3.	Referencias . . . . .	42
3.1.1.2.4.	Control de flujo . . . . .	42
3.1.1.2.5.	Condiciones . . . . .	43
3.1.1.2.6.	Procesamiento interno: Acciones . . . . .	43
3.1.1.2.7.	Threads . . . . .	43
3.1.1.2.8.	Comentarios . . . . .	43
3.1.1.3.	Descripción de combinadores de servicio empleando MSC Básico con expresiones inline . . . . .	44
3.1.1.4.	Variables y tipos de datos . . . . .	47
3.1.1.5.	Funciones misceláneas . . . . .	47
3.1.1.5.1.	PrintLn . . . . .	47
3.1.1.5.2.	ReadLn . . . . .	47
3.1.1.5.3.	Markup . . . . .	47
3.1.1.5.4.	Text . . . . .	47
3.1.2.	Un ejemplo . . . . .	48
3.2.	MSC2WEBL: Diagrama de bloques del sistema . . . . .	48
3.2.1.	Schemas XML . . . . .	48
3.2.1.1.	Schema MSC para descripción de tareas Web . . . . .	48
3.2.1.2.	Esquema proyecto MSC2WebL . . . . .	48
3.2.2.	Transformaciones XSLT . . . . .	48
3.2.2.1.	Transformación a código L <sup>A</sup> T <sub>E</sub> X . . . . .	50
3.2.2.2.	Transformación a código WebL . . . . .	51
3.2.3.	Interfaz gráfico de usuario . . . . .	57
3.2.3.1.	Editor MSC . . . . .	57
3.2.3.2.	Editor L <sup>A</sup> T <sub>E</sub> X . . . . .	58
3.2.3.3.	Editor WebL y consola de ejecución . . . . .	58
3.2.3.4.	Visualizador Marcado Páginas . . . . .	60
3.2.3.5.	Implementación con PyGTK y Glade . . . . .	60
<b>4.</b>	<b>Ejemplos de código</b>	<b>61</b>
4.1.	Ejemplo de estructura . . . . .	61
4.1.1.	Ejemplo A . . . . .	61
4.1.2.	Ejemplo B . . . . .	68
4.2.	Descarga de tira semanal de cómic . . . . .	71
4.3.	Consulta en Google . . . . .	71
4.4.	Cabeceras de Barrapunto . . . . .	78
4.5.	Ejemplo de Threads . . . . .	86
4.6.	Lectura del correo de alumnos de la Universidad Carlos III . . . . .	91
4.6.1.	Documento . . . . .	91
4.6.2.	Programa principal . . . . .	91
4.6.3.	Función <i>login</i> . . . . .	91
4.6.4.	Función <i>mensajes</i> . . . . .	93
4.6.5.	Función <i>imprimir</i> . . . . .	94
4.6.6.	Código WebL final y resultado de ejecución . . . . .	94

<b>5. Conclusiones y trabajos futuros</b>	<b>105</b>
5.1. Conclusiones	105
5.1.1. Conclusiones sobre el modelo propuesto	105
5.1.2. Sobre la aplicación MSC2WEBL	106
5.2. Líneas de trabajo futuro	107
5.2.1. Sobre la metodología propuesta	107
5.2.2. Sobre la aplicación MSC2WEBL	108
<b>6. Presupuesto del proyecto</b>	<b>109</b>
6.1. Historia del proyecto	109
6.1.1. Documentación y configuración del sistema	109
6.1.2. Desarrollo inicial con DIA	109
6.1.3. Desarrollo de la arquitectura XML	110
6.1.4. Desarrollo de la interfaz gráfica de usuario	110
6.1.5. Pruebas y depuración de la aplicación	110
6.1.6. Confección de la batería de ejemplos	110
6.1.7. Redacción de la documentación del proyecto	110
6.1.8. Corrección y depuración final	110
6.1.9. Calendario	110
6.2. Presupuesto	110
6.2.1. Costes de material	111
6.2.2. Costes de personal	111
6.2.3. Presupuesto total	112
<b>Apéndice A: Manual de la aplicación</b>	<b>113</b>
A-1. Introducción	113
A-2. La aplicación MSC2WEBL	113
A-2.1. Vista Propiedades del Proyecto	113
A-2.2. Vista Editor MSC	114
A-2.3. Vista Latex	114
A-2.4. Vista WebL	115
A-2.5. Vista Explorador de Mercado Web	117
A-3. Descripción de tareas Web mediante MSC	118
A-3.1. Documentos MSC	118
A-3.2. Diagramas MSC	119
A-3.3. Entidades	120
A-3.4. Mensajes	120
A-3.5. Referencias	121
A-3.6. Control de flujo	121
A-3.7. Acciones	122
A-3.7.1. Selección de formularios	125
A-3.7.2. Creación de una nueva variable	125
A-3.7.3. Asignación de valor a una variable	125
A-3.7.4. Borrado de elementos	127
A-3.7.5. Impresión por pantalla	127
A-3.7.6. Salvar a disco	127
A-3.7.7. Expresión WebL	127
A-3.8. Descripción de combinadores de servicio	127
A-3.8.1. Combinador Secuencial ( $S_1?S_2$ )	128
A-3.8.2. Combinador Concurrente ( $S_1 S_2$ )	129
A-3.8.3. Timeout( $t,S$ )	129
A-3.8.4. Repetición	130
A-3.9. Threads	130
A-3.10. Comentarios	131
A-3.11. Variables y tipos de datos	131
A-3.12. Funciones Misceláneas	133
A-3.12.1. Markup	134
A-3.12.2. Text	134
A-4. Opciones de configuración de la aplicación	134
A-5. Creación de proyectos MSC2WEBL	135
A-6. Guía Instalación	135



<b>Apéndice B: Guía de desarrollo e instalación</b>	<b>137</b>
B-1. El sistema operativo Linux: Debian SID . . . . .	137
B-2. Python . . . . .	137
B-3. Procesado XML con Python . . . . .	138
B-3.1. Jaxml . . . . .	138
B-3.2. 4Suite . . . . .	138
B-3.3. XSV . . . . .	138
B-4. GTK y Glade . . . . .	138
B-5. Dibujo de los diagramas . . . . .	139
B-6. WebL . . . . .	139
<b>Bibliografía</b>	<b>141</b>



# Índice de figuras

1.1. Crecimiento de el WWW . . . . .	2
2.1. Ejemplo de documento RSS (Barrapunto.com) . . . . .	9
2.2. Ejemplo de documento RSS (NewsForge.com) . . . . .	10
2.3. Diferencia visual entre los sitios de Barrapunto.com y NewsForge.com . . . . .	11
2.4. Sistema mediador . . . . .	13
2.5. Ejemplo de documento XML . . . . .	15
2.6. Ejemplo de DTD . . . . .	16
2.7. XML Schema para el ejemplo de catálogo de joyas . . . . .	16
2.8. Ejemplo de plantilla XSLT . . . . .	19
2.9. Ejemplos de entidades . . . . .	21
2.10. Ejemplos de mensajes . . . . .	21
2.11. Ejemplo Acción . . . . .	22
2.12. Ejemplo de temporizadores . . . . .	22
2.13. Ejemplos de condiciones . . . . .	23
2.14. Ejemplo de expresiones inline . . . . .	24
2.15. Ejemplo de referencia . . . . .	25
2.16. Ejemplo de corrección . . . . .	25
2.17. Ejemplo de creación dinámica de entidades . . . . .	26
2.18. Ejemplo de documento MSC . . . . .	26
2.19. Ejemplo de diagrama MSC de alto nivel . . . . .	27
2.20. Código WebL del script de comprobación de enlaces . . . . .	35
2.21. Código WebL del script de descarga de página Web . . . . .	36
2.22. Ejemplo del uso de Forms.webl . . . . .	37
3.1. Ejemplo de Documento de Proyecto . . . . .	41
3.2. Ejemplo de mensaje . . . . .	42
3.3. Ejemplo de referencia . . . . .	42
3.4. Ejemplo de control del flujo . . . . .	43
3.5. Ejemplo de acción XPath . . . . .	44
3.6. Ejemplo del uso de Threads . . . . .	44
3.7. Ejemplo del uso de comentarios . . . . .	44
3.8. Combinador de Servicio <i>secuencial</i> en MSC . . . . .	45
3.9. Combinador de Servicio <i>concurrente</i> en MSC . . . . .	45
3.10. Combinador de Servicio <i>repetición</i> en MSC . . . . .	46
3.11. Ejemplo de los combinadores de servicio <i>stall</i> y <i>timer</i> en MSC . . . . .	46
3.12. Ejemplo de tarea Web representada mediante MSC . . . . .	49
3.13. Arquitectura del sistema . . . . .	50
3.14. Arquitectura del interfaz gráfico de usuario . . . . .	50
3.15. Elementos definidos por el Schema MSC . . . . .	51
3.16. Schema XML de datos de proyecto MSC2WEBL . . . . .	52
3.17. Parámetros definibles por el usuario en un diagrama MSC . . . . .	53
3.18. Código XML antes de aplicar la plantilla de transformación a $\text{\LaTeX}$ . . . . .	54
3.19. Código $\text{\LaTeX}$ después de aplicar la plantilla XSLT . . . . .	55
3.20. Variables y módulos WebL predefinidos . . . . .	56
3.21. Código WebL después de aplicar la plantilla XSLT . . . . .	56
3.22. Ventana de edición de diagramas MSC (Modo Gráfico) . . . . .	57
3.23. Ventana de edición de diagramas MSC (Modo XML) . . . . .	58
3.24. Ventana de edición $\text{\LaTeX}$ . . . . .	59
3.25. Ventana de edición y ejecución de código WebL . . . . .	59

4.1. Ejemplo A: Documento MSC . . . . .	62
4.2. Representación gráfica de la composición de los bloques en el ejemplo <i>ejEstructuraA</i> . . . . .	62
4.3. Ejemplo A: Diagrama MSC de <i>util1</i> . . . . .	62
4.4. Ejemplo A: Diagrama de <i>util2</i> . . . . .	63
4.5. Ejemplo A: Diagrama MSC de <i>util3</i> . . . . .	63
4.6. Ejemplo A: Diagrama MSC de <i>bloqueA</i> . . . . .	63
4.7. Ejemplo A: Diagrama MSC de <i>bloqueB</i> . . . . .	64
4.8. Ejemplo A: Código XML del documento MSC . . . . .	64
4.9. Ejemplo A: Código XML del diagrama MSC de <i>util1</i> . . . . .	64
4.10. Ejemplo A: Código XML del diagrama MSC de <i>util2</i> . . . . .	65
4.11. Ejemplo A: Código XML del diagrama MSC de <i>util3</i> . . . . .	65
4.12. Ejemplo A: Código XML del diagrama MSC <i>bloqueA</i> . . . . .	66
4.13. Ejemplo A: Código XML del diagrama MSC <i>bloqueB</i> . . . . .	66
4.14. Código WebL correspondiente al Ejemplo A de estructuras MSC . . . . .	67
4.15. Ejemplo B: Diagrama MSC . . . . .	68
4.16. Salidas de los programas <i>ejemplo A</i> y <i>ejemplo B</i> . . . . .	68
4.17. Ejemplo B: Código XML correspondiente . . . . .	69
4.18. Ejemplo B: Código WebL correspondiente . . . . .	70
4.19. Tira Ecol . . . . .	71
4.20. Ejemplo tira Ecol: Diagrama MSC . . . . .	71
4.21. Ejemplo Tira Ecol: Código XML . . . . .	72
4.22. Ejemplo Tira Ecol: Código WebL . . . . .	72
4.23. Marcado de la página principal de Google.es . . . . .	73
4.24. Marcado de una página de resultados de búsqueda en Google . . . . .	73
4.25. Ejemplo Google: Diagrama MSC . . . . .	74
4.26. Ejemplo Google: Código XML del diagrama MSC . . . . .	76
4.27. Ejemplo Google: Código WebL correspondiente . . . . .	76
4.28. Ejemplo Google: Resultado de la ejecución . . . . .	77
4.29. Fichero RSS de Barrapunto.com . . . . .	78
4.30. Ejemplo BarraPunto: diagrama descargaRSS . . . . .	79
4.31. Ejemplo Barrapunto: diagrama imprimir . . . . .	80
4.32. Código XML del diagrama descargaRSS . . . . .	81
4.33. Código XML del diagrama imprimir . . . . .	83
4.34. Ejemplo barraPunto: código WebL . . . . .	84
4.35. Salida del programa de descarga de cabeceras de Barrapunto . . . . .	85
4.36. Diagrama MSC del ejemplo con Threads . . . . .	86
4.37. Resultado de ejecución del ejemplo de Threads . . . . .	87
4.38. Código XML del diagrama MSC del ejemplo con Threads . . . . .	89
4.39. Código WebL del ejemplo de Threads . . . . .	90
4.40. Ejemplo Correo Alumnos: Documento MSC . . . . .	91
4.41. Ejemplo Correo Alumnos: Código XML del Documento MSC . . . . .	91
4.42. Ejemplo Correo Alumnos: Diagrama MSC del programa principal . . . . .	92
4.43. Ejemplo Correo Alumnos: Código XML del diagrama MSC del programa principal . . . . .	92
4.44. Ejemplo Correo Alumnos: Marcado de la página de acceso a la aplicación . . . . .	93
4.45. Ejemplo Correo Alumnos: Diagrama MSC de <i>login</i> . . . . .	94
4.46. Ejemplo Correo Alumnos: Código XML del diagrama MSC <i>login</i> . . . . .	96
4.47. Ejemplo Correo Alumnos: Marcado HTML de la página inbox . . . . .	96
4.48. Ejemplo Correo Alumnos: Diagrama MSC de <i>mensajes</i> . . . . .	97
4.49. Ejemplo Correo Alumnos: Código XML del diagrama <i>mensajes</i> . . . . .	98
4.50. Ejemplo Correo Alumnos: Diagrama MSC de <i>imprimir</i> . . . . .	99
4.51. Ejemplo Correo Alumnos: Código XML del diagrama <i>imprimir</i> . . . . .	101
4.52. Ejemplo Correo Alumnos: Código WebL correspondiente . . . . .	103
4.53. Ejemplo Correo Alumnos: Salida correspondiente a la ejecución del script . . . . .	103
6.1. Calendario del proyecto . . . . .	111
A-2. Ventana de propiedades de proyecto . . . . .	114
A-3. Ventana del editor MSC (Representación Gráfica) . . . . .	115
A-4. Ventana del editor MSC (Representación XML) . . . . .	115
A-5. Ventana del editor Latex . . . . .	116
A-6. Ventana WebL (edición y ejecución) . . . . .	116

A-7. Ventana Navegador . . . . .	117
A-8. Documento MSC fileDownloading . . . . .	118
A-9. Dialogo Nuevo Documento MSC . . . . .	119
A-10.Ejemplo de diagrama MSC . . . . .	119
A-11.Dialogo Nuevo Diagrama MSC . . . . .	120
A-12.Ejemplo Entidad . . . . .	120
A-13.Ejemplo mensaje . . . . .	121
A-14.Dialogo Nuevo Mensaje . . . . .	121
A-15.Ejemplo de referencia MSC . . . . .	122
A-16.Dialogo Nueva Referencia . . . . .	122
A-17.Dialogo Nueva Expresión de Control de Flujo . . . . .	123
A-18.Ejemplo de expresiones de control de flujo anidadas . . . . .	124
A-19.Dialogo Nueva Acción de selección de formulario . . . . .	125
A-20.Dialogo Nueva Acción de creación de variable . . . . .	126
A-21.Dialogo Nueva Acción de asignación . . . . .	126
A-22.Ejemplo de acciones . . . . .	127
A-23.Diálogo Nueva Acción de impresión por pantalla . . . . .	128
A-24.Ejemplo de combinador secuencial . . . . .	129
A-25.Diálogo Nuevo combinador secuencial de servicio . . . . .	129
A-26.Código XML de un combinador de servicio . . . . .	130
A-27.Diálogo Nuevo combinador concurrente de servicio . . . . .	131
A-28.Ejemplo de <i>Timeout</i> . . . . .	131
A-29.Diálogo Nuevo combinador de servicio timeout . . . . .	132
A-30.Diálogo Nuevo combinador de servicio repetición . . . . .	132
A-31.Ejemplo de combinador <i>Repetición</i> . . . . .	133
A-32.Ejemplo de Threads . . . . .	133
A-33.Diálogo Nuevo Thread de ejecución . . . . .	133
A-34.Diálogo Nuevo comentario . . . . .	133
A-35.Diálogo opciones de la aplicación . . . . .	134
B-1. Fuentes oficiales de Debian Sid en <a href="http://rediris.es">rediris.es</a> . . . . .	137



# Capítulo 1

## Introducción

En este primer capítulo introductorio se incluye una breve descripción de la historia de Internet con el fin de introducir el marco de trabajo y las motivaciones de este proyecto. Se hace una breve descripción de los distintos objetivos a alcanzar, así como una descomposición del proyecto en fases de trabajo y una breve descripción de la estructura de la memoria.

### 1.1. Breve historia de Internet

La agencia gubernamental estadounidense ARPA (Agencia para Proyectos de Investigación Avanzados, hoy en día DARPA) y un grupo de científicos de varias universidades estadounidenses se unieron en la década de los 60 con el objetivo de sacar mayor partido a los escasos, incompatibles y caros ordenadores de la época. En aquellos tiempos, los ordenadores eran enormes máquinas que ocupaban varios metros cuadrados, y que, para ser utilizados, necesitaban de un terminal de conexión. Cada fabricante producía equipos que eran incompatibles entre sí, ya que su funcionamiento interno y la forma de trabajar con ellos era distinta. Como solución a este problema, se vio que era necesario implementar una red que interconectase los diferentes ordenadores de una forma estándar.

Como resultado de estas investigaciones, se creó una red, de nombre ARPANET, que fue el embrión de lo que hoy se conoce como Internet. El 2 de septiembre de 1969, en la Universidad de California en Los Ángeles (UCLA), se conectó el primer ordenador a esta primitiva red. En pocos meses se unieron más equipos de distintas universidades, convirtiéndose así en una magnífica herramienta de comunicación para los investigadores.

Con el tiempo, fueron surgiendo otras redes paralelas a ARPANET. Sin embargo, todas eran cerradas e incompatibles entre sí. Así que, buscando que las diferentes redes pudieran interconectarse, se creó el protocolo TCP/IP en 1973, que todavía sigue usándose en la actualidad.

Sin embargo, Internet todavía tardó mucho en madurar y durante casi dos décadas pasó totalmente desapercibido para el gran público. Fue en los ámbitos universitarios y científicos dónde se le sacó mayor provecho, mediante el uso del correo electrónico, la transferencia de ficheros (FTP) o la publicación de mensajes en los grupos de noticias. También existían sistemas como gopher o BBS, considerados los antecesores de las páginas Web actuales y que permitían el acceso a la información de formas más primitivas.

En la década de los 90, gracias al trabajo de Tim Berners-Lee en el CERN (Center for European Particle Research) se desarrolló un nuevo sistema de organización y acceso a la información contenida en Internet. Nació así la World Wide Web, haciendo crecer el interés empresarial por el fenómeno y dando el espaldarazo definitivo para que Internet se convirtiera en lo que es hoy día, el mayor repositorio de información públicamente accesible hasta la fecha.

### 1.2. Motivación

La evolución y asentamiento del WWW ha modificado de forma profunda muchos de los ámbitos de la vida cotidiana, no sólo en el campo empresarial sino también en el del usuario doméstico. La difusión de información de todo tipo ha encontrado en el Web un nuevo soporte, tanto a nivel de Internet como medio de comunicación global para publicar información general, servir de escaparate a las tiendas electrónicas o,

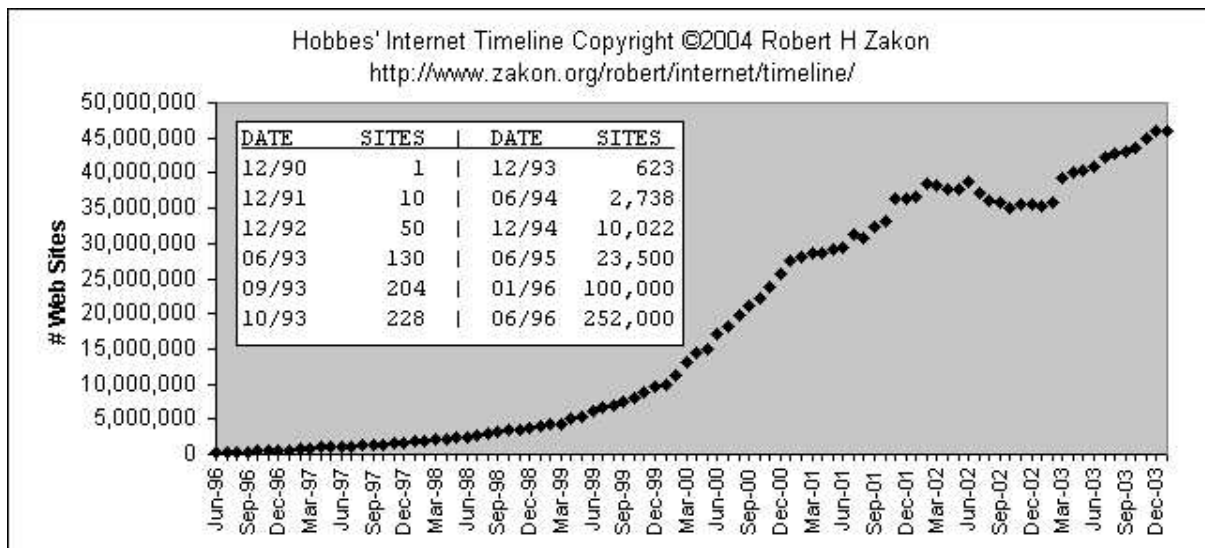


Figura 1.1: Crecimiento de el WWW

cada vez más, en las intranets corporativas de muchas empresas.

Cada vez es más común el uso de aplicaciones remotamente accesibles desde el Web para diversos propósitos, por ejemplo la compra de artículos o la consulta de movimientos bancarios. Al mismo tiempo, la cantidad de información disponible en el WWW crece día a día y con ella el número de usuarios que acceden a el Web para el envío y recepción de información.

Se estima [1] que actualmente existen aproximadamente más de 46 millones de servidores, muchos de los cuales pueden albergar más de un sitio Web. A pesar de que estos datos son aproximados y ya obsoletos, dan una idea del ritmo de crecimiento de Internet en sus 35 años de vida. La gráfica 1.1 representa el crecimiento de sitios Web desde 1996 hasta 2004.

En muchos de los accesos a Internet es necesario un alto grado de interacción por parte del usuario con la red. Esta interacción se realiza, en la mayoría de los casos, empleando navegación manual con la simple ayuda del teclado y el ratón, usando navegadores cuyo único fin es la visualización paginada de la información contenida en la red y el envío de información empleando formularios.

La heterogeneidad de datos en el Web, cada vez mayor, el aumento del número y variedad de aplicaciones y la aparición de nuevas tecnologías (muchas de ellas no estandarizadas y propietarias) ponen en evidencia muchos de los problemas y carencias de la navegación tradicional, al tiempo que incrementan los tiempos de procesamiento manual de la información en el WWW.

Muchos son los lenguajes de programación para el Web, no sólo del lado del servidor, que queda fuera del alcance de este proyecto, sino también del lado cliente, que permiten automatizar tareas de navegación y procesamiento de información en Internet. Los más conocidos son Java, C, C++, C#, Perl, Python, etc, que cuentan con un gran conjunto de librerías específicas para el Web. Otro lenguaje menos conocido, y empleado en este proyecto es WebL[2][3][4], escrito en Java y que auna sencillez de desarrollo con un modelo de programación para el Web muy robusto y potente.

Si bien estos lenguajes para el Web y los nuevos modelos de programación para Internet han de tener en cuenta las características de este nuevo medio para garantizar unas cotas de robustez y funcionalidad en un medio tan dinámico y sujeto a fallos como es el Web.

### 1.3. Enfoque

Este proyecto plantea cómo los Diagramas de Secuencia de Mensajes[5][6] pueden ser aplicados a la especificación de requisitos de sistemas diseñados para automatizar tareas o recopilar información residente en Internet, de tal forma que se facilite y abarate el proceso de desarrollo y mantenimiento de estos sistemas.



La norma MSC es muy amplia y no fue diseñada inicialmente para el tratamiento de tareas Web. Por ello es necesario seleccionar el conjunto de elementos necesarios para este fin, así como introducir algunos nuevos y variaciones sobre otros existentes, para por ejemplo representar los combinadores de servicio[7]. Una vez definido el sublenguaje MSC adecuado, este podrá emplearse directamente a la automatización de tareas en Internet.

Para demostrar esto, se ha creado una herramienta CASE de desarrollo Web basado en los diagramas MSC. Esta herramienta incluye un editor MSC de forma que los diagramas se construyan de forma sencilla, empleando para funcionamiento interno una gramática textual, en forma de esquema XML[8] fácilmente transformable a representaciones gráficas, más intuitivas. La aplicación incluye así mismo los mecanismos de transformación de los diagramas MSC a código WebL ejecutable, un entorno de ejecución de los scripts de navegado y un explorador de código XHTML [9] de las páginas Web susceptibles de ser navegadas de forma automática.

## 1.4. Objetivos del proyecto

Varios son los objetivos marcados con este proyecto. Los enumeramos a continuación:

1. Analizar la automatización de tareas en el WWW desde el punto de vista del cliente, descubriendo cuáles son los principales problemas que plantea y qué soluciones se han propuesto hasta la fecha.
2. Estudiar por qué los diagramas MSC son apropiados como mecanismo de especificación de requisitos de sistemas de navegación automatizada, y cómo hay que adaptarlos para que se puedan utilizar para esta tarea.
3. Estudiar y aprovechar las ventajas que proporciona XML, no sólo en cuanto a la confección de páginas Web (usando XHTML o mediante plantillas XSLT), sino también qué ventajas aportan las páginas basadas en XML a la hora de ser navegadas por aplicaciones software (en oposición a la navegación manual).
4. Estudiar el lenguaje WebL y ver cómo aplica los conceptos de programación Web, desarrollando ejemplos concretos y funcionales y estudiando las ventajas e inconvenientes de este lenguaje frente a otros existentes en el mercado.
5. Implementar una herramienta CASE que aglutine las fases de especificación, desarrollo y documentación de sistemas Web (en el lenguaje WebL) basada en los Diagramas de Secuencia de Mensajes, facilitando los desarrollos incluso a personas sin demasiados conocimientos de programación. Para ello se empleará internamente un conjunto de esquemas XML y transformaciones XSLT definidas para este fin.
6. Confeccionar una batería de ejemplos, desarrollados tanto manualmente como utilizando la aplicación, con el fin de comparar el desarrollo manual frente al desarrollo usando la aplicación.

## 1.5. Fases del proyecto

El desarrollo del presente proyecto se ha descompuesto, desde su inicio hasta su conclusión, en varias fases distintas cubriendo cada una de ellas diferentes aspectos del desarrollo. Podemos descomponer el proyecto en seis fases que enumeramos a continuación:

- **Fase 0:** documentación y configuración del sistema.
- **Fase 1:** desarrollo inicial con DIA[10].
- **Fase 2:** desarrollo de la arquitectura XML.
- **Fase 3:** desarrollo de la interfaz gráfica de usuario.
- **Fase 4:** pruebas y depuración de la aplicación.
- **Fase 5:** colección de la colección de ejemplos.
- **Fase 6:** redacción de la documentación del proyecto.
- **Fase 7:** corrección.

## 1.6. Contenido de la memoria

En el siguiente capítulo se introduce el estado del arte, analizando en profundidad la automatización de tareas Web, presentando las características de Internet y su influencia en los modelos de programación Web, así como los distintos tipos de programas de navegación automática y los retos de la programación en el WWW. Se introducen algunas de las tecnologías implicadas en el proyecto como son el lenguaje XML (eXtensive Markup Language) y los lenguajes XPath y XSLT. Se estudian los Diagramas de Secuencias de Mensajes y el lenguaje de programación WebL. Por último se incluyen las conclusiones más relevantes sobre el estudio del arte.

El capítulo tercero presenta la arquitectura de la aplicación generada, MSC2WEBL, basada por entero en el lenguaje XML. Se delimita el conjunto de los elementos MSC necesarios para su aplicación a la descripción de sistemas Web y se documentan el esquema XML definido para representar los diagramas MSC y las distintas transformaciones XSLT[11] para generar la representación gráfica de los diagramas y el código WebL ejecutable. Así mismo se explica la interfaz gráfica del programa y cómo ésta ha sido implementada con Python[12] y PyGTK[13].

El cuarto capítulo incluye una batería de ejemplos de sistemas de navegación automática, incluyendo para todos ellos no sólo el código WebL ejecutado, sino también los diagramas MSC que lo representan (tanto en forma gráfica como textual, basada en el esquema XML definido).

El capítulo quinto presenta las conclusiones del proyecto y las líneas de trabajo futuro posibles.

El capítulo sexto incluye una descripción de la historia del proyecto, descompuesto en fases, así como un presupuesto detallado del mismo.

Por último, además de las referencias bibliográficas, se incluyen una serie de apéndices que contienen:

- **Apéndice A:** manual de usuario de la aplicación.
- **Apéndice B:** guía de desarrollo e instalación, explicando una por una las librerías y programas utilizados en el desarrollo y como conseguirlos e instalarlos.

# Capítulo 2

## Estado del arte

En este capítulo se hace un breve repaso de las principales características del Web como medio de programación, presentando las diferencias entre la navegación automática y la tradicional, los diferentes tipos de sistemas para procesado del WWW, el modelo de programación sobre el Web y los costes derivados de la creación de asistentes de navegación Web automática.

Se presentan a continuación el lenguaje de marcas XML y los lenguajes XPath y XSLT, familia de tecnologías derivadas de XML que están teniendo una influencia decisiva en el desarrollo Web actual, no sólo en el lado del servidor, donde cada vez son más las páginas Web escritas en XHTML o XML, sino también del lado cliente, donde las distintas técnicas permiten la extracción de información relevante y el procesado de datos dentro de documentos XML de forma eficiente.

Algunas de las metodologías existentes de programación en Internet, tienen enfoques similares o han sido incluidas en la presentada en este proyecto. Se comentan brevemente aquí los *Combinadores de Servicio* y se introducen los diagramas MSC y algunas de las aproximaciones hasta la fecha de aplicación de los MSC a la programación del Web.

Por último, se presenta el lenguaje WebL, clave en el desarrollo del proyecto.

### 2.1. Análisis de las tareas en el Web

Internet se ha convertido en el mayor repositorio de conocimiento de la humanidad. Cada vez es mayor el número de de servicios<sup>1</sup>, información almacenada y aplicaciones remotamente accesibles mediante el Web para diversos propósitos. El acceso a muchas de estas fuentes de información se realiza mediante interfaces diseñados para el Web que, en muchos de los casos, no se adecuan a las necesidades concretas de todos los posibles usuarios, empleando para ello navegadores Web. Estos están limitados como herramientas de trabajo, ya que carecen de cualquier tipo de propósito más allá que el de la mera presentación paginada de información al usuario y la recogida de datos mediante formularios.

La heterogeneidad actual de el WWW es enorme: no sólo existen infinitas formas posibles de organizar y estructurar los contenidos dentro de cada página Web, sino que además existen múltiples variantes para distribuir la información entre las distintas páginas enlazadas unas a otras. Esta heterogeneidad dificulta la integración de datos en el Web que, bien sea para obtener datos con valor añadido (sindicación de contenido) o para alimentar otras aplicaciones, es un campo de gran interés, al ser cada vez más necesario el automatizar el manejo de grandes volúmenes de datos en Internet.

En ocasiones, los usuarios se enfrentan a la tarea de rellenar varias veces el mismo conjunto de formularios o seguir el mismo conjunto de enlaces para obtener una información deseada o manejar una aplicación Web. Estos usuarios no disponen de otras herramientas más allá del navegador, e invierten para ello gran cantidad de tiempo en tareas que, dado su automatismo, sería muchas veces posible que se realizaran de forma autónoma por un ordenador. Para que esto sea posible, han de surgir nuevos modelos de programación adecuados a las características de el Web de tal forma que se facilite la creación y mantenimiento de este tipo de sistemas.

---

<sup>1</sup>Se define servicio Web como toda aplicación accesible desde el Web o conjunto de páginas lógicamente enlazadas para un fin común y publicadas bajo una misma estructura.

### 2.1.1. Navegación automática

El conjunto de servicios y aplicaciones en el Web, así como la cantidad de información en ella es cada vez mayor haciendo que el tratamiento manual de estas tareas sea cada vez más costoso en tiempo y esfuerzo. En el caso del Web, los beneficios de la automatización se hacen más patentes en aquellas tareas que necesiten procesar grandes volúmenes de información o que deban ser frecuentemente ejecutadas. La automatización de tareas Web persigue incrementar la productividad de las personas, minimizando los errores propios de la naturaleza humana y mejorando así la eficiencia en la realización de trabajos y en la optimización de recursos utilizados.

#### 2.1.1.1. Diferencias entre navegación manual y navegación automática

Las principales diferencias entre la navegación manual y la automática, resumidas en la tabla 2.1, aparecen detalladas a continuación.

	Navegación manual	Navegación automática
<b>Intervención</b>	Humana	Ordenador
<b>Esfuerzo</b>	De navegación	De programación
<b>Errores</b>	De navegación	De programación
<b>Tiempo de respuesta</b>	Significativo	Ínfimo
<b>Requisitos</b>	Conexión y navegador	Conexión
<b>Repositorios</b>	No programables	Programables
<b>Procesamiento</b>	Cálculo mental	Automatizable
<b>Volumen de datos</b>	Limitado	Enorme
<b>Implantación</b>	Factible	Costosa
<b>Adaptabilidad ante cambios</b>	Tolerable	Costosa

Cuadro 2.1: Diferencias entre la navegación manual y la navegación automática

**2.1.1.1.1. Esfuerzo** El Web actual está diseñado para ser navegado de forma interactiva, seleccionando uno a uno cada enlace a seguir, lo que supone en muchas ocasiones un gran esfuerzo. Por el contrario, una tarea automatizada por un programa capaz de navegar en lugar del usuario, emulando el comportamiento de la actuación conjunta de éste y el navegador, puede reducir significativamente ese coste de navegación.

**2.1.1.1.2. Propensión a errores** La necesidad de interacción de las personas en la navegación manual aumenta en gran medida la propensión a errores durante la ejecución de la tarea, en especial cuando el conjunto de datos que debe ser manipulado es voluminoso. Por el contrario, un programa que navegue automáticamente por el Web puede manipular eficiente y adecuadamente grandes volúmenes de información, incluso a pesar de que ésta se encuentre distribuida en varias fuentes de datos.

**2.1.1.1.3. Tiempo de respuesta** Pese a que el rendimiento de una aplicación Web está principalmente condicionado por el tiempo de respuesta del servidor y de las conexiones que comunican a éste con el cliente, lo cierto es que, las personas, cuando navegamos delante de un navegador en el que debemos introducir interactivamente nuestras órdenes, tenemos unos tiempos de respuesta significativos. Además, incluso en un entorno donde el usuario esté sumamente concentrado en el uso del navegador, el hecho de tener que activar mecánicamente unos dispositivos de introducción de datos, como el teclado y el ratón consume un determinado tiempo.

**2.1.1.1.4. Requisitos hardware y software** Uno de los grandes problemas de los navegadores es que, además de la intervención humana para poder proceder al seguimiento de enlaces, requieren de hardware y software especializado, de forma que gran parte del Web ahora mismo está pensada exclusivamente para ser accesible desde ordenadores personales con unas resoluciones de pantalla determinadas, y con unos requisitos software específicos muy concretos.

Sin embargo, la proliferación de un cada vez mayor número de dispositivos electrónicos conectados a Internet, como PDA's, cabinas públicas de acceso, teléfonos móviles, etc, está aumentando el número de dispositivos con reducidas capacidades de visualización de datos, haciendo evidente que la navegación manual

está prácticamente limitada a un conjunto muy pequeño de dispositivos, restricciones a las que no está sujeta la navegación automática.

**2.1.1.1.5. Adecuación de repositorios** Las personas al navegar por Internet, recolectan los datos relevantes muy a menudo mediante métodos poco automatizables e inadecuados para grandes volúmenes de datos. Desde el punto de vista de la automatización de tareas, es deseable la utilización de repositorios de datos accesibles desde programas de ordenador, como variables de memoria o ficheros con algún tipo de estructura conocida por el programador.

**2.1.1.1.6. Procesamiento** En muchas ocasiones, el procesamiento que se debe realizar sobre los datos Web es sencillo. A pesar de ello, el volumen de datos en el Web dificulta el realizar mentalmente estos procesamientos. Estas tareas pueden ser fácilmente realizadas por un programa capaz de acceder a los datos involucrados si estos se encuentran convenientemente almacenados en un repositorio adecuado de datos como los mencionados en el punto anterior.

**2.1.1.1.7. Coste de implantación y adaptabilidad** Desarrollar aplicaciones que automaticen las tareas de navegación en el Web es costoso. Existen pocas técnicas específicamente orientadas a minimizar los costes de implantación de este tipo de sistemas[14] [15][16][17]. Además hay que tener en cuenta que cualquier cambio en la estructura de las páginas involucradas en una tarea puede acabar requiriendo una ardua labor de mantenimiento. Por el contrario, estos cambios en la estructura de los sitios Web apenas afectan a la navegación manual, puesto que el comportamiento humano es fácilmente adaptable a las nuevas circunstancias.

## 2.1.2. Características de los datos de el Web

Para procesar automáticamente información en el Web hay que tener en cuenta las principales características de los datos cuyo tratamiento se pretende automatizar. Enumero a continuación algunas de las características más relevantes de los datos en el Web.

### 2.1.2.1. Orientación a la visualización

Una de las grandes barreras de la automatización del Web actual es que está muy orientado a la visualización. La gran mayoría de los desarrolladores no han diseñado sus páginas para ser procesadas por otro tipo de aplicaciones distintas a los navegadores e incluso, en muchos de los casos, éstas sólo pueden visualizarse usando ciertos navegadores concretos.

La gran mayoría de sitios centran aún el uso de su marcado HTML[18][19] en aspectos primordialmente visuales (tamaños y tipos de letra, colores, alineamientos, imágenes, etc) enfocados a impactar visualmente al usuario en lugar de hacer fácilmente accesible la información contenida. Existen algunas tecnologías como las hojas de estilo CSS[20] y otros lenguajes de marcado más apropiados que HTML que permiten independizar la información contenida de la visualización que se hace de la misma. Sin embargo, al nacer estas tecnologías con posterioridad al propio Web, la gran mayoría de sitios emplean aún de forma mayoritaria marcado HTML y no tienen en cuenta generalmente estos aspectos.

### 2.1.2.2. Accesibilidad Web

La correcta operatividad con independencia del navegador utilizado o el dispositivo de acceso, o incluso para personas con alguna discapacidad física o sistemas de navegación automática, es una de las grandes necesidades del Web actual. El W3C[21] ha definido unas pautas para la mejora de la accesibilidad de los sitios Web[22] que están encontrando últimamente un decidido apoyo en numerosas personas e instituciones para conseguir que el acceso a los contenidos Web sea funcional, con independencia de cuál sea la plataforma de acceso, tanto hardware como software. El objetivo principal de estas iniciativas consiste en que las páginas publicadas en el Web sean accesibles a sus usuarios independientemente del dispositivo de acceso empleado por cada uno de ellos, a costa de reducir al mínimo el uso del marcado HTML orientado a la visualización, delegando esta tarea en las hojas de estilo y simplificando así el marcado estructural de las páginas.

Sin embargo, la escasa implantación práctica de las normas y recomendaciones del W3C durante muchos años hace prever que la transición hacia un Web accesible será aún lenta, debido a la inercia del Web, asentada en su gran tamaño, y en la generalmente escasa orientación de los desarrolladores a estos aspectos.

### 2.1.2.3. Distribución de la información

Los datos que aparecen en el Web y son necesarios para una tarea no suelen aparecer todos integrados en el mismo documento, sino que suelen aparecer distribuidos en varios de ellos por cuestiones de organización. La información que necesita ser combinada para realizar una tarea puede encontrarse distribuida de muy diversas formas:

- Entre los documentos de diversos sitios Web, por ejemplo de la misma temática.
- Entre cada uno de los documentos individuales que componen un documento multimedia (por ejemplo una página con frames frames).
- Mezclada dentro de una misma página con información no relevante para la tarea (como por ejemplo publicidad en un sitio Web).
- Entre diversas páginas enlazadas entre sí.

### 2.1.2.4. Ausencia de semántica en el marcado

Cualquier dato que aparezca en una página HTML no puede distinguirse fácilmente de forma automatizada ya que el marcado HTML no distingue entre los distintos tipos de datos. Por otro lado, cualquier combinación de etiquetas pueden usarse para marcar unos datos u otros, por lo que es el usuario final, quien en función del contexto en que aparezcan, infiere la semántica de los datos que visualiza. En la navegación manual se debe tomar a menudo decisiones sobre que formularios rellenar o que enlaces seguir dependiendo de los datos visualizados en pantalla, pero que el ordenador no es capaz de distinguir al ser HTML un lenguaje poco orientado a la descripción semántica y con un uso muy orientado a la visualización.

Aunque en general queda fuera del alcance de este proyecto, que busca la automatización de tareas en lo que se conoce como *Web legado*<sup>2</sup>, en los últimos tiempos se ha definido y comienza a emplearse lo que se conoce como Web Semántico. La Web Semántica[23] *es una extensión del Web actual en la cuál la información se presenta con una semántica bien definida, con lo que se permite a personas y ordenadores trabajar en cooperación.* Permite un marco de trabajo común que permite la compartición y reutilización de datos entre distintas aplicaciones, empresas y miembros de comunidades. Está basada en RDF[24], *Resource Description Framework*, que integra varias aplicaciones para la descripción semántica de los datos contenidos en las páginas, usando XML para la sintaxis y URIs para el nombrado de los elementos, basado en meta datos. De esta forma, numerosos sitios actuales sindicán sus contenidos usando RDF de forma que el intercambio de información entre aplicaciones es muy sencillo.

Uno de los ejemplos ejecutables que acompañan el proyecto está basado en este concepto. Es un script para descargar las cabeceras de Barrapunto.com[25], que al estar sindicado mediante RDF (ver figura 2.1), se hace de forma sencilla y que puede ser aplicado sin cambio alguno a otros sitios Web que empleen la misma tecnología, por ejemplo NewsForge[26] cuyo código aparece en la figura 2.2.

El empleo de esta nueva tecnología por los desarrolladores Web, permite un mecanismo de acceso rápido a la información para aplicaciones de minería de datos o automatización de tareas en el WWW. Por otro lado, mediante el uso de RSS, sitios Web de aspecto visual muy diferente se navegan automáticamente de forma similar. Véase en la figura 2.3 el diferente aspecto visual de las páginas anteriores.

Esta tecnología esta aún en un estado de incipiente estudio y desarrollo, sin embargo es cada vez más usada en el ámbito científico, los sitios de información (no necesariamente técnica) y los blogs y páginas personales de muchos usuarios. No obstante no lo trataremos en este proyecto y nos centraremos en la automatización de tareas en el *Web Legado*, que si adolece frecuentemente de esta ausencia de semántica en el marcado.

### 2.1.2.5. Voluminosidad

Desde la creación del WWW hasta hoy día, el número de aplicaciones accesibles desde cualquier punto del planeta mediante el Web ha crecido enormemente, proporcionando una ingente cantidad de información para la que no existen apenas medios adecuados de manipulación. El procesado manual de toda esta información se hace en muchos de los casos inabordable.

---

<sup>2</sup>Web tradicional, basado en HTML y orientado a la visualización en navegadores.

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://purl.org/rss/1.0/"
  xmlns:slash="http://slashcode.com/rss/1.0/modules/Slash/"
  xmlns:taxo="http://purl.org/rss/1.0/modules/taxonomy/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:syn="http://purl.org/rss/1.0/modules/syndication/"
  xmlns:admin="http://Webns.net/mvcb/"
>
<channel rdf:about="http://barrapunto.com/">
  <title>Barrapunto</title>
  <link>http://barrapunto.com/</link>
  <description>La informaci#243;n que te interesa</description>
  <dc:language>es</dc:language>
  <dc:rights>Copyright &amp;copy; 2003, Barrapunto</dc:rights>
  <dc:date>2004-08-24T10:14:07+00:00</dc:date>
  <dc:publisher>Barrapunto S.L.</dc:publisher>
  <dc:creator>Webmaster@barrapunto.com</dc:creator>
  <dc:subject>Software libre</dc:subject>
  <syn:updatePeriod>hourly</syn:updatePeriod>
  <syn:updateFrequency>1</syn:updateFrequency>
  <syn:updateBase>1970-01-01T00:00+00:00</syn:updateBase>
  <items>
    <rdf:Seq>
      <rdf:li rdf:resource="http://barrapunto.com/article.pl?sid=04/08/24/101230" />
      <rdf:li rdf:resource="http://barrapunto.com/article.pl?sid=04/08/24/0942243" />
      <rdf:li rdf:resource="http://barrapunto.com/article.pl?sid=04/08/23/2316215" />
      <rdf:li rdf:resource="http://barrapunto.com/article.pl?sid=04/08/23/2126252" />
    </rdf:Seq>
  </items>
  <image rdf:resource="http://barrapunto.com/topics/topicbarrapunto.png" />
  <textinput rdf:resource="http://barrapunto.com/search.pl" />
</channel>

<image rdf:about="http://barrapunto.com/topics/topicbarrapunto.png">
  <title>Barrapunto</title>
  <url>http://barrapunto.com/topics/topicbarrapunto.png</url>
  <link>http://barrapunto.com/</link>
</image>

...

<item rdf:about="http://barrapunto.com/article.pl?sid=04/08/22/1732202">
  <title>Arranca la aKademy</title>
  <link>http://barrapunto.com/article.pl?sid=04/08/22/1732202</link>
  <dc:creator>Chewie</dc:creator>
  <dc:date>2004-08-22T17:11:08+00:00</dc:date>
  <dc:subject>kde</dc:subject>
  <slash:department>heKaDEmeia</slash:department>
  <slash:hitparade></slash:hitparade>
  <slash:section>eventos</slash:section>
  <slash:comments>24</slash:comments>
</item>
<textinput rdf:about="http://barrapunto.com/search.pl">
  <title>Search Barrapunto</title>
  <description>Search Barrapunto stories</description>
  <name>query</name>
  <link>http://barrapunto.com/search.pl</link>
</textinput>
</rdf:RDF>

```

Figura 2.1: Ejemplo de documento RSS (Barrapunto.com)

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://purl.org/rss/1.0/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:slash="http://purl.org/rss/1.0/modules/slash/"
  xmlns:taxo="http://purl.org/rss/1.0/modules/taxonomy/"
  xmlns:admin="http://Webns.net/mvcb/"
  xmlns:syn="http://purl.org/rss/1.0/modules/syndication/"
>

<channel rdf:about="http://www.newsforge.com/">
  <title>NewsForge</title>
  <link>http://www.newsforge.com/</link>
  <description>The Online Newspaper for Linux and Open Source</description>
  <dc:language>en-us</dc:language>
  <dc:rights>Copyright 2004, OSTG - Open Source Technology Group, Inc.</dc:rights>
  <dc:date>2004-08-24T10:43:06+00:00</dc:date>
  <dc:publisher>NewsForge</dc:publisher>
  <dc:creator>Webmaster@newsforge.com</dc:creator>
  <dc:subject>Information Technology</dc:subject>
  <syn:updatePeriod>hourly</syn:updatePeriod>
  <syn:updateFrequency>1</syn:updateFrequency>
  <syn:updateBase>1970-01-01T00:00+00:00</syn:updateBase>
  <items>
    <rdf:Seq>
      <rdf:li rdf:resource="http://www.newsforge.com/article.pl?sid=04/08/19/1420241" />
      <rdf:li rdf:resource="http://www.newsforge.com/article.pl?sid=04/08/23/1258257" />
      <rdf:li rdf:resource="http://www.newsforge.com/article.pl?sid=04/08/20/0826255" />
      <rdf:li rdf:resource="http://www.newsforge.com/article.pl?sid=04/08/20/1245202" />
      <rdf:li rdf:resource="http://www.newsforge.com/article.pl?sid=04/08/19/0017206" />
      <rdf:li rdf:resource="http://www.newsforge.com/article.pl?sid=04/08/18/2149208" />
    </rdf:Seq>
  </items>
</channel>

...

<item rdf:about="http://www.newsforge.com/article.pl?sid=04/08/18/2041215">
  <title>Linux to help Marines become Semper Wi-Fi</title>
  <link>http://www.newsforge.com/article.pl?sid=04/08/18/2041215</link>
  <description>One of the benefits of attending conferences like Blackhat Briefings
    and Defcon is the networking that occurs in the background. In this case, wireless
    networking...
  </description>
  <dc:creator>warthawg</dc:creator>
  <dc:subject>trends</dc:subject>
  <dc:date>2004-08-19T08:00:00+00:00</dc:date>
  <slash:section>trends</slash:section>
  <slash:comments>10</slash:comments>
  <slash:hitparade>10,10,0,0,0,0,0</slash:hitparade>
</item>
<textinput rdf:about="http://www.newsforge.com/search.pl">
  <title>Search NewsForge</title>
  <description>Search NewsForge stories</description>
  <name>query</name>
  <link>http://www.newsforge.com/search.pl</link>
</textinput>
</rdf:RDF>

```

Figura 2.2: Ejemplo de documento RSS (NewsForge.com)





Figura 2.3: Diferencia visual entre los sitios de Barrapunto.com y NewsForge.com

### 2.1.2.6. Heterogeneidad

La proliferación de fuentes de información desarrolladas de forma totalmente independiente unas respecto de las otras ha llevado a un escenario en el que mucha información potencialmente útil para las necesidades particulares de los usuarios esté dispersa y almacenada según esquemas y estructuras de almacenamiento con grandes heterogeneidades. Muchos son los tipos de documentos que podemos encontrar en el WWW y, en muchos de los casos, estos no pueden visualizarse correctamente en función del navegador usado, o de los *plugins* que éste tenga instalados. La Web actual va más allá de la visualización de texto e imágenes en el navegador. Ficheros de audio y vídeo, aplicaciones Java o en algún lenguaje de script en el servidor, accesos a bases de datos SQL o PHP, pueden aparecer mezclados en los sitios que se quieren procesar. Toda esta heterogeneidad no sólo de formatos de archivos sino también de estructuras de marcado diferentes complican los esfuerzos en el desarrollo de sistemas capaces de procesar esa información de forma autónoma.

### 2.1.2.7. Regularidad estructural

El hecho de que exista cierta *regularidad estructural* en las páginas HTML publicadas en un mismo sitio tiene un inmenso valor para la extracción de datos del Web. Sin embargo, que esta estructura no sea explícita, permite que los sitios Web puedan cambiar en cualquier momento sus estructuras de marcado HTML sin apenas contemplaciones e incluso, pueden aparecer irregularidades puntuales en la estructura de las páginas, como por ejemplo es el caso de que algún producto de un catálogo online contenga algún dato adicional que otros productos del mismo no contengan.

En cualquier caso, reconocer esta regularidad estructural en una página permite extraer sus datos relevantes. Existe actualmente una gran necesidad de aplicaciones que puedan integrar la información semi-estructurada de varias fuentes diversas preexistentes. La heterogeneidad en la forma de marcar datos en cada fuente hace necesario muchas veces un tratamiento previo particularizado, consistente en adaptar los datos provenientes de diversas fuentes a un formato común, para que puedan luego ser procesados adecuadamente sin tener en cuenta su procedencia.

### 2.1.2.8. Varios lenguajes de marcado

HTML es el lenguaje de marcado en el que se encuentran embebidos mayoritariamente los datos del Web. Apenas los Web Services y unos pocos segmentos de negocio usan XML como formato de intercambio de datos, mientras que algunos más son los sitios XHTML. El uso de XML y los lenguajes derivados crece poco

a poco y es presumible que acabe imponiéndose, pero la mayor parte de las páginas del Web Legado siguen siendo HTML. Además, si tenemos en cuenta que existen varias versiones de cada uno de estos lenguajes de marcado, es lógico pensar que la variedad de lenguajes de marcado posibles será muy grande y dificultará en cierta medida la integración de datos de diversas fuentes, que pueden aparecer escritas en lenguajes de marcado distintos, o en distintas versiones de ellos.

### 2.1.3. Tipos de programas de navegación automatizada

Varios son los criterios para clasificar los distintos tipos de programas de navegación automatizada. En función del grado de particularización de las páginas que se pretenden procesar podemos distinguir los siguientes sistemas:

**Navegación genérica no adaptada** Son programas no particularizados a ningún sitio Web concreto y que por tanto pueden ser utilizados en cualquiera de ellos. Normalmente se dedican a solicitar interactivamente al usuario una lista de enlaces de un sitio Web para realizar tareas generalmente sencillas sobre cada una de las páginas, como por ejemplo la indexación por palabras de cada una de sus páginas (robots), la comprobación de enlaces rotos, la descarga completa de sitios Web (FlashGet [27], GetRight[28], etc) o el aviso de que ciertas páginas han sido actualizadas recientemente. Estos programas carecen normalmente de un contexto semántico y son incapaces de particularizar su comportamiento al significado semántico de los datos de las páginas visitadas y pueden ser usados únicamente en tareas sencillas.

**Navegación genérica adaptada** Son programas que, siendo en principio utilizables en cualquier Web, necesitan meta-información acerca del mismo para poder navegarlo de forma adaptada a sus características particulares. El Web Semántico es un prometedor exponente de esta forma de automatización de tareas en el Web.

**Navegación particularizada** Son programas totalmente particularizados a las peculiaridades de un sitio Web concreto, por lo que en principio no son reutilizables en otros sitios Web. Estos programas se denominan comúnmente *wrappers* o código envoltorio. Mediante el seguimiento, particularizado al sitio Web, de enlaces pre-programados estáticamente según la semántica que el programador refleja implícitamente en el código de estos programas, prácticamente cualquier tarea concebida por el usuario puede ser desarrollada de forma eficiente. Este tipo de programas son muy costosos, tanto de desarrollar, como de mantener, ya que están afectados por cualquier cambio en la estructura de las páginas accedidas.

Por otro lado, en función de la tarea que se quiere realizar también podremos distinguir entre:

**Asistentes de navegación Web** Para automatizar tareas en el Web se necesitan programas capaces de eliminar la necesidad de que el usuario deba interactuar incansablemente con el ordenador durante la ejecución de la tarea. Se deben programar previamente todas esas activaciones en un algoritmo. El resultado de ese algoritmo normalmente consistirá en presentarle al usuario sólo los datos que le interesan, evitando los datos para él irrelevantes. El programa capaz de automatizar de esta forma las tareas de usuario conforme a sus intereses se denomina **asistente de navegación Web**.

**Sistemas mediadores** Un foco importante de investigación en la actualidad es la construcción de sistemas capaces de integrar de manera sencilla datos semiestructurados heterogéneos y dispersos de múltiples fuentes accesibles por medios telemáticos, de forma que se obtenga de los mismos una visión similar a la proporcionada por una única base de datos convencional. Los llamados *sistemas mediadores* mantienen los datos en sus fuentes originales y emplean un sistema intermedio, llamado mediador, que se encarga de proporcionar a los usuarios la ilusión de que existe una única fuente en la que se encuentran todos los datos combinados y unificados de manera coherente de acuerdo a un único esquema global. Cuando el mediador recibe una consulta sobre el esquema global, ésta se reformula en diversas subconsultas que se realizan directamente sobre las fuentes originales. La interacción directa con cada fuente se hace mediante wrappers adaptados a cada servidor. Estos se encargan de recibir peticiones del mediador, traducirlas como subconsultas ajustadas al formato particular de cada fuente, ejecutarlas sobre la misma y obtener los resultados que se entregan al mediador para ser ajustados al esquema global y devueltos al usuario. La idea de este enfoque consiste en ocultar todo tipo de heterogeneidad de cada una de las fuentes de datos accedidas de forma que al usuario se le proporcione una visión global de todo el sistema y pueda manejar cada uno de los recursos de la red de forma uniforme, pese a que cada sistema trabaje en realidad de forma distinta y tenga sus propias particularidades sintácticas[29][30] [31]. Este esquema está representado en la figura 2.4.

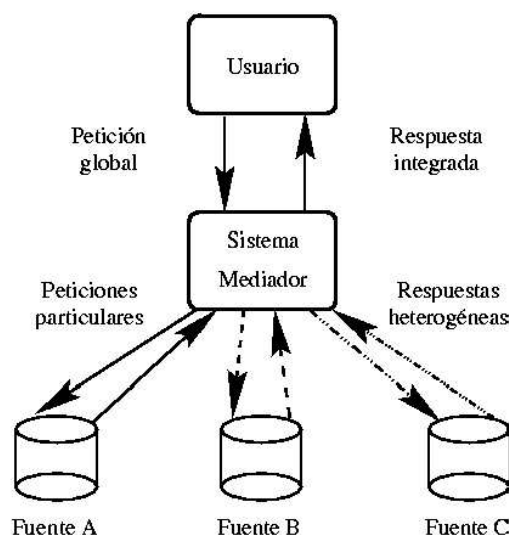


Figura 2.4: Sistema mediador

#### 2.1.4. Modelo de computación sobre el Web

El Web puede ser considerado entonces como una gran base de datos distribuida, de la cual la información es pedida y procesada. Sin embargo, mientras las bases de datos tienen lenguajes de programación especializados, como por ejemplo SQL, para acceder y procesar sus contenidos, todavía no existen, o son muy pocos, los lenguajes especializados para este objetivo en el Web. Un típico modelo de computación sobre el Web, que tiene en cuenta las características concretas aquí presentadas, distingue generalmente tres fases:

##### Fase de entrada

Incluye el acceso a una o más páginas Web para su posterior procesamiento. Durante esta fase hay que tener en cuenta la distribución geográfica de el Web y las ineficiencias de su arquitectura. Por ejemplo, una o más de las siguientes situaciones pueden ocurrir cuando se está intentando obtener una página de un servidor Web:

- La página está disponible y puede ser obtenida con éxito.
- El servidor no está disponible o proporciona servicio intermitente debido a que está sobrecargado.
- La página no está disponible (quizás temporalmente) o fue redireccionada a otro servidor.
- La conexión es terminada de manera inesperada o la velocidad de transferencia cae enormemente.

##### Fase de procesamiento

Esta fase típicamente requiere la extracción de datos de las páginas y la realización de cálculos sobre estos valores, o su utilización. En las páginas marcadas con XML o HTML se puede aprovechar la estructura del contenido de la página para facilitar estas tareas, sujetas en general a los problemas anteriormente descritos.

##### Fase de salida

La fase de salida de una computación típica requiere la generación de documentos Web utilizando valores calculados durante la fase de procesamiento, y que son almacenados en el Web, o la presentación en pantalla de la información procesada. Esta fase no siempre está presente.

#### 2.1.5. Costes de la navegación automatizada

El desarrollo de sistemas de navegación automática del Web, basada en el modelo anteriormente presentado, es una tarea costosa, no sólo en cuanto se refiere a la creación de los programas sino también, y quizá de forma más acusada, en cuanto al mantenimiento de los desarrollos, que, como hemos indicado

anteriormente, está fuertemente ligado a cambios en la estructura interna de las páginas cuya navegación se desee automatizar. Además, en función de la relevancia de la tarea, podemos hablar de un coste de ejecución fallida, sin embargo no estudiaremos éste en el proyecto.

#### 2.1.5.1. Coste de desarrollo

El desarrollo de este tipo de sistemas debe tener en cuenta las características peculiares del medio que va a automatizar (brevemente resumidas en la sección 2.2). Los distintos tipos de aplicaciones presentados pueden desarrollarse en multitud de lenguajes existentes, Java, Perl, Python, WebL, etc. Sin embargo, no todas estas alternativas son igual de flexibles y adecuadas en cada caso. Para minimizar los costes es importante que el API de desarrollo proporcione un buen acceso a datos Web (soportando varios protocolos), que sea robusto frente a fallos ocasionales de la comunicación y que permita localizar fácilmente los errores en ejecución. Es importante además que el API tenga un adecuado nivel de abstracción, y que permita de forma sencilla la manipulación, extracción y almacenamiento de datos semi-estructurados.

Un factor que condiciona sin duda el coste de desarrollo es el grado de complejidad de la tarea que se desea automatizar. Es importante que el lenguaje en el que se desarrolle sea sencillo y con un nivel de abstracción tal que permita codificar acciones sencillas o típicas, como la obtención de una página dada su URL, en una o pocas líneas de código.

Una de las principales fuentes de problemas a la hora de desarrollar estos programas es la falta de soporte adecuado para muchas acciones ocultas al usuario durante la navegación y que son ejecutadas por los navegadores. Lo ideal es que este tipo de acciones estén soportadas de forma transparente para que no trasciendan al programador, aunque en general esto no está siempre logrado y el desarrollador tiene en muchos de los casos que programar teniendo esto en cuenta.

#### 2.1.5.2. Coste de mantenimiento

Los programas de navegación genérica no adaptada apenas necesitan labor de mantenimiento y no se ven afectadas por las modificaciones en la regularidad estructural de las páginas. Esto si ocurre para los sistemas genéricos adaptados y los particularizados. En el caso del Web Semántico esas modificaciones deben quedar convenientemente reflejadas en los meta-datos del sitio Web, de forma que el mantenimiento se reduce a una labor declarativa. Por el contrario, en los sistemas de navegación particularizada, este mantenimiento es una tarea muy costosa que debe realizarse habitualmente en el propio código de la aplicación. Por muchas medidas de robustez frente a errores y cambios que introduzca el desarrollador, puede existir y existirá algún cambio no contemplado que hace necesario una labor regular de mantenimiento. Para que este mantenimiento se haga de forma sencilla es deseable que los programas sean fácilmente legibles, breves y simples, para que incluso el mantenimiento pueda ser hecho por otra persona distinta al desarrollador original.

## 2.2. XML y tecnologías derivadas

### 2.2.1. eXtensible Markup Language

El lenguaje extensible de marcas XML (eXtensible Markup Language) es un formato estándar para la estructuración de datos. La definición actual XML 1.0[8] es una recomendación del W3C[21] de Febrero de 1998 y está basado en el estándar SGML (Standard Generalized Markup Language, ISO 8879), que data de 1986.

XML es en sí mismo un meta-lenguaje para definir lenguajes. Un documento XML aparece como una jerarquía estrictamente anidada de elementos. Los elementos tienen atributos y pueden contener texto u otros elementos como hijos. Los elementos y atributos y su orden en la jerarquía forman el lenguaje usado por el documento. En XML no hay etiquetas predefinidas, como ocurre en HTML. Es el usuario quien define cuáles son sus elementos y las etiquetas asociadas. La semántica del documento XML la proporciona la aplicación o el usuario que utiliza el documento.

Como ejemplo se presenta un documento XML que describe un catálogo de collares de una joyería. El ejemplo aparece en la figura 2.5. De dicho ejemplo puede inferirse que el lenguaje contiene, al menos, dos elementos: *catálogo* e *item*. El elemento catálogo puede contener elementos de tipo *item* y los atributos *name* y *autor*; así mismo, *item* contiene texto además de la información incluida en sus atributos.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogo name="coleccion collares" autor="Enplata S.A.">
  <item ref="collar01" precio="150">Collar de plata con incrustaciones de malaquita</item>
  <item ref="collar02" precio="80">Colgante plata de ley</item>
</catalogo>
```

Figura 2.5: Ejemplo de documento XML

XML aporta numerosas y significativas soluciones a la hora de estructurar los documentos de una forma mucho más clara que la manera en la que HTML lo permite actualmente. Gracias a su mayor expresividad, y a su facilidad para ser procesado por programas no orientados a la mera visualización, el uso de XML se ha extendido rápidamente como solución al intercambio de datos estructurados en muy diversos ámbitos. Las principales cualidades de los documentos XML son las siguientes:

### Sintaxis fácilmente procesable

XML presenta una sintaxis textual muy simplificada que permite estructurar fácilmente los documentos en forma de árbol. Al contrario de su predecesor SGML, los programas que procesan XML pueden ser muy fácilmente creados debido a la simplicidad del formato, algo que sin embargo no le resta flexibilidad y escalabilidad para poder definir documentos complejos.

### Independencia de la presentación

XML permite dotar fácilmente a los documentos de una estructura sintáctica que esté adecuada a la naturaleza propia del documento, prescindiendo absolutamente de la forma en la que éste pueda ser visualizado, labor que queda delegada en las hojas de estilo.

### Estructuración de datos

Las etiquetas XML no definen propiedades acerca de cómo deben ser visualizadas, sino que simplemente describen los datos que contienen. Con el fin de dar una descripción formal a los posibles contenidos que puede tener una etiqueta dentro de un documento, XML incluye la posibilidad opcional de describir esas sintaxis mediante los DTD o XML Schema.

### Contexto semántico

Algo que no forma parte de la especificación de XML y para lo que aún no se ha definido formato estandarizado de representación es el contexto semántico, capaz de dotar de significados a cada una de las partes de esa estructuración sintáctica (etiquetas y atributos XML), es posible así que cada uno de los datos que aparecen en un documento pueda tener significado semántico para los usuarios. De esta forma, las distintas partes de los documentos pueden presentar un significado conocido y sin ambigüedades, adecuado para ser comprensible por las máquinas, algo que las páginas HTML que forman parte del Web actual no cumplen, pues están orientadas a la mera visualización.

## 2.2.2. Definición de lenguajes XML

Puesto que se ha definido XML como un meta-lenguaje, deberán existir mecanismos para la descripción de los lenguajes definidos. Esta gramática puede hacerse explícita en un esquema. Los autores de documentos XML (humanos o aplicaciones) podrían, usando este esquema, asegurar que sus documentos son válidos conforme a la gramática que éste representa. Además esta gramática podrá ampliarse, si fuera necesario, sin que ello supusiera invalidar documentos que atendiesen a una especificación anterior más restringida. En este sentido, XML es adaptable: permite definir marcas propias y crear las relaciones estructurales necesarias, especificarlas en un esquema y compartirlo. Existen dos mecanismos fundamentales de definición de lenguajes XML: DTD's y XML Schema.

### 2.2.2.1. DTD's

La propia especificación XML 1.0 incluye un tipo de esquema denominado DTD (Document Type Definition). Una DTD puede estar contenida en el propio documento, en un fichero externo o repartida entre ambos y describe de forma sencilla la gramática de un lenguaje definido sobre XML.

En un DTD habrá principalmente tres tipos de declaraciones:

- **Elementos:** Identifican los nombres de los elementos y la naturaleza de su contenido;
- **Lista de atributos:** Identifica el nombre y el tipo de los posibles atributos de un elemento dado;
- **Entidades:** Permiten asociar un nombre a un fragmento de texto;

La DTD para el ejemplo anterior aparece en la figura 2.6. El ejemplo, muy sencillo, incluye únicamente un par de elementos con sus correspondientes listas de atributos.

```
<!ELEMENT catalogo (item+)>
<!ATTLIST catalogo name CDATA #REQUIRED autor CDATA #OPTIONAL >
<!ELEMENT item (#PCDATA)>
<!ATTLIST item ref CDATA #REQUIRED precio CDATA #REQUIRED >
```

Figura 2.6: Ejemplo de DTD

Un esquema basado en una DTD tiene bastantes limitaciones. La falta de flexibilidad en una DTD imposibilita la definición de elementos con contenido mixto, es decir, que incluyan otros elementos además de texto. Además no es posible indicar a qué tipo de dato (número, fecha, moneda) ha de corresponder un atributo o el texto de un elemento. A pesar de estas limitaciones, la definición de documentos XML mediante DTDs, al ser un proceso sencillo, está muy extendida.

#### 2.2.2.2. XML Schema

EL W3C promovió la especificación de un nuevo tipo de esquema de descripción de documentos XML: XML Schema[32] basándose a su vez en propuestas existentes que intentaban superar las limitaciones comentadas de los DTDs. Un esquema XML Schema es en sí mismo un documento XML mientras que una DTD no lo es. XML Schema ya incluye la noción de espacios de nombres lo que permite definir elementos con el mismo nombre con características diferenciadas siempre que correspondan a contextos diferentes o a distintos espacios de nombres.

Respecto a la definición de tipos simples de datos de atributos o del contenido textual de los elementos, XML Schema permite utilizar una gama amplia de tipos predefinidos, definir tipos propios, imponer restricciones, por ejemplo, patrones, etc. Además, XML Schema tiene ciertas características heredadas de la orientación a objetos, por ejemplo, la derivación de tipos por extensión o restricción, clases de equivalencia (o grupos de sustitución) y elementos abstractos, etc.

La figura 2.7 representa el Schema XML correspondiente al anterior ejemplo del catálogo de joyas.

```
<xs:element name="catalogo">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="item">
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute name="ref" use="required" type="xs:string">
              <xs:attribute name="precio" use="required" type="xs:int">
                </xs:extension>
              </xs:simpleContent>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="name" use="required" type="xs:int">
            <xs:attribute name="autor" use="optional" type="xs:string">
              </xs:complexType>
            </xs:element>
```

Figura 2.7: XML Schema para el ejemplo de catálogo de joyas

A pesar de la potencia expresiva de XML Schema frente a una DTD, no siempre será descartable utilizar esta última. De hecho, no todo el software que permite analizar XML soporta XML Schema pero todos los

analizadores soportan DTD. Por ejemplo, en aplicaciones donde el control de tipo de datos no sea relevante, conviene definir la o las correspondientes DTDs y validar los documentos XML frente a éstas. Utilizar una DTD es por ahora la solución más portable y la más eficiente puesto que los validadores para XML Schema son más pesados.

### 2.2.3. XPath

XPath es una recomendación del W3C para el direccionamiento de datos en documentos XML. El actual estándar es la versión 1.0[33], recomendada desde noviembre de 1999. No obstante, a lo largo de los últimos años los grupos de trabajo del W3C de XML Query y de XSL han estado trabajando en una nueva versión del estándar, la versión 2.0[34], que actualmente está todavía en fase de borrador.

XPath permite una especificación clara y funcional de consultas de datos en documentos y colecciones de elementos XML. La expresividad de XPath permite que las expresiones de consulta sean lo suficientemente potentes como para resolver la mayoría de las búsquedas usando apenas una expresión simple y fácilmente comprensible que ocupa normalmente una única línea de texto. Por otro lado, XPath sirve como base de muchos otros lenguajes que recorren la estructura de documentos XML, tales como XSLT, XPointer, XQuery o XUpdate.

Los bloques básicos de construcción de XPath son expresiones. El lenguaje contempla varios tipos distintos que pueden ser contruidos con palabras clave, símbolos y operandos.

#### 2.2.3.1. Secuencias

El valor de una expresión es siempre una secuencia de valores. Una secuencia es una colección ordenada de varios valores, con un número de elementos ilimitado, pero finito, incluyendo también el caso de la secuencia vacía, es decir, la secuencia que no contiene ningún valor. Los elementos posibles de una secuencia son valores de tipos denominados como atómicos, esto es, valores simples no formados por la composición de otros valores. Ejemplos de valores atómicos son los valores numéricos, valores lógicos (o booleanos), las cadenas de caracteres o los nodos (referencias a elementos del documento XML, incluyendo el elemento raíz o página).

#### 2.2.3.2. Variables

Las variables son repositorios en memoria que albergan resultados de expresiones XPath y que constituyen la forma de comunicar XPath con su entorno. Gracias a las variables, expresiones XPath pueden utilizar los resultados de otras expresiones XPath. El acceso al valor de una variable dentro de XPath se representa con el símbolo \$ seguido del nombre de la variable. Las variables de XPath son accesibles por el lenguaje anfitrión de XPath de forma que éste también es capaz de manipular los resultados de expresiones XPath.

#### 2.2.3.3. Operadores aritmético-lógicos y de comparación

XPath permite especificar operaciones aritméticas (sumas, restas, multiplicaciones, divisiones) así como operaciones lógicas (and, or, not) y comparaciones de diversos tipos (igualdad, menor que, menor o igual que, mayor que), incluyendo también operadores para comparar el orden de aparición de nodos dentro del documento. Todo ello permite la creación de expresiones sencillas capaces de calcular operaciones habituales de análisis de datos en páginas XML que pueden ser combinadas fácilmente.

#### 2.2.3.4. Ejes de navegación

Los ejes de navegación son indicadores acerca de los caminos que deben seguirse para saltar de un nodo a otro dentro del árbol del documento. Normalmente esta navegación necesita hacerse en varios saltos, razón por la cual las expresiones XPath están divididas en distintos pasos, delimitados por barras inclinadas /.

Dentro de cada paso, los saltos entre nodos están condicionados por los denominados ejes de navegación que indican la relación vecinal existente entre el nodo de origen y el nodo destino de cada salto. Estas distintas formas de proximidad vecinal son contempladas por el estándar de forma que sean así explorables todos los nodos, tanto internos a un nodo cualquiera (child y descendant), como externos al mismo (parent, ancestor, preceding-sibling y following-sibling).

El acceso a los atributos XML de un nodo se puede realizar atravesando el eje attribute. XPath proporciona a su vez una sintaxis abreviada para cada uno de esos ejes de navegación, lo cual permite la construcción de expresiones más cortas y legibles. La tabla 2.2 muestra un listado de los ejes de XPath.

Eje	Nodos considerados
ancestor	Cualquier nodo en el camino a la raíz
ancestor-or-self	Igual que ancestor pero incluyendo el nodo actual
attribute	Los nodos atributo del árbol
child	Los nodos directamente contenidos por el actual
descendant	Los nodos del subárbol cuya raíz es el nodo actual
descendant-or-self	Lo mismo pero incluyéndolo
following	Nodos posteriores al actual, excluyendo descendientes
following-sibling	Nodos hermanos posteriores al actual
parent	Ascendiente directo de un nodo
preceding	Nodos anteriores al actual excluyendo ancestros
preceding-sibling	Nodos hermanos anteriores al actual
self	El nodo actual

Cuadro 2.2: Ejes de navegación XPath

### 2.2.3.5. Predicados

Si bien suele ser necesario a menudo atravesar varios nodos antes de poder llegar a los elementos que interesa finalmente acceder, lo cierto es que muchas veces el recorrer ciertos de esos nodos que aparecen direccionables por un eje de navegación resultan no deseable. El uso de los ejes de navegación, aunque esté bien combinado con una adecuada elección de los nombres de los elementos, resulta muchas veces insuficiente para evitar la navegación por nodos no deseados. Por esa razón, XPath proporciona mecanismos adecuados para permitir la selección, conforme a criterios especificables por el usuario, de los elementos de una secuencia obtenida en cada paso de una expresión XPath, de manera que sólo sean seleccionados del documento XML aquellos componentes que cumplan un conjunto de requisitos establecidos, siendo descartados aquellos elementos de la secuencia que no cumplan esa propiedad.

Las funciones encargadas de filtrar o seleccionar aquellos elementos de una secuencia que cumplen las propiedades especificadas en una condición se denominan predicados, y son una parte muy importante de XPath, pues en ellos reside gran parte de su expresividad. Los predicados se representan en XPath rodeados de corchetes [] que, situados a la parte derecha de cada paso, rodean la condición que deben cumplir cada uno de los elementos seleccionados en ese paso para poder formar parte del resultado final.

### 2.2.3.6. Funciones

Con el fin de poder especificar tratamientos especializados, XPath permite la llamada a funciones parametrizables con argumentos. Dichas funciones aparecen detalladas en una biblioteca definida para el lenguaje, e incluye múltiples tipos de tratamientos para cada uno de los distintos tipos de datos manejables en XPath. No obstante, el conjunto de funciones resulta un tanto limitado ya que XPath no permite la definición de funciones de usuario y la biblioteca utilizable tiene un conjunto cerrado de primitivas.

Se presentan a continuación varios ejemplos de expresiones XPath:

- `./@href` Selecciona todos los atributos href descendientes del nodo contexto.
- `child::section[position()<6] / descendant::cite / attribute::href` Selecciona todos los atributos href en los elementos de tipo cite de las cinco primeras secciones de un documento XML.

## 2.2.4. XSLT

XSLT[11] (eXtensible Stylesheet Language for Transformations) es un estándar de la organización W3C que define una forma de transformar documentos XML en otros e incluso a formatos que no son XML. Las hojas de estilo XSLT realizan la transformación del documento utilizando una o varias reglas definidas mediante plantillas. Junto con el documento XML fuente, esas plantillas alimentan a un procesador de XSLT, el cual realiza las transformaciones deseadas colocando el resultado en un archivo de salida o, como en el caso de una



pagina Web, directamente en un dispositivo de presentación, como el monitor de un usuario o una impresora.

XSLT sigue una sintaxis XML, donde se utiliza el concepto de plantilla para aplicar unas reglas de transformación a elementos de documentos XML direccionados con XPath. La figura 2.8 incluye un sencillo ejemplo de plantilla XSLT para generar una página Web a partir de un catálogo XML de discos.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <body>
      <h2>My CD Collection</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
          <th align="left">Title</th>
          <th align="left">Artist</th>
        </tr>
        <xsl:for-each select="catalog/cd">
          <tr>
            <td><xsl:value-of select="title"/></td>
            <td><xsl:value-of select="artist"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>
```

Figura 2.8: Ejemplo de plantilla XSLT

Actualmente, XSLT es muy usado en la edición Web, generando paginas HTML o XHTML de forma dinámica. La unión de XML y XSLT permite separar contenido y presentación, aumentando así la productividad en la creación de sitios Web complejos. Varios son también los procesadores XSLT existentes, e incluso las últimas versiones de los navegadores más populares soportan ya el uso de plantillas XSLT.

## 2.3. Combinadores de servicio

Con el fin de reproducir el comportamiento humano al navegar a través de el Web, es necesario tener en cuenta sus reacciones ante tasas de transmisión bajas y caídas de ciertos enlaces. Cardelli y Davies definieron en 1996 un formalismo para tener en cuenta los conceptos de fallo y tasa de transmisión, los *Combinadores de Servicio*[7], que pueden ser incorporados a los distintos modelos de programación para el Web, basados en el protocolo HTTP.

El servicio básico con el que trabajaremos será una petición o envío de una pagina Web desde o hacia una URL. Este servicio puede realizarse satisfactoriamente o fallar. La idea es combinar distintos servicios que puedan estar sujetos a fallos para obtener servicios virtuales más fiables. Los combinadores definidos por Cardelli y Davies fueron:

- **Ejecución secuencial:** dados dos servicios  $S_1$  y  $S_2$ , primero ejecutamos  $S_1$  y si falla entonces  $S_2$ . En caso de que los dos servicios fallen, la combinación también falla.
- **Ejecución concurrente:** se ejecutan  $S_1$  y  $S_2$  concurrentemente y se devuelve el resultado del que primero termina. Si los dos fallan la combinación también.
- **Límite temporal:** el combinador funciona como el servicio S excepto que falla si después de t segundos S no ha finalizado.
- **Repetición:** se repite el servicio S hasta que finaliza correctamente.
- **No-terminación (stall):** este combinador no termina nunca. Es útil cuando se combina con otros.

- **Tasa de transmisión:** funciona como el servicio S salvo que si la tasa de transmisión cae por debajo de un determinado límite el servicio falla.
- **Fallo:** este servicio falla siempre.

El empleo de estos combinadores de servicio permite que los programas de navegación automatizada reproduzcan el comportamiento humano de una navegación Web típica, al tiempo que aumentan el grado de fiabilidad en la consecución de las distintas tareas sobre un medio tan poco fiable como es Internet, frecuentemente sujeto a caídas de los distintos servidores, disminuciones drásticas de las tasas de transmisión, etc.

## 2.4. Los Diagramas de Secuencias de Mensajes

Los MSC constituyen representaciones abstractas, capaces de ser usadas en muy diversos ámbitos y siendo además fáciles de entender por mucha gente, incluso no expertos. Aunque en su nacimiento, los MSC estuvieron inicialmente pensados para representar el intercambio de señales entre componentes electrónicos de sistemas de telecomunicaciones, su incorporación a los formalismos establecidos por la ITU (International Telecommunication Union) de la mano de otro formalismo bien conocido como SDL (Specification and Description Language)[35][36] supuso una gran aceptación por muy diversos colectivos y muy variados usos, llegando a realizar importantes aportaciones al mundo de la ingeniería del software, como lo demuestra su decisiva influencia sobre la representación gráfica de los diagramas de secuencia de UML[37][36], que pueden ser considerados como una versión orientada a objetos de los MSC con pequeñas diferencias que están intentando ser solventadas por las revisiones de ambas especificaciones.

Además de la importancia que conceden a la representación gráfica, los MSC incorporan la valiosa aportación de poder ser representados textualmente, de forma que resultan igualmente modificables y analizables por herramientas no gráficas. Dicha representación textual sirve de base para la definición de la semántica formal que incorporan los MSC y que les permite estar sujetos a la demostración automatizada de algunas propiedades de los escenarios reproducidos como, por ejemplo, la imposibilidad de que determinado mensaje A sea enviado con antelación a la recepción de otro evento en el sistema, como la recepción de otro mensaje B en algún otro componente o la ejecución de alguna acción.

Los MSC se han convertido en una potente técnica de especificación del comportamiento de un sistema desde el punto de vista de las interacciones de sus componentes. Al ser fácilmente comprensibles por no expertos en programación, suelen formar parte de los documentos de especificación de requisitos que se intercambian ingenieros y analistas con sus clientes, constituyendo una representación visual muy descriptiva de la interacción bajo distintos escenarios entre diversos componentes, especialmente entre aquellos que forman parte de un sistema distribuido. Los MSC pueden ser usados desde los primeros pasos del diseño del software, gracias a lo cual, los errores detectados mediante su uso adelantado son resueltos con un menor coste que en etapas posteriores.

En los últimos tiempos se ha motivado mucho el desarrollo de algoritmos para una variedad de análisis de MSC consistentes en la detección de condiciones de carreras, conflictos de tiempos, toma de decisiones no locales, o incluso capacidad de generación de autómatas finitos concurrentes que simulen el sistema modelado por el MSC, por lo que son varias las herramientas utilizables para analizar los problemas derivables en etapas de diseño gracias al examen de los MSC. Desde el año en el que nacieron (1992) hasta la actualidad, los MSC ha venido incorporando nuevas funcionalidades en sucesivas ampliaciones revisadas por la ITU, la entidad que lo ha estandarizado. Los MSC se han convertido en una tecnología particularmente útil en el modelado de las interacciones propias de protocolos en las comunicaciones, así como también para la representación de llamadas a procedimientos o métodos, incluyendo RPC. Su interés ha suscitado también en los últimos tiempos, el desarrollo de herramientas capaces de transformar representaciones de MSC en otros tipos de formalismos de especificación, como SDL o Promela[38], herramientas de validación de especificaciones MSC[39][40][41] o, incluso, los MSC han sido empleados como método de especificación en el desarrollo de aplicaciones alojables en servidores Web[14].

A continuación se mencionan los componentes más importantes que define la norma. Estos son:

- Diagrama MSC básico;
- Documento MSC;
- Diagrama MSC de alto nivel;

### 2.4.1. Diagramas MSC básicos

Los diagramas MSC básicos, o MSC, describen un comportamiento parcial de un sistema mediante la representación gráfica o textual del intercambio de mensajes entre los distintos componentes presentes en el sistema y su entorno. Para cada uno de los elementos del sistema involucrados en un MSC hay un eje de entidad. Las distintas entidades se comunican entre sí mediante el paso de mensajes. A continuación se mencionan los elementos más importantes que se pueden encontrar en un diagrama MSC básico.

#### 2.4.1.1. Entidades

Están representadas por líneas verticales, y modelizan cada uno de los componentes (hardware o software) que forman parte del sistema. Las líneas verticales que representan a los componentes sirven a su vez como eje temporal, de forma que aquellos eventos que se representan en la parte superior del dibujo de un componente le suceden a éste con anterioridad a los que están representados en las partes inferiores. Ello implica que ningún mensaje debe ser enviado o recibido y ninguna acción debe ser ejecutada por un componente del sistema hasta que, desde el punto de vista de su representación gráfica, no se hayan procesado todos los eventos anteriores que consten previamente en su representación. Una representación de tres entidades puede encontrarse en la figura 2.9.

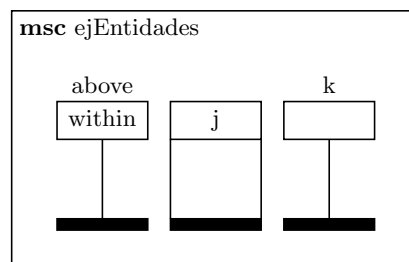


Figura 2.9: Ejemplos de entidades

#### 2.4.1.2. Mensajes

Están representados por las flechas, horizontales o diagonales, que nacen de una entidad y mueren en otra. El esquema permite a su vez la representación de mensajes intercambiados con el entorno externo del sistema representado, así como posibles mensajes enviados perdidos por el camino e incapaces de llegar a sus destinos. Un mensaje representado por una flecha horizontal entre dos componentes indica que su tiempo de latencia es poco considerable, mientras que otro mensaje que aparezca con mayor ángulo representa un mayor retardo desde el momento del envío por el componente emisor hasta el momento de la llegada al componente receptor. Una representación del intercambio de mensajes entre varios componentes puede encontrarse en la figura 2.10.

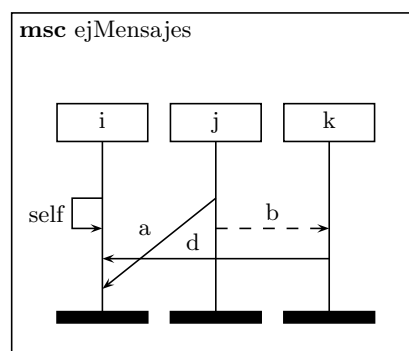


Figura 2.10: Ejemplos de mensajes

#### 2.4.1.3. Acciones

Otros tipos de elementos representables en un MSC lo constituyen las acciones que cada componente puede realizar sin interacción con las demás entidades, consistentes en cálculos internos. Estos procesamientos

de datos pueden ser realizados para dar respuesta a una petición codificada en un mensaje, para preparar el lanzamiento de nuevos mensajes que serán lanzados a otras entidades, o para analizar el resultado de respuestas recibidas. Las acciones se representan en los MSC con rectángulos. Las acciones pueden involucrar la ejecución de sentencias como llamadas a funciones o procedimientos ejecutados por la propia entidad y sólo se consideran terminadas cuando los cómputos terminan o esas llamadas a funciones o procedimientos retornan. Una representación de una acción ejecutada entre la recepción de un mensaje a y el envío de una respuesta b puede contemplarse en la figura 2.11.

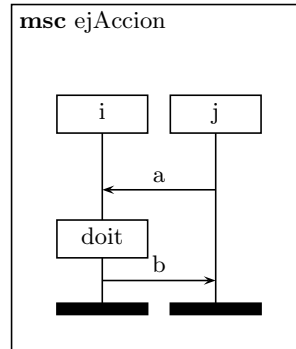


Figura 2.11: Ejemplo Acción

#### 2.4.1.4. Temporizadores

Si bien el eje vertical de cada uno de los componentes de un sistema representa la secuenciación relativa de los eventos que suceden en ese componente, lo cierto es que en dicho eje no existe medida alguna que represente temporizaciones ni medidas acerca del momento exacto en el que se espera que ocurran los eventos. En un MSC se refleja el orden posible de los eventos que ocurren globalmente en el sistema, pero no necesariamente se dan medidas acerca de los tiempos que deben transcurrir entre dichos eventos. En la figura 2.11 aparece claramente representado un sistema de dos entidades i y j de forma que presentan el requisito de que, en primer lugar, la entidad j envía un mensaje a a la entidad i, la cual realiza una acción y a continuación envía un mensaje de respuesta b a la entidad j. Sin embargo, en ningún lugar del modelo aparecen restricciones acerca de cuál debe ser el tiempo transcurrido entre dichos eventos, ni es tampoco predecible ni acotable el tiempo existente entre el envío del mensaje a y la recepción del mensaje b por la entidad j.

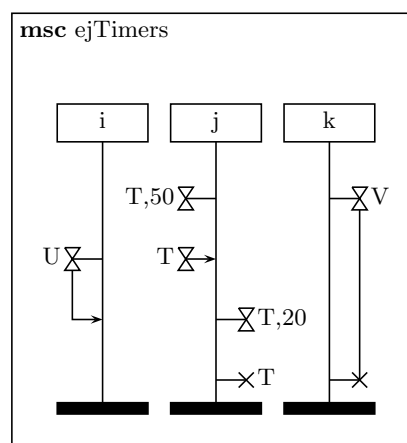


Figura 2.12: Ejemplo de temporizadores

Para poder contemplar en el modelo la duración máxima de estos intervalos entre eventos, los MSC permiten definir temporizadores, de forma que al vencimiento de un temporizador en una entidad se pueda provocar el comportamiento de una acción especial en dicha entidad que haga un tratamiento de la situación. Los MSC ofrecen para ello operadores para crear y desactivar temporizadores, así como una marca para señalar el evento de activación del temporizador. Distintos tipos de operadores de temporizadores, cada uno

de ellos nombrado con un nombre distinto, y para los cuales hay distintos comportamientos representados (establecimiento, desactivación, vencimiento, ...) pueden contemplarse en la figura 2.12.

#### 2.4.1.5. Condiciones

Una condición es el cumplimiento de una expresión de verdad, y sirve para indicar que el sistema ha entrado en un cierto estado. Una condición, que desde el punto de vista de programación se puede asociar con una expresión booleana, puede afectar a más de una entidad en un sistema. El símbolo de condición en un MSC, que es un hexágono, aparece superpuesto a todas las entidades a las que afecta. Las demás entidades no afectadas por la condición se dibujan sin intersecar con el símbolo de la condición, o bien aparecen superpuestas a él. En la figura 2.13 aparece representado un MSC donde la primera condición afecta a las entidades i y k, la segunda condición afecta sólo a la entidad i y la tercera condición afecta a todas las entidades del sistema.

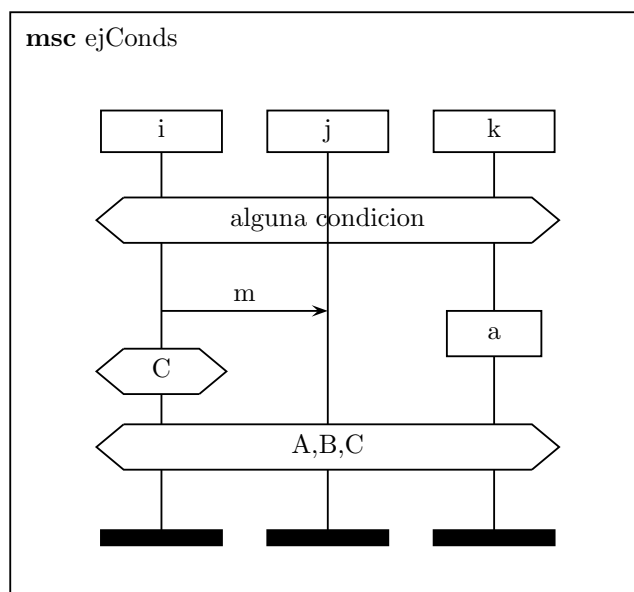


Figura 2.13: Ejemplos de condiciones

#### 2.4.1.6. Expresiones inline

Una expresión inline es una parte especial de un MSC sobre la cual está definido un operador que conjuga la forma en la que deben aparecer los eventos que forman parte de la expresión. Por ejemplo, una expresión inline de bucle indica que los eventos que forman parte de ella pueden ser ejecutados en secuencia varias veces. Existen varios ejemplos posibles de operadores, que aparecen en la tabla 2.3. Por ejemplo, existen operadores para indicar que dos o más partes de un MSC deben ejecutarse en paralelo (concurrentemente), o para indicar que, de varias partes de un MSC, se debe escoger sólo alguna, quizá incluso dependiendo de una condición que determine cuál es la parte que responde al modelado de los siguientes eventos. Existen operadores también para indicar que la secuencia de eventos producidos en un fragmento de un MSC se deben repetir varias veces hasta que se cumpla una condición de salida del bucle.

Expresión inline	Significado
alt	Bloques alternativos
loop	Bucle
opt	Bloque opcional
par	Acciones ejecutadas concurrentemente
exc	Excepciones

Cuadro 2.3: Expresiones inline definidas

Las expresiones inline están representadas por rectángulos, donde los operandos están separados por líneas horizontales discontinuas, y donde el nombre del operador aparece en la esquina superior izquierda

del rectángulo. Las estructuras inline pueden anidarse unas dentro de otras. Una representación de varias expresiones inline de un MSC, una de ellas anidada dentro de otra, puede contemplarse en la figura 2.14.

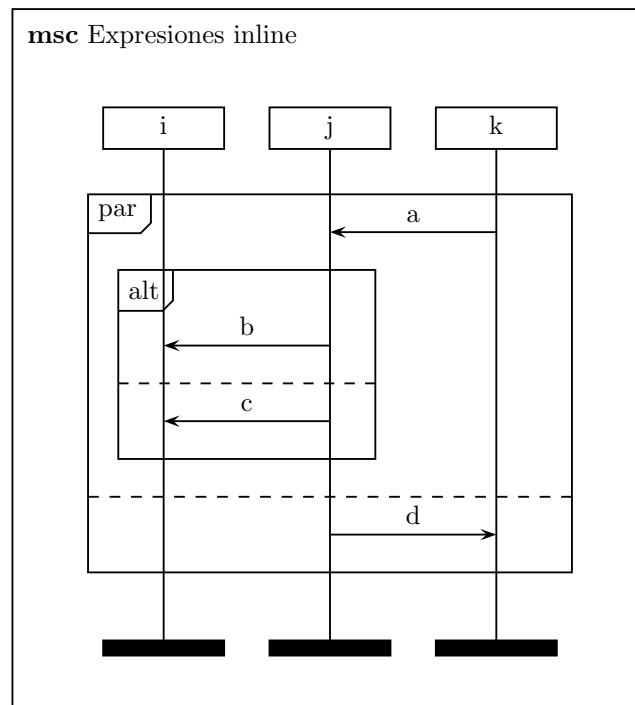


Figura 2.14: Ejemplo de expresiones inline

#### 2.4.1.7. Descomposición modular: Referencias

Es habitual que en los sistemas más complejos, el nivel de complejidad de un MSC presente dificultades de legibilidad en la representación gráfica. El número de eventos y de expresiones inline puede fácilmente llegar a ser demasiado grande, o la estructura de anidamientos puede presentar demasiados niveles, por lo que suele ser necesario simplificar la representación con construcciones más sencillas, encapsulando las partes de más bajo nivel para que puedan ser convenientemente referenciadas, de la misma forma en la que se usan procedimientos y funciones en los lenguajes de programación. Desde el punto de vista de los MSC, la solución habitual consiste en distribuir la información del modelo en varios MSC más simples de forma que unos puedan ser referenciados por otros. Tales referencias se representan con unos rectángulos con las esquinas redondeadas, cubriendo dichos rectángulos aquellas entidades que intervienen en el MSC referenciado. Una representación de un MSC que referencia a otros dos MSC externos puede contemplarse en la figura 2.15.

#### 2.4.1.8. Correcciones

En ocasiones, el orden de los eventos de un MSC no está completamente determinado al tener un conjunto de eventos que pueden ocurrir en cualquier orden, de forma no determinista. En aras de que las representaciones sean lo suficientemente flexibles y permitan representar en un mismo gráfico todas las posibles variantes de esas ordenaciones, típicamente atribuibles a aspectos no controlables como retrasos impredecibles en las comunicaciones, en la recepción o el envío de varios mensajes, los MSC permiten la definición de zonas específicas en las entidades, llamadas correcciones representadas por fragmentos dibujados con trazos discontinuos en las entidades, de forma que los mensajes que parten o llegan a una corrección pueden ser enviados o recibidos en cualquier orden posible respecto del resto de los eventos definidos dentro de la misma corrección. Así pues, dentro de una corrección con  $n$  eventos, se estarán representando los  $n!$  casos posibles de ordenación de una sola vez. Un ejemplo de corrección definida para el componente  $j$  puede contemplarse en la figura 2.16.

#### 2.4.1.9. Creación y destrucción dinámica de entidades

Las entidades pueden estar creadas antes del momento del que parte el modelado del MSC y seguir existiendo después del momento en el que termina el modelado. En ese caso, se las considera entidades

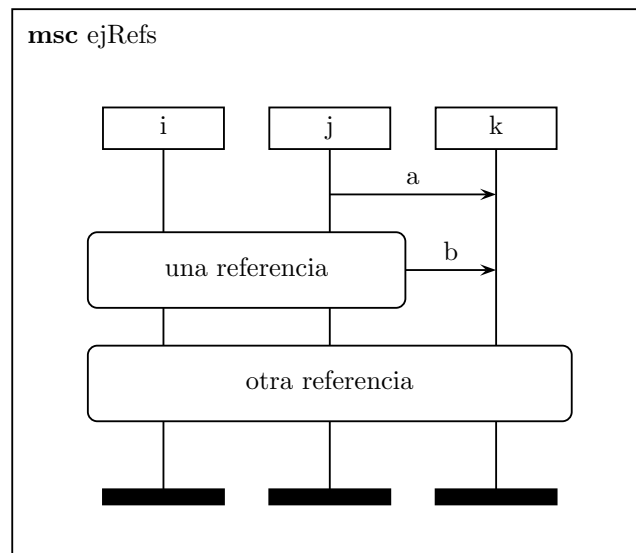


Figura 2.15: Ejemplo de referencia

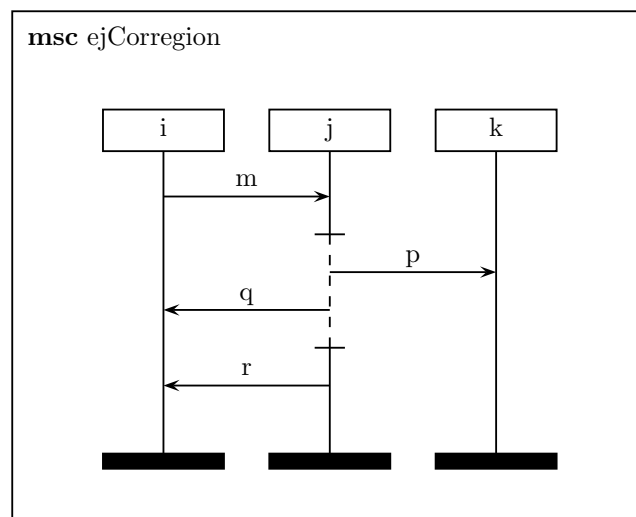


Figura 2.16: Ejemplo de corrección

de creación estática, y su creación y destrucción no forma parte del modelado del sistema. No obstante, algunas entidades también pueden ser creadas y destruidas dinámicamente en tiempo de ejecución, dentro del modelado del sistema, como por ejemplo, como respuesta a algún tipo de evento. Tanto la creación como la destrucción de este tipo de entidades puede ser representada en un MSC fácilmente. La creación de mensajes se representa con una flecha discontinua desde la entidad creadora (la que invoca la creación) a la entidad creada (la que aparece como resultado de la invocación anterior). La destrucción de mensajes se representa al final de cada entidad con unas aspas en forma de cruz, momento a partir del cual, la entidad deja de existir. Una representación de la creación de dos entidades y la muerte de dos de ellas puede contemplarse en la figura 2.17.

### 2.4.2. Documentos MSC

Un documento MSC define un tipo de instancia y una colección asociada de diagramas MSC básicos que definen escenarios de interacción entre las instancias del tipo definido en el documento. Estos documentos pueden definirse tanto en forma textual como gráfica, pero cuentan siempre con una cabecera y un cuerpo del documento. En la cabecera se define el nombre del documento. El cuerpo se descompone en una parte definidora, o de definiciones, y una parte utilitaria, o de utilidades. La figura 2.18 representa un documento MSC en el que podemos distinguir las distintas partes.

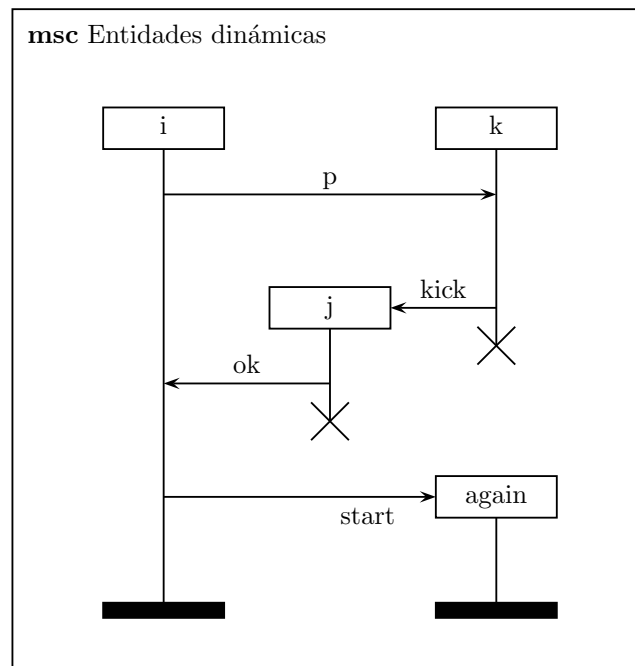


Figura 2.17: Ejemplo de creación dinámica de entidades

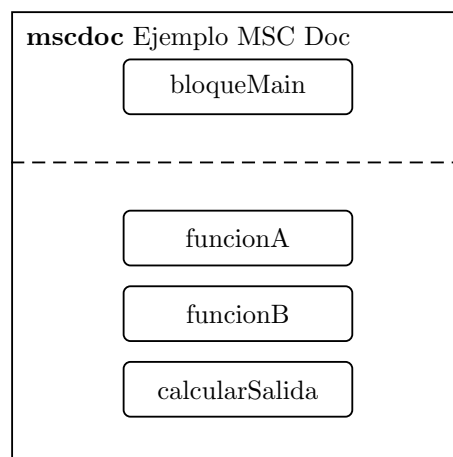


Figura 2.18: Ejemplo de documento MSC

La parte de definiciones describe la colección de diagramas MSC que componen el sistema, mientras que la parte de utilidades contiene únicamente patrones utilizados por los elementos de la parte de definiciones (a modo de funciones que pueden ser llamadas en cualquier punto de la parte de definiciones). Por tanto, los documentos MSC se emplean para describir la arquitectura de sistemas que no se pueden describir en un único diagrama MSC básico.

### 2.4.3. Documentos MSC de alto nivel (hMSC)

Un diagrama MSC de alto nivel, hMSC, describe cómo varios diagramas básicos se combinan para dar lugar a casos más complicados. En un hMSC, los diagramas básicos se representan mediante referencias y pueden ser compuestos de forma secuencial, concurrente o como alternativas. Además, este tipo de gráficos no representa instancias ni mensajes. Los elementos que pueden aparecer en un diagrama hMSC son:

- Símbolos de principio y fin (triángulos);
- Condiciones restrictivas;
- Referencias a MSC's;



- Nodos de conexión (círculos pequeños);
- Líneas de conexión entre elementos;

La figura 2.19 incluye un ejemplo de diagrama hMSC.

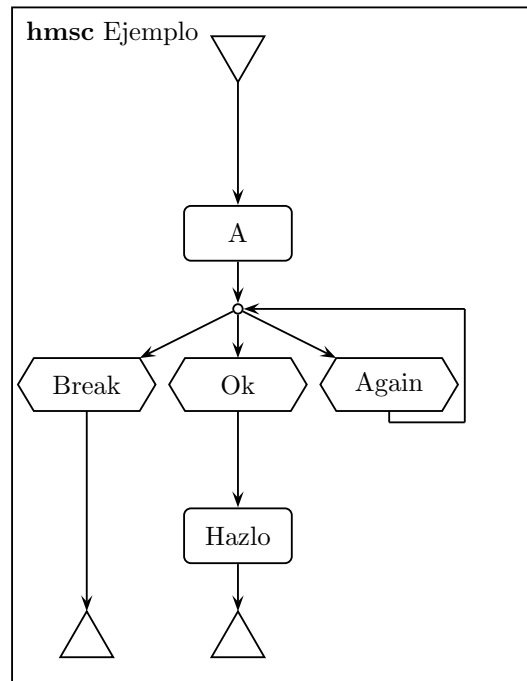


Figura 2.19: Ejemplo de diagrama MSC de alto nivel

#### 2.4.4. MSC para especificación de requisitos y diseño de clientes Web

Los MSC aportan una serie de ventajas para el modelado de clientes Web. Las más importantes son:

- Su intuitiva notación gráfica que ayuda a los diseñadores a visualizar la estructura del sistema y sus interfaces.
- Si bien MSC es un lenguaje gráfico, existe una forma textual de la norma principalmente para el intercambio entre aplicaciones y como base para el análisis formal automático.
- Permiten elevados niveles de abstracción.
- Se hace distinción entre el *sistema* y su *entorno*.
- Permiten una descripción operacional del sistema. No se tratan cuestiones relacionadas con cada implementación particular.
- Soporta diseños estructurados: escenarios simples (descritos en MSC básico) pueden combinarse para formar especificaciones más complejas (hMSC's). El diseño modular está soportado en mecanismos de descomposición y reutilización.
- Soporta especificaciones completas o incompletas (en fases tempranas de desarrollo o para documentación).
- MSC se usa a menudo en conjunción con otros lenguajes y métodos. Su definición formal permite la validación de MSC con respecto a un modelo descrito en un lenguaje diferente.
- Sirven para describir combinadores de servicio de forma sencilla, resultando apropiados para modelos de computación Web.
- Son fácilmente ampliables para soportar nuevas características.

La utilización de MSC para modelar el comportamiento de aplicaciones en el Web no es novedosa de este trabajo. Existen precedentes del uso de MSC para el modelado y especificación de aplicaciones orientadas al Web pero en general se trata de aplicaciones centradas en el modelado de aplicaciones Web desde el punto de vista de los servidores Web, salvo el caso de XPlore.

#### 2.4.4.1. XPlore

XPlore[14] es un lenguaje de programación de alto nivel de abstracción para la creación de programas de navegación automática (asistentes de navegación, programas envoltorio, etc) basada en los MSC y XPath. XPlore es concebido como una adaptación restringida con reglas de la representación textual de los MSC y hereda, por tanto, sus principales características, utilizando el subconjunto de elementos más representativos de este formalismo. XPlore es un lenguaje imperativo que permite construcciones típicas de los lenguajes de esa naturaleza, como secuenciación de sentencias, bucles, sentencias condicionales, uso de variables y funciones.

XPlore respeta la sintaxis inicial de MSC, pero añade una serie de restricciones semánticas con el objetivo de evitar que determinadas construcciones, siendo sintácticamente correctas desde el punto de vista de los MSC, sean semánticamente inadmisibles desde el punto de vista de un algoritmo de navegación Web. De esta manera, se reducen aspectos como la ambigüedad o la posibilidad de especificar sistemas incoherentes o no traducibles a lenguajes de programación, sin que la pérdida de flexibilidad provocada por esas restricciones afecte a la esencia principal del lenguaje de expresión de algoritmos ni tampoco a la expresividad del mismo.

Al contrario de lo que ocurre con la mayoría de los lenguajes diseñados en los trabajos relacionados, XPlore no dispone de una plataforma que le dé directamente soporte de ejecución, es decir, no es directamente interpretable por ninguna aplicación.

## 2.5. El lenguaje WebL

Analizando los lenguajes actualmente existentes para el desarrollo de aplicaciones capaces de navegar automáticamente en el Web pocos de ellos resultan adecuados para la automatización de aplicaciones Web. Muchas de las bibliotecas desarrolladas para ser manejadas por lenguajes de programación habituales (C, Java, Perl, etc) permiten especificar fácilmente sencillas pautas de navegación, básicamente visitando páginas de direcciones conocidas de antemano. Muy pocas de ellas permiten la especificación de caminos de navegación complejos, donde se deben seguir varios enlaces y en los que se debe hacer referencia con algún patrón de acceso a enlaces extraídos de los documentos recientemente visitados. No proporcionan al usuario los mecanismos de extracción de datos necesarios para parametrizar convenientemente las distintas primitivas HTTP en caminos de navegación compleja o establecer sesiones con los servidores Web accedidos.

Por otro lado, las soluciones existentes no tienen un nivel de abstracción lo suficientemente elevado capaz de sintetizar en pocas líneas de código operaciones comunes de la navegación o el procesamiento, lo cual da lugar a grandes programas difíciles de mantener. Se requiere poder representar fácilmente el diálogo formado por las peticiones y respuestas que se deben desarrollar con las aplicaciones cuyo uso se desea automatizar, pero a la vez integrando ese diálogo con el manejo de otros recursos, como el manejo del sistema de ficheros e interacción con el usuario, el uso de temporizadores y gestión de hilos concurrentes para manejar varias comunicaciones en paralelo, así como operaciones de control de flujo habituales en lenguajes de programación imperativos (bucles, condiciones, variables, llamadas a funciones, etc).

Web Language, o como se lo denominó en sus orígenes, WebL[2][3][4] es un lenguaje de script para procesar documentos sobre la World Wide Web desarrollado en los laboratorios del Compaq Systems Research Center en 1998. Está bien preparado para recuperar documentos de el Web, extraer su información, y de esa manera manejar sus contenidos. En contraste con otros lenguajes de propósito general, WebL está específicamente diseñado para automatizar tareas sobre el Web. No sólo el lenguaje está construido sobre el conocimiento de los protocolos Web, como HTTP y FTP, sino que también es capaz de procesar documentos en los formatos de texto, HTML y XML.

El manejo flexible del marcado encontrado en documentos HTML y XML es una importante característica del lenguaje. Además, WebL soporta características que simplifican el manejo de fallos de comunicación, la exploración de documentos replicados sobre múltiples servicios Web para aumentar la fiabilidad, y la posibilidad de realizar múltiples tareas en paralelo, además de contar con las características de los lenguajes de programación imperativos tradicionales como objetos, módulos, estructuras de control, etc.

Además de las estructuras y tipos de datos típicos de los lenguajes de programación, WebL tiene dos características especiales para ayudar a los programadores de aplicaciones Web: los combinadores de servicios, que ayudan a manejar los servicios poco fiables; y el álgebra de marcado para manejar estructuras de texto como las de HTML y XML.

### 2.5.1. Características generales

A continuación se recoge un breve resumen de las principales características de WebL:

- El sistema y lenguaje de WebL está diseñado para un rápido prototipado de la computación sobre el Web. Es muy adecuado para la automatización de tareas sobre la WWW.
- El énfasis de WebL está más centrado en una alta flexibilidad y un alto nivel de abstracción, que en una rápida computación. Es, por tanto, más adecuado como una herramienta de rápido prototipado que como una herramienta de alto volumen de producción.
- WebL es implementado como una aplicación stand-alone que toma y procesa páginas Web de acuerdo con los scripts programados.
- WebL es un lenguaje de alto nivel, imperativo, interpretado, dinámicamente tipado, con soporte para hilos de ejecución, y basado en expresiones.
- Tiene una sintaxis fácil de leer, con expresiones parecidas a las de C, y estructuras de control similares a las de Modula.
- WebL soporta todos los protocolos soportados por Java, como HTTP, FTP y FILE. WebL no provee soporte para protocolos adicionales.
- WebL está escrito en Java. (WebL es completamente portable en plataformas UNIX; para Windows habrá que hacer alguna comprobación en algún API y archivos .dll).
- Es muy fácil hacer puentes de código WebL a Java. Los objetos de Java pueden ser llamados directamente desde código WebL sin necesidad de extender el sistema WebL.

### 2.5.2. Núcleo del lenguaje

Las características especiales de WebL como los combinadores de servicios y el álgebra de marcas, están integradas en un pequeño lenguaje de programación. Al margen de estas características, el núcleo del lenguaje es conceptualmente similar al de la mayoría de los otros lenguajes de programación. Por ello, se hace simplemente referencia a las características de éste núcleo, sin entrar en detalles.

#### 2.5.2.1. Expresiones, constantes y operadores

Los programas WebL consisten en una secuencia de expresiones separadas por punto y coma. Ejecutar un programa WebL involucra evaluar las expresiones en secuencia. Cada expresión o evalúa a un valor (o resultado) o causa una excepción que hace que la evaluación del programa termine en ese punto, en cuyo caso se dice que la expresión lanza una excepción.

Las constantes son expresiones simples que se evalúan a sí mismas. Son las expresiones más simples de WebL. Se permiten constantes de tipo nil, boolean, integer, real, character y string.

#### 2.5.2.2. Tipos de datos

WebL es un lenguaje de programación dinámicamente tipado. Los tipos de valores definidos de WebL son: nil, boolean, int, real, char, string, fun, meth, set, list, object, page, piece, pieceset, tag.

#### 2.5.2.3. Variables y Contextos

Las variables deben ser declaradas antes de usarse y una variable sólo puede ser declarada una vez en un contexto dado y sólo podrá ser utilizada en un contexto específico en todas las posiciones sintácticamente siguientes a donde haya sido declarada.

#### 2.5.2.4. Sentencias

WebL usa constructores típicamente de lenguajes imperativos de programación. Algunas de las sentencias posibles en WebL son las sentencias if, while, repeat, try, every, lock, begin y return.

### 2.5.2.5. Constructores

WebL soporta listas de valores, conjuntos de valores y objetos con campos. Los constructores realizan la creación de esos tipos de valores a partir de valores más simples. Los tipos list se construyen con `[ ]`, los set con `set` y los objetos con `[. .]`.

También existen otros constructores, como por ejemplo el constructor de listas en paralelo `[| |]`.

### 2.5.2.6. Funciones construidas en WebL

Existen muchas funciones ya implementadas en WebL y que facilitarán al programador su tarea. El programador también puede definir sus propias funciones.

### 2.5.2.7. Módulos

Para facilitar la reutilización de código, WebL permite empaquetar rutinas en un módulo. Un módulo no es más que una secuencia de expresiones almacenadas en un fichero con extensión `.web1`. Además de los módulos ya definidos en WebL, un programador puede crear sus propios módulos. Para utilizar un módulo dentro de un programa, el programador debe importarlo, y hacer referencia a las variables exportadas por este módulo. La referencia a esas variables se hace con el siguiente esquema:

`NombreMódulo_NombreVariable`

WebL incluye un conjunto de módulos estándar ya implementados que recogen funcionalidades utilizadas a menudo. Algunos de estos módulos estándar proporcionados por WebL son:

- **Files:** Contiene funciones para procesar ficheros locales y descargar páginas a ficheros.
- **Java:** Soporta la integración WebL-Java
- **Servlet:** Soporte para servlets de Java.
- **Str:** Facilidades para el tratamiento de strings.
- **Url:** Funciones para la manipulación de URLs.
- **Otros:** WebServer, Base64, Browser, Cookies, Farm y WebCrawler,...

### 2.5.2.8. Comentarios

En WebL los comentarios se pueden añadir de dos formas. Con `//`, se introduce un comentario hasta el final de la línea, con `/* Comentario */` todo lo contenido entre `/*` y `*/` es un comentario.

## 2.5.3. Combinadores de Servicios

Los combinadores de servicios sirven para escribir programas capaces de reproducir comportamientos humanos ante velocidades de transmisión bajas, fallos, muchos enlaces simultáneos, etc. Los humanos saben cuándo parar y volver a intentar de nuevo un acceso. Estos, basados en los de Cardelli y Davies[7] están directamente integrados en el núcleo del lenguaje.

Ejemplo de estos comportamientos son:

- Repetir peticiones, por ejemplo tomando pequeñas pausas entre ambas.
- Terminar una petición que va muy lenta.
- Cambiar a un servidor que esté menos cargado y posea la misma información.
- Monitorizar la tasa de transferencia de información, y decidir si esperar o no.
- Ejecutar varios accesos en paralelo, esperando al primero que acabe, y detener el resto.

Como se puede imaginar muchas cosas pueden fallar cuando se trabaja sobre una red distribuida como Internet. Por tanto, la naturaleza impredecible de el Web hace que trabajar sobre ésta cause muchas más excepciones que si se trabaja en un entorno no distribuido. Los combinadores de servicios permiten combinar varios servicios de distintas formas, de tal manera que la computación sea más fiable, y en algunos casos se consigue mejorar la velocidad (cuando se explota el paralelismo inherente de servicios replicados).

Otra forma de definir qué es un combinador de servicios, es decir, que es una estructura de control que determina el orden de ejecución de sus servicios basado en el éxito o fallo. WebL incorpora directamente los combinadores de servicios en operadores del lenguaje. Soporta varias combinaciones: ejecución secuencial, ejecución concurrente, timeout, repetición y no terminación.

Antes de estudiar los distintos combinadores de servicios proporcionados por WebL, es necesario ver qué se entiende por servicio.

### 2.5.3.1. Servicio

Un servicio es una acción (como por ejemplo traer una página Web) o un conjunto de acciones que involucran estructuras de control o componentes de servicios. Es decir, por servicio se entiende cualquier operación en WebL y por tanto puede terminar con éxito o fallo, y en ese momento, produce un valor (o no). La mayoría de los servicios básicos implica el acceso a una página Web. Por ejemplo, son servicios las funciones ya definidas en WebL `GetURL` y `PostURL` que implementan los servicios para traer páginas Web desde un URL indicado.

`GetURL` utiliza el protocolo HTTP GET y devuelve un objeto WebL que encapsula la estructura analizada de la página. Si el proceso de captura de la página falla, entonces la función falla. `GetURL` tiene un segundo argumento opcional que proporciona los argumentos de la petición de acceso y un tercer argumento para cabeceras HTTP. `PostURL` es similar, pero usa el protocolo HTTP POST, que se utiliza en los formularios de entrada de las páginas Web.

Para el resto de esta sección, `S` y `T` denotan servicios, que pueden contener primitivas para traer páginas u operaciones generales de WebL.

### 2.5.3.2. Ejecución secuencial `S ? T`

El combinador de ejecución secuencial `?` permite a un servicio secundario ser consultado en caso de que el primer servicio falle por cualquier motivo. El servicio `S ? T` actúa como `S` excepto en el caso de que `S` falle, entonces se ejecutaría `T`. Si `T` también falla, entonces falla `S ? T`.

### 2.5.3.3. Ejecución concurrente `S | T`

El combinador `|` permite que dos servicios sean ejecutados concurrentemente. En el servicio `S | T` los servicios `S` y `T` empiezan a ejecutarse a la vez, y se devuelve el resultado del que acabe con éxito primero. Una vez que uno de los servicios acaba exitosamente, el otro servicio se para. Si ambos fallan, entonces el combinador de servicios también falla.

### 2.5.3.4. Time-out `Timeout(t,S)`

El combinador `Timeout` permite fijar un tiempo límite para la ejecución de un servicio. El servicio `Timeout(t,S)` actúa como `S` excepto si no acaba con éxito en `t` milisegundos, en cuyo caso falla. Si no ha acabado en `t` milisegundos `S` es parado.

### 2.5.3.5. Repetición `Retry(S)`

El combinador de repetición proporciona un camino para invocar repetidamente un servicio hasta que éste resulte exitoso. `Retry(S)` actúa como `S` excepto si falla, en cuyo caso `S` vuelve a empezar de nuevo. El bucle podría acabarse con `Timeout(t,Retry(S))`.

### 2.5.3.6. Non-termination `Stall()`

El combinador `Stall` nunca se completa, o falla.

En resumen, los combinadores de servicios son utilizados para manejar fallos de los servicios, y explotar la replicación de servidores. Hacen más fácil imitar los comportamientos y reflexiones de los humanos, tales

como recarga de enlaces congestionados, reintentos de búsquedas tras una pausa, conexiones con servidores menos utilizados, y ejecución de múltiples peticiones en paralelo.

#### 2.5.4. Álgebra de marcado

El álgebra de marcado de WebL se usa para manipular páginas Web y extraer datos de ellas. La extracción de información puede consistir en operaciones simples como contar los enlaces de una página u operaciones más complejas que completan formularios Web y procesan los resultados retornados desde el servidor. Los operadores del álgebra de marcado son diseñados para hacer más fácil la obtención de información selectiva de una página, ignorando el desorden subyacente. Todas las funciones y operadores trabajan bajo el concepto de alto nivel sobre una página Web analizada, y la necesidad de manejar strings en un nivel más bajo es pequeña.

##### 2.5.4.1. Páginas, etiquetas, partes y conjuntos de partes

Después de que una página se obtiene de el Web y se analiza de acuerdo a su tipo MIME, la página y su contenido están accesibles para operaciones posteriores en WebL. El procesado que se puede realizar sobre una página está determinado por el álgebra de marcado de WebL.

**2.5.4.1.1. Páginas** La representación interna que WebL usa para un recurso Web es un objeto de página. Las funciones `GetURL` y `PostURL` traen una página de el Web, y retornan un objeto de página.

El álgebra de marcado se basa en tres conceptos: etiquetas, partes y conjuntos de partes. En términos simples, una etiqueta corresponde a una etiqueta de marcado, una parte identifica una subregión contigua de una página, y un conjunto de partes es una colección de partes.

**2.5.4.1.2. Etiquetas** El primer paso en analizar una página Web es identificar todas las etiquetas de marcado en la página (encerradas entre los caracteres `<` y `>`). Cada una de las etiquetas se convierte en un valor WebL de tipo `tag`. Conceptualmente la página consiste de una lista de etiquetas y segmentos de texto (o datos de tipo carácter).

El modelo WebL también soporta etiquetas sin nombre. Lo equivalente en HTML o en XML es `<>` (que en la práctica no ocurre). WebL usa estas etiquetas para ubicar marcadores en una página. Como su nombre lo indica, las etiquetas sin nombre no tienen ni nombre ni atributos.

**2.5.4.1.3. Partes** Una parte es un tipo de valor WebL que denota una región de la página. Cada parte se referencia con dos etiquetas: la etiqueta de inicio que denota el comienzo de la región, y la etiqueta de fin que denota el final de la región. Notar que la etiqueta de inicio y fin pueden ser las mismas. Otro hecho importante es que las partes nunca apuntan a segmentos de texto.

Los tipos más comunes de partes son aquellos que corresponden a los elementos de una página. Para permitir que el programador acceda a los atributos de los elementos, los atributos de una etiqueta de inicio de un elemento se copian dentro de variables de campo de cada parte. De manera que, una parte es muy similar a un tipo de valor objeto.

**2.5.4.1.4. Conjunto de Partes** Como su nombre lo indica, un conjunto de partes es una colección de partes que pertenecen a la misma página. Un conjunto de partes también es una lista debido a que las partes de un conjunto de partes están ordenadas. El ordenamiento de las partes en un conjunto se basa en las posiciones de las etiquetas de inicio y fin de una parte en una página.

Los conjuntos de partes juegan un rol muy importante en WebL. Ellos forman la base de muchas de las operaciones de extracción de datos de páginas Web.

##### 2.5.4.2. Funciones de búsqueda

Los conjuntos de partes se pueden crear de varias formas: buscando los elementos de marcado de un nombre determinado (por ejemplo todos los elementos `.a.o "li"`); buscando patrones de caracteres que concuerden con una expresión regular, buscando segmentos de texto, o secuencias de elementos de marcado, etc.

Algunas de las funciones de búsqueda más importantes son:

**2.5.4.2.1. Búsqueda de Elementos** La función `Elem` retorna un conjunto de partes de todos los elementos que concuerdan con un nombre específico. La función también restringe el alcance de búsqueda a una página o parte. Entonces, una parte se construye para cada par de etiquetas de inicio y fin de un elemento de marcado con el nombre indicado en el alcance indicado. Las partes resultantes se colocan dentro de un conjunto de partes resultado.

Por ejemplo, el siguiente programa trae una página, calcula el conjunto de partes con todas las imágenes, e imprime el atributo "src" de cada una de las imágenes:

```
var P = GetURL("http://www.url.com");
var images = Elem(P, "img");
every image in images do
    PrintLn(image.src)
end;
```

Como se puede ver en el ejemplo, la sentencia `every` también permite la iteración sobre los elementos de un conjunto de partes.

**2.5.4.2.2. Búsqueda de Patrones** La función `Pat` busca en una página por patrones de caracteres que concuerdan con una expresión regular o por grupos de patrones agrupados en una expresión Perl5. La función `Pat` ignora las etiquetas en una página, solamente se busca en el texto puro. Se crea una nueva parte con cada ocurrencia del patrón. Se insertan etiquetas sin nombre antes y después de la ocurrencia del patrón para mantener pistas de la ubicación.

A un conjunto de partes creado con las funciones `Elem` o `Pat`, se le pueden aplicar operadores y funciones de WebL para ejecutar nuevos cálculos. Por ejemplo, indexando un conjunto de partes con el operador de indexación `[]`, se puede extraer el elemento `n`. A partir de un conjunto de partes se puede muy fácilmente hacer un filtrado, utilizando para ello la función `Select`.

#### 2.5.4.3. Funciones Misceláneas

El álgebra de marcas incluye varias funciones misceláneas para realizar conversiones entre diferentes tipos de valores, por ejemplo de string a page, y viceversa. O para acceder a las etiquetas de principio y fin de algunos elementos. El siguiente ejemplo muestra la conversión de un string en page.

```
var P = NewPage("<html><body>
  <h1>Test Page</h1>
  <table>
    <tr>
      <td> A </td>
    </tr>
  </table>
</body></html>", "text/html");
```

El segundo argumento de `NewPage` define el parser a ser usado para codificar el string dentro de una página.

#### 2.5.4.4. Operadores de conjuntos de partes y funciones

Se define un operador de conjuntos de partes (piece set operator) como una operación algebraica entre dos conjuntos de partes `P` y `Q`, que devuelve como resultado otro conjunto de partes. Para los siguientes puntos, `P` y `Q` representan conjuntos de partes. Un elemento de `P` se representa como `p` y uno de `Q` como `q`.

**2.5.4.4.1. Operadores básicos** Los operadores básicos son utilizados para una manipulación básica de conjuntos. Son los operadores de unión (`P+Q`), exclusión (`P-Q`) e intersección (`P*Q`).

**2.5.4.4.2. Operadores posicionales** Estos operadores expresan las relaciones entre las partes de un conjunto, de acuerdo a su posición en la página. La mayoría de estos operadores tienen su versión negativa, que se indica con el símbolo `!`. Son los siguientes:

- indexing `P[i]`

- P before/!before Q
- P after/!after Q
- P directlybefore/!directlybefore Q
- P directlyafter/!directlyafter Q
- P overlap/!overlap Q

**2.5.4.4.3. Operadores jerárquicos** Estos operadores expresan las relaciones entre las partes de un conjunto, de acuerdo a su anidamiento jerárquico en el árbol. Son:

- P inside/!inside Q
- P contain/!contain Q
- P directlyinside/!directlyinside Q
- P directlycontain/!directlycontain Q

**2.5.4.4.4. Operadores regionales** Estos operadores construyen nuevas partes para identificar partes de una página, aunque suelen ser menos utilizados que los anteriores. Son:

- P without Q
- P intersect Q

**2.5.4.4.5. Funciones misceláneas** Son: Children(p), Parent(p), Flatten(p) y Content(p).

#### 2.5.4.5. Modificación de páginas

Esta es una parte importante del álgebra de marcado. Como ya se ha comentado, los atributos de los elementos de marcado pueden ser fácilmente modificados accediendo a los campos de la parte. A continuación se muestran las funciones utilizadas para insertar partes en una página, borrarlas o sustituirlas.

**2.5.4.5.1. Crear partes** Hay muchas maneras de crear nuevas partes. Las funciones creadas para este propósito son `NewPiece`, `NewNamedPiece`.

**2.5.4.5.2. Insertar partes** Una vez creada una piece, ésta puede ser insertada en una posición específica de una página. Para ello se usan `InsertBefore` e `InsertAfter`.

**2.5.4.5.3. Borrar partes** Para borrar una parte de una página se usa `Delete`.

**2.5.4.5.4. Sustituir partes** La función `Replace` sustituye unas partes (las borra) por otras (las inserta).

### 2.5.5. Elección de WebL

Uno de los objetivos de este proyecto era el desarrollo de una herramienta que transforme representaciones MSC del comportamiento de un asistente de navegación automática en código ejecutable en algunos de los lenguajes existentes. En principio no había restricciones en cuanto al lenguaje objetivo final, si bien era necesario utilizar un lenguaje que facilitará la programación de aplicaciones capaces de interactuar con el Web: navegar por sus enlaces, rellenar y enviar formularios, extraer y procesar la información procedente de páginas Web, y todo ello de manera automática, sin la ayuda del usuario.

El lenguaje WebL cuenta con una serie de ventajas frente a otros existentes ya que fue específicamente diseñado para recuperar documentos de el Web, extraer su información, y de esa manera manejar sus contenidos. No sólo el lenguaje está construido sobre el conocimiento de los protocolos Web como HTTP y FTP, sino que también sabe procesar documentos en los formatos de texto, HTML y XML. Además presenta algunas de las características indicadas en el análisis de la automatización de tareas en el Web ya que cuenta con un adecuado nivel de abstracción y el código escrito en WebL es muy sencillo y corto en comparación al generado en otros lenguajes.



Otra ventaja importante es la posibilidad de importar módulos externos. En el caso concreto de este proyecto se ha trabajado con un módulo WebL de tratamiento especializado de formularios y corrección de errores y transformación de las páginas a marcado XHTML (usando Tidy[42]). Este módulo define una serie de primitivas que alivian al programador de hacer un manejo a bajo nivel de estos procesos.

En cualquier caso, hay que decir que WebL también presenta algunos inconvenientes frente a otras opciones. En primer lugar, WebL es un lenguaje relativamente lento en ejecución. Esto se debe, sin duda, a que es un lenguaje interpretado, y que el intérprete está escrito en Java, sobre el que también se cuestiona su velocidad. Para procesados intensivos existen otras alternativas que agilizan los tiempos de ejecución. Por otro lado WebL no es libre, el sistema de licencias de WebL, si bien permite su uso para fines académicos y personales, no aporta toda la flexibilidad y ventajas de las licencias GPL[43].

Además, el uso de WebL no se ha extendido y es minoritario. Convendría el uso de un lenguaje más extendido entre el público general, como Python (que si es libre), que está cobrando gran interés a día de hoy y que ya se plantea como un serio competidor de Java, al aparecer cada día nuevas API's y módulos, no sólo para el tratamiento de tareas Web, tanto del lado del cliente como del servidor, sino para cualquier propósito que pudiera plantearse

### 2.5.6. Ejemplos WebL

El objetivo de este apartado es mostrar al lector algunas líneas de código de WebL. A continuación se muestran un par de ejemplos sencillos de código WebL. Para un mayor detalle puede consultar los ejemplos que aparecen en el capítulo octavo, que, nótese, han sido generados por la aplicación MSC2WEBL.

#### 2.5.6.1. Ejemplo 1: Consulta de enlaces rotos

```
var url = ARGS[1];
var P0 = GetURL(url);
var links=Elem(P0,"a");
var invalidos={};
every link in links do
  try
    var p = HeadURL(link.href);
  catch E
  on E.type == "NoSuchField" do
    PrintLn("Enlace vac'io con el texto "+Text(link))
  on E.statuscode == 400 do
    PrintLn(E);
    PrintLn("El tipo MIME del fichero al que apunta el enlace no esta soportado.");
    PrintLn("El enlace no soportado es: " +link.href+"\n")
  on E.statuscode == 404 do
    invalidos=invalidos+{link}
  on true do
    PrintLn("Error indeterminado en el enlace "+link.href+"\n");
  end
end;
if(Size(invalidos)>0) then
  PrintLn("La lista de enlaces incorrectos en el sitio " + url);
  every link in invalidos do
    PrintLn(link.href);
  end;
end;
```

Figura 2.20: Código WebL del script de comprobación de enlaces

El objetivo es construir una aplicación que automatice la verificación del estado de todos los enlaces de un sitio Web indicado por su URL. Esta es una tarea típica de los creadores de sitios, con lo que sería de gran utilidad el poder automatizarla. Vemos como esto se hace de forma sencilla empleando WebL (figura 2.20).

### 2.5.6.2. Ejemplo 2: Descarga de una página Web para lectura off-line

El siguiente ejemplo descarga a disco la página indicada como argumento, junto con todas las imágenes presentes en la página Web dada. El código WebL aparece en la figura 2.21.

```
import Str, Files, Url;

var Filename = fun(path)
var p = Str_LastIndexOf("/", path);
if p != -1 then Select(path, p+1, Size(path))
else ""
end
end;

var Download = fun(url, dir)
  var downloaded = {};
  var page = GetURL(url);
  every img in Elem(page, "img") do
    var u = Url_Split(img.src);
    var fname = Filename(u.path);
    if !(fname member downloaded) then
      PrintLn(img.src, "->", dir + fname);
      Files_GetURL(img.src, dir + fname);
      downloaded = downloaded + {fname};
    end;
    img.src = fname;
  end;

  var output = Filename(Url_Split(url).path);
  if output == "" then output = "root.html" end;

  Files_SaveToFile(dir + output, Markup(page));
end;

Download(ARGS[1], "/home/jones/tmp/");
```

Figura 2.21: Código WebL del script de descarga de página Web

### 2.5.6.3. Ejemplo 3: Uso de la biblioteca Forms.webl

La aplicación MSC2WEBL, implementada para este proyecto, hace uso de una librería WebL para la corrección y transformación de las páginas recibidas a código XHTML, el tratamiento de formularios, y la sustitución de los servicios básicos `GetURL` y `PostURL` por otros más complejos, `Forms_Get` y `Forms_Post`.

Para ilustrar el uso de esta biblioteca se ha desarrollado un pequeño script de búsqueda de códigos postales<sup>[44]</sup> conociendo el nombre de la calle y la localidad a buscar. El código de la aplicación aparece en la figura 2.22. Obsérvese la definición de una serie de variables predefinidas que permiten identificar a los clientes en el servidor como cualquiera de los navegadores típicos (*headers*), o eliminar por defecto cualquier etiqueta que se desee de las páginas recibidas (*eliminables*). El uso de esta biblioteca es muy simple. Las páginas se descargan usando `Forms_Get` pudiendo definir cabeceras no predefinidas si fuera necesario y posteriormente se convierten a XHTML con la función `Forms_xhtmliza`. Los formularios se normalizan antes de introducir los datos adecuados en ellos usando `Forms_Post`.

## 2.6. Conclusiones sobre el estado del arte

Numerosos han sido los trabajos que se han elaborado sobre el tema y todos coinciden en señalar que este tipo de soluciones tiene serios costes de desarrollo, problemas de aplicabilidad en diversas fuentes de datos (dadas sus enormes heterogeneidades), una elevada fragilidad ante errores y cambios en el servidor, y, principalmente, elevados costes de mantenimiento que sin duda han influido en su escasa utilización. Como conclusión podemos apuntar algunos de los retos de la programación Web:

```
//Modulos Importados
import Forms,XPath,Url,Str;

//Variables de "entorno"
var home="/home/jones/";
var temporal= home + "pfc/webl/tmp/";
var headers=[. "User-Agent"=Forms_useragent .];
var eliminables = "";

var consulta = fun(localidad,calle)
  var P0 = Forms_Get("http://www.meguias.com/cp.php",nil,headers);
  P0 = Forms_xhtmliza(P0,"P0",eliminables,temporal);
  var form0 = Forms_Normaliza(Elem(P0,"form")[2],true);
  var tabla0 = Elem(form0,"table")[0];
  Select(Elem(tabla0,"input"), fun(a) a.name=="empresa" ? false end)[0].value = localidad;
  Select(Elem(tabla0,"input"), fun(a) a.name=="calle" ? false end)[0].value = calle;
  var P1 = Forms_Post(form0,P0.URL,[. mimetype="text/plain" .], true);
  P1=Forms_xhtmliza(P1,"P1",eliminables,temporal);
  var tabla = Elem(Elem(P1,"table")[0],"table")[8];
  var patron = Elem(Elem(tabla,"tr")[0],"center")[1];
  var resultados = Elem(tabla,"tr");
  resultados=resultados-resultados[0]-resultados[1]-resultados[2]-resultados[3]-resultados[4];
  PrintLn(Text(patron));
  every r in resultados do
    PrintLn(Text(r));
  end;
end;

consulta("Madrid","RODRIGUEZ SAMPEDRO");
```

Figura 2.22: Ejemplo del uso de Forms.webl

1. Proponer nuevas metodologías de desarrollo, de forma que los costes se vean reducidos frente al empleo de las opciones actuales. Estos métodos deberán estar basados en estándares y tener un adecuado nivel de abstracción para favorecer la comprensión por un mayor número de personas.
2. Proporcionar mecanismos adecuados para favorecer la robustez de los programas ante fallos en las conexiones TCP/IP y adaptabilidad ante cambios en las páginas.
3. Es necesario proporcionar mecanismos para la minimización del coste de mantenimiento debido al carácter dinámico de el Web, de forma que el mantenimiento quede reducido a una mínima labor.
4. Permitir que esa automatización se pueda aplicar prácticamente a cualquier fuente o aplicación del Web legado. Se debe poder aplicar no sólo a páginas correctamente construidas teniendo en cuenta todos los criterios de accesibilidad. Se deben incluir métodos de corrección de errores en las páginas recuperadas y solventar los problemas derivados de la existencia de applets, rutinas JavaScript o animaciones Flash (entre otras).
5. Hacer extensible estas técnicas al procesado de documentos en cualquier lenguaje definido sobre XML, siempre que al menos éste sea conocido por el desarrollador, como por ejemplo RDF, RSS, WML, etc.

Existen muchos estándares para realizar diseños y especificaciones con buen nivel de abstracción, pero prácticamente ninguno de ellos ha sido usado en el ámbito de la automatización de tareas en el Web, al menos desde el punto de vista de la creación de clientes, que es el de interés para este proyecto. Uno de los estándares más adecuados para estas representaciones han sido los MSC en conjunción con el lenguaje XPath, de forma parecida a como se emplean en XPlore. Esta metodología es en la que se centra el presente estudio, al proponer un método de especificación que se demostrará efectivo al crear los mecanismos de transformación de las representaciones a código WebL ejecutable.

## Capítulo 3

# Herramienta CASE de desarrollo Web basado en los MSC y el lenguaje WebL

Uno de los objetivos primordiales del proyecto era programar una herramienta CASE que permitiera, de forma sencilla, crear y mantener programas de navegación automatizada. Ya ha quedado manifiesto cuáles son las razones de combinar para este propósito el lenguaje de programación WebL y los diagramas MSC.

Para lograr la creación y visualización de los diagramas se emplea el lenguaje de presentación de textos  $\text{\LaTeX}$  para el que se ha desarrollado una macro[45] para representar este tipo de diagramas. La aplicación realiza una transformación entre la descripción en formato XML de los distintos diagramas y el código  $\text{\LaTeX}$  de su representación gráfica de forma automatizada. Por otro lado deberá existir otra transformación entre la descripción XML intermedia y el código WebL que se quiere generar.

En este capítulo se explica cómo ha sido desarrollada la herramienta, describiendo el subconjunto de la norma MSC empleado para representar los sistemas de navegación Web, así como el lenguaje XML definido para describir sistemas conformes con este subconjunto. Siguiendo este schema para representar las tareas Web que se quieren automatizar, es posible la aplicación de distintas plantillas XSLT para obtener tanto la representación gráfica de los diagramas como el código WebL final. Se explican también los distintos bloques que componen el interfaz gráfico de usuario.

### 3.1. MSC para representación de tareas en la World Wide Web

Los diagramas MSC presentados en el capítulo anterior pueden utilizarse como base para la definición de metodologías para el automatismo de tareas en el Web en función de las necesidades concretas de cada usuario o aplicación.

La mayoría de aproximaciones [16][17][14], trabajan en general con la sintaxis textual de la norma, apoyados eso sí, con diagramas con el fin de documentar los desarrollos. Nuestra idea es que, aplicando la forma gráfica de la norma a las fases de especificación, diseño y desarrollo de sistemas de navegación automática del Web, estas tareas se facilitan, disminuyendo tiempo y costes, e incluso brindando la posibilidad a no expertos en la materia de desarrollar sus propios asistentes de navegación Web.

Existen técnicas y herramientas de validación[46][40][41] para evitar discrepancias entre una especificación MSC y su interpretación (o potencial implementación) en aquellos casos en que sea necesario proveer información explícita sobre sincronización entre procesos, la arquitectura de red subyacente y las estrategias de cola, etc. Ejemplos de este tipo de situaciones son los fallos en la red, o la pérdida de mensajes.

Nuestra propuesta, sin embargo, es la utilización únicamente de diagramas básicos, sin tener en cuenta este tipo de problemas. Las únicas combinaciones no deterministas permitidas serán aquellas incluidas en los combinadores de servicio, evitando así los problemas anteriores de falta de especificación.

Esta restricción impuesta permite que los MSC puedan aplicarse en la mayor parte de desarrollos. Para sistemas demasiado complejos que escapen al alcance de esta restricción, se puede optar por emplearla en

las fases iniciales para obtener fácilmente prototipos antes de implementar las funcionalidades que aporten dichas complejidades, pudiéndose emplear para ello diagramas de alto nivel (lo que no se ha estudiado en este proyecto).

Está claro que el intercambio de mensajes propio de las tareas Web es fácilmente representable mediante MSC. No es tan evidente la forma de representar el procesamiento que se hace internamente de la información recibida. La norma da libertad a la hora de elegir el lenguaje de especificación de estas tareas. La opción elegida para ello es el uso de XPath[33], en su versión 1.0, ya que la siguiente versión está siendo aún objeto de revisión. XPath permite representar de forma sencilla las distintas partes de un documento XML, de tal forma que es fácil seleccionar distintas partes dentro de un documento y operar con ellas. Dado que la mayor parte de las páginas existentes en el Web actual están aún en formato HTML, debe existir una capa intermedia de adaptación, de tal forma que las páginas HTML tras ser revisadas en busca de fallos de sintaxis, que se corrigen automáticamente, se transforman a formato XHTML, con lo que el direccionamiento XPath es posible.

La versión actual de XPath no incorpora algunos elementos de direccionamiento útiles para las tareas de procesado Web. No es posible, por ejemplo, seleccionar partes de un documento entre dos etiquetas concretas, ni extraer secuencias de etiquetas de la página, posibilidades que sí están incluidas en el álgebra de marcado de WebL. La siguiente versión de XPath, la 2.0[34], sí incluye ésta y otras posibilidades más complejas. No obstante, y puesto que está aún en fase de desarrollo, se ha optado por emplear la versión vigente, ya que la mayor parte de las tareas se puede representar, aunque sea de forma más compleja, mediante más de una expresión XPath.

Al contrario que en XPlore[14], se pretende aplicar la forma gráfica y no textual de la norma. Esto quiere decir que aunque el enfoque parte de la misma idea, combinar MSC y XPath, la forma de desarrollarlo es ligeramente diferente.

### 3.1.1. Elementos de la norma necesarios

Para reducir al máximo los elementos necesarios y conseguir lograr un enfoque metodológico, nos hemos apoyado en dos conceptos básicos de la recomendación: documentos MSC (para definir la estructura general del sistema) y MSC básico (para definir cada uno de los módulos del sistema). Además hemos definido nuevas expresiones inline para representar los combinadores *secuencial* y *concurrente*, así como una manera estándar de representación mediante MSC básico de los combinadores *timer* y *stall*.

#### 3.1.1.1. Documentos MSC

Cada proyecto nuevo se definirá dentro de un documento MSC donde se representarán de forma clara y sencilla todos los módulos que componen la aplicación que se quiere generar. Los datos que define nuestro documento MSC son:

- Nombre del documento (proyecto) y opcionalmente nombre del fichero de salida asociado (por ejemplo código java).
- Una parte de *definiciones* que especifica los distintos diagramas MSC que conforman la aplicación. Cada uno de estos se referencian por su nombre (y opcionalmente los parámetros de entrada y salida necesarios). Los distintos MSC's referenciados en la parte de definiciones se ejecutan secuencialmente para realizar la tarea que se quiere automatizar.
- Opcionalmente también puede existir una parte de *utilidades* con referencias a los módulos que pueden ser invocados por los elementos de la parte de definiciones. Este esquema permite la definición de funciones externas que pueden ser llamadas en cualquier punto del programa principal las veces que sea necesario.

La figura 3.1 ilustra lo anteriormente comentado. Se observa cómo, mediante la estructuración usando documentos MSC, se puede descomponer el desarrollo de sistemas complejos en varios módulos distintos, facilitando las tareas de programación, y ensamblarlos posteriormente siguiendo el orden expresado gráficamente de forma muy sencilla mediante el documento MSC.

#### 3.1.1.2. Descripción de tareas Web mediante diagramas MSC

Una gran parte de los programas de navegación automática se limitan generalmente a solicitar una serie de páginas o recursos Web (estableciendo las sesiones que fueran necesarias para ello) y procesar la información

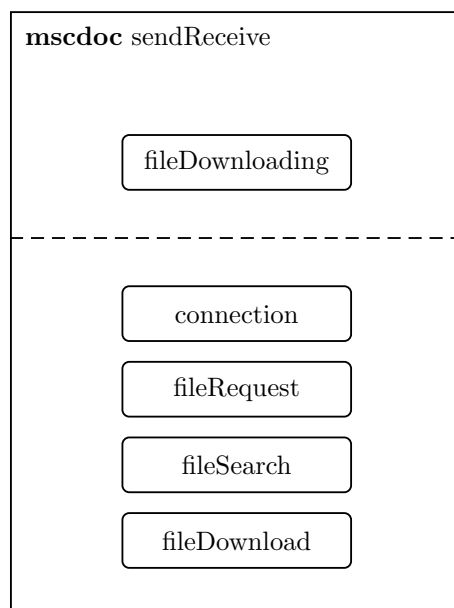


Figura 3.1: Ejemplo de Documento de Proyecto

contenida en éstas, generalmente de forma ordenada y secuencial, mediante composiciones deterministas. Este comportamiento típico es fácilmente representable empleando un subconjunto muy reducido de la sintaxis gráfica de MSC básico. Los elementos con que trabajamos son:

- Entidades.
- Mensajes.
- Referencias.
- Expresiones inline para control de flujo y combinadores de servicio.
- Condiciones.
- Acciones.
- Temporizadores.
- Correcciones.
- Creación dinámica de entidades, para modelar el lanzamiento de Threads de ejecución.
- Comentarios.

**3.1.1.2.1. Entidades** Todo escenario Web consta de varios elementos que se comunican entre sí para la consecución de la tarea. Los distintos elementos que intervienen en el proceso se representan por entidades. En nuestro caso, en todo desarrollo que hagamos habrá obligatoriamente una entidad, generalmente denominada *Cliente*, que interactuará consigo mismo o con otra u otras entidades. Cada entidad se identifica con un nombre distintivo.

El eje vertical de la entidad representa de forma gráfica la evolución temporal de los eventos. Los eventos posibles dentro de una entidad serán el envío o recepción de información (identificado mediante mensajes), la invocación de referencias o acciones, expresiones de control de flujo, temporizadores o combinadores de servicio, comentarios y creación o finalización de Threads de ejecución, etc.

**3.1.1.2.2. Mensajes** La operación más básica que tendremos dentro de una entidad es el envío y recepción de mensajes que representan el intercambio de información. Es obvio que deben existir diferentes tipos de mensajes, en función de las distintas primitivas del protocolo HTTP. Tendremos entonces por lo menos los siguientes tipos de mensajes:

- GET

- **POST**
- **HEAD**

Puesto que se pretenden automatizar acciones desde el lado cliente, y que éste generalmente no tiene privilegios ni necesidad de modificar los recursos del servidor, no se han incluido los métodos *PUT*, *DELETE*, etc. Se otorga además libertad al programador de definir tipos nuevos si fuera necesario.

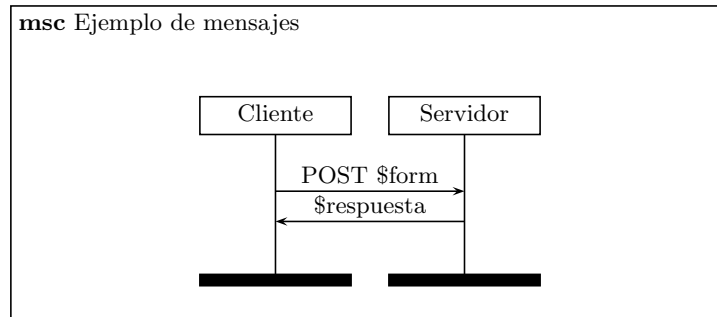


Figura 3.2: Ejemplo de mensaje

En general, y salvo para el caso de combinadores de servicio, no se tendrá en cuenta la pérdida de mensajes, se supone que la plataforma de desarrollo en que posteriormente se crearán los programas maneja este tipo de errores. Esto quiere decir que todos los mensajes se envían y reciben correctamente y en caso de que no fuera así, la ejecución de la tarea se interrumpe.

Los mensajes, representados mediante flechas, identifican los extremos de la comunicación, el tipo del mensaje y los parámetros necesarios para identificar la comunicación (URL, formulario, etc...). Un ejemplo puede verse en la figura 3.2.

**3.1.1.2.3. Referencias** Se permiten referencias a otros documentos MSC, previamente definidos en el documento MSC. Esto permite entre otras cosas crear un programa principal a base de construir subrutinas y luego enlazarlas juntas mediante referencias. Las referencias se hacen empleando el nombre del módulo que se quiere invocar e indicando los parámetros, tanto de entrada como de salida, que pudieran ser necesarios.

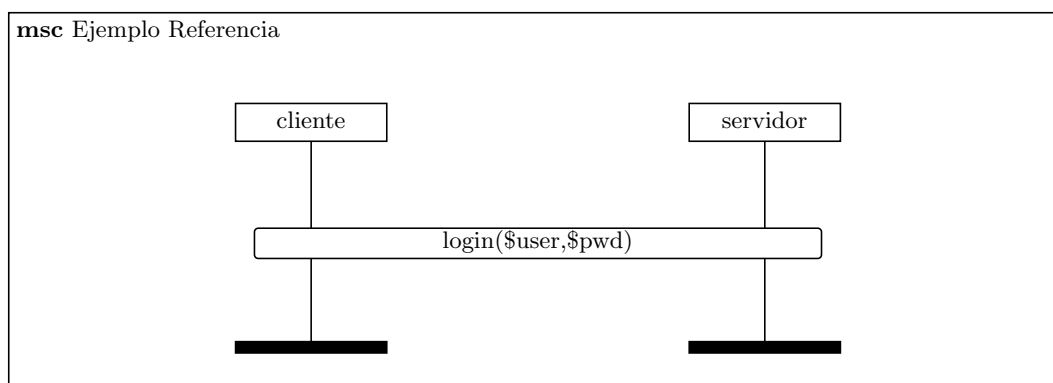


Figura 3.3: Ejemplo de referencia

**3.1.1.2.4. Control de flujo** En un cliente típico podrán aparecer las siguientes expresiones de control de flujo:

- **If-Then**
- **If-Then-Else**
- **Repeat-Until**



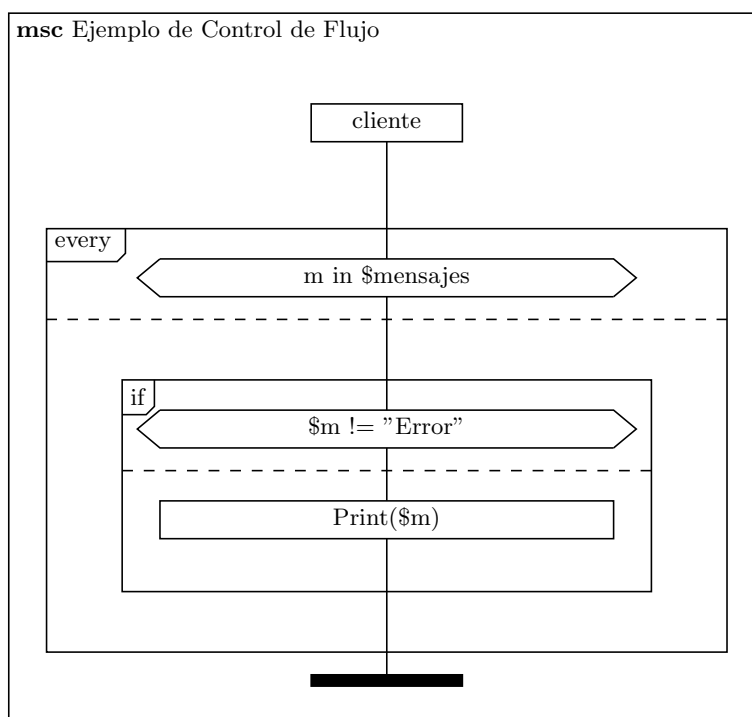


Figura 3.4: Ejemplo de control del flujo

- **While**
- **Every**

Para expresarlas en MSC crearemos nuevas *expresiones inline* ligeramente distintas de las aparecidas en el estándar. El símbolo es el típico de las expresiones inline, únicamente cambia la leyenda, para identificar cada una de los posibles expresiones de control de flujo. La última expresión *Every* es muy útil en computación Web, donde recorrer iterativamente todos los elementos de un determinado tipo en una página (enlaces, imágenes, etc) es una operación muy común. Además, estas expresiones pueden ser anidadas.

**3.1.1.2.5. Condiciones** Dentro o fuera de expresiones inline sirven para reflejar estados del sistema o guardas. Si aparecen fuera de una expresión de control de flujo, sirven para identificar el estado de una sesión. Esto permite indicar en un punto del diagrama que ya se ha accedido en el sistema o que se espera la recepción de un recurso antes de continuar procesando la tarea. Si están dentro de una expresión de control de flujo representan la condición a evaluar por la expresión.

**3.1.1.2.6. Procesamiento interno: Acciones** Además del intercambio de mensajes, pueden definirse acciones en los MSC que representan las operaciones que se pueden hacer con los datos obtenidos mediante ese intercambio de mensajes. Mediante XPath es sencillo seleccionar una determinada tabla dentro de un documento Web o rellenar un formulario con un dato concreto, por ejemplo.

Será necesario definir variables dentro de los escenarios que representen páginas descargadas o elementos dentro de una tabla. Mediante acciones podremos variar el contenido de esas variables o crear otras nuevas. La sintaxis recomendada es la de XPath 1.0, sin embargo es posible el uso de cualquier lenguaje de especificación que se desee.

**3.1.1.2.7. Threads** En ocasiones puede ser útil realizar dos procesados de forma concurrente. El modelo permite el lanzamiento de dos hilos concurrentes de ejecución. En principio hay libertad para la definición de N de estos hilos, sin embargo el representar más de dos de estos hilos en un diagrama de dos dimensiones es complicado. Un ejemplo simple del uso de Threads aparece representado en la figura 3.6.

**3.1.1.2.8. Comentarios** Podremos introducir comentarios en los MSC a fin de documentar aspectos del código. La figura 3.7 incluye un ejemplo de comentario en un diagrama MSC.

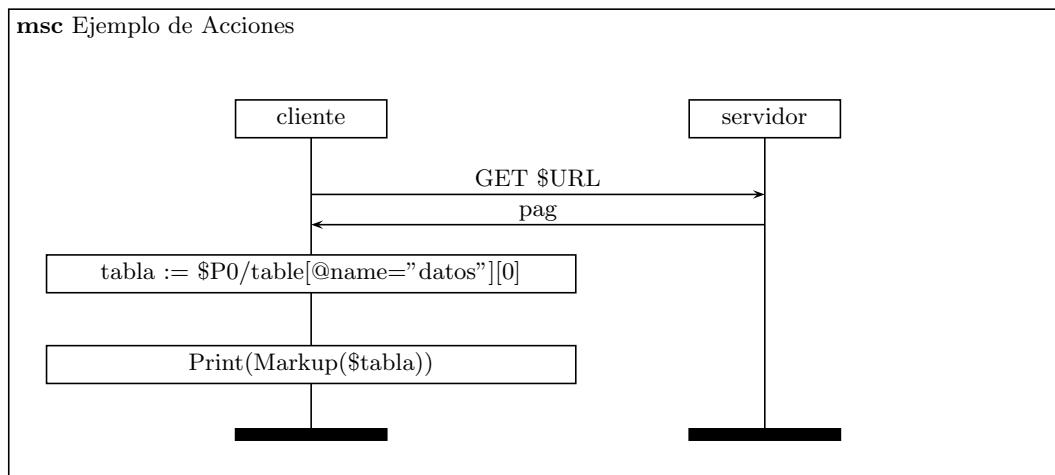


Figura 3.5: Ejemplo de acción XPath

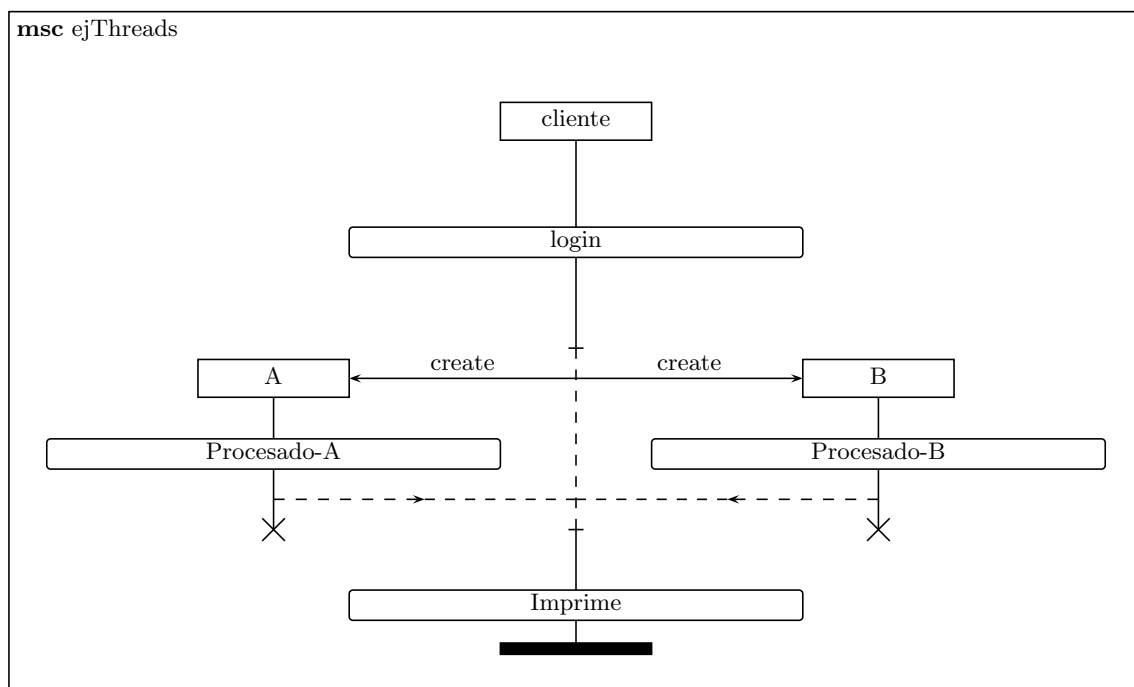


Figura 3.6: Ejemplo del uso de Threads

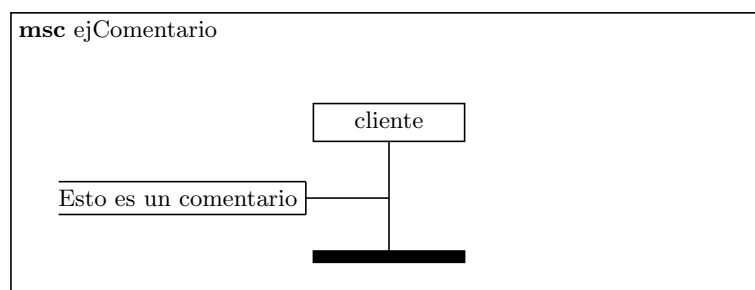


Figura 3.7: Ejemplo del uso de comentarios

### 3.1.1.3. Descripción de combinadores de servicio empleando MSC Básico con expresiones inline

Los combinadores que hemos incluido en el modelo son:

- Ejecución secuencial.
- Límite temporal (timeout).
- Repetición.
- No-terminación (stall).

Se ha optado por definir nuevas expresiones inline para representar los combinadores *secuencial*, *concurrente* y *repetición*. Para los combinadores *timeout* y *stall* empleamos dos elementos que si que existen en la recomendación, los temporizadores, cuyo símbolo gráfico es un reloj de arena; y las regiones de suspensión cuyo símbolo gráfico es una caja con línea de puntos asociada al eje de la entidad suspendida.

Los combinadores secuencial, concurrente y repetición pueden generar fallos, es decir, las páginas seleccionadas pueden no llegar a descargarse, con lo que la ejecución del diagrama MSC debe finalizar en el momento en que falla alguno de los servicios combinados. Una vez más corresponde a la plataforma de desarrollo/ejecución el manejar este tipo de errores.

Se incluyen a continuación varias gráficas que muestran nuestra propuesta de representación de los distintos combinadores de servicio expresados mediante MSC. La figura 3.11 representa un posible uso del combinador de servicio *stall* (representado en el diagrama como una región de suspensión) en conjunción con temporizadores. Se repite consecutivamente hasta que se tenga éxito la petición de P1 esperando 10 segundos entre intento e intento.

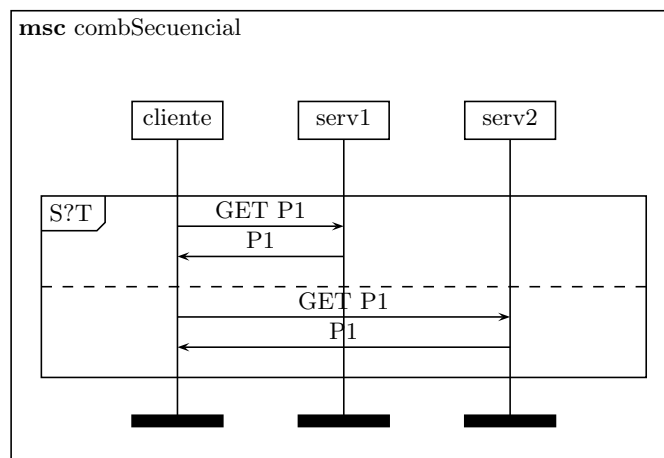


Figura 3.8: Combinador de Servicio *secuencial* en MSC

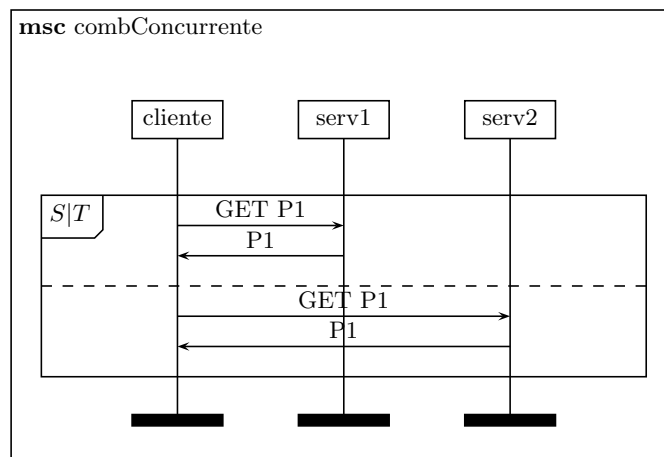


Figura 3.9: Combinador de Servicio *concurrente* en MSC

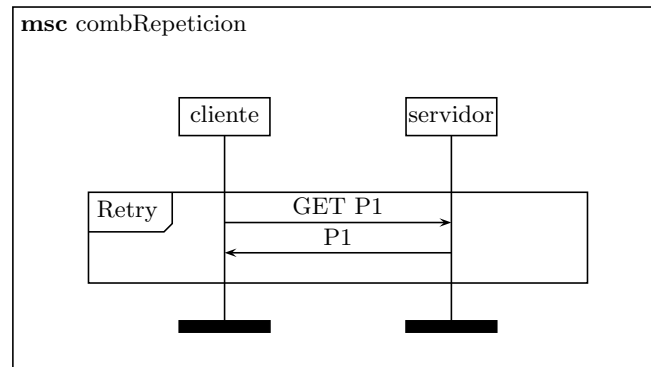


Figura 3.10: Combinador de Servicio *repetición* en MSC

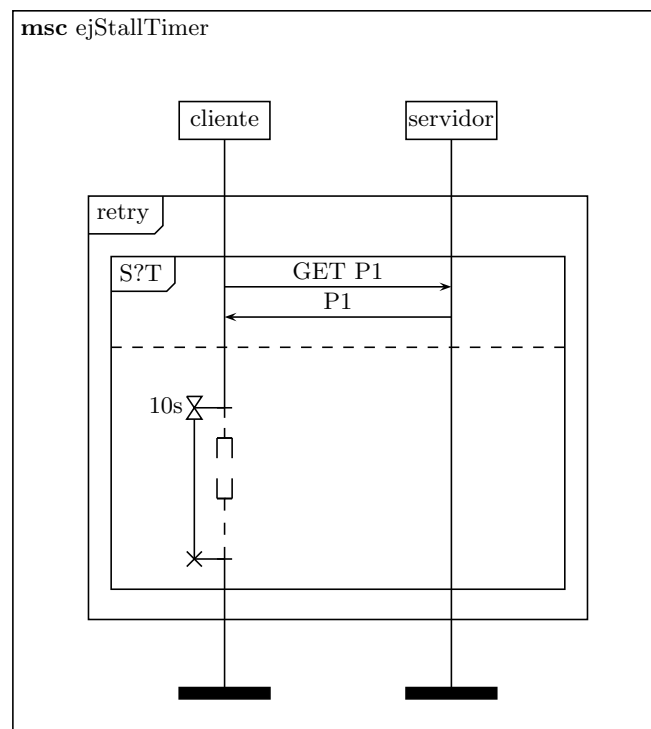


Figura 3.11: Ejemplo de los combinadores de servicio *stall* y *timer* en MSC

Es importante no confundir las expresiones inline de control de flujo (generalmente vinculadas a una única entidad) de las que expresan combinadores de servicio, que relacionan al menos dos entidades (cliente servidor), salvo en el caso de *stall* que es una condición que únicamente afecta a una entidad concreta.

#### 3.1.1.4. Variables y tipos de datos

El modelo no hace un tratamiento específico de las variables y tipos de datos posibles. Sin embargo conviene tener claro cuáles son los tipos de datos relevantes para la automatización de tareas en el Web, y el manejo interno de las variables.

Una variable en XPath y por tanto en el modelo propuesto, consta de un identificador único. La referencia a estas variables se hace antecediendo el símbolo "\$" al identificador de la misma. Los tipos de datos posibles específicos de la programación Web, además de los propios de lenguajes de programación tradicionales (int, bool, real, char, string, list, set) son:

- **Página:** representa una página descargada. Incluye tanto su código HTML como una serie de propiedades entre las que se incluye su URL;
- **Parte:** identifica una subregión contigua de una página;
- **Conjunto de partes:** una colección de partes;

Para acceder a alguna de las propiedades de una variable de los tipos anteriores se hace indicando el nombre del atributo a continuación del identificador de variable y precedido por un punto. Por ejemplo, `pag.URL` devuelve dicho atributo de la variable `pag`.

Estos tipos, basados en los definidos en el álgebra de marcado de WebL, son apropiados para este tipo de programación, y no son exclusivos del lenguaje WebL. Sin embargo, el posible uso de estos depende del nivel de abstracción del lenguaje final. Si no estuvieran presentes en el lenguaje utilizado, habría que definir una capa previa de conversión de estos tipos abstractos a otros presentes en el lenguaje.

#### 3.1.1.5. Funciones misceláneas

Para la automatización de tareas en el Web es necesario incluir varias funciones para ayudar al procesamiento de la información. En ocasiones es necesario leer datos desde el teclado, o imprimir en pantalla. Otras funciones también muy útiles son aquellas que afectan al marcado HTML de las páginas. Se han definido cuatro de estas funciones, sin embargo se da libertad a los diseñadores a incluir todas aquellas que fueran necesarias, con la única previsión de que si la plataforma de ejecución no dispone de éstas, habrá que programarlas. Los lenguajes típicos de programación para el Web incluyen éstas y muchas otras, como por ejemplo funciones de creación de nuevos documentos HTML a partir de los datos procesados.

**3.1.1.5.1. PrintLn** Imprimir los resultados obtenidos es una tarea común en la programación Web. Es necesario por tanto disponer de una función para ello. El nivel de abstracción de esta función queda indefinido y sujeto al lenguaje de programación en que se transformará la especificación MSC.

La sintaxis de la función es `PrintLn(dato)`. El dato a imprimir será generalmente de tipo textual, sin embargo por lo comentado anteriormente, si el lenguaje final lo implementa podrá tener cualquier otro tipo.

**3.1.1.5.2. ReadLn** Generalmente los programas definidos no requieren interacción con el usuario salvo quizá por el uso de parámetros de ejecución o variables predefinidas. En ocasiones será necesario también la introducción de algún dato por parte del usuario a través del teclado. Para ello hemos incluido la función *ReadLn* cuya sintaxis es `variable = ReadLn`. El dato leído se almacena en *variable* para su posterior uso.

**3.1.1.5.3. Markup** Bien sea para construir una nueva página, o para analizar el código HTML de una página descargada o de parte de ella, resulta apropiada la definición de una función que sea capaz de devolver el código HTML de una variable dada. Se ha adoptado el uso de la función *Markup()* dentro del modelo, de forma que sea sencillo calcular este marcado de los elementos.

**3.1.1.5.4. Text** Una página HTML o una parte de ellas, en forma de conjunto de etiquetas, tiene información textual embebida, que en muchos de los casos ha de ser accedida y procesada. La función *Text()* devuelve todo el texto contenido en la variable argumento, que ha de ser de tipo página o parte de página.

### 3.1.2. Un ejemplo

A continuación se ilustra (figura 3.12) lo anteriormente comentado con un ejemplo ficticio en el que se incluyen muchos de los elementos definidos en las secciones anteriores. El ejemplo se encarga de descargar una página, *P0*, del servidor, extraer el primer formulario de la página para introducir los valores adecuados de *login* y *password* para enviar dicho formulario y así obtener una página de resultados, almacenados dentro de una tabla. Una vez seleccionados aquellos resultados de interés se imprimen empleando una referencia a la función *imprimirResultado*.

## 3.2. MSC2WEBL: Diagrama de bloques del sistema

La arquitectura del sistema, representada en la figura 3.13, está basada básicamente en el lenguaje definido para representar los diagramas MSC mediante un Schema XML. A partir de los documentos válidos contra este Schema se pueden generar las transformaciones pertinentes empleando plantillas XSLT. Este esquema permitiría la invocación de las transformaciones desde una consola de texto, incluso en un terminal remoto, empleando cualquiera de los analizadores XSLT existentes, como por ejemplo Xalan[47].

Sin embargo, lo que pretende la aplicación es facilitar este tipo de desarrollos y el mecanismo manual de ejecución en una consola de texto de alguno de los analizadores existentes presenta algunas complejidades, sobre todo a personas no expertas en estas tareas. Para solucionar esto, se desarrolla sobre la arquitectura básica un interfaz gráfico que incluye mejoras de facilidad de uso al incorporar un editor de diagramas MSC, que evita la aparición de errores en el marcado y permite trabajar con las representaciones gráficas de los diagramas MSC, manejo interno de las transformaciones XSLT, un explorador de marcado de páginas HTML (o derivados) y una consola de ejecución de los programas generados. El esquema de la arquitectura del interfaz gráfico aparece en la figura 3.14.

Para el funcionamiento interno del interfaz gráfico se ha definido otro Schema XML para describir los datos asociados a cada proyecto. A continuación vemos en más detalle los elementos que componen la arquitectura del sistema.

### 3.2.1. Schemas XML

La aplicación trabaja con dos lenguajes distintos. El más complejo se crea para la representación MSC de las tareas que se quieren programar; el otro, muy simple, define un nuevo tipo de datos para representar un proyecto MSC2WEBL, con todas sus propiedades (descripción, autor, ficheros asociados, etc).

#### 3.2.1.1. Schema MSC para descripción de tareas Web

Empleando este lenguaje, se permite crear de forma sencilla documentos y diagramas MSC y validarlos para probar la corrección de los documentos conforme a este Schema. Los documentos válidos pueden ser transformados mediante plantillas XSLT a la representación gráfica y al código WeBL.

Los elementos definidos por el Schema aparecen representados en la figura 3.15. Por simplicidad no se han incluido los distintos atributos de cada elemento. Este Schema es además fácilmente ampliable para incluir futuras mejoras si fuera necesario. De esta forma comportamientos que no se habían tenido en cuenta inicialmente pueden ser incluidos sin demasiados problemas. Basándose en este Schema se puede desarrollar una aplicación de ayuda a la creación de los diagramas MSC minimizando el número de errores de validación al mínimo. Esto es lo que se ha hecho con el editor MSC incluido con la aplicación.

#### 3.2.1.2. Esquema proyecto MSC2WebL

El Schema XML de datos de proyecto define los distintos elementos que conforman un proyecto MSC2WEBL. Estos son el nombre del proyecto y opcionalmente una descripción y autor asociados, y los datos referentes a los ficheros de datos (de proyecto, de diagramas MSC, de código L<sup>A</sup>T<sub>E</sub>X y código WeBL). El Schema completo aparece en la figura 3.16.

### 3.2.2. Transformaciones XSLT

Las principales funcionalidades del sistema se obtienen a partir de la representación XML de los diagramas MSC aplicando transformaciones XSLT que convierten dichas representaciones tanto a código L<sup>A</sup>T<sub>E</sub>X como a código WeBL. Además de estas transformaciones se han definido otras, mucho más sencillas para

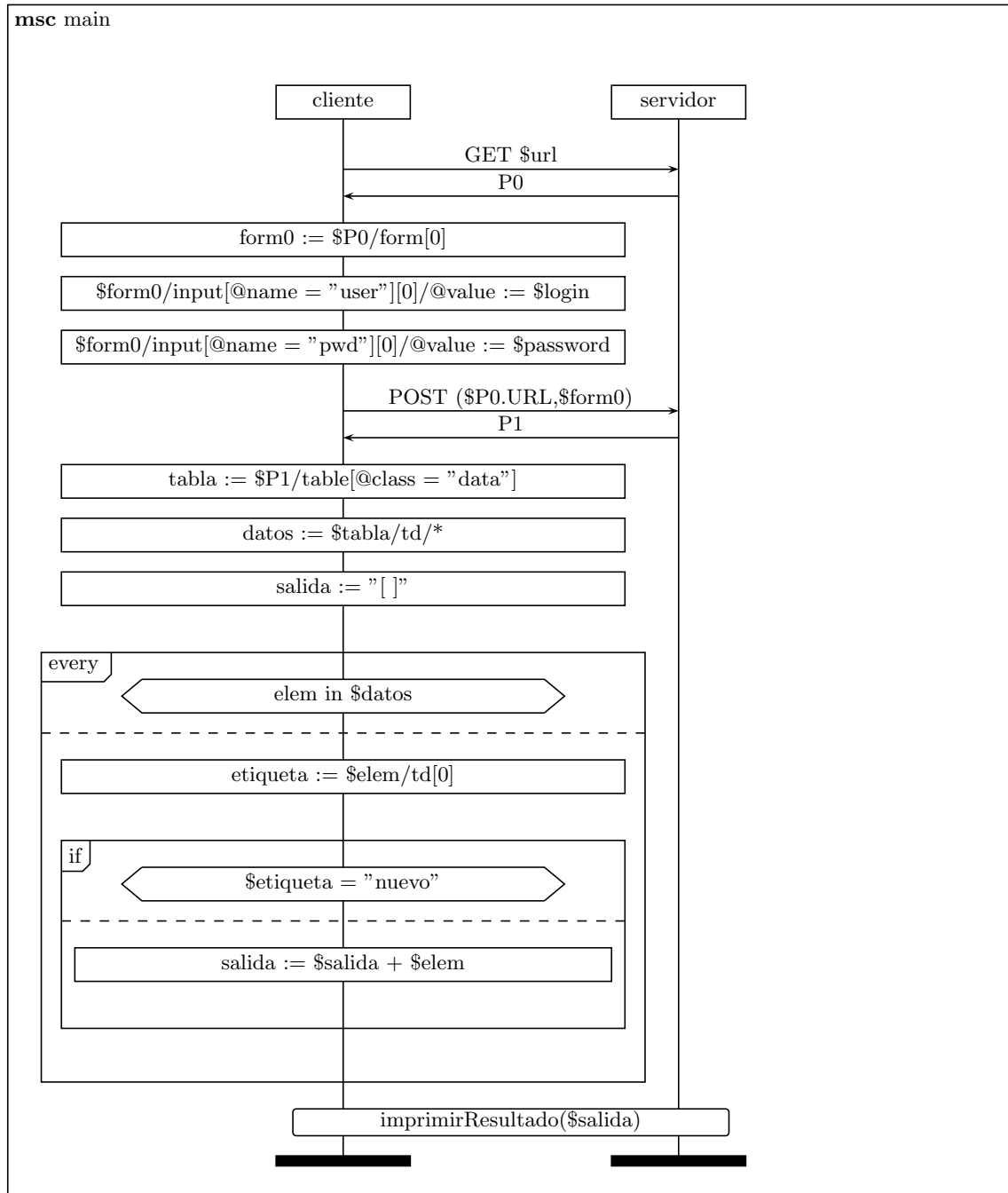


Figura 3.12: Ejemplo de tarea Web representada mediante MSC

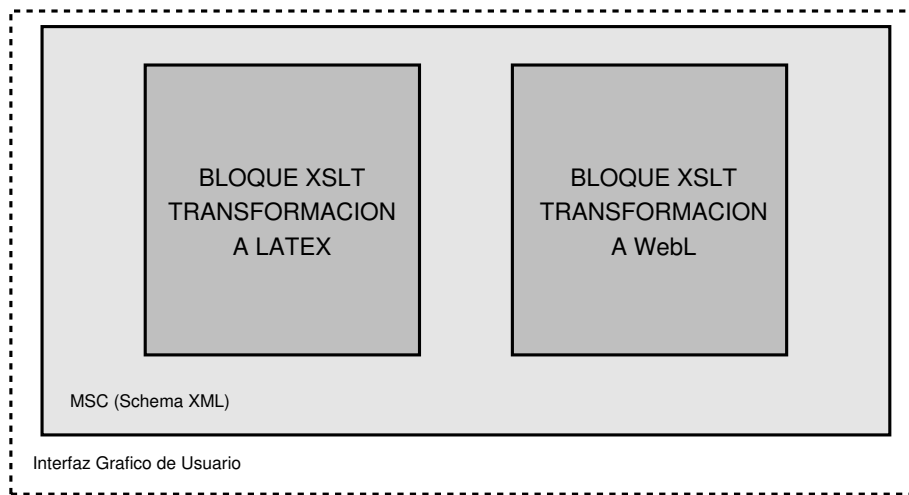


Figura 3.13: Arquitectura del sistema

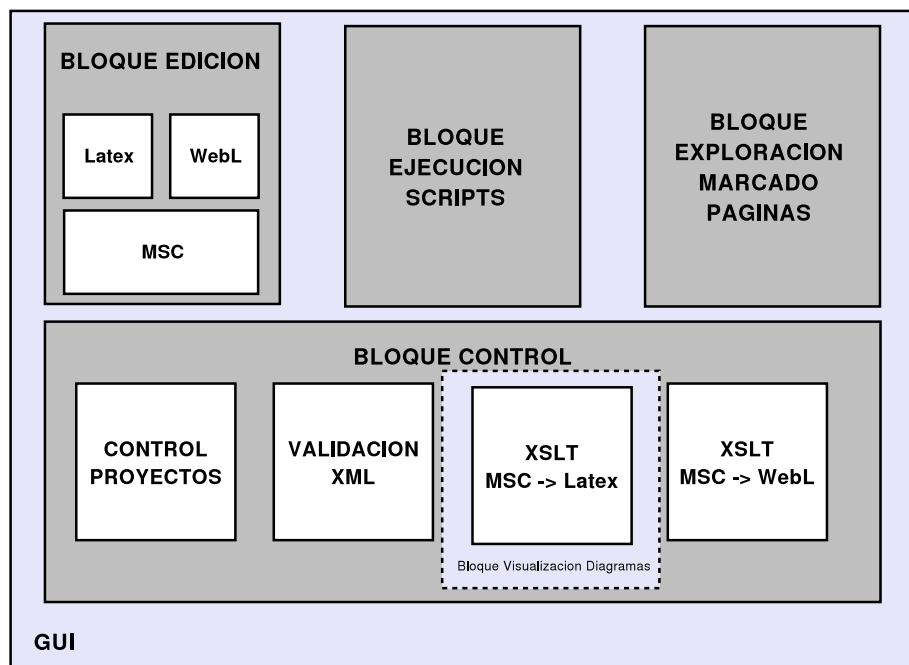


Figura 3.14: Arquitectura del interfaz gráfico de usuario

funcionamiento interno de la aplicación. De esta forma, la extracción de ciertos datos relevantes para el funcionamiento del programa se hace de forma sencilla sin necesidad de recurrir a la búsqueda por patrones de texto en los ficheros donde se guarda esta información. Por tanto se han definido cuatro transformaciones XSLT:

- Transformación a código  $\text{\LaTeX}$ .
- Transformación a código WebL.
- Transformación para la extracción de propiedades de un proyecto.
- Transformación para la extracción de entidades de un proyecto.

### 3.2.2.1. Transformación a código $\text{\LaTeX}$

Es la transformación más compleja de las creadas para el proyecto. Su objetivo es ayudar al dibujo en pantalla de los diagramas MSC a partir de las representaciones textuales en XML. A partir del código  $\text{\LaTeX}$ , este puede convertirse a formato PDF, para ayudar a las tareas de documentación o bien a formato JPG,



- msc
  - documento
    - definiciones
      - ◊ refdoc
    - utilidades
      - ◊ refdoc
  - diagramaMSC
    - listaInstancias
      - ◊ inst
    - eventos
      - ◊ saltoNivel
      - ◊ mensaje
      - ◊ accion
      - ◊ ref
      - ◊ if
      - ◊ ifelse
      - ◊ repeat
      - ◊ while
      - ◊ every
      - ◊ combseq
      - ◊ combpar
      - ◊ combretry
      - ◊ timeout
      - ◊ thread
      - ◊ comentario

Figura 3.15: Elementos definidos por el Schema MSC

que puede ser empotrado dentro de la aplicación gráfica. Uno de los problemas que presenta este modelo, es que la visualización de los diagramas no se hace en tiempo real. Sin embargo, el tiempo de compilación del fichero y su posterior conversión a JPG es pequeño en un ordenador relativamente moderno.

La plantilla se ha diseñado con tres tamaños estándar (grande, mediano y pequeño) para acomodarse a la mayor parte de los esquemas, si bien en algunos casos, habrá que modificar la salida de forma que algunos elementos se ajusten mejor. Puede ocurrir por ejemplo que una acción sea demasiado larga como para entrar en su caja de texto, por lo que habría que variar el ancho predeterminado de la plantilla con la que se trabaje. Existen varios ficheros con las distintas plantillas de cada tamaño<sup>1</sup> Para ello, en cada uno de estos ficheros se incluyen unas líneas con el valor por defecto de las distancias y tamaños más importantes en un diagrama. Estos valores por defecto pueden cambiarse para obtener el diagrama deseado. El paquete para dibujar los diagramas MSC define además otras distancias que, en alguno de los casos pueden resultar interesantes. Éstas aparecen en la figura 3.17.

Al ser esta plantilla XSLT perfectamente modificable, se pueden introducir descripciones textuales de los diferentes diagramas, índices, comentarios o cualquier elemento que se desee. Si bien, conviene notar que para ello es necesario tener unos mínimos conocimientos de L<sup>A</sup>T<sub>E</sub>Xy del paquete para dibujar diagramas MSC.

Se ilustra con un pequeño ejemplo la salida resultante de aplicar esta plantilla a un documento válido. La figura 3.18 incluye el código origen antes de aplicar la transformación. La figura 3.19 el código L<sup>A</sup>T<sub>E</sub>Xresultante.

### 3.2.2.2. Transformación a código WebL

Para obtener el código WebL ejecutable empleamos una nueva transformación XSLT sobre los documentos XML válidos contra el schema MSC-XML. El funcionamiento de esta plantilla es distinto: al aplicarla

<sup>1</sup>Estos aparecen ocultos en el directorio `xm1`.

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  attributeFormDefault="unqualified">

  <xs:element name="proyecto">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nombre" type="xs:string"/>
        <xs:element name="autor" type="xs:string" minOccurs="0"/>
        <xs:element name="descripcion" type="xs:string" minOccurs="0"/>
        <xs:element name="ficheros">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="fichero" type="tipoFichero" maxOccurs="4"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="tipoFichero">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="tipo" use="required">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="proyecto"/>
              <xs:enumeration value="msc"/>
              <xs:enumeration value="latex"/>
              <xs:enumeration value="webl"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

</xs:schema>

```

Figura 3.16: Schema XML de datos de proyecto MSC2WEBL

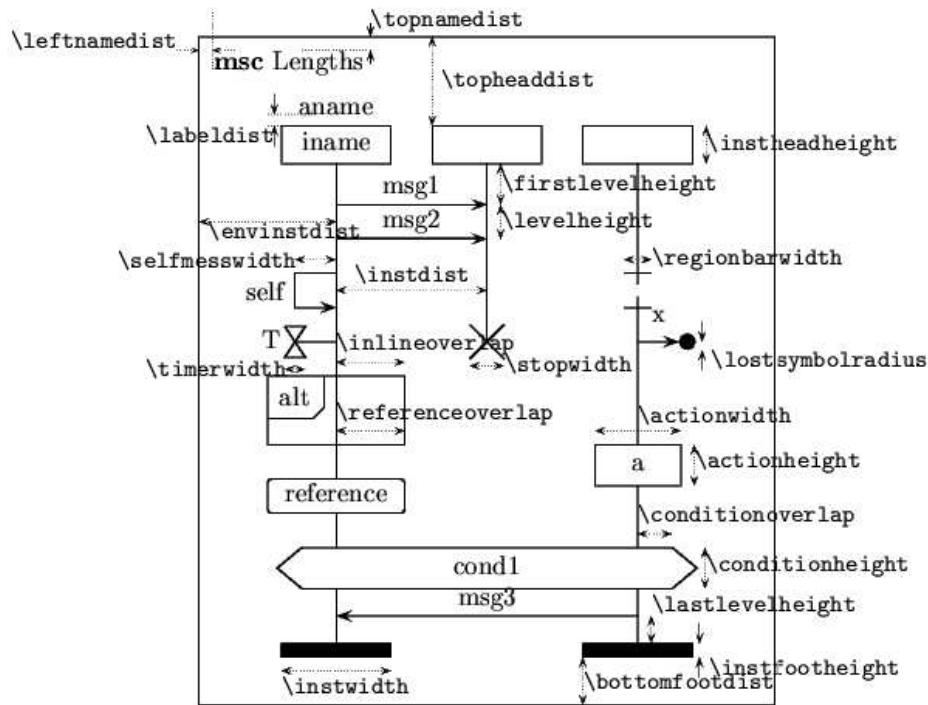


Figura 3.17: Parámetros definibles por el usuario en un diagrama MSC

sobre un documento XML, en primer lugar, todos los diagramas referenciados en la parte de utilidades se transforman a funciones WebL, cada una con los parámetros de entrada y salida que fueran necesarios, mientras que los diagramas referenciados en la parte de definiciones se transforman a su código WebL y se empalman uno a continuación del otro para dar lugar al programa principal. Por tanto, para el funcionamiento de esta plantilla XSLT es necesario definir la estructura del sistema mediante un documento MSC o el sistema no será capaz de componer los distintos diagramas MSC para generar el programa final.

Esta plantilla incluye en todos los desarrollos los valores contenidos en unas variables del sistema que determinan algunos parámetros importantes. La figura 3.20 contiene estas variables predefinidas. Los módulos WebL importados por defecto son *Forms* y *Files* que permiten la transformación de las páginas recibidas a XHTML, el manejo eficiente de formularios Web y leer y salvar datos a disco. La variable *home* es base para calcular la ruta del directorio temporal, donde se guardan archivos intermedios e información temporal. La variable *headers* permite identificar nuestro cliente Web como si fuera cualquiera de los navegadores existentes, así como incluir datos específicos en las cabeceras HTML que enviamos. Por último, La variable *eliminables* sirve para, a la hora de convertir las páginas HTML a XHTML, eliminar aquellas etiquetas HTML que no fueran relevantes para la tarea que se desea automatizar. De esta forma es fácil eliminar las etiquetas de estilo de una página, ya que sólo afectan al aspecto visual de la información contenida en la página y no a la información misma.

La figura 3.21 contiene el código WebL generado resultado de aplicar la plantilla XSLT al código XML de la figura 3.18.

```

<?xml version="1.0" encoding="UTF-8"?>
<msc xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../xml/msc.xsd">

<diagramaMSC nombre="main" autor="Daniel Garcia Jones">
  <listaInstancias>
    <inst nombre="cliente" tipo="normal"/>
    <inst nombre="ecol" tipo="normal"/>
  </listaInstancias>
  <eventos>
    <mensaje in="ecol" out="cliente" tipo="get" varSalida="P0">
      'http://tira.escomposlinux.org/'
    </mensaje>
    <saltoNivel/>
    <mensaje in="cliente" out="ecol" tipo="reply" varSalida="P0">P0</mensaje>
    <saltoNivel/>
    <accion inst="cliente" tipo="var" varSalida="imagen">
      <seleccion tipo="simple">
        <varOrigen>$P0</varOrigen>
        <elemBusqueda>img</elemBusqueda>
        <pos>1</pos>
      </seleccion>
    </accion>
    <saltoNivel/>
    <mensaje in="ecol" out="cliente" tipo="getfile" varSalida="$ARGS[1]">$imagen.src</mensaje>
    <saltoNivel/>
    <mensaje in="cliente" out="ecol" tipo="replyFile" varSalida="$ARGS[1]">$ARGS[1]</mensaje>
  </eventos>
</diagramaMSC>
</msc>

```

Figura 3.18: Código XML antes de aplicar la plantilla de transformación a L<sup>A</sup>T<sub>E</sub>X

```

\documentclass[a4paper,8pt]{article}
\usepackage[latin1]{inputenc}
\usepackage[activeacute,spanish]{babel}
\usepackage{graphicx}
\usepackage{amsmath}
\usepackage{amsfonts}
\usepackage{amssymb}
\usepackage{msc}
\pagestyle{empty}

\begin{document}
% Diagrama MSC (main)
% Autor: Daniel Garcia Jones
\begin{msc}{main}
%
%Parametros por defecto (SE PUEDEN EDITAR PARA CAMBIAR EL ASPECTO DEL DIAGRAMA)
%
\setmscvalues{small}
\setlength{\firstlevelheight}{0.1cm}
\setlength{\instdist}{7cm}
\setlength{\instwidth}{2cm}
\setlength{\actionwidth}{8cm}
\setlength{\envinstdist}{5cm}
\setlength{\conditionoverlap}{3cm}
\setlength{\inlineoverlap}{4.5cm}
%
% Fin parametros por defecto

\declinst{cliente}{}{cliente}
\declinst{ecol}{}{ecol}
\mess{GET ''http://tira.escomposlinux.org/''}{cliente}{ecol}
\nextlevel
\mess{P0}{ecol}{cliente}
\nextlevel
\action{imagen := \P0/img[1]}{cliente}
\nextlevel
\nextlevel
\mess{GET \$imagen.src}{cliente}{ecol}
\nextlevel
\mess{ARGS[1]}{ecol}{cliente}
\end{msc}
\end{document}

```

Figura 3.19: Código L<sup>A</sup>T<sub>E</sub>X después de aplicar la plantilla XSLT

```
//  
// Modulos importados por defecto  
import Forms,Files;  
  
//  
// Variables predeterminadas  
var home="/home/jones/";  
var temporal= home + "msc2webl/tmp/";  
var headers=[. "User-Agent"=Forms_useragent .];  
var eliminables = "";
```

Figura 3.20: Variables y módulos WebL predefinidos

```
//  
// Modulos importados por defecto  
import Files,Forms;  
//  
// Variables predeterminadas  
var home="/home/jones/pfc/msc2webl-1.1/";  
var temporal="/home/jones/pfc/msc2webl-1.1/tmp/";  
var headers=[. "User-Agent"=Forms_useragent .];  
var eliminables = "tr td table center font";  
  
// Programa: ejEcol  
// Autor: Daniel Garcia Jones  
//  
// Codigo WebL del diagrama main  
var P0 = Forms_Get("http://tira.escomposlinux.org/",nil,headers);  
P0 = Forms_xhtmliza(P0,"P0",eliminables,temporal);  
var imagen = Elem(P0,"img")[1];  
Files_GetURL(imagen.src,ARGS[1]);
```

Figura 3.21: Código WebL después de aplicar la plantilla XSLT

### 3.2.3. Interfaz gráfico de usuario

Para facilitar el uso del esquema propuesto se ha programado en Python un interfaz gráfico que incluye los elementos necesarios para crear fácilmente programas dentro de un entorno unificado. El empleo de este interfaz gráfico sobre la arquitectura descrita convierten al conjunto en una herramienta CASE para el desarrollo de sistemas de navegación automática del Web basado en los diagramas MSC.

La aplicación integra las siguientes funcionalidades:

- **Editor MSC:** permite de forma sencilla crear los diagramas MSC que definen el comportamiento que se quiere programar. Es posible hacerlo tanto en forma gráfica como textual, editando manualmente el código XML que, una vez válido contra el esquema, puede ser transformado a código WebL o código  $\text{\LaTeX}$ .
- **Editor  $\text{\LaTeX}$ :** incluye un sencillo editor de código para modificar si fuera necesario el código generado a partir del fichero XML.
- **Editor WebL y consola de ejecución:** Para editar el código WebL generado y visualizar el resultado de la ejecución de los scripts.
- **Visualizador de Marcado de Páginas HTML:** incluye un pseudonavegador capaz de mostrar el código de páginas HTML residentes en el Web o en disco.

#### 3.2.3.1. Editor MSC

Si bien es posible la edición de los documentos y diagramas manualmente siguiendo el Schema definido, mediante el editor MSC podemos crearlos de forma sencilla, visualizando a cada paso los elementos introducidos, ya que la aplicación es capaz de generar los diagramas correspondientes en formato jpg invocando la plantilla XSLT de transformación a código  $\text{\LaTeX}$ .

Muchos son los editores XML existentes en el mercado que permiten la edición de documentos XML y la validación de estos contra el documento sobre el que se definen, bien sea mediante el uso de DTDs o Schemas. Sin embargo el uso de un entorno de edición exclusivo para este tipo de documentos mediante el uso de botones y diálogos para cada uno de los posibles elementos en el documento facilita enormemente la creación de los programas, al tiempo que minimiza los errores, tanto si el resultado se observa gráfica o textualmente. El aspecto de este editor MSC se incluye en las figuras 3.22 y 3.23.

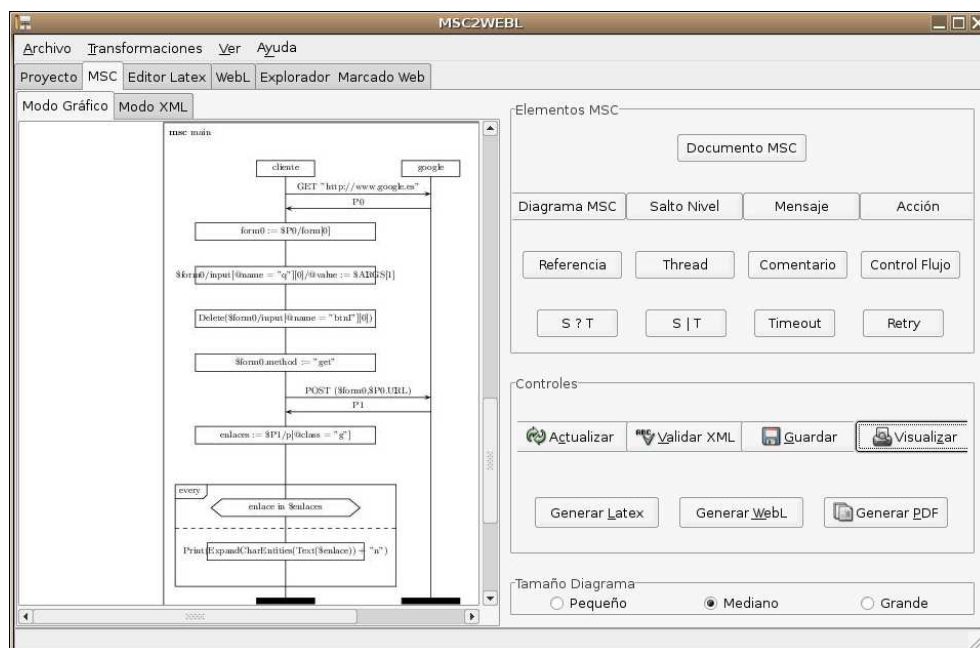


Figura 3.22: Ventana de edición de diagramas MSC (Modo Gráfico)

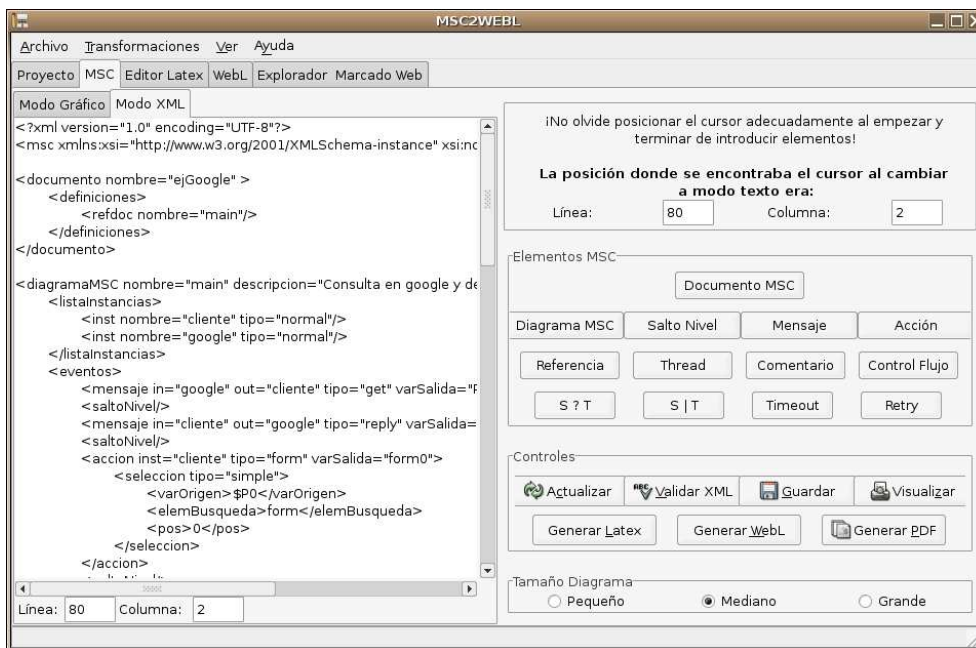


Figura 3.23: Ventana de edición de diagramas MSC (Modo XML)

Para la creación de los documentos basta con ir pulsando en los botones de los diferentes elementos que queremos introducir. Si editamos en la subventana de representación gráfica, el posicionado de los elementos dentro del diagrama se hace de forma automática. Si por el contrario editamos empleando la sintaxis XML, los elementos se irán creando en la posición que indique el cursor de la ventana de edición<sup>2</sup>. Generalmente al pulsar el botón del elemento deseado aparecerá un diálogo para introducir los parámetros necesarios. En todo momento podremos validar el documento editado, indicándonos si éste es correcto o, en que caso de que no lo fuera, en que línea aparece el error<sup>3</sup>

### 3.2.3.2. Editor $\text{\LaTeX}$

El proceso de visualización de los diagramas necesita de la transformación a código  $\text{\LaTeX}$  de los diagramas generados en código XML. Una vez obtenido este código, se compila y transforma a formato pdf. Este fichero pdf puede guardarse a disco, y emplearse en tareas de documentación, o convertirse, usando la librería *libmagick*, a formato jpg (que puede ser incrustado en tiempo de ejecución en el interfaz gráfico).

La ventana de edición de  $\text{\LaTeX}$  aparece en la figura 3.24. El código  $\text{\LaTeX}$  generado a partir del fichero XML puede ser editado en esta ventana, para incluir todas aquellas modificaciones que fuera necesario, como, por ejemplo, ajustar los tamaños predefinidos de los elementos de los diagramas para ajustarlos a cada caso concreto.

### 3.2.3.3. Editor WebL y consola de ejecución

El código WebL generado requiere en algunos de los casos de retoques finales, para la asignación de valores concretos a las variables predefinidas o incluso para indentarlo. Se incluye un pequeño editor para introducir todos los cambios que fueran necesarios en los scripts generados antes de guardarlos a disco. Se añade además la posibilidad de ejecutar en la misma ventana aquellos scripts que no requieran entrada de datos a través del teclado (salvo por el uso de parámetros) y observar el resultado de la ejecución. Esto ayuda en las fases de desarrollo y depuración de los sistemas. Una vez finalizados los programas se guardarán a disco y se ejecutarán, generalmente, desde una consola de texto o de forma automática al meterlos en el planificador de tareas del sistema. La ventana del editor WebL aparece en la figura 3.25.

<sup>2</sup>Se nos avisa en todo momento de la posición del cursor antes de pasar a modo XML.

<sup>3</sup>Para ello se emplea el validador XSV (implementado en Python). En algunas ocasiones, este validador no es capaz de describir correctamente el error presente en el código XML.



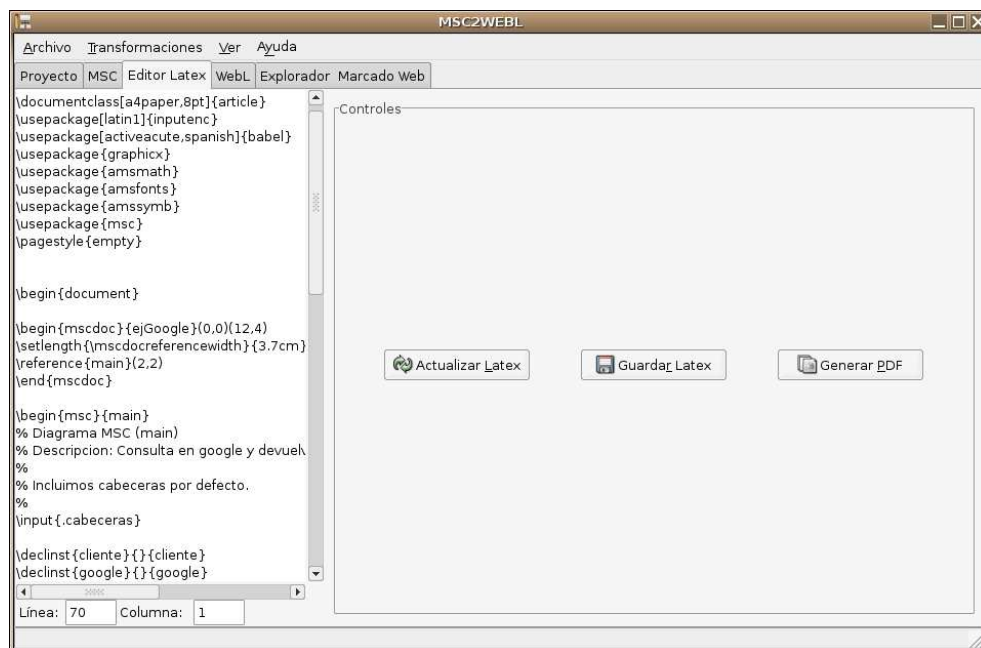
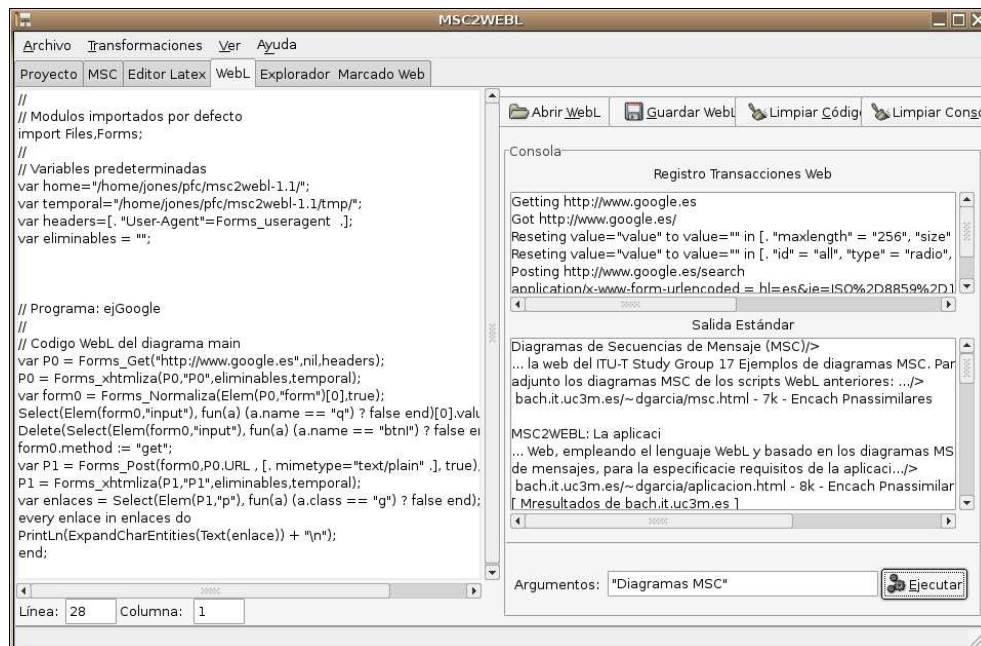
Figura 3.24: Ventana de edición  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 

Figura 3.25: Ventana de edición y ejecución de código WebL

#### 3.2.3.4. Visualizador Marcado Páginas

Se ha incluido un explorador de código HTML de páginas Web. Esto facilita las tareas de desarrollo ya que es una tarea muy común, al crear este tipo de sistemas, examinar el código de las páginas sobre las que se quiere automatizar ciertas tareas, empleando para ello generalmente un navegador en otra ventana externa, sobre la cuál, una vez descargadas las páginas, hay que indicar de forma explícita que se quiere examinar el marcado. Además, al emplear WebL y la librería *Forms* para descargar las páginas es posible eliminar ciertas etiquetas de la página Web de tal forma que el código que muestra este navegador es el mismo que se almacenará en el programa WebL que, por lo explicado anteriormente, puede ser distinto del original.

#### 3.2.3.5. Implementación con PyGTK y Glade

El interfaz gráfico se ha programado por entero empleando Python, Python-GTK y Glade. Python-GTK es un puente de Python a la librería gráfica GTK+ para creación de interfaces gráficas. Glade es una herramienta de especificación de aplicaciones GTK+ de forma sencilla empleando una herramienta de dibujo que permite crear ventanas con distintos elementos que luego serán manejados por eventos y señales. El fichero Glade de especificación, en formato XML, es interpretado en tiempo de ejecución y transformado al código Python-GTK correspondiente. De esta forma la creación del interfaz gráfico es una tarea mucho más sencilla que escribir directamente el código GTK.

Además de Python-GTK y Glade para la parte gráfica, se han empleado varios módulos Python para la validación de los documentos XML contra el Schema definido (XSV), la aplicación de las plantillas XSLT a los documentos ya válidos (4Suite) y la creación de los documentos de datos de proyecto (Jaxml). Estas librerías se explican con más detalle en el apéndice B.

# Capítulo 4

## Ejemplos de código

A continuación presentaremos una serie de ejemplos de código generado con la aplicación. Para cada uno de los ejemplos se incluye una descripción del marcado de las páginas navegadas y de los aspectos estructurales más relevantes para las tareas que se quieren automatizar. Se adjuntan también el código XML, los diagramas MSC y el código WebL correspondiente a cada uno de los programas. Los ejemplos se presentan ordenados en función de su complejidad y sirven para ilustrar no sólo el funcionamiento de la aplicación y del esquema MSC, sino también la metodología de programación para el Web introducida en este proyecto.

### 4.1. Ejemplo de estructura

Con el fin de entender cómo se componen los programas MSC2WEBL a partir de ensamblar distintos diagramas MSC básicos, presentamos un ejemplo muy sencillo que ilustra los mecanismos de descomposición estructural definidos. El programa simplemente imprime una serie de líneas por pantalla y descarga un par de páginas, la primera de ellas indicada al pasar como parámetro de ejecución su URL (para introducir también el uso de parámetros de ejecución); la segunda es el primer enlace que aparezca en la primera. Puesto que el ejemplo simplemente pretende ilustrar el concepto de estructura de un proyecto, el marcado de las páginas tratadas es irrelevante.

Se presentan dos estructuras de programa distintas con la misma funcionalidad. De esta manera se ilustra cómo el comportamiento deseado puede obtenerse de muchas formas diferentes al tener gran libertad para descomponer los programas usando diagramas y documentos MSC.

#### 4.1.1. Ejemplo A

En este ejemplo (ver figura 4.1) se ha descompuesto el programa principal en dos bloques, **bloqueA** (figura 4.6) y **bloqueB** (figura 4.7) que a su vez hacen referencia a otros diagramas MSC (definidos en la parte de utilidades del documento) **util1** (figura 4.3), **util2** (figura 4.4) y **util3** (figura 4.5). Los últimos son funciones WebL que son llamadas en algún punto de los diagramas **bloqueA** y **bloqueB**, que se ensamblan uno a continuación del otro para dar lugar al programa final, **ejEstructuraA**. La figura 4.2 muestra esta composición de los distintos bloques.

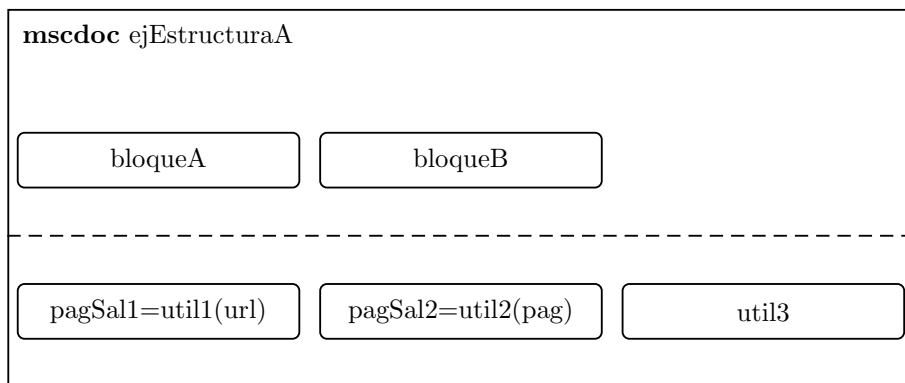
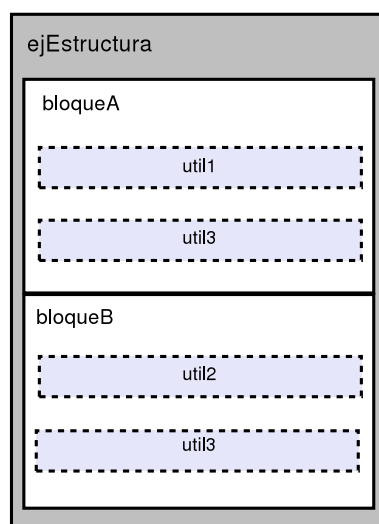
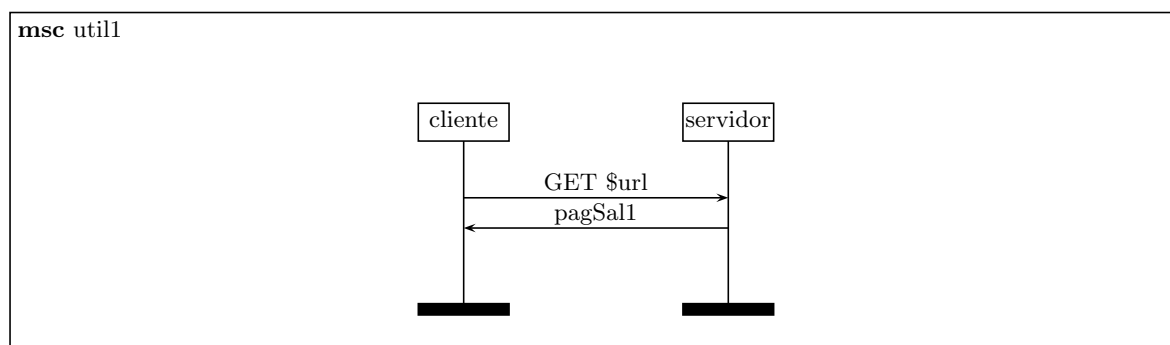
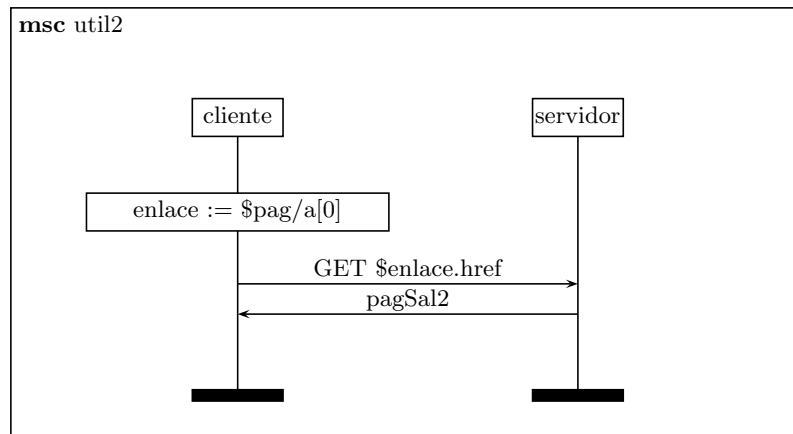
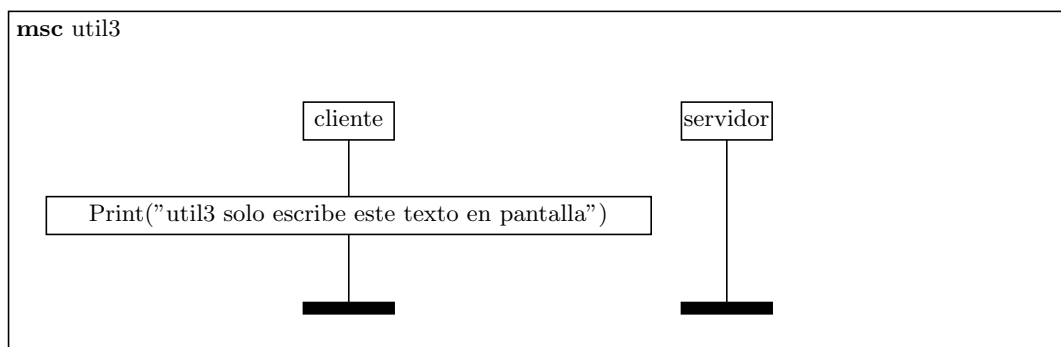
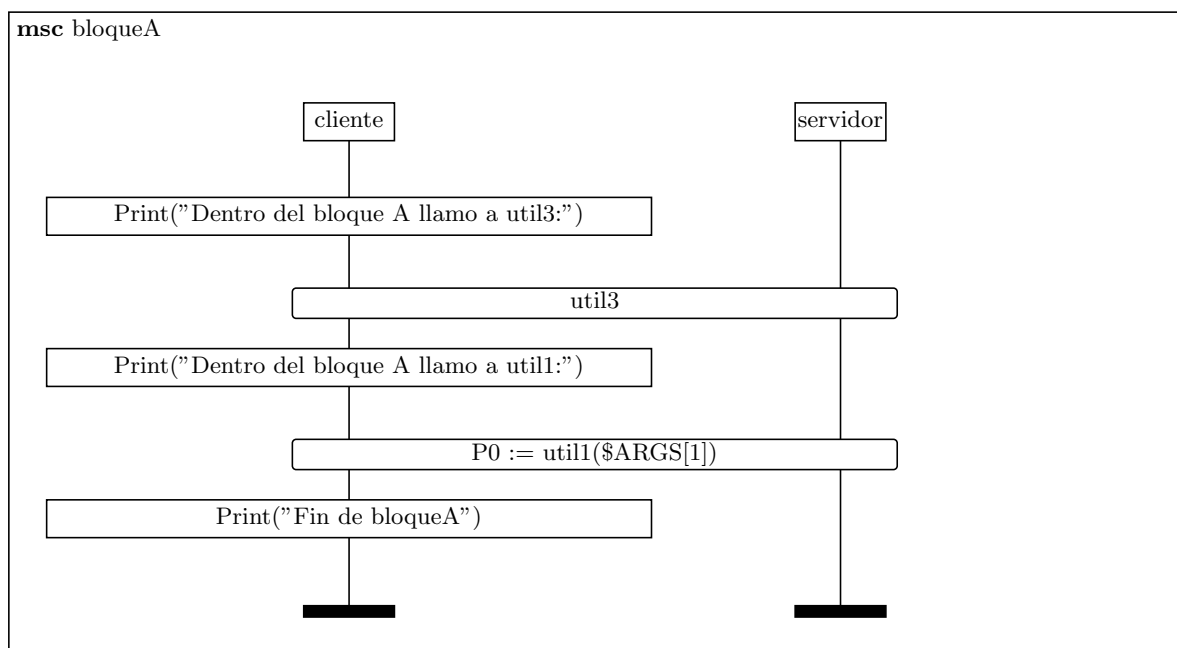
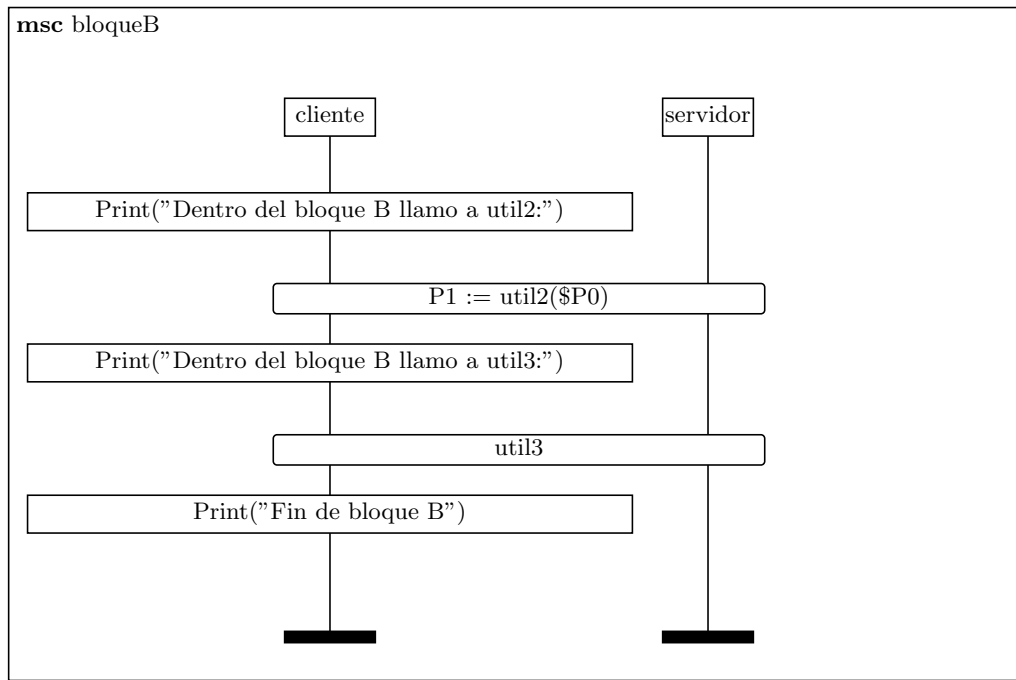


Figura 4.1: Ejemplo A: Documento MSC

Figura 4.2: Representación gráfica de la composición de los bloques en el ejemplo `ejEstructuraA`Figura 4.3: Ejemplo A: Diagrama MSC de `util1`

Figura 4.4: Ejemplo A: Diagrama de *util2*Figura 4.5: Ejemplo A: Diagrama MSC de *util3*Figura 4.6: Ejemplo A: Diagrama MSC de *bloqueA*

Figura 4.7: Ejemplo A: Diagrama MSC de *bloqueB*

```
<documento nombre="ejEstructuraA" autor="Daniel Garcia Jones">
  <definiciones>
    <refdoc nombre="bloqueA"/>
    <refdoc nombre="bloqueB"/>
  </definiciones>
  <utilidades>
    <refdoc nombre="util1" varSalida="pagSal1" parametros="url"/>
    <refdoc nombre="util2" varSalida="pagSal2" parametros="pag"/>
    <refdoc nombre="util3"/>
  </utilidades>
</documento>
```

Figura 4.8: Ejemplo A: Código XML del documento MSC

```
<diagramaMSC nombre="util1">
  <listaInstancias>
    <inst nombre="cliente" tipo="normal"/>
    <inst nombre="servidor" tipo="normal"/>
  </listaInstancias>
  <eventos>
    <mensaje in="servidor" out="cliente" tipo="get" varSalida="pagSal1">$url</mensaje>
    <saltoNivel/>
    <mensaje in="cliente" out="servidor" tipo="reply" varSalida="pagSal1">pagSal1</mensaje>
  </eventos>
</diagramaMSC>
```

Figura 4.9: Ejemplo A: Código XML del diagrama MSC de *util1*

```

<diagramaMSC nombre="util2">
  <listaInstancias>
    <inst nombre="cliente" tipo="normal"/>
    <inst nombre="servidor" tipo="normal"/>
  </listaInstancias>
  <eventos>
    <accion inst="cliente" tipo="var" varSalida="enlace">
      <seleccion tipo="simple">
        <varOrigen>pag</varOrigen>
        <elemBusqueda>a</elemBusqueda>
        <pos>0</pos>
      </seleccion>
    </accion>
    <saltoNivel/>
    <mensaje in="servidor" out="cliente" tipo="get" varSalida="pagSal2">$enlace.href</mensaje>
    <saltoNivel/>
    <mensaje in="cliente" out="servidor" tipo="reply" varSalida="pagSal2">pagSal2</mensaje>
  </eventos>
</diagramaMSC>

```

Figura 4.10: Ejemplo A: Código XML del diagrama MSC de util2

```

<diagramaMSC nombre="util3">
  <listaInstancias>
    <inst nombre="cliente" tipo="normal"/>
    <inst nombre="servidor" tipo="normal"/>
  </listaInstancias>
  <eventos>
    <accion inst="cliente" tipo="print">
      <valor>'util3 solo escribe este texto en pantalla'</valor>
    </accion>
  </eventos>
</diagramaMSC>

```

Figura 4.11: Ejemplo A: Código XML del diagrama MSC de util3

```

<diagramaMSC nombre="bloqueA">
  <listaInstancias>
    <inst nombre="cliente" tipo="normal"/>
    <inst nombre="servidor" tipo="normal"/>
  </listaInstancias>
  <eventos>
    <accion inst="cliente" tipo="print">
      <valor>'Dentro del bloque A llamo a util3:'</valor>
    </accion>
    <saltoNivel/>
    <ref>util3</ref>
    <saltoNivel/>
    <accion inst="cliente" tipo="print">
      <valor>'Dentro del bloque A llamo a util1:'</valor>
    </accion>
    <saltoNivel/>
    <ref varSalida="P0" parametros="$ARGS[1]">util1</ref>
    <saltoNivel/>
    <accion inst="cliente" tipo="print">
      <valor>'Fin de bloqueA'</valor>
    </accion>
  </eventos>
</diagramaMSC>

```

Figura 4.12: Ejemplo A: Código XML del diagrama MSC bloqueA

```

<diagramaMSC nombre="bloqueB">
  <listaInstancias>
    <inst nombre="cliente" tipo="normal"/>
    <inst nombre="servidor" tipo="normal"/>
  </listaInstancias>
  <eventos>
    <accion inst="cliente" tipo="print">
      <valor>'Dentro del bloque B llamo a util2:'</valor>
    </accion>
    <saltoNivel/>
    <ref varSalida="P1" parametros="$P0">util2</ref>
    <saltoNivel/>
    <accion inst="cliente" tipo="print">
      <valor>'Dentro del bloque B llamo a util3:'</valor>
    </accion>
    <saltoNivel/>
    <ref>util3</ref>
    <saltoNivel/>
    <accion inst="cliente" tipo="print">
      <valor>'Fin de bloque B'</valor>
    </accion>
    <saltoNivel/>
  </eventos>
</diagramaMSC>
</msc>

```

Figura 4.13: Ejemplo A: Código XML del diagrama MSC bloqueB



La figura 4.14 incluye el código WebL final del ejemplo A.

```
//
// Modulos importados por defecto
import Files,Forms;
//
// Variables predeterminadas
var home="/home/jones/pfc/msc2webl-1.1/";
var temporal="/home/jones/pfc/msc2webl-1.1/tmp/";
var headers=[. "User-Agent"=Forms_useragent .];
var eliminables = "tr td table center font";

// Programa: estructura
// Autor: Daniel Garcia Jones

// Funcion util1
var util1 = fun (url)
var pagSal1 = Forms_Get(url,nil,headers);
pagSal1 = Forms_xhtmliza(pagSal1,"pagSal1",eliminables,temporal);
end;

// Funcion util2
var util2 = fun (pag)
var enlace = Elem(pag,"a")[0];
var pagSal2 = Forms_Get(enlace.href,nil,headers);
pagSal2 = Forms_xhtmliza(pagSal2,"pagSal2",eliminables,temporal);
end;

// Funcion util3
var util3 = fun ()
PrintLn("util3 solo escribe este texto en pantalla");
end;

//
//Codigo WebL del diagrama bloqueA
PrintLn("Dentro del bloque A llamo a util3:");
util3();
PrintLn("Dentro del bloque A llamo a util1:");
var P0 = util1(ARGS[1]);
PrintLn("Fin de bloqueA");

//
//Codigo WebL del diagrama bloqueB
PrintLn("Dentro del bloque B llamo a util2:");
var P1 = util2(P0);
PrintLn("Dentro del bloque B llamo a util3:");
util3();
PrintLn("Fin de bloque B");
```

Figura 4.14: Código WebL correspondiente al Ejemplo A de estructuras MSC

### 4.1.2. Ejemplo B

Este ejemplo B únicamente consta de un diagrama MSC, llamado **main** (figura 4.15), que implementa la misma funcionalidad que el ejemplo A. La figuras 4.17 y 4.18 incluyen el código XML y Webl asociado respectivamente. Como se puede observar en la figura 4.16, no hay diferencias en el ordenamiento de los elementos en los programas Ejemplo A y Ejemplo B.

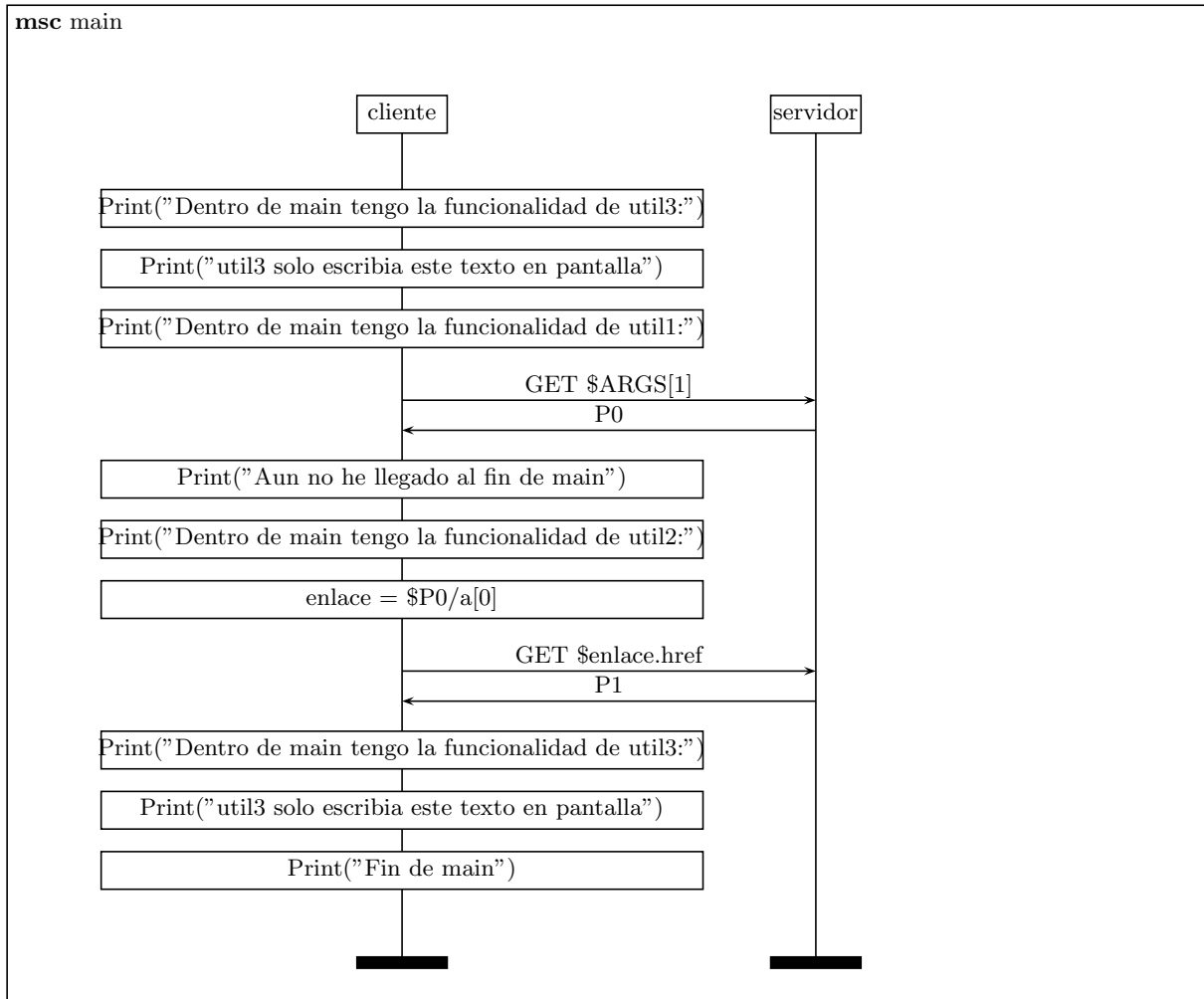


Figura 4.15: Ejemplo B: Diagrama MSC

The image shows two terminal windows side-by-side. The left window, titled 'jones@bajo: /home/jones/msc2webl-1.1/prjs', shows the output of 'ejemplo A'. The right window, titled 'jones@bajo: /home/jones/pfc/msc2webl-1.1/prjs', shows the output of 'ejemplo B'. Both windows display a series of print statements and network requests, demonstrating the execution flow of the programs.

```

jones@bajo: /home/jones/msc2webl-1.1/prjs$
Dentro del bloque A llamo a util3:
util3 solo escribe este texto en pantalla
Dentro del bloque A llamo a util1:
Getting http://www.google.com
Got http://www.google.es/
Fin de bloqueA
Dentro del bloque B llamo a util2:
Getting http://www.google.es/imghp?hl=es?wi=UTF-8
Got http://www.google.es/imghp?hl=es?wi=UTF-8
Dentro del bloque B llamo a util3:
util3 solo escribe este texto en pantalla
Fin de bloque B
jones@bajo: /home/jones/msc2webl-1.1/prjs$

jones@bajo: /home/jones/pfc/msc2webl-1.1/prjs$
Dentro de main tengo la funcionalidad de util3:
util3 solo escribe este texto en pantalla
Dentro de main tengo la funcionalidad de util1:
Getting http://google.com
Got http://www.google.es/
Aun no he llegado al final de main
Dentro de main tengo la funcionalidad de util2:
Getting http://www.google.es/imghp?hl=es?wi=UTF-8
Got http://www.google.es/imghp?hl=es?wi=UTF-8
Dentro de main tengo la funcionalidad de util3:
util3 solo escribe este texto en pantalla
Fin de main
jones@bajo: /home/jones/pfc/msc2webl-1.1/prjs$
  
```

Figura 4.16: Salidas de los programas ejemplo A y ejemplo B

```

<diagramaMSC nombre="main">
  <listaInstancias>
    <inst nombre="cliente" tipo="normal"/>
    <inst nombre="servidor" tipo="normal"/>
  </listaInstancias>
  <eventos>
    <accion inst="cliente" tipo="print">
      <valor>'Dentro de main tengo la funcionalidad de util3:'</valor>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="print">
      <valor>'util3 solo escribia este texto en pantalla'</valor>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="print">
      <valor>'Dentro de main tengo la funcionalidad de util1:'</valor>
    </accion>
    <saltoNivel/>
    <mensaje in="servidor" out="cliente" tipo="get" varSalida="P0">$ARGS[1]</mensaje>
    <saltoNivel/>
    <mensaje in="cliente" out="servidor" tipo="reply" varSalida="P0">P0</mensaje>
    <saltoNivel/>
    <accion inst="cliente" tipo="print">
      <valor>'Aun no he llegado al fin de main'</valor>
    </accion>
    <accion inst="cliente" tipo="print">
      <valor>'Dentro de main tengo la funcionalidad de util2:'</valor>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="var" varSalida="enlace">
      <seleccion tipo="simple">
        <varOrigen>$P0</varOrigen>
        <elemBusqueda>a</elemBusqueda>
        <pos>0</pos>
      </seleccion>
    </accion>
    <saltoNivel/>
    <mensaje in="servidor" out="cliente" tipo="get" varSalida="P1">$enlace.href</mensaje>
    <saltoNivel/>
    <mensaje in="cliente" out="servidor" tipo="reply" varSalida="P1">P1</mensaje>
    <saltoNivel/>
    <accion inst="cliente" tipo="print">
      <valor>'Dentro de main tengo la funcionalidad de util3:'</valor>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="print">
      <valor>'util3 solo escribia este texto en pantalla'</valor>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="print">
      <valor>'Fin de main'</valor>
    </accion>
    <saltoNivel/>
  </eventos>
</diagramaMSC>

```

Figura 4.17: Ejemplo B: Código XML correspondiente

```
// Programa: ejEstructura
// Autor: Daniel Garcia Jones

//
// Modulos importados por defecto
import Forms,Files,Str,Url;

//
// Variables predeterminadas
var home="/home/jones/";
var temporal= home + "pfc/webl/tmp/";
var headers=[. "User-Agent"=Forms_useragent .];
var eliminables = "";

//
//Codigo WebL del diagrama main
PrintLn("Dentro de main tengo la funcionalidad de util3:");
PrintLn("util3 solo escribia este texto en pantalla");
PrintLn("Dentro de main tengo la funcionalidad de util1:");
var P0 = Forms_Get(ARGS[1],nil,headers);
P0 = Forms_xhtmliza(P0,"P0",eliminables,temporal);
PrintLn("Aun no he llegado al fin de main");
PrintLn("Dentro de main tengo la funcionalidad de util2:");
var enlace = Elem(P0,"a")[0];
var P1 = Forms_Get(enlace.href,nil,headers);
P1 = Forms_xhtmliza(P1,"P1",eliminables,temporal);
PrintLn("Dentro de main tengo la funcionalidad de util3:");
PrintLn("util3 solo escribia este texto en pantalla");
PrintLn("Fin de main");
```

Figura 4.18: Ejemplo B: Código WebL correspondiente

Una vez entendido cómo pueden descomponerse estructuralmente los sistemas empleando documentos y diagramas MSC, para el resto de los ejemplos, se omitirá generalmente la inclusión de los documentos MSC (tanto en su representación gráfica como en la XML), a no ser que el ejemplo en cuestión tenga una estructura compleja que se vea aclarada al incluir los documentos MSC.

## 4.2. Descarga de tira semanal de cómic

El grupo de noticias de Linux es.comp.os.linux[48], en su sitio Web publica semanalmente una tira cómica[49] relacionada con el mundo de Linux (figura 4.19). Se incluye un script que descarga a disco esta tira cómica, dejándola en el directorio que se indique como argumento.



Figura 4.19: Tira Ecol

La página que se quiere navegar es muy sencilla. Explorando el código HTML, se observa que la imagen correspondiente a la tira gráfica es la segunda de las que aparecen en la página. Consultando el atributo `src` de la imagen dada, se obtiene la URL de la tira cómica. La figura 4.20 representa el diagrama MSC de este programa. El código WebL correspondiente aparece en la figura 4.22.

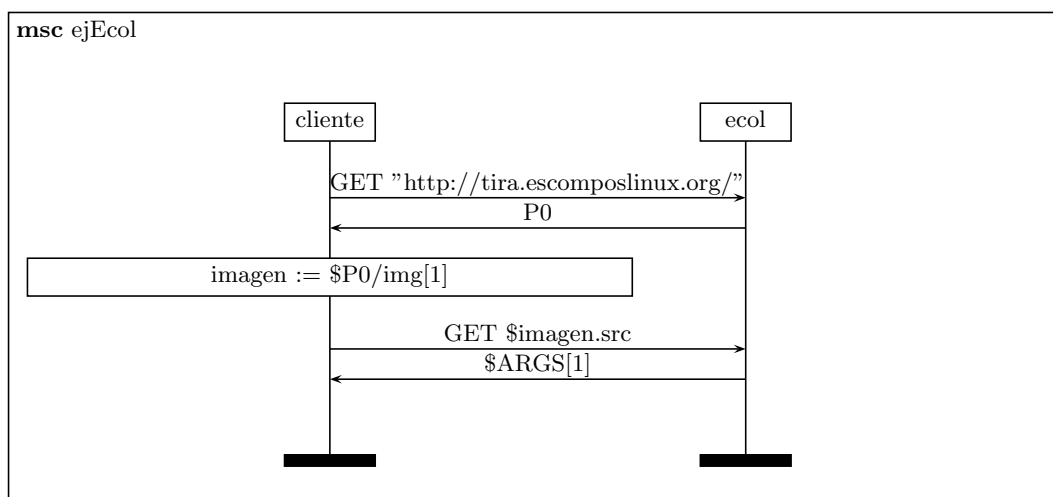


Figura 4.20: Ejemplo tira Ecol: Diagrama MSC

## 4.3. Consulta en Google

Se ha desarrollado un script para consulta automática de Google. El patrón de búsqueda se pasa como argumento de ejecución y el script devuelve por pantalla la lista de los enlaces en la primera página de resultados, cada uno con su descripción.

Este ejemplo incluye el procesamiento de un formulario para realizar la búsqueda. Una vez seleccionado el formulario, el primero de los que aparecen (ver figura 4.23), hay que introducir el patrón de búsqueda, pasado en este caso como argumento (`ARGS[1]`), y borrar el input cuyo nombre es `btnI`. Al generar el

```

<diagramaMSC nombre="ejEcol" autor="Daniel Garcia Jones">
  <listaInstancias>
    <inst nombre="cliente" tipo="normal"/>
    <inst nombre="ecol" tipo="normal"/>
  </listaInstancias>
  <eventos>
    <mensaje in="ecol" out="cliente" tipo="get" varSalida="P0">
      'http://tira.escomposlinux.org/'
    </mensaje>
    <saltoNivel/>
    <mensaje in="cliente" out="ecol" tipo="reply" varSalida="P0">P0</mensaje>
    <saltoNivel/>
    <accion inst="cliente" tipo="var" varSalida="imagen">
      <seleccion tipo="simple">
        <varOrigen>$P0</varOrigen>
        <elemBusqueda>img</elemBusqueda>
        <pos>1</pos>
      </seleccion>
    </accion>
    <saltoNivel/>
    <mensaje in="ecol" out="cliente" tipo="getFile" varSalida="$ARGS[1]">$imagen.src</mensaje>
    <saltoNivel/>
    <mensaje in="cliente" out="ecol" tipo="replyFile" varSalida="$ARGS[1]">$ARGS[1]</mensaje>
  </eventos>
</diagramaMSC>

```

Figura 4.21: Ejemplo Tira Ecol: Código XML

```

//
// Modulos importados por defecto
import Files,Forms;
//
// Variables predeterminadas
var home="/home/jones/pfc/msc2webl-1.1/";
var temporal="/home/jones/pfc/msc2webl-1.1/tmp/";
var headers=[. "User-Agent"=Forms_useragent .];
var eliminables = "tr td table center font";

// Programa: ejEcol
// Autor: Daniel Garcia Jones
//
// Codigo Webl del diagrama main
var P0 = Forms_Get("http://tira.escomposlinux.org/",nil,headers);
P0 = Forms_xhtmliza(P0,"P0",eliminables,temporal);
var imagen = Elem(P0,"img")[1];
Files_GetURL(imagen.src,ARGS[1]);

```

Figura 4.22: Ejemplo Tira Ecol: Código Webl

programa, nos dimos cuenta de que para que funcionara era necesario asignar el método del formulario que se iba a enviar posteriormente. Esto es una peculiaridad de este sitio Web, ya que en general, si un formulario requiere un método concreto, este aparece definido de forma explícita. Una vez enviado el formulario, se recibe la página de resultados, P1.

Si observamos el código HTML de una página de resultados de búsqueda (figura 4.24), notamos que cada uno de los resultados se incluyen dentro de un párrafo de clase "g" (<p class="g"> ... </p>). De esta forma, podremos seleccionarlos todos ellos e imprimir el contenido textual para obtener los detalles de cada uno de los resultados.

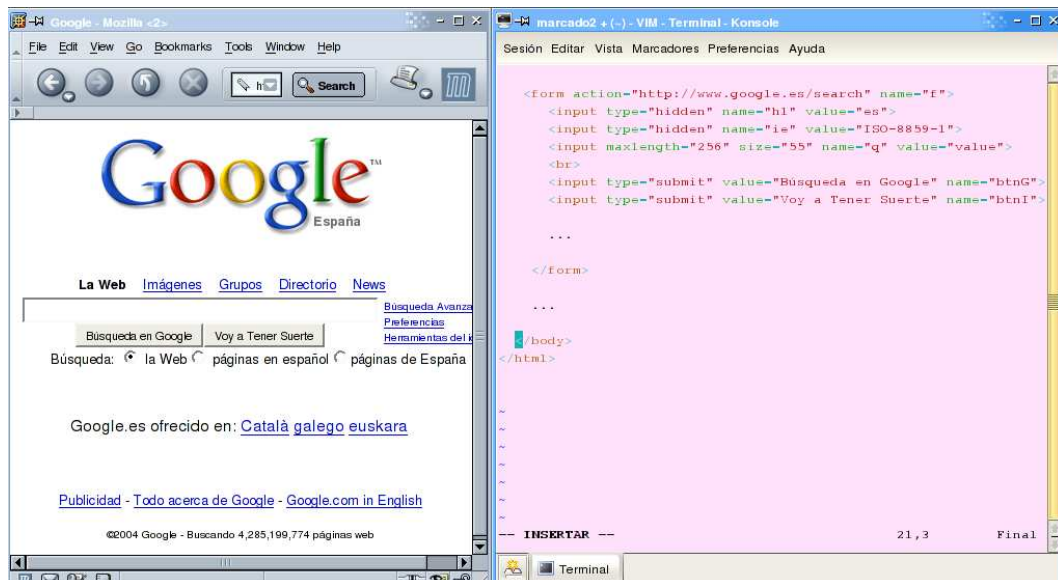


Figura 4.23: Marcado de la página principal de Google.es

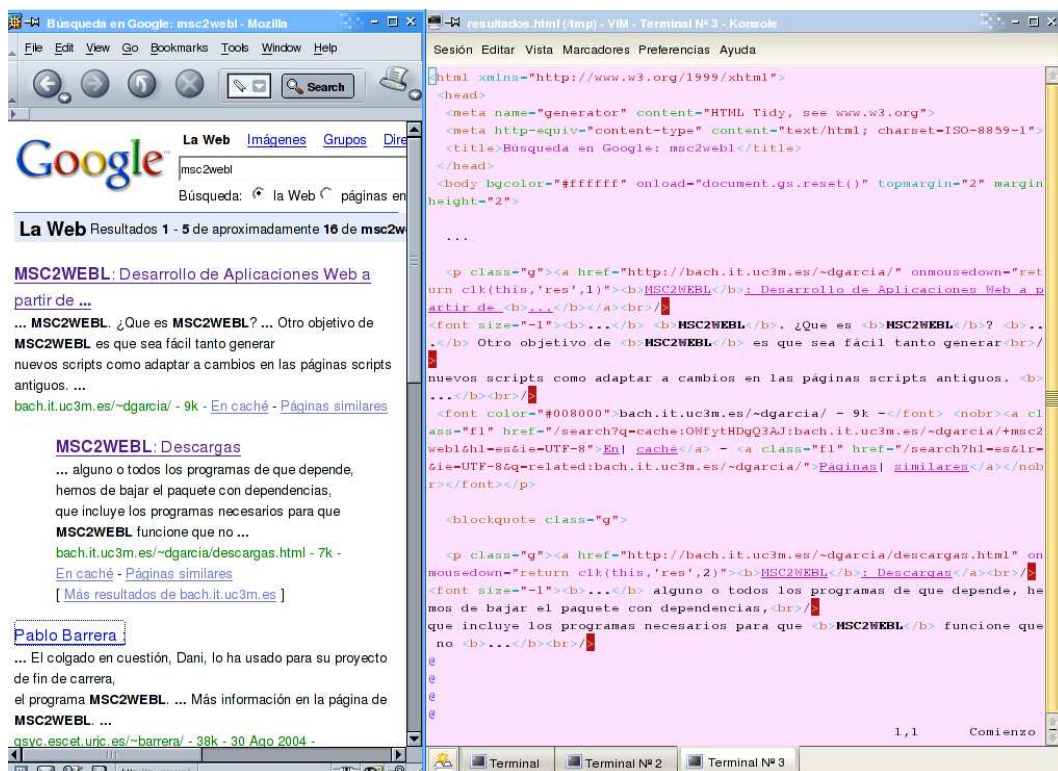


Figura 4.24: Marcado de una página de resultados de búsqueda en Google

La figura 4.25 incluye el diagrama MSC que representa al programa y la figura 4.26 su código XML. La figura 4.27 incluye el código WebL correspondiente. La figura 4.28 muestra el resultado de la ejecución del programa.

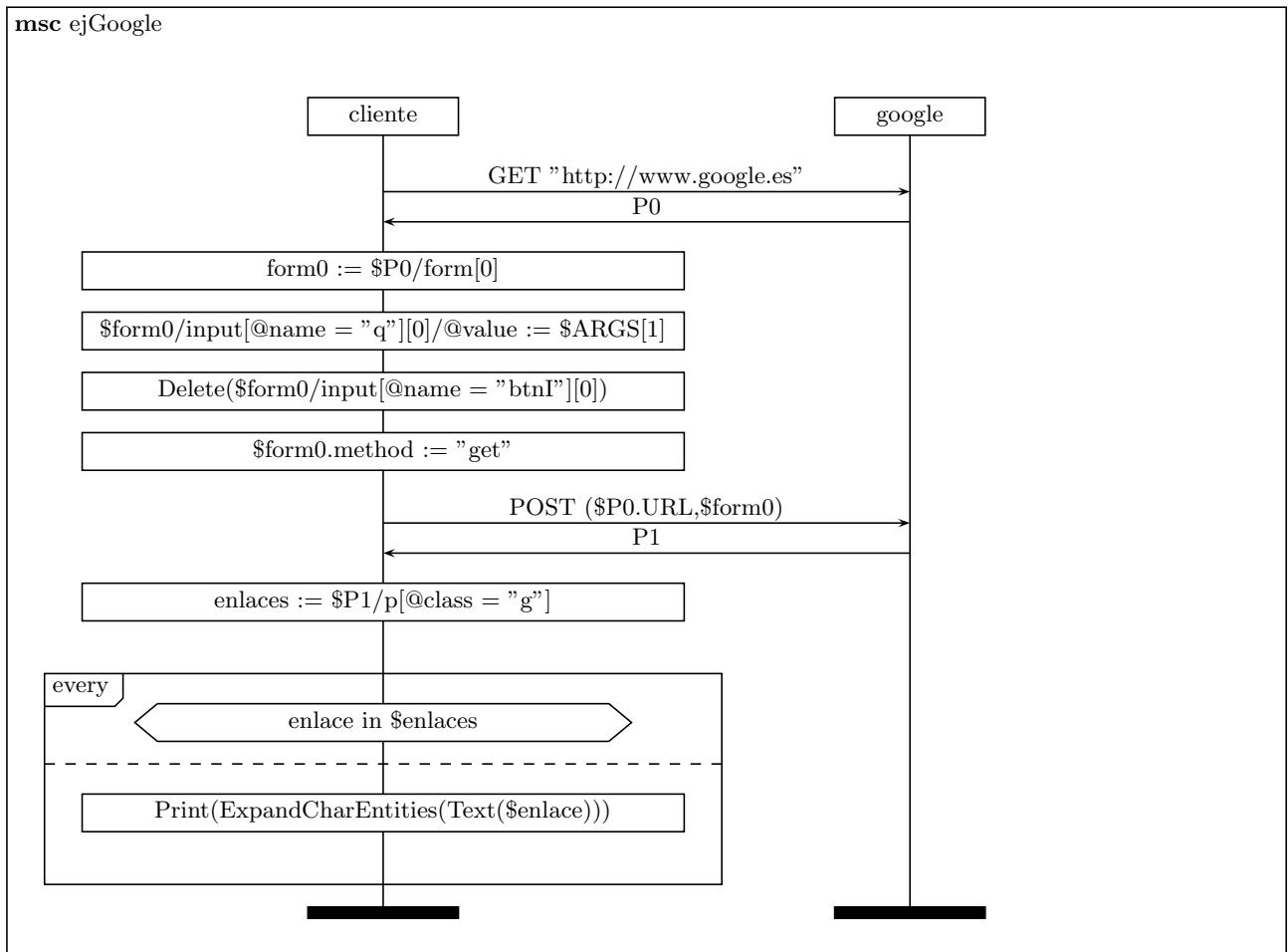


Figura 4.25: Ejemplo Google: Diagrama MSC



```

<diagramaMSC nombre="ejGoogle">
  <listaInstancias>
    <inst nombre="cliente"/>
    <inst nombre="google"/>
  </listaInstancias>
  <eventos>
    <mensaje in="google" out="cliente" tipo="get" varSalida="P0">
      'http://www.google.es'
    </mensaje>
    <saltoNivel/>
    <mensaje in="cliente" out="google" tipo="reply" varSalida="P0">P0</mensaje>
    <saltoNivel/>
    <accion inst="cliente" tipo="form" varSalida="form0">
      <seleccion tipo="simple">
        <varOrigen>$P0</varOrigen>
        <elemBusqueda>form</elemBusqueda>
        <pos>0</pos>
      </seleccion>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="asignacion">
      <seleccion tipo="complejo">
        <varOrigen>$form0</varOrigen>
        <elemBusqueda>input</elemBusqueda>
        <att-busqueda>name</att-busqueda>
        <val-busqueda>'q'</val-busqueda>
        <pos>0</pos>
      </seleccion>
      <attsel>value</attsel>
      <valor>$ARGS[1]</valor>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="borrado">
      <seleccion tipo="complejo">
        <varOrigen>$form0</varOrigen>
        <elemBusqueda>input</elemBusqueda>
        <att-busqueda>name</att-busqueda>
        <val-busqueda>'btnI'</val-busqueda>
        <pos>0</pos>
      </seleccion>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="asignacion">
      <valor>$form0</valor>
      <attsel>method</attsel>
      <valor>'get'</valor>
    </accion>
    <saltoNivel/>
    <mensaje in="google" out="cliente" tipo="post" varSalida="P1" form="form0">
      $P0.URL
    </mensaje>
    <saltoNivel/>
    <mensaje in="cliente" out="google" tipo="reply" varSalida="P1">P1</mensaje>
    <saltoNivel/>
    <accion inst="cliente" tipo="var" varSalida="enlaces">
      <seleccion tipo="complejo">
        <varOrigen>$P1</varOrigen>
        <elemBusqueda>p</elemBusqueda>
        <att-busqueda>class</att-busqueda>
        <val-busqueda>'g'</val-busqueda>
      </seleccion>
    </accion>
  </eventos>
</diagramaMSC>

```

```

<saltoNivel/>
<every inst="cliente">
  <test>
    <id>enlace</id>
    <conjunto>$enlaces</conjunto>
  </test>
  <argumento>
    <accion inst="cliente" tipo="print">
      <valor>ExpandCharEntities(Text($enlace)) + '\n'</valor>
    </accion>
  </argumento>
</every>
</eventos>
</diagramaMSC>
</msc>

```

Figura 4.26: Ejemplo Google: Código XML del diagrama MSC

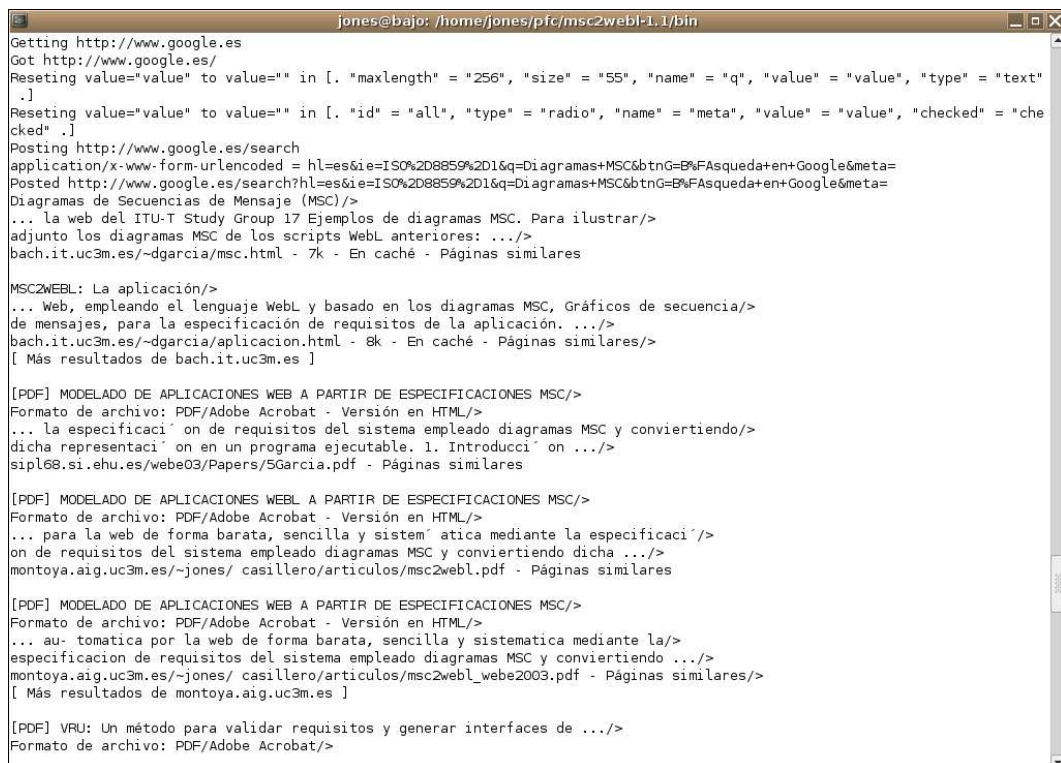
```

//
// Modulos importados por defecto
import Files,Forms;
//
// Variables predeterminadas
var home="/home/jones/pfc/msc2webl-1.1/";
var temporal="/home/jones/pfc/msc2webl-1.1/tmp/";
var headers=[. "User-Agent"=Forms_useragent .];
var eliminables = "tr td table center font";

// Programa: ejGoogle
//
// Codigo Webl del diagrama main
var P0 = Forms_Get("http://www.google.es",nil,headers);
P0 = Forms_xhtmliza(P0,"P0",eliminables,temporal);
var form0 = Forms_Normaliza(Elem(P0,"form")[0],true);
Select(Elem(form0,"input"), fun(a) (a.name == "q") ? false end)[0].value := ARGS[1];
Delete(Select(Elem(form0,"input"), fun(a) (a.name == "btnI") ? false end)[0]);
form0.method := "get";
var P1 = Forms_Post(form0,P0.URL , [. mimetype="text/plain" .], true);
P1 = Forms_xhtmliza(P1,"P1",eliminables,temporal);
var enlaces = Select(Elem(P1,"p"), fun(a) (a.class == "g") ? false end);
every enlace in enlaces do
  PrintLn(ExpandCharEntities(Text(enlace)) + "\n");
end;

```

Figura 4.27: Ejemplo Google: Código Webl correspondiente



```

jones@bajo: /home/jones/pfc/msc2webl-1.1/bin
Getting http://www.google.es
Got http://www.google.es/
Reseting value="value" to value="" in [. "maxlength" = "256", "size" = "55", "name" = "q", "value" = "value", "type" = "text"
.]
Reseting value="value" to value="" in [. "id" = "all", "type" = "radio", "name" = "meta", "value" = "value", "checked" = "che
cked" .]
Posting http://www.google.es/search
application/x-www-form-urlencoded = hl=es&ie=ISO%2D8859%2D1&q=Diagramas+MSC&btnG=B%FAsqueda+en+Google&meta=
Posted http://www.google.es/search?hl=es&ie=ISO%2D8859%2D1&q=Diagramas+MSC&btnG=B%FAsqueda+en+Google&meta=
Diagramas de Secuencias de Mensaje (MSC)/>
... la web del ITU-T Study Group 17 Ejemplos de diagramas MSC. Para ilustrar/>
adjunto los diagramas MSC de los scripts Webl anteriores: .../>
bach.it.uc3m.es/~dgarcia/msc.html - 7k - En caché - Páginas similares

MSC2WEBL: La aplicación/>
... Web, empleando el lenguaje Webl y basado en los diagramas MSC, Gráficos de secuencia/>
de mensajes, para la especificación de requisitos de la aplicación. .../>
bach.it.uc3m.es/~dgarcia/aplicacion.html - 8k - En caché - Páginas similares/>
[ Más resultados de bach.it.uc3m.es ]

[PDF] MODELADO DE APLICACIONES WEB A PARTIR DE ESPECIFICACIONES MSC/>
Formato de archivo: PDF/Adobe Acrobat - Versión en HTML/>
... la especificación de requisitos del sistema empleado diagramas MSC y convirtiendo/>
dicha representación en un programa ejecutable. 1. Introducci on .../>
sipl68.si.ehu.es/webe03/Papers/5Garcia.pdf - Páginas similares

[PDF] MODELADO DE APLICACIONES WEB A PARTIR DE ESPECIFICACIONES MSC/>
Formato de archivo: PDF/Adobe Acrobat - Versión en HTML/>
... para la web de forma barata, sencilla y sistemática mediante la especificaci on/>
on de requisitos del sistema empleado diagramas MSC y convirtiendo dicha .../>
montoya.aig.uc3m.es/~jones/ casillero/articulos/msc2webl.pdf - Páginas similares

[PDF] MODELADO DE APLICACIONES WEB A PARTIR DE ESPECIFICACIONES MSC/>
Formato de archivo: PDF/Adobe Acrobat - Versión en HTML/>
... au- tomática por la web de forma barata, sencilla y sistemática mediante la/>
especificación de requisitos del sistema empleado diagramas MSC y convirtiendo .../>
montoya.aig.uc3m.es/~jones/ casillero/articulos/msc2webl_webe2003.pdf - Páginas similares/>
[ Más resultados de montoya.aig.uc3m.es ]

[PDF] VRU: Un método para validar requisitos y generar interfaces de .../>
Formato de archivo: PDF/Adobe Acrobat/>

```

Figura 4.28: Ejemplo Google: Resultado de la ejecución

## 4.4. Cabeceras de Barrapunto

Barrapunto.com[25] es un sitio Web de noticias sobre informática y ciencia en general. La información que contiene es enviada por distintos usuarios en forma de noticias, a las que se pueden añadir comentarios. Se propone la creación de un sencillo script que descargue las cabeceras de las noticias que aparecen en la portada de Barrapunto. Este sitio Web, syndica su información empleando RSS (ver capítulos 2 y 3) lo que permite de forma sencilla acceder a las cabeceras de las noticias. La página principal contiene un enlace, cuyo texto es "RSS", que apunta al fichero XML con la información deseada. Una vez obtenido dicho fichero, si observamos su código (figura 4.29), descubrimos lo sencillo que es acceder la información deseada.

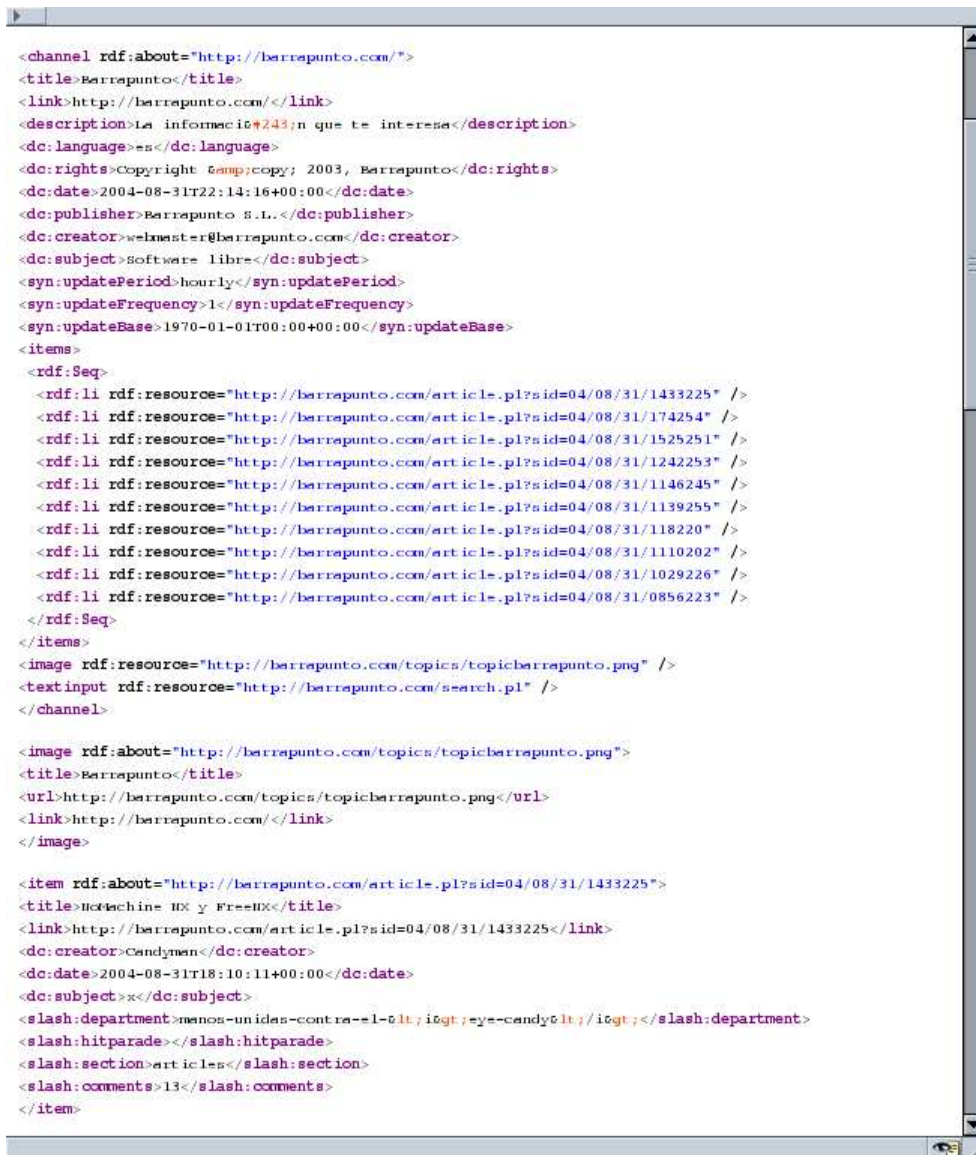


Figura 4.29: Fichero RSS de Barrapunto.com

Para describir la tarea, definimos dos diagramas MSC, `descargaRSS` que se encarga de descargar el fichero RSS, e `imprimeCabeceras` que muestra la información deseada. Las figuras 4.30 y 4.31 representan los diagramas MSC que describen el programa. Las figuras 4.32 y 4.33 contienen el código XML correspondiente a los anteriores diagramas. El código WebL correspondiente a este ejemplo aparece en la figura 4.34. El resultado de la ejecución del script se representa en la figura 4.35.

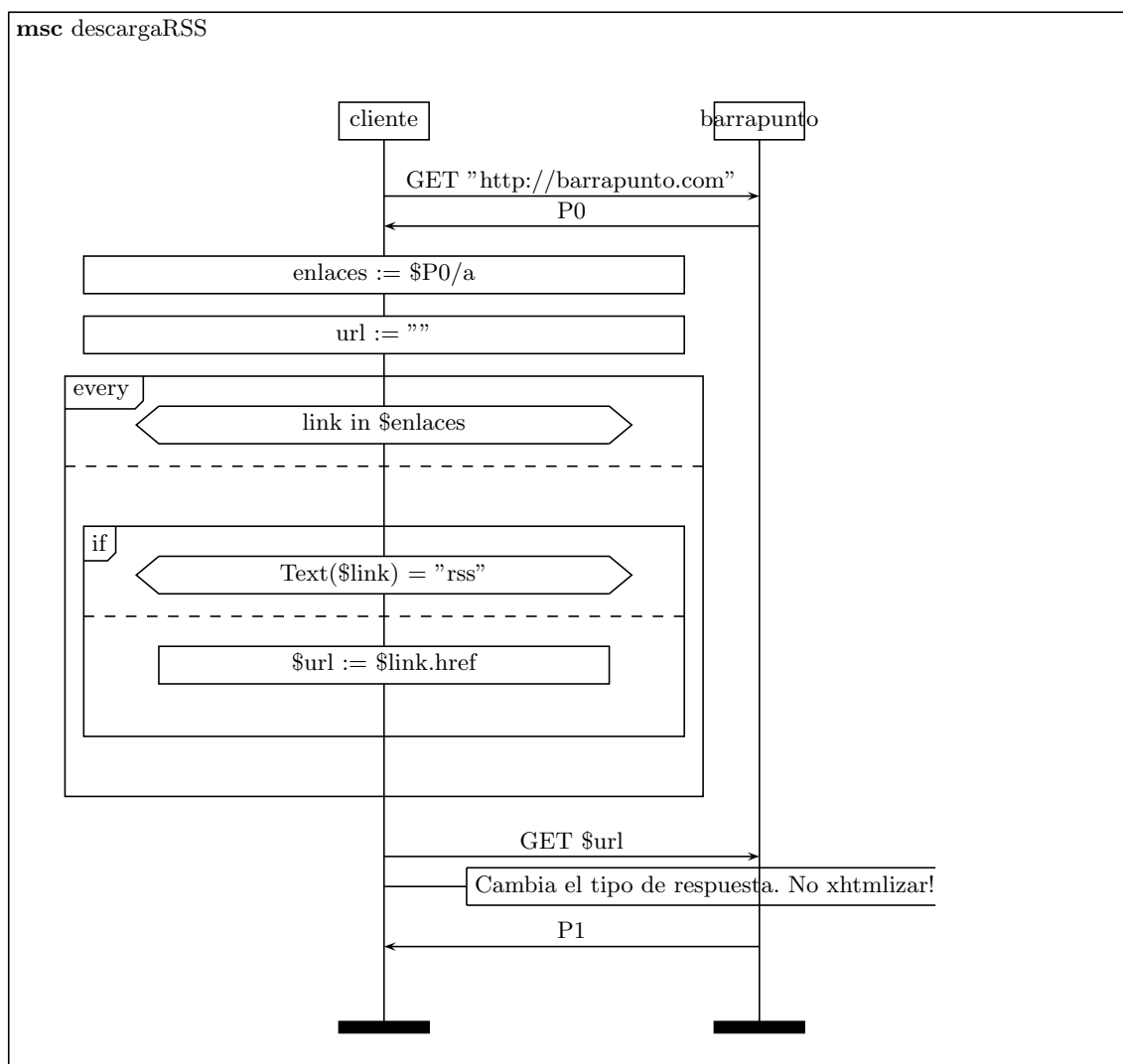


Figura 4.30: Ejemplo BarraPunto: diagrama descargaRSS

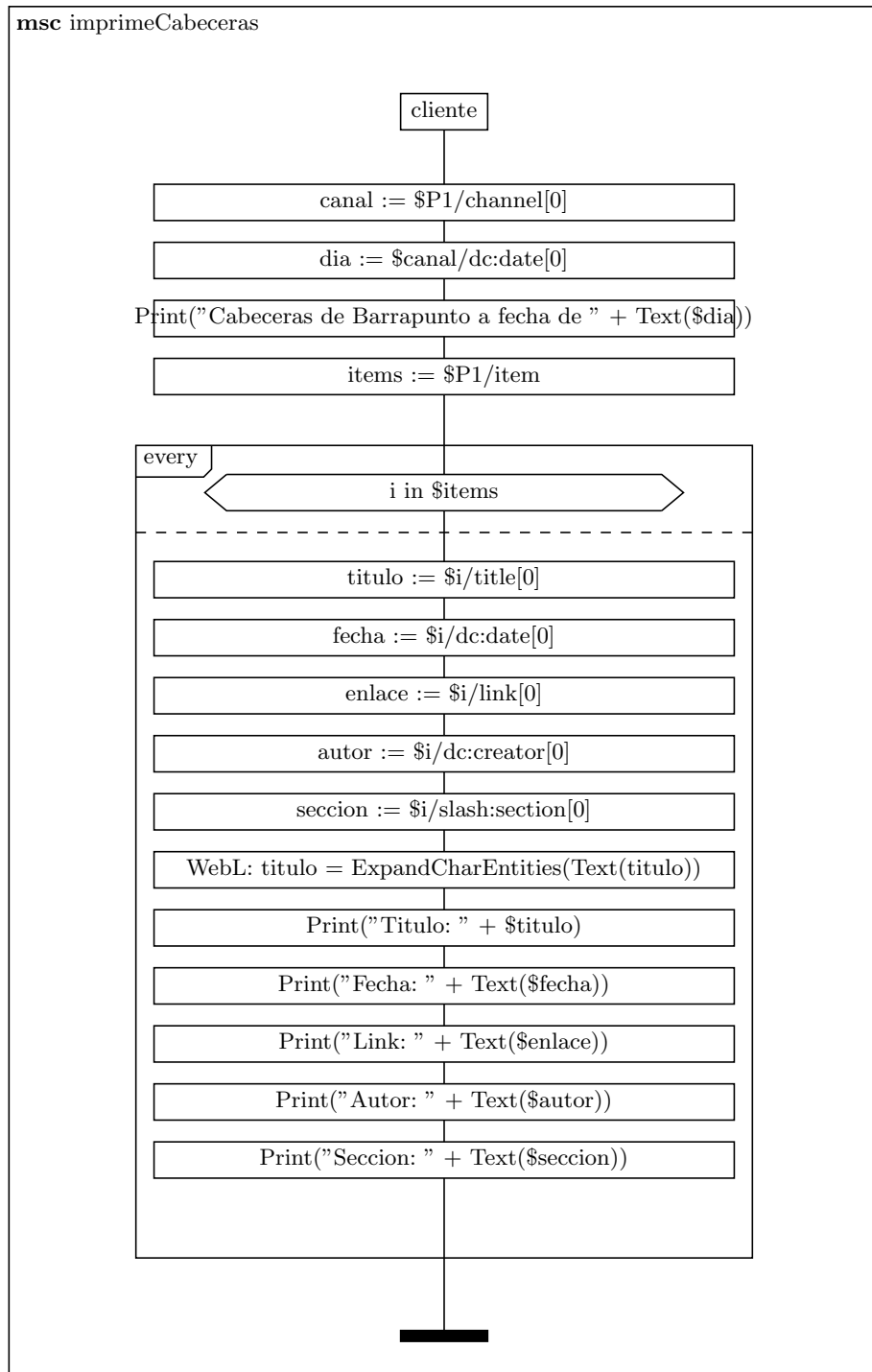


Figura 4.31: Ejemplo Barrapunto: diagrama imprimir

```

<diagramaMSC nombre="descargaRSS">
  <listaInstancias>
    <inst nombre="cliente" tipo="normal"/>
    <inst nombre="barrapunto" tipo="normal"/>
  </listaInstancias>
  <eventos>
    <mensaje in="barrapunto" out="cliente" tipo="get" varSalida="P0">
      'http://barrapunto.com'
    </mensaje>
    <saltoNivel/>
    <mensaje in="cliente" out="barrapunto" tipo="reply" varSalida="P0">P0</mensaje>
    <saltoNivel/>
    <accion inst="cliente" tipo="var" varSalida="enlaces">
      <seleccion tipo="simple">
        <varOrigen>$P0</varOrigen>
        <elemBusqueda>a</elemBusqueda>
      </seleccion>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="var" varSalida="url">
      <valor>'</valor>
    </accion>
    <saltoNivel/>
    <every inst="cliente">
      <test>
        <id>link</id>
        <conjunto>$enlaces</conjunto>
      </test>
      <argumento>
        <if inst="cliente">
          <test>Text($link) = 'rss'</test>
          <argumento>
            <accion inst="cliente" tipo="asignacion">
              <valor>$url</valor>
              <valor>$link.href</valor>
            </accion>
          </argumento>
        </if>
      </argumento>
    </every>
    <saltoNivel/>
    <mensaje in="barrapunto" out="cliente" tipo="get" varSalida="P1">$url</mensaje>
    <saltoNivel/>
    <comentario inst="cliente">Cambio manual del tipo de respuesta. No xhtmlizar!</comentario>
    <saltoNivel/>
    <mensaje in="cliente" out="barrapunto" tipo="replyFile" varSalida="P1">P1</mensaje>
  </eventos>
</diagramaMSC>

```

Figura 4.32: Código XML del diagrama descargaRSS

```

<diagramaMSC nombre="imprimeCabeceras">
  <listaInstancias>
    <inst nombre="cliente" tipo="normal"/>
  </listaInstancias>
  <eventos>
    <accion inst="cliente" tipo="var" varSalida="canal">
      <seleccion tipo="simple">
        <varOrigen>$P1</varOrigen>
        <elemBusqueda>channel</elemBusqueda>
        <pos>0</pos>
      </seleccion>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="var" varSalida="dia">
      <seleccion tipo="simple">
        <varOrigen>$canal</varOrigen>
        <elemBusqueda>dc:date</elemBusqueda>
        <pos>0</pos>
      </seleccion>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="print">
      <valor>'Las Cabeceras de Barrapunto a fecha de ' + Text($dia) </valor>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="var" varSalida="items">
      <seleccion tipo="simple">
        <varOrigen>$P1</varOrigen>
        <elemBusqueda>item</elemBusqueda>
      </seleccion>
    </accion>
    <saltoNivel/>
    <every inst="cliente">
      <test>
        <id>i</id>
        <conjunto>$items</conjunto>
      </test>
      <argumento>
        <accion inst="cliente" tipo="var" varSalida="titulo">
          <seleccion tipo="simple">
            <varOrigen>$i</varOrigen>
            <elemBusqueda>title</elemBusqueda>
            <pos>0</pos>
          </seleccion>
        </accion>
        <saltoNivel/>
        <accion inst="cliente" tipo="var" varSalida="fecha">
          <seleccion tipo="simple">
            <varOrigen>$i</varOrigen>
            <elemBusqueda>dc:date</elemBusqueda>
            <pos>0</pos>
          </seleccion>
        </accion>
        <saltoNivel/>
        <accion inst="cliente" tipo="var" varSalida="enlace">
          <seleccion tipo="simple">
            <varOrigen>$i</varOrigen>
            <elemBusqueda>link</elemBusqueda>
            <pos>0</pos>
          </seleccion>
        </accion>
        <saltoNivel/>

```



```

<accion inst="cliente" tipo="var" varSalida="autor">
  <seleccion tipo="simple">
    <varOrigen>$i</varOrigen>
    <elemBusqueda>dc:creator</elemBusqueda>
    <pos>0</pos>
  </seleccion>
</accion>
<saltoNivel/>
<accion inst="cliente" tipo="var" varSalida="seccion">
  <seleccion tipo="simple">
    <varOrigen>$i</varOrigen>
    <elemBusqueda>slash:section</elemBusqueda>
    <pos>0</pos>
  </seleccion>
</accion>
<saltoNivel/>
<accion inst="cliente" tipo="webl">
  <valor>titulo = ExpandCharEntities(Text(titulo))</valor>
</accion>
<saltoNivel/>
<accion inst="cliente" tipo="print">
  <valor>'Titulo: ' + $titulo</valor>
</accion>
<saltoNivel/>
<accion inst="cliente" tipo="print">
  <valor>'Fecha: ' + Text($fecha)</valor>
</accion>
<saltoNivel/>
<accion inst="cliente" tipo="print">
  <valor>'Link: ' + Text($enlace)</valor>
</accion>
<saltoNivel/>
<accion inst="cliente" tipo="print">
  <valor>'Autor: ' + Text($autor)</valor>
</accion>
<saltoNivel/>
<accion inst="cliente" tipo="print">
  <valor>'Seccion: ' + Text($seccion) </valor>
</accion>
<saltoNivel/>
</argumento>
</every>
</eventos>
</diagramaMSC>

```

Figura 4.33: Código XML del diagrama imprimir

```

//
// Modulos importados por defecto
import Files,Forms;
//
// Variables predeterminadas
var home="/home/jones/pfc/msc2webl-1.1/";
var temporal="/home/jones/pfc/msc2webl-1.1/tmp/";
var headers=[. "User-Agent"=Forms_useragent .];
var eliminables = "tr td table center font";

// Programa: barraPunto
// Autor: Daniel Garcia Jones
//
// Codigo WebL del diagrama descargaRSS
var P0 = Forms_Get("http://barrapunto.com",nil,headers);
P0 = Forms_xhtmliza(P0,"P0",eliminables,temporal);
var enlaces = Elem(P0,"a");
var url = "";
every link in enlaces do
if (Text(link) == "rss") then
url = link.href;
end;
end;
var P1 = Forms_Get(url,nil,headers);
// Cambio manual del tipo de respuesta (no xhtmlizar)

//
// Codigo WebL del diagrama imprimeCabeceras
var canal = Elem(P1,"channel")[0];
var dia = Elem(canal,"dc:date")[0];
PrintLn("Las Cabeceras de Barrapunto a fecha de " + Text(dia) );
var items = Elem(P1,"item");
every i in items do
var titulo = Elem(i,"title")[0];
var fecha = Elem(i,"dc:date")[0];
var enlace = Elem(i,"link")[0];
var autor = Elem(i,"dc:creator")[0];
var seccion = Elem(i,"slash:section")[0];
titulo = ExpandCharEntities(Text(titulo));
PrintLn("Titulo: " + titulo);
PrintLn("Fecha: " + Text(fecha));
PrintLn("Link: " + Text(enlace));
PrintLn("Autor: " + Text(autor));
PrintLn("Seccion: " + Text(seccion) );
end;

```

Figura 4.34: Ejemplo barraPunto: código WebL



```
jones@bajo: /home/jones/pfc/msc2webl-1.1/prjs
Getting http://barrapunto.com
Got http://barrapunto.com/
Getting http://barrapunto.com/barrapunto.rss
Got http://backends.barrapunto.com/barrapunto.rss
Las Cabeceras de Barrapunto a fecha de 2004-12-02T16:13:03+00:00
Titulo: Google por dentro
Fecha: 2004-12-02T14:18:21+00:00
Link: http://barrapunto.com/article.pl?sid=04/12/02/1418226
Autor: rvr
Seccion: articles
Titulo: Microsoft presenta MSN Spaces
Fecha: 2004-12-02T13:59:27+00:00
Link: http://barrapunto.com/article.pl?sid=04/12/02/1415241
Autor: rvr
Seccion: articles
Titulo: HispaLinux en las III Jornadas Ceres
Fecha: 2004-12-02T13:54:09+00:00
Link: http://barrapunto.com/article.pl?sid=04/12/02/1356247
Autor: rvr
Seccion: eventos
Titulo: Perspectiva tecnológica para el año 2014
Fecha: 2004-12-02T13:47:34+00:00
Link: http://barrapunto.com/article.pl?sid=04/12/02/1353212
Autor: rvr
Seccion: articles
Titulo: El Gobierno reitera su negativa a las patentes de software
Fecha: 2004-12-02T13:36:19+00:00
Link: http://barrapunto.com/article.pl?sid=04/12/02/1340225
Autor: rvr
Seccion: ciberderechos
Titulo: Inkscape 0.40
Fecha: 2004-12-02T13:30:17+00:00
Link: http://barrapunto.com/article.pl?sid=04/12/02/1334219
Autor: rvr
Seccion: softlibre
Titulo: Ataque a la web de la campaña anti-spam de Lycos
Fecha: 2004-12-02T12:17:02+00:00
Link: http://barrapunto.com/article.pl?sid=04/12/02/1236243
Autor: alo
Seccion: articles
Titulo: 'Blog', la palabra del año, según Merriam-Webster
```

Figura 4.35: Salida del programa de descarga de cabeceras de Barrapunto

## 4.5. Ejemplo de Threads

Con el fin de ilustrar el uso de threads de ejecución en la aplicación y explicar sus diferencias con los combinadores de servicio se ha incluido un sencillo script que primero accede a las páginas de Google.es y Google.com mediante un combinador de servicio concurrente. Después hace lo mismo con threads. La figura 4.36 incluye el diagrama MSC correspondiente a este ejemplo. La figura 4.38 incluye el código XML del mismo.

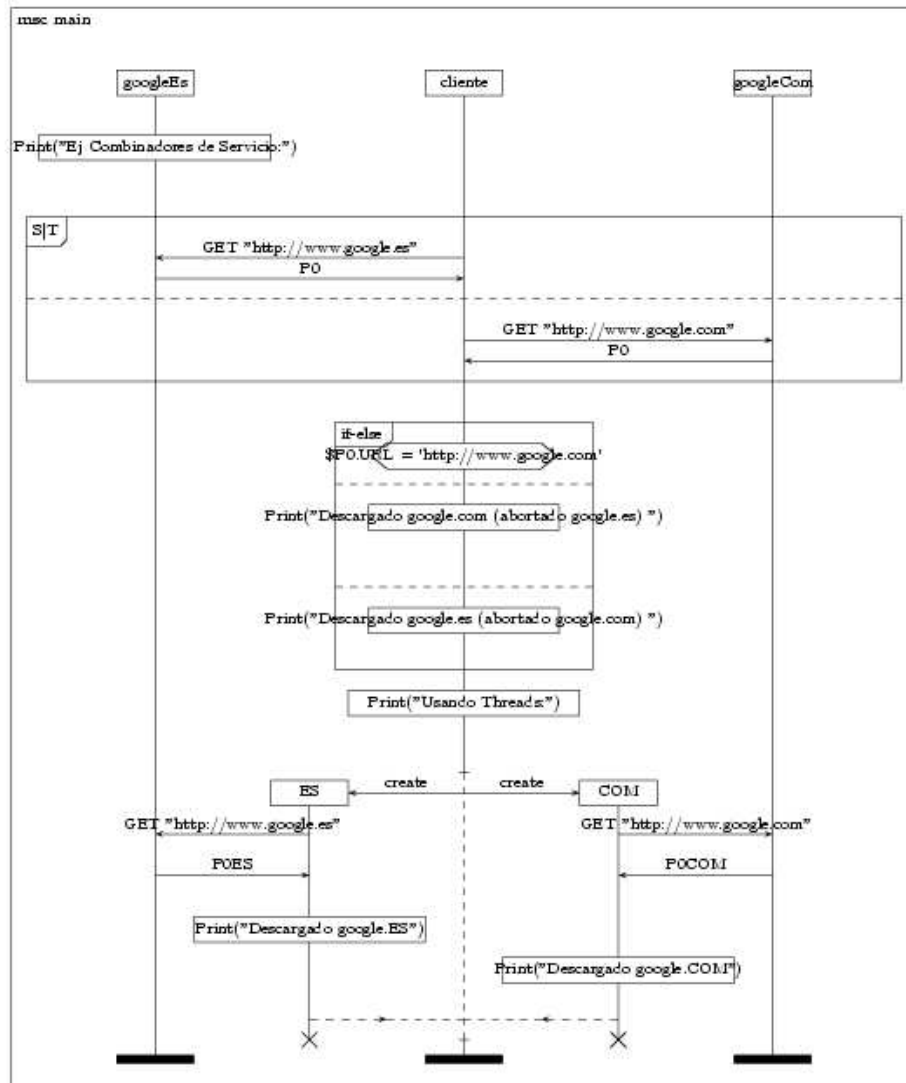


Figura 4.36: Diagrama MSC del ejemplo con Threads

Al ejecutar este script vemos (figura 4.37) que en primer lugar, cuando se emplea el combinador de servicio, únicamente se descarga la página que tarde menos, abortando la obtención de la otra. Por el contrario, empleando threads, ambas páginas son descargadas concurrentemente y, cuando uno de los threads acaba, no aborta la ejecución del otro sino que lo espera antes de seguir con el hilo principal de ejecución. Para este ejemplo, no afecta el marcado de las páginas descargadas puesto que no se hace ningún procesado con ellas. El código WebL correspondiente aparece representado en la figura 4.39.



```
jones@bajo: /home/jones/msc2webl-1.1/prj$  
Ej Combinadores de Servicio:  
Getting http://www.google.es  
Getting http://www.google.com  
Got http://www.google.es/  
Descargado google.es (abortado google.com)  
Usando Threads:  
Getting http://www.google.es  
Getting http://www.google.com  
Got http://www.google.es/  
Descargado google.ES  
Got http://www.google.es/  
Descargado google.COM  
jones@bajo:~/msc2webl-1.1/prjs$
```

Figura 4.37: Resultado de ejecución del ejemplo de Threads

```

<diagramaMSC nombre="main" >
  <listaInstancias>
    <inst nombre="googleEs" tipo="normal"/>
    <inst nombre="ES" tipo="dinamica"/>
    <inst nombre="cliente" tipo="normal"/>
    <inst nombre="COM" tipo="dinamica"/>
    <inst nombre="googleCom" tipo="normal"/>
  </listaInstancias>
  <eventos>
    <accion inst="cliente" tipo="print">
      <valor>'Prueba con Combinadores de Servicio'</valor>
    </accion>
    <saltoNivel/>
    <combpar varSalida="P0">
      <listaInst>
        <inst nombre="googleEs" tipo="normal"/>
        <inst nombre="googleCom" tipo="normal"/>
      </listaInst>
      <operando1>
        <mensaje in="googleEs" out="cliente" tipo="get" varSalida="P0">
          'http://www.google.es'
        </mensaje>
        <saltoNivel/>
        <mensaje in="cliente" out="googleEs" tipo="reply" varSalida="P0">P0</mensaje>
      </operando1>
      <operando2>
        <mensaje in="googleCom" out="cliente" tipo="get" varSalida="P0">
          'http://www.google.com'
        </mensaje>
        <saltoNivel/>
        <mensaje in="cliente" out="googleCom" tipo="reply" varSalida="P0">P0</mensaje>
      </operando2>
    </combpar>
    <saltoNivel/>
    <ifelse inst="cliente">
      <test>$P0.URL = 'http://www.google.com'</test>
      <argumento>
        <accion inst="cliente" tipo="print">
          <valor>'Descargado google.com (descarga de google.es abortada)'</valor>
        </accion>
      </argumento>
      <argumento>
        <accion inst="cliente" tipo="print">
          <valor>'Descargado google.es (descarga de google.com abortada)'</valor>
        </accion>
      </argumento>
    </ifelse>
    <saltoNivel/>
    <accion inst="cliente" tipo="print">
      <valor>'Probamos ahora algo similar pero usando Threads.\n'</valor>
    </accion>
    <saltoNivel/>
  </eventos>
</diagramaMSC>

```

```
<thread nombreA="ES" nombreB="COM" creador="cliente">
  <eventosA>
    <mensaje in="googleEs" out="ES" tipo="get" varSalida="POES">
      'http://www.google.es'
    </mensaje>
    <saltoNivel/>
    <mensaje in="ES" out="googleEs" tipo="reply" varSalida="POES">POES</mensaje>
    <saltoNivel/>
    <accion inst="ES" tipo="print">
      <valor>'Se ha recibido correctamente google.ES'</valor>
    </accion>
  </eventosA>
  <eventosB>
    <mensaje in="googleCom" out="COM" tipo="get" varSalida="POCOM">
      'http://www.google.com'
    </mensaje>
    <saltoNivel/>
    <mensaje in="COM" out="googleCom" tipo="reply" varSalida="POCOM">POCOM</mensaje>
    <saltoNivel/>
    <accion inst="COM" tipo="print">
      <valor>'Se ha recibido correctamente google.COM'</valor>
    </accion>
  </eventosB>
</thread>
</eventos>
</diagramaMSC>
```

Figura 4.38: Código XML del diagrama MSC del ejemplo con Threads

```
//
// Modulos importados por defecto
import Files,Forms;
//
// Variables predeterminadas
var home="/home/jones/pfc/msc2webl-1.1/";
var temporal="/home/jones/pfc/msc2webl-1.1/tmp/";
var headers=[. "User-Agent"=Forms_useragent .];
var eliminables = "tr td table center font";

// Programa: threads
//
// Codigo WebL del diagrama main
PrintLn("Ej Combinadores de Servicio:");

//Combinador Servicio Paralelo:
var P0 = (Forms_Get("http://www.google.es",nil,headers)) |
(Forms_Get("http://www.google.com",nil,headers));
P0 = Forms_xhtmliza(P0,"P0",eliminables,temporal);
if (P0.URL == "http://www.google.com") then
PrintLn("Descargado google.com (abortado google.es) ");
else
PrintLn("Descargado google.es (abortado google.com) ");
end;
PrintLn("Usando Threads:");

// Thread
[|
begin
var POES = Forms_Get("http://www.google.es",nil,headers);
POES = Forms_xhtmliza(POES,"POES",eliminables,temporal);
PrintLn("Descargado google.ES");
end
,
begin
var POCOM = Forms_Get("http://www.google.com",nil,headers);
POCOM = Forms_xhtmliza(POCOM,"POCOM",eliminables,temporal);
PrintLn("Descargado google.COM");
end
|];
```

Figura 4.39: Código WebL del ejemplo de Threads



## 4.6. Lectura del correo de alumnos de la Universidad Carlos III

Como último ejemplo se incluye un programa que descarga las cabeceras de los mensajes en la bandeja de entrada del correo de alumnos de la Universidad Carlos III[50]. Este programa es más complejo que los anteriores e ilustra además el uso de funciones WebL embebidas dentro de los diagramas MSC. Esto es debido a las carencias de XPath señaladas en las anteriores secciones. Puesto que la estructura es ya algo más compleja, se incluye el documento MSC. Vemos en más detalle cada uno de los elementos que lo componen.

### 4.6.1. Documento

La estructura de este ejemplo si queda más clara al mostrar el documento MSC que lo describe. Se incluyen a continuación las representaciones gráficas y XML de dicho documento (figuras 4.40 y 4.41) donde se observa cómo el programa se define en un único diagrama MSC llamado *main* que hace llamadas a las funciones *login*, *obtenerMensajes* e *imprimir*.

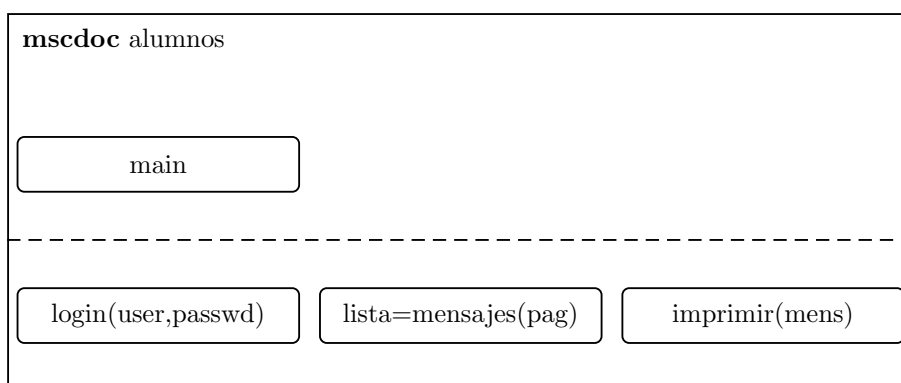


Figura 4.40: Ejemplo Correo Alumnos: Documento MSC

```

<documento nombre="alumnos" autor="Daniel Garcia Jones">
  <definiciones>
    <refdoc nombre="main"/>
  </definiciones>
  <utilidades>
    <refdoc nombre="login" parametros="user,passwd"/>
    <refdoc nombre="mensajes" varSalida="lista" parametros="pag"/>
    <refdoc nombre="imprimir" parametros="mens"/>
  </utilidades>
</documento>
  
```

Figura 4.41: Ejemplo Correo Alumnos: Código XML del Documento MSC

### 4.6.2. Programa principal

Podemos descomponer el programa principal en tres bloques:

1. Proceso de login en el sistema;
2. Obtención de la lista completa de mensajes;
3. Impresión de las cabeceras de mensaje;

### 4.6.3. Función *login*

Este bloque del programa maneja el proceso de login en el sistema. Esto implica cierto manejo de formularios y la obtención de tres páginas distintas hasta llegar a la que contiene la carpeta inbox. Analizando el código de la página principal de la aplicación, vemos que es una página con frames. El que nos interesa es el

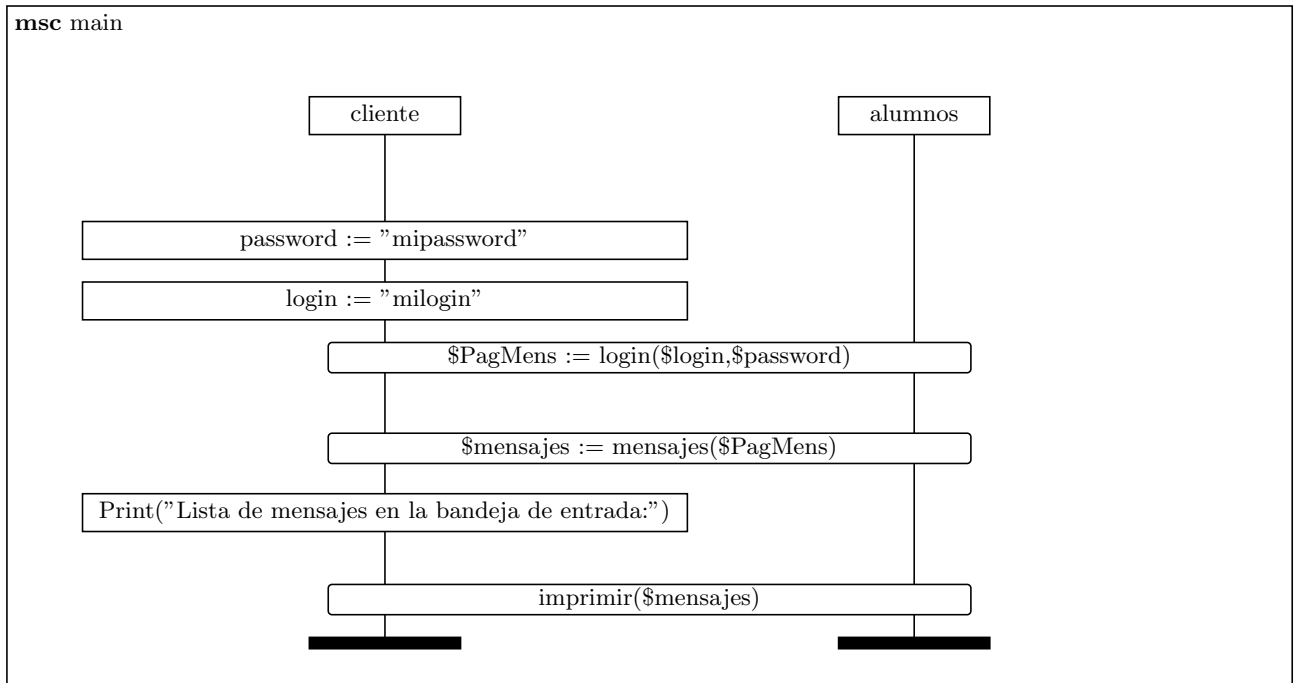


Figura 4.42: Ejemplo Correo Alumnos: Diagrama MSC del programa principal

```

<diagramaMSC nombre="main" autor="Daniel Garcia Jones">
  <listaInstancias>
    <inst nombre="cliente"/>
    <inst nombre="alumnos"/>
  </listaInstancias>
  <eventos>
    <accion inst="cliente" tipo="var" varSalida="password">
      <valor>'mipassword'</valor>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="var" varSalida="login">
      <valor>'milogin'</valor>
    </accion>
    <saltoNivel/>
    <ref varSalida="PagMens" parametros="$usuario,$password">login</ref>
    <saltoNivel/>
    <ref varSalida="mensajes" parametros="$PagMens">mensajes</ref>
    <saltoNivel/>
    <accion inst="cliente" tipo="print">
      <valor>'Lista de mensajes en la bandeja de entrada:'</valor>
    </accion>
    <saltoNivel/>
    <ref parametros="$mensajes">imprimir</ref>
  </eventos>
</diagramaMSC>
  
```

Figura 4.43: Ejemplo Correo Alumnos: Código XML del diagrama MSC del programa principal

primero de ellos, que es a su vez otra página con frames de la que hay que descargar la primera página del frameset. Hecho esto, ya tenemos en la variable P2, cuyo marcado aparece en la figura 4.44, la página que contiene el formulario de acceso. Dentro de P2, seleccionamos el primer formulario que aparece e introducimos los valores adecuados dentro de los inputs de usuario (de nombre *user*) y de password (de nombre *passwd*).

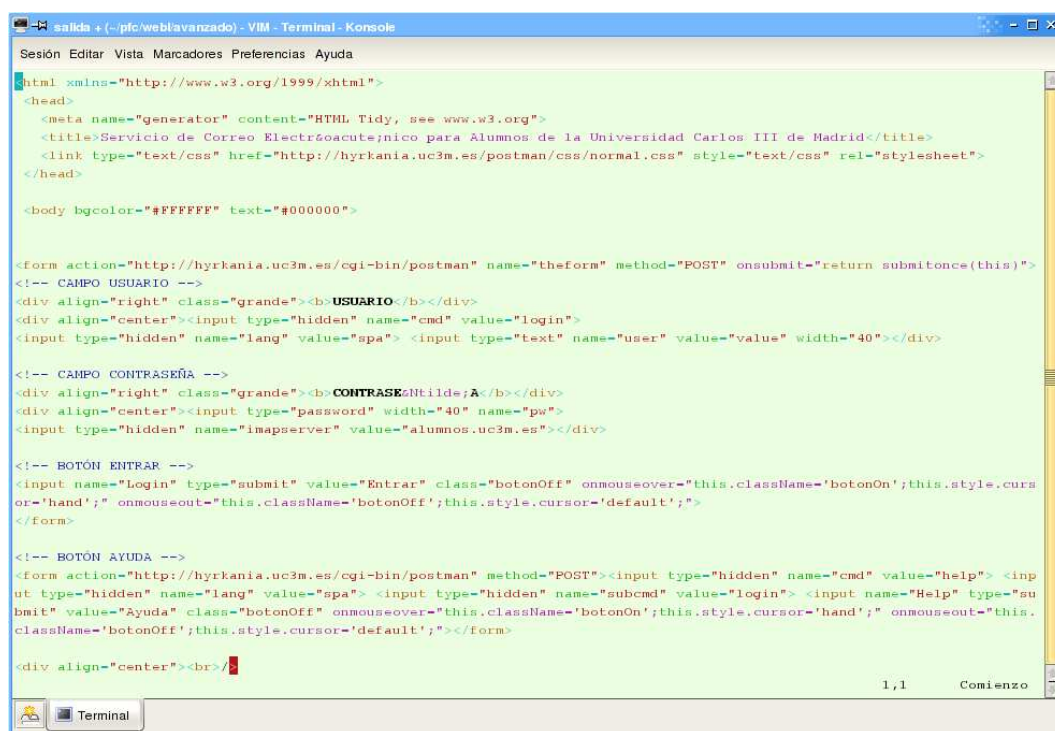


Figura 4.44: Ejemplo Correo Alumnos: Marcado de la página de acceso a la aplicación

Las figuras 4.45 y 4.46 representan el diagrama MSC de este bloque y su código XML correspondiente.

#### 4.6.4. Función *mensajes*

Este es el bloque más complejo de la aplicación. Su misión es obtener la lista completa de los mensajes que hay en la bandeja de entrada, navegando para ello a través de todas las páginas que fueran necesarias. Para entender su funcionamiento es necesario entender la regularidad estructural de las páginas de la aplicación, una vez dentro del sistema. El código HTML de una de estas páginas aparece en la figura 4.47.

Los datos de los diferentes mensajes están situados dentro de la página siempre entre dos elementos de tipo input. Entre cada uno de estos inputs hay una secuencia de seis elementos de tipo `<b>`, cada uno conteniendo los diferentes campos del mensaje. La manera más sencilla de seleccionar los distintos mensajes es elegir los elementos entre dos etiquetas input que tengan en su interior una secuencia como la indicada. Este tipo de selección no se puede hacer directamente con XPath, por lo que se han empleado expresiones WebL. Una vez hecho esto, la página se transforma para eliminar los caracteres extraños.

El mecanismo anterior permite seleccionar los mensajes en una página de la aplicación. Sin embargo, puede ocurrir que haya más de una página de mensajes, con lo que habrá que navegar entre ellas, una a una, para obtener todos los mensajes en la bandeja de entrada. Puesto que al entrar en la bandeja de correo, lo hacemos en la última de las páginas, tendremos que ir navegando hacia atrás, hasta que no queden más páginas que consultar. El enlace que navega por las páginas en sentido inverso tiene el texto "Página Previa"<sup>1</sup>. Sin embargo, puesto que esto añade bastante complejidad al código, en el script incluido únicamente se descargan los mensajes que aparecen en la portada del interfaz Web.

El diagrama MSC resultante, así como su código XML correspondiente, aparecen en las figuras 4.48 y 4.49.

<sup>1</sup>En realidad el texto es "Página Previa"

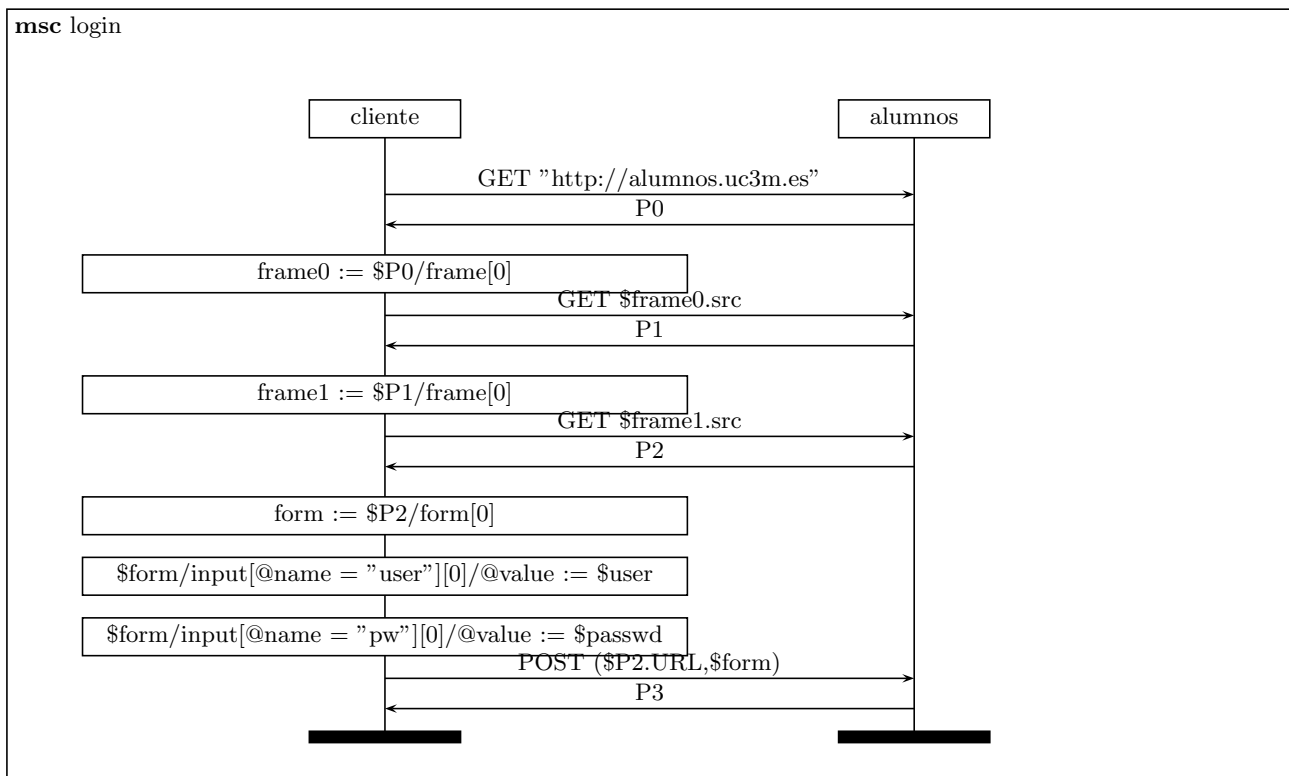


Figura 4.45: Ejemplo Correo Alumnos: Diagrama MSC de *login*

#### 4.6.5. Función *imprimir*

Una vez obtenidos la lista completa de mensajes, imprimimos por pantalla los datos relevantes. La información que almacena cada mensaje, en el orden en el que aparece (dentro de las etiquetas `<b>` `</b>`) es la siguiente:

1. Número del mensajes;
2. Marcas gráficas (indican si el mensaje es nuevo, respondido o está marcado para borrar);
3. Fecha del mensaje;
4. Remitente;
5. Asunto;
6. Tamaño;

Hemos diseñado el script de forma que únicamente imprima las marcas, el número del mensaje, el asunto y el remitente, pero ampliar la funcionalidad para imprimir el resto de elementos es sencillo. Debido al formato en que el bloque *obtenerMensajes* devuelve la lista de los mensajes, es necesario crear el bucle `While` para que recorra una a una cada lista de mensajes en las distintas páginas navegadas. El diagrama MSC de este bloque aparece en la figura 4.50 y su código XML correspondiente en la 4.51.

#### 4.6.6. Código WebL final y resultado de ejecución

El código final aparece reflejado en la figura 4.52. Observándolo, vemos que antes de ejecutarlo hay que introducir los valores adecuados de las variables *usuario* y *pwd*. Además, hemos de tener en cuenta que, para este ejemplo concreto, hemos de variar el valor de *eliminables* para eliminar de las páginas descargadas todas las etiquetas de tipo `"tr"`, `"td"`, `"table"`, `"center"` o `"font"`. Esto habremos de hacerlo de forma manual una vez obtenido el código WebL correspondiente a los diagramas. La figura 4.53 contiene la salida de la ejecución de este script para mi cuenta de correo de alumnos.

```

<diagramaMSC nombre="login" autor="Daniel Garcia Jones">
  <listaInstancias>
    <inst nombre="cliente" tipo="normal"/>
    <inst nombre="alumnos" tipo="normal"/>
  </listaInstancias>
  <eventos>
    <mensaje in="alumnos" out="cliente" tipo="get" varSalida="P0">'http://alumnos.uc3m.es'</mensaje>
    <saltoNivel/>
    <mensaje in="cliente" out="alumnos" tipo="reply" varSalida="P0">P0</mensaje>
    <saltoNivel/>
    <accion inst="cliente" tipo="var" varSalida="frame0">
      <seleccion tipo="simple">
        <varOrigen>$P0</varOrigen>
        <elemBusqueda>frame</elemBusqueda>
        <pos>0</pos>
      </seleccion>
    </accion>
    <saltoNivel/>
    <mensaje in="alumnos" out="cliente" tipo="get" varSalida="P1">$frame0.src</mensaje>
    <saltoNivel/>
    <mensaje in="cliente" out="alumnos" tipo="reply" varSalida="P1">P1</mensaje>
    <saltoNivel/>
    <accion inst="cliente" tipo="var" varSalida="frame1">
      <seleccion tipo="simple">
        <varOrigen>$P1</varOrigen>
        <elemBusqueda>frame</elemBusqueda>
        <pos>0</pos>
      </seleccion>
    </accion>
    <saltoNivel/>
    <mensaje in="alumnos" out="cliente" tipo="get" varSalida="P2">$frame1.src</mensaje>
    <saltoNivel/>
    <mensaje in="cliente" out="alumnos" tipo="reply" varSalida="P2">P2</mensaje>
    <saltoNivel/>
    <accion inst="cliente" tipo="form" varSalida="form">
      <seleccion tipo="simple">
        <varOrigen>$P2</varOrigen>
        <elemBusqueda>form</elemBusqueda>
        <pos>0</pos>
      </seleccion>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="asignacion">
      <seleccion tipo="complejo">
        <varOrigen>$form</varOrigen>
        <elemBusqueda>input</elemBusqueda>
        <att-busqueda>name</att-busqueda>
        <val-busqueda>'user'</val-busqueda>
        <pos>0</pos>
      </seleccion>
      <attsel>value</attsel>
      <valor>'user'</valor>
    </accion>
    <saltoNivel/>
  </eventos>
</diagramaMSC>

```

```

<accion inst="cliente" tipo="asignacion">
  <seleccion tipo="complejo">
    <varOrigen>$form</varOrigen>
    <elemBusqueda>input</elemBusqueda>
    <att-busqueda>name</att-busqueda>
    <val-busqueda>'pw'</val-busqueda>
    <pos>0</pos>
  </seleccion>
  <attsel>value</attsel>
  <valor>'passwd'</valor>
</accion>
<saltoNivel/>
<mensaje in="alumnos" out="cliente" tipo="post" varSalida="P3" form="$form">$P2.URL</mensaje>
<saltoNivel/>
<mensaje in="cliente" out="alumnos" tipo="reply" varSalida="P3">P3</mensaje>
</eventos>
</diagramaMSC>

```

Figura 4.46: Ejemplo Correo Alumnos: Código XML del diagrama MSC *login*

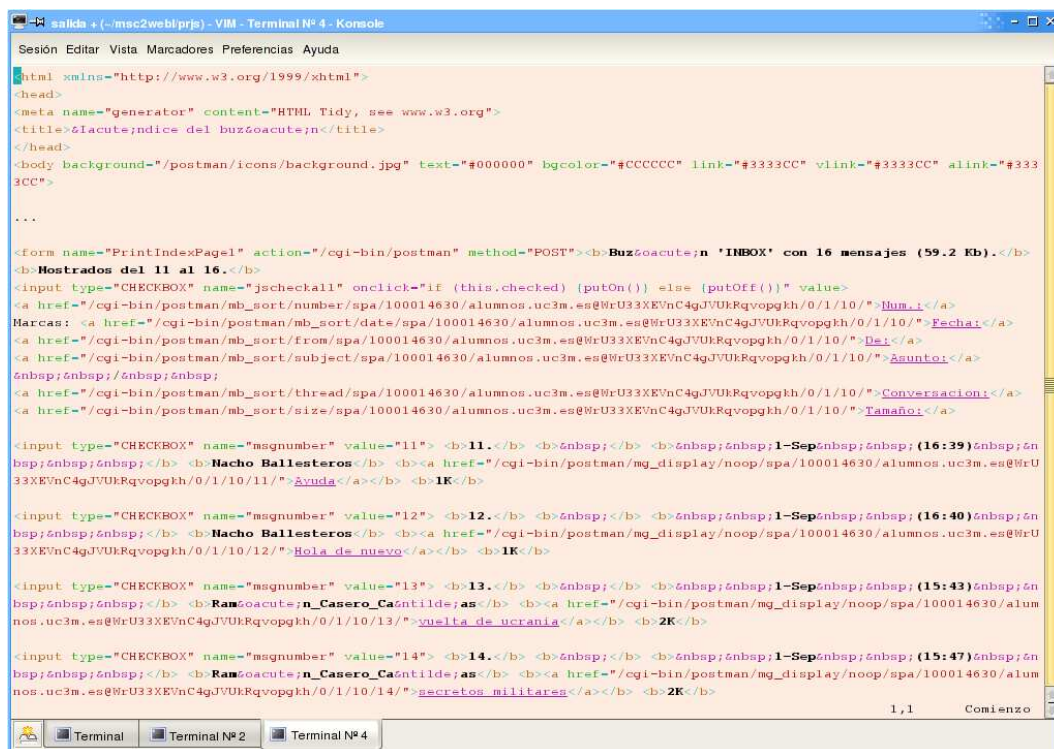


Figura 4.47: Ejemplo Correo Alumnos: Marcado HTML de la página inbox

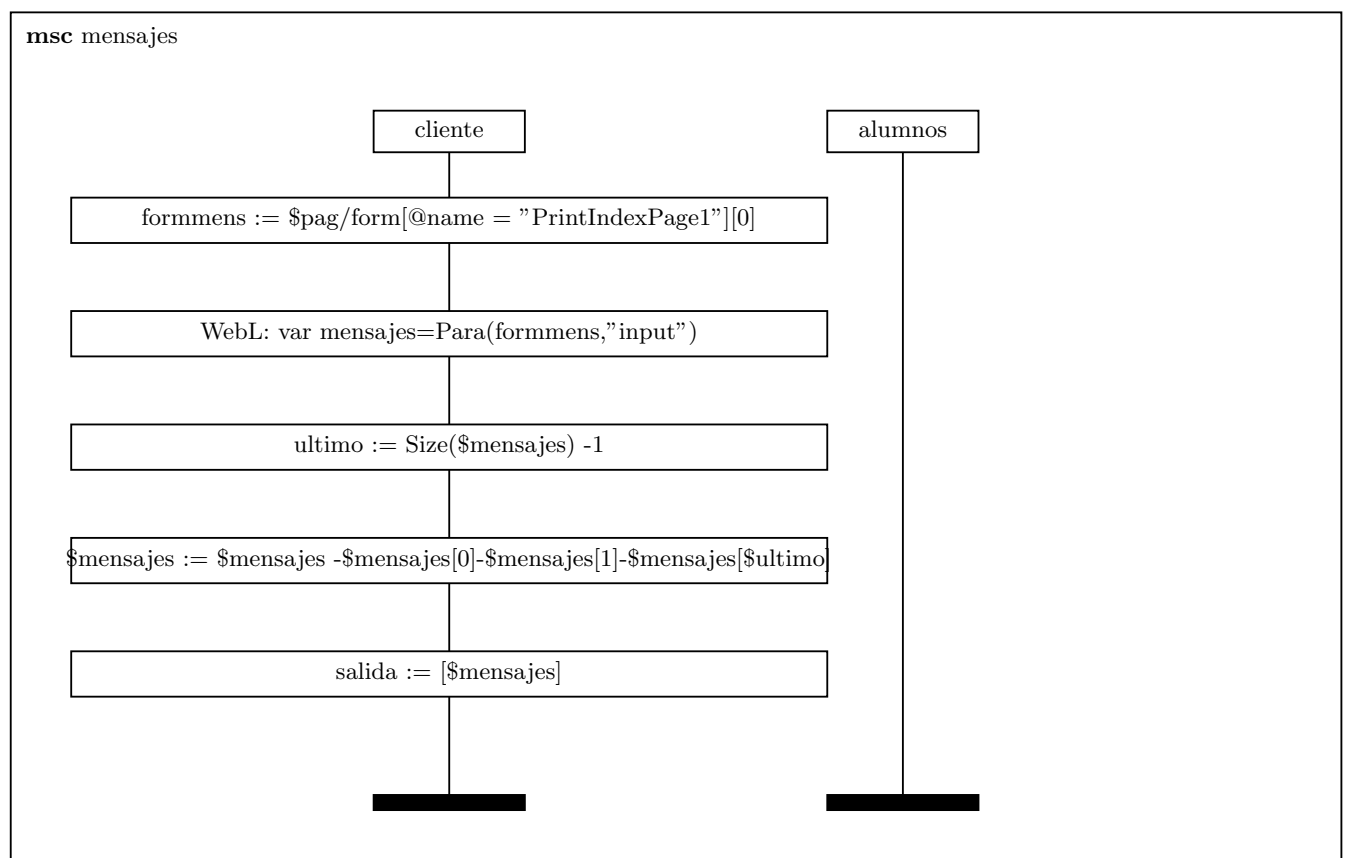


Figura 4.48: Ejemplo Correo Alumnos: Diagrama MSC de *mensajes*

```

<diagramaMSC nombre="mensajes" autor="Daniel Garcia Jones"
  descripcion="Obtiene la lista de todos los mensajes en la lista" >
  <listaInstancias>
    <inst nombre="cliente" tipo="normal"/>
    <inst nombre="alumnos" tipo="normal"/>
  </listaInstancias>
  <eventos>
    <accion inst="cliente" tipo="var" varSalida="formmens">
      <seleccion tipo="complejo">
        <varOrigen>$pag</varOrigen>
        <elemBusqueda>form</elemBusqueda>
        <pos>0</pos>
        <att-busqueda>name</att-busqueda>
        <val-busqueda>'PrintIndexPage1'</val-busqueda>
      </seleccion>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="webl">
      <valor>var mensajes=Para(formmens,"input")</valor>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="var" varSalida="ultimo">
      <valor>Size($mensajes)-1</valor>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="asignacion">
      <valor>$mensajes</valor>
      <valor>$mensajes -$mensajes[0]-$mensajes[1]-$mensajes[$ultimo]</valor>
    </accion>
    <saltoNivel/>
    <accion inst="cliente" tipo="var" varSalida="salida">
      <valor>[$mensajes]</valor>
    </accion>
    <saltoNivel/>
  </eventos>
</diagramaMSC>

```

Figura 4.49: Ejemplo Correo Alumnos: Código XML del diagrama *mensajes*



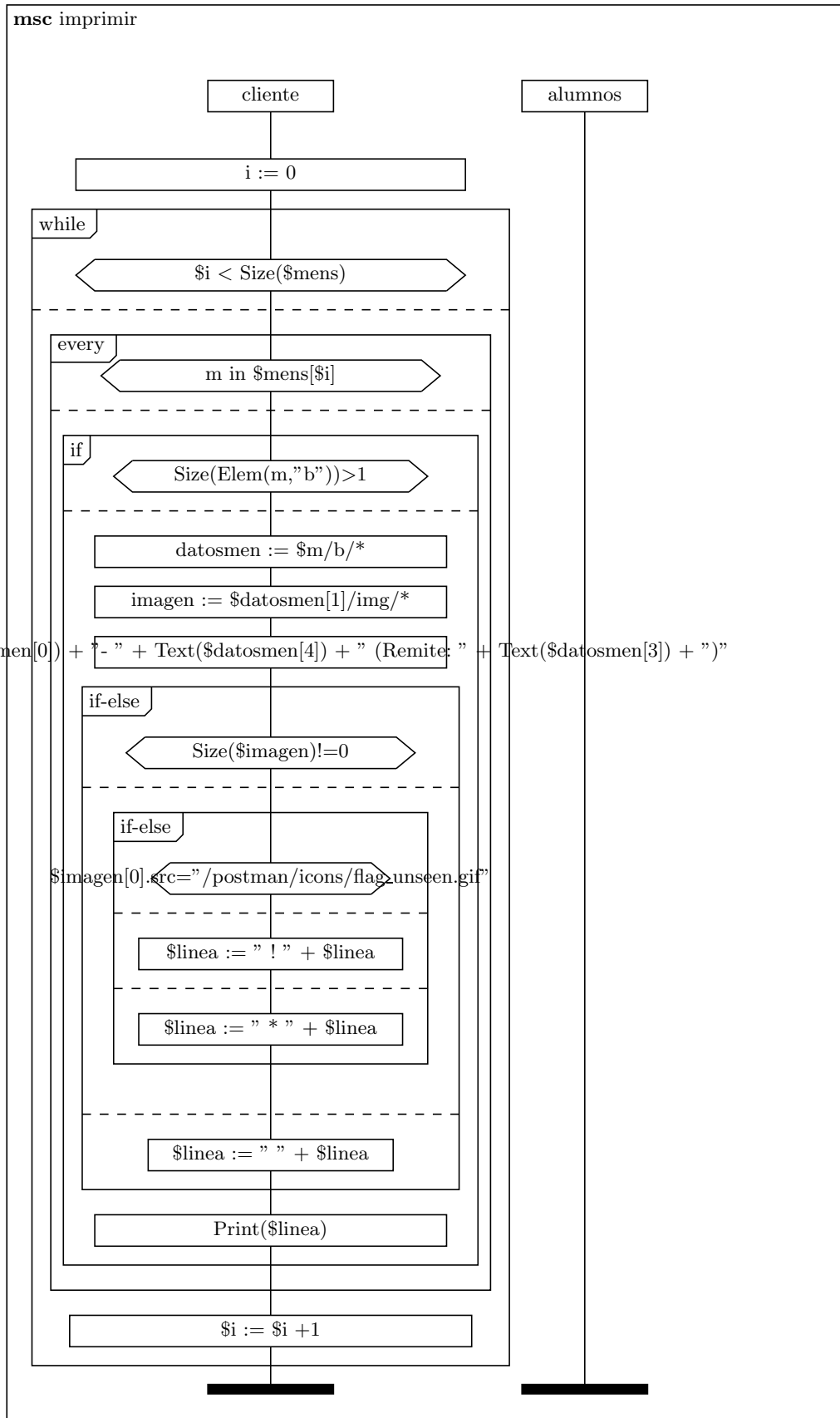


Figura 4.50: Ejemplo Correo Alumnos: Diagrama MSC de *imprimir*

```

<diagramaMSC nombre="imprimir" autor="Daniel Garcia Jones">
  <listaInstancias>
    <inst nombre="cliente" tipo="normal"/>
    <inst nombre="alumnos" tipo="normal"/>
  </listaInstancias>
  <eventos>
    <accion inst="cliente" tipo="var" varSalida="i">
      <valor>0</valor>
    </accion>
    <saltoNivel/>
    <while inst="cliente">
      <test>i < Size($mens)</test>
      <argumento>
        <every inst="cliente">
          <test>
            <id>m</id>
            <conjunto>$mens[$i]</conjunto>
          </test>
          <argumento>
            <if inst="cliente">
              <test>Size(Elem(m,"b"))&gt;1</test>
              <argumento>
                <accion inst="cliente" tipo="var" varSalida="datosmen">
                  <seleccion tipo="simple">
                    <varOrigen>$m</varOrigen>
                    <elemBusqueda>b</elemBusqueda>
                  </seleccion>
                </accion>
                <saltoNivel/>
                <accion inst="cliente" tipo="var" varSalida="imagen">
                  <seleccion tipo="simple">
                    <varOrigen>$datosmen[1]</varOrigen>
                    <elemBusqueda>img</elemBusqueda>
                  </seleccion>
                </accion>
                <saltoNivel/>
                <accion inst="cliente" tipo="var" varSalida="linea">
                  <valor>
                    Text($datosmen[0])+'- '+Text($datosmen[4])+' (Remite: '+Text($datosmen[3])+')'
                  </valor>
                </accion>
                <saltoNivel/>
                <ifelse inst="cliente">
                  <test>Size($imagen)!=0</test>
                  <argumento>
                    <ifelse inst="cliente">
                      <test>$imagen[0].src='/postman/icons/flag_unseen.gif'</test>
                      <argumento>
                        <accion inst="cliente" tipo="asignacion">
                          <valor>$linea</valor>
                          <valor>' ! ' + $linea</valor>
                        </accion>
                      </argumento>
                    </ifelse>
                    <argumento>
                      <accion inst="cliente" tipo="asignacion">
                        <valor>$linea</valor>
                        <valor>' * ' + $linea</valor>
                      </accion>
                    </argumento>
                  </ifelse>
                </argumento>
              </if>
            </every>
          </argumento>
        </while>
      </test>
    </while>
  </eventos>
</diagramaMSC>

```

```

        <argumento>
            <accion inst="cliente" tipo="asignacion">
                <valor>$linea</valor>
                <valor>'    ' + $linea</valor>
            </accion>
        </argumento>
    </ifelse>
    <accion inst="cliente" tipo="print">
        <valor>$linea</valor>
    </accion>
</argumento>
</if>
</argumento>
</every>
<accion inst="cliente" tipo="asignacion">
    <valor>$i</valor>
    <valor>$i +1</valor>
</accion>
</argumento>
</while>
</eventos>
</diagramaMSC>

```

Figura 4.51: Ejemplo Correo Alumnos: Código XML del diagrama *imprimir*

```

//
// Modulos importados por defecto
import Files,Forms;
//
// Variables predeterminadas
var home="/home/jones/pfc/msc2webl-1.1/";
var temporal="/home/jones/pfc/msc2webl-1.1/tmp/";
var headers=[. "User-Agent"=Forms_useragent .];
var eliminables = "tr td table center font";

// Programa: alumnos
// Autor: Daniel Garcia Jones

// Funcion login
var login = fun (user,passwd)
var P0 = Forms_Get("http://alumnos.uc3m.es",nil,headers);
P0 = Forms_xhtmliza(P0,"P0",eliminables,temporal);
var frame0 = Elem(P0,"frame")[0];
var P1 = Forms_Get(frame0.src,nil,headers);
P1 = Forms_xhtmliza(P1,"P1",eliminables,temporal);
var frame1 = Elem(P1,"frame")[0];
var P2 = Forms_Get(frame1.src,nil,headers);
P2 = Forms_xhtmliza(P2,"P2",eliminables,temporal);
var form = Forms_Normaliza(Elem(P2,"form")[0],true);
Select(Elem(form,"input"), fun(a) (a.name == "user") ? false end)[0].value := user;
Select(Elem(form,"input"), fun(a) (a.name == "pw") ? false end)[0].value := passwd;
var P3 = Forms_Post(form,P2.URL , [. mimetype="text/plain" .], true);
P3 = Forms_xhtmliza(P3,"P3",eliminables,temporal);
end;

// Funcion mensajes
var mensajes = fun (pag)
var formmens = Select(Elem(pag,"form"), fun(a) (a.name == "PrintIndexPage1") ? false end)[0];
var mensajes=Para(formmens,"input");
var ultimo = Size(mensajes) -1;
mensajes = mensajes -mensajes[0]-mensajes[1]-mensajes[ultimo];
var salida = [mensajes];
end;

```

```

// Funcion imprimir
var imprimir = fun (mens)
var i = 0;
while (i < Size(mens)) do
every m in mens[i] do
if (Size(Elem(m,"b"))>1) then
var datosmen = Elem(m,"b");
var imagen = Elem(datosmen[1],"img");
var linea = Text(datosmen[0]) + "- " + Text(datosmen[4]) + " (Remite: " + Text(datosmen[3]) + ")";
if (Size(imagen)!=0) then
if (imagen[0].src=="/postman/icons/flag_unseen.gif") then
linea = " ! " + linea;
else
linea = " * " + linea;
end;
else
linea = "  " + linea;
end;
println(linea);
end;
end;
i = i +1;
end;
end;

//
//Codigo WebL del diagrama main
var password = "mipassword";
var milogin = "login";
var PagMens = login(login,password);
var mens = mensajes(PagMens);
println("Bandeja de entrada:");
imprimir(mens);

```

Figura 4.52: Ejemplo Correo Alumnos: Código WebL correspondiente

```

jones@bajo: /home/jones/pfc/msc2webl-1.1/bin
Posted http://hyrkania.uc3m.es/cgi-bin/postman
Bandeja de entrada:
! 26.- [tio_matt] 4by1 FAQ (Remite: Adrian)
! 27.- Re: [tio_matt] 4by1 FAQ (Remite: BNL)
! 29.- [Todos] Pruebas de Selecci... para Forempleo 2 (Remite: MARIA JOSE MORENO CRESPIN)
! 28.- [Todos] (Remite: DOLORES MORELL SANZ)
! 30.- [Todos] ACCESO A TITULACIONES DE SEGUNDO CICL (Remite: CELIA GAVILAN MARFE)
! 32.- [Todos] Cursos Formaci... Ocupacional 2004/200 (Remite: Curso Extension)
! 31.- [Todos] Desayunos de Trabajo Forempleo 2004 (Remite: MARIA JOSE MORENO CRESPIN)
! 33.- [Todos] Fe de erratas Pruebas de Selecci... de (Remite: MARIA JOSE MORENO CRESPIN)
! 34.- [Todos] PRUEBA... (Remite: JOAQUIN BALLESTEROS TORRE)
! 35.- [tio_matt] M...s 4by1 (Remite: Adrian)
! 36.- [Todos] BECAS PARA COLABORAR EN ACCIONES DE P (Remite: CELIA GAVILAN MARFE)
! 37.- [Todos] Convocatoria de Becas para Forempleo (Remite: MARIA JOSE MORENO CRESPIN)
! 38.- [Todos] BECAS DE COLABORACI...N EN PROYECTOS DE (Remite: CELIA GAVILAN MARFE)
! 39.- [Todos] SELECCIONES DEPORTIVAS (Remite: JOAQUIN BALLESTEROS TORRE)
! 40.- [tio_matt] M...s de lo mismo (Remite: Adrian)
! 41.- [Todos] BECAS PARA ESTUDIAR IDIOMAS EN LA UNI (Remite: CELIA GAVILAN MARFE)
! 42.- [Todos] el s...bado... (Remite: JOAQUIN BALLESTEROS TORRE)
! 43.- [Todos] Conferencias sobre la mejora personal (Remite: Admon. Correo Electronico)
! 44.- [Todos] Rectificaci...n Wed Idiomas (Remite: Centro de idiomas)
! 45.- [Todos] Cursos FSE (Remite: Cursos Formacion Ocupacio)
! 46.- [Todos] Seminario "LOS DERECHOS DE LOS NI...OS (Remite: MARTA ISABEL CAAMI&Atilde;&lsquo;A DOM)
! 47.- [Todos] CONCESION DE BECA DE HONOR A EDURNE P (Remite: C.M.-R.E.FERNANDO ABRIL M)
! 48.- [Todos] LA UNIVERSIDAD DEL SIGLO XXI (Remite: C.M.-R.E.FERNANDO ABRIL M)
! 50.- [Todos] ACTIVIDADES (Remite: JOAQUIN BALLESTEROS TORRE)
! 49.- [Todos] Carod Rovira en la Universidad (Remite: Residencia Antonio Machad)
jones@bajo:~/pfc/msc2webl-1.1/bin$

```

Figura 4.53: Ejemplo Correo Alumnos: Salida correspondiente a la ejecución del script



## Capítulo 5

# Conclusiones y trabajos futuros

### 5.1. Conclusiones

En este proyecto hemos realizado un análisis detallado de la automatización de tareas en el Web y de las técnicas más relevantes aplicadas a este campo. Se han estudiado los Diagramas de Secuencias de Mensajes, que se demuestran adecuados para ser incorporados en metodologías de desarrollo para el Web. Por otro lado, se ha seleccionado un subconjunto de la norma MSC necesario para estos menesteres y se han analizado las posibilidades de XPath como lenguaje embebido en los MSC para representación del procesamiento realizado con los datos obtenidos de el Web.

Para probar de forma práctica el análisis teórico realizado, se ha implementado una aplicación funcional basada en este modelo de forma que, a partir de representaciones gráficas de los MSC, se obtiene código WebL directamente ejecutable. Esta aplicación se ha desarrollado sobre una arquitectura XML y diversas tecnologías estándar asociadas, como son XML Schema y XSLT.

Por último se ha confeccionado una batería de ejemplos de forma que no sólo el funcionamiento de la herramienta ha sido probado sino también, y mucho más importante, el funcionamiento del modelo. Durante el desarrollo de este proyecto se escribió además un artículo[51] publicado en el III Taller de Ingeniería del Software Orientada al Web (JISBD'2003) y se creó una página Web XHTML para albergar la documentación y la aplicación desarrollada.

Tras el estudio realizado, se han extraído una serie de conclusiones que podemos separar en aquellas relativas al modelo propuesto (aplicación de los MSC a la descripción de sistemas de navegación automática del Web) y aquellas que conciernen a la aplicación MSC2WEBL.

#### 5.1.1. Conclusiones sobre el modelo propuesto

Se ha delimitado un subconjunto de la norma MSC suficiente para describir la mayor parte de los asistentes de navegación automática típicos. Se ha procurado introducir el menor número de elementos nuevos para alejarse lo menos posible de la norma que, no obstante, permite la definición de elementos nuevos a partir de los ya existentes. En un principio estudiamos la posibilidad de definir los elementos no concurrentes empleando diagramas de alto nivel hMSC. Sin embargo vimos que esto añadía una complejidad innecesaria, ya que mediante el uso de expresiones inline es sencillo describir estos comportamientos y se hacen más fáciles de entender los diagramas generados.

El nivel de abstracción del modelo propuesto permite de forma sencilla la extracción, manipulación y almacenamiento de los datos Web, independientemente de los protocolos de comunicación empleados, no incluidos explícitamente en los diagramas generados. El lenguaje planteado es, además, sencillo de entender, incluso por personas no expertas en el tema.

Se han introducido los documentos MSC como ayuda a los desarrollos grandes, ya que puede ayudar a visualizar la estructura de proyectos de desarrollo complejos. Para automatizar tareas simples es posible prescindir de estos formalismos. Esto aporta un grado más de flexibilidad al método propuesto y facilita las tareas de mantenimiento. Se han introducido también los Combinadores de Servicio para dotar de una mayor robustez los programas descritos e imitar de la forma más fiel posible el comportamiento humano al procesar tareas en el Web de forma manual.

Desarrollando los mecanismos de transformación necesarios, se pueden convertir las especificaciones MSC a programas ejecutables en cualquiera de los lenguajes de procesamiento Web existentes. Para esta capa de transformación intermedia puede emplearse la gramática textual definida por la norma, si bien consideramos más apropiado emplear un lenguaje de representación basado en XML, ya que existen multitud de APIs y lenguajes de procesamiento de documentos XML que facilitarían la tarea de desarrollar herramientas de desarrollo basadas en esta técnica.

Otra importante ventaja es que modelo definido es fácilmente ampliable para cubrir nuevas necesidades o usos que surjan en el futuro. Sin embargo, este método presenta una serie de limitaciones. La principal de ellas es la, en ocasiones insuficiente, expresividad de XPath 1.0 como lenguaje de acceso de partes en documentos estructurados. A pesar de la potencia de este lenguaje, algunos métodos útiles para el desarrollo de sistemas de navegación automática no están incluidos en la actual versión. Esto se corregirá al lanzarse la nueva versión que incluirá muchas mejoras.

Este método de descripción satisface por tanto muchos de los aspectos señalados en el capítulo 2 y permite reducir los costes de desarrollo y mantenimiento de este tipo de sistemas, aunque sigue siendo necesario un conocimiento mínimo del modelo de programación del Web y de los distintos lenguajes de marcado para poder realizar los distintos desarrollos.

### 5.1.2. Sobre la aplicación MSC2WEBL

La aplicación MSC2WEBL está compuesta principalmente por un conjunto de Schemas XML y plantillas XSLT, así como un complejo interfaz gráfico de usuario. La representación interna de los distintos datos con que trabaja la aplicación se hacen empleando documentos XML conformes a los distintos Schemas definidos.

El método de dibujo de los diagramas MSC se apoya en el lenguaje de procesamiento de textos  $\text{\LaTeX}$ , lo que requiere en todo caso que, para visualizar un diagrama, sea necesario un proceso de compilación. Inicialmente la aplicación no compilaba paso a paso si no cuando el usuario así lo indicara, editando directamente el fichero XML. Finalmente se ha optado por dejar a la elección del usuario si quiere o no compilar el código  $\text{\LaTeX}$  en cada paso. No obstante, el tiempo de compilación en un ordenador relativamente moderno es pequeño y apenas produce molestias.

Hay que tener en cuenta también que a pesar de que se puede variar el aspecto de los diagramas dibujados, existirán casos en que haya que hacer algún retoque manual para la correcta visualización de determinadas composiciones. Un ejemplo claro de esto es el uso de Threads, que originan bastantes problemas en la visualización. No obstante hay que recordar que, para un código XML válido e independientemente de su visualización, el código WebL final será siempre funcional.

El manejo de expresiones XPath en la aplicación se hace de forma transparente al usuario. Aunque internamente se trabaje con XPath, existe un enmascaramiento de este lenguaje para que la aplicación pueda ser empleada incluso por personas que no lo conozcan. Por tanto, aunque es deseable un conocimiento previo de su sintaxis, éste no es necesario para emplear la aplicación ya que la creación de los distintos elementos del diagrama se hace empleando una serie de diálogos adecuados a cada uno de estos elementos, de forma que las expresiones se construyen de forma automática.

Puesto que la herramienta incluye un gran número de opciones es recomendable leer el manual de usuario así como tener alguna noción de programación sobre el Web para sacar un mayor partido de la aplicación. El hecho de que se incluyan todos los elementos necesarios para la creación, ejecución documentación y mantenimiento de los sistemas disminuye los costes en todas estas fases del proceso de programación del Web.

Por otro lado, es posible el uso del sistema desde un editor XML de los existentes en el mercado, sin necesidad de usar el interfaz gráfico de usuario, empleando los distintos Schemas y plantillas XSLT de forma manual. En entornos donde no exista un servidor gráfico de aplicaciones, podremos seguir usando las distintas plantillas XSLT sobre los documentos editados manualmente, obteniendo la misma funcionalidad que con la aplicación. De esta forma se podrían desarrollar los sistemas en una consola remota o incluso incorporarlo directamente en un servidor Web con soporte para transformaciones XSLT.

Como resumen, indico a continuación las principales conclusiones del proyecto:

- Los diagramas MSC en conjunción con XPath son una técnica adecuada para la especificación de



requisitos de sistemas Web. Mediante su uso, los tiempos de desarrollo de estos sistemas disminuyen notablemente al facilitarse en gran medida la especificación de requisitos de los sistemas empleando para ello una sintaxis gráfica sencilla. Estas descripciones gráficas son fáciles de entender, incluso por personas no expertas en la materia.

- La heterogeneidad de datos en el Web y la falta de regularidad son dos de los factores que más complican la programación sobre Internet. Una Web accesible, basada en estándares, con un marcado correcto que independice presentación de información contenida, contribuye a facilitar enormemente los desarrollos sobre ella. La mayor parte de páginas del Web legado no cumplen estos requisitos. Es necesario por tanto concienciar y educar en estos aspectos a los desarrolladores de sitios Web, de forma que la navegación automática de Internet sea en general más viable a como es hoy día.
- La versión actual de XPath, la 1.0, es algo limitada en cuanto a su flexibilidad. Algunas construcciones útiles en procesado Web no están incluidas en ella mientras que sí que lo están en el actual borrador de la futura versión 2.0 de XPath. Esta futura versión presenta por tanto características más adecuadas que su predecesora. Sin embargo, el hecho de que esté aún en fase de revisión hace conveniente que no sea incluida (al menos de momento) en la metodología propuesta, que pretende basarse en estándares reconocidos.
- El empleo de la herramienta CASE creada en este proyecto disminuye los costes de desarrollo y documentación de sistemas. Los costes de mantenimiento son también algo menores debido a que la representación gráfica de los diagramas permite de forma más sencilla detectar los cambios o errores a corregir en la fase de mantenimiento. Sin embargo, hubiera sido de interés proveer de algún mecanismo extra de ayuda al mantenimiento automática o, al menos, con la menor intervención humana posible. Por ejemplo, sería interesante que la herramienta localizara y marcara el punto en el diagrama en que el programa a mantener falle, bien sea por cambios en el marcado de la página o cualquier otra razón.
- La intención de apoyarse en estándares reconocidos no es caprichosa. Siguiendo este enfoque se permite que haya abundante información al respecto de las técnicas empleadas. Esto ha facilitado no sólo el desarrollo del proyecto en sí, sino que además permite un acercamiento fácil y ampliamente documentado al enfoque propuesto por terceras personas, de forma que este nuevo método e incluso la aplicación desarrollada puedan ser empleados por el mayor número de personas.
- El conjunto de herramientas de software libre disponible proporciona una gran ayuda a los desarrollos software, sean del tipo que sean, no sólo por la gratuidad de la mayor parte de programas y sistemas operativos, sino por la abundante documentación, bibliotecas y aplicaciones existentes para todo tipo de propósitos.

Los objetivos iniciales del proyecto han sido satisfechos: se ha demostrado que la metodología propuesta es válida y facilita los desarrollos, la aplicación MSC2WEBL es completamente funcional y se ha incluido un completo manual de usuario además de un conjunto de ejemplos variados que ilustran la mayor parte de los elementos incluidos en el método propuesto.

## 5.2. Líneas de trabajo futuro

Varias son las posibles líneas de trabajo futuro. Éstas, al igual que las conclusiones, están divididas en dos frentes, aquellas que se centran en la metodología propuesta; y aquellas que afectan a la aplicación desarrollada. Las enumero a continuación.

### 5.2.1. Sobre la metodología propuesta

Los diagramas MSC han demostrado ser válidos para la especificación de requisitos de sistemas de navegación automática. Sin embargo a lo largo del presente estudio se han evidenciado ciertas carencias de este método que podrían ser subsanadas en futuros estudios. Las principales líneas de trabajo futuro en el ámbito de la metodología propuesta se resumen a continuación.

- Adaptar el método propuesto para que emplee la versión 2.0 de XPath que subsana muchas de las limitaciones ya comentadas de XPath. Para poder incluir esta versión en el modelo es necesario o bien que se concluya la especificación, o bien que el borrador de trabajo se encuentre en una fase más avanzada a la actual.
- El método propuesto se ha aplicado a la descripción de clientes Web. Una de las líneas de trabajo futuras más claras se centraría en el estudio de los MSC aplicados al lado del servidor de forma que cualquier tipo de aplicación pueda ser descrita empleando MSC.

- El Schema XML definido no es completamente genérico al estar parcialmente basado en el lenguaje WebL en que posteriormente se transforman los diagramas. Sería adecuado definir un Schema general e independiente del lenguaje final de salida y que implemente de forma más completa el lenguaje XPath, permitiendo cualquier profundidad en las expresiones de selección de partes de documentos XML.
- La especificación mediante diagramas MSC facilita las tareas de mantenimiento de los sistemas, si bien sería de interés proveer de mecanismos de ayuda al mantenimiento especializados, de forma que la intervención humana sea reducida al mínimo para estas tareas.

### 5.2.2. Sobre la aplicación MSC2WEBL

Se ha desarrollado la aplicación MSC2WEBL como ejemplo práctico de aplicación de los diagramas MSC a la descripción de sistemas Web. A pesar de que la aplicación es completamente funcional, muchos son los aspectos que podrían modificarse para aumentar el grado de facilidad de uso de la aplicación y la robustez de los sistemas generados. Enumero a continuación varias líneas futuras relacionadas con este aspecto.

- El lenguaje WebL ha demostrado ser muy potente y sencillo de utilizar. Sin embargo, también presenta ciertas limitaciones, como por ejemplo el hecho de depender de Java y no ser libre. Existen multitud de módulos y bibliotecas para programación Web en el lenguaje Python. El empleo de Python como lenguaje de salida de la aplicación proporciona enormes ventajas. No sólo Python es libre, sino que la aplicación gráfica ya esta desarrollada en este lenguaje. Además el uso de este lenguaje está mucho más extendido que el del lenguaje WebL y la velocidad de ejecución de los programas Python es mayor.
- Indistintamente del lenguaje de salida elegido, varios aspectos de la herramienta CASE pueden ser mejorados para aumentar la facilidad de uso de la aplicación.
  - Implementar una consola de ejecución completa que permita la introducción de parámetros de entrada, de forma que los scripts puedan ser completamente interactivos.
  - El navegador de código XHTML incluido facilita las tareas de programación de los sistemas. No obstante también está bastante limitado, en tanto en cuanto que únicamente permite la exploración del código. Un navegador que permita explorar tanto el aspecto de la página, como su código interno y que permita calcular la expresión XPath que define a un elemento de la página, facilitaría enormemente los desarrollos.
  - El mecanismo de dibujo de los diagramas MSC no es en tiempo real. Sería interesante poder dibujar directamente los diagramas MSC, sin necesidad en ninguno de los casos de editar el código XML directamente. Crear una herramienta en que los diagramas sean directamente editables en la ventana de dibujo queda fuera del alcance de este proyecto, pero facilitaría aún más los distintos desarrollos.
- El proceso de instalación de la aplicación esta basado en un script que funciona únicamente en distribuciones basadas en Debian. Facilitar el proceso de instalación de la aplicación independientemente de la distribución Linux empleada es necesario para que la aplicación pueda ser utilizada por el mayor número de personas.
- Las bibliotecas y módulos empleados están portados en casi su totalidad a sistemas operativos Windows. La aplicación debería por tanto poder migrarse sin demasiados problemas a Windows, sistema operativo mayoritario hoy día, facilitando aún más el uso de la aplicación, independizándolo en mayor medida de la plataforma de trabajo de los usuarios.
- El conjunto de ejemplos propuesto no incluye una aplicación compleja de un tamaño (líneas de código) importante, como por ejemplo las presentadas en [29][30][31]. Una interesante línea de trabajo futuro es el desarrollo de un proyecto grande empleando la aplicación MSC2WEBL para estudiar de manera sistemática en que medida se facilita el desarrollo empleando la aplicación desarrollada.

## Capítulo 6

# Presupuesto del proyecto

### 6.1. Historia del proyecto

El inicio del presente proyecto se sitúa en Septiembre de 2003 y se puede descomponer en las siguientes fases:

- **Fase 0:** documentación y configuración del sistema.
- **Fase 1:** desarrollo inicial con DIA.
- **Fase 2:** desarrollo de la arquitectura XML.
- **Fase 3:** desarrollo de la interfaz gráfica de usuario.
- **Fase 4:** pruebas y depuración de la aplicación.
- **Fase 5:** confección de ejemplos.
- **Fase 6:** redacción de la documentación del proyecto.
- **Fase 7:** corrección y depuración final.

#### 6.1.1. Documentación y configuración del sistema

A lo largo del desarrollo del proyecto ha sido necesario realizar una tarea intensiva de documentación. No sólo se han analizado la norma MSC y el lenguaje WebL, sino que se ha realizado un estudio de la automatización de tareas en el Web y estudiado el lenguaje XML junto con algunas de las tecnologías desarrolladas sobre éste, como son XML Schema, XPath y XSLT. También ha sido necesario aprender los lenguajes Python y PyGTK. En este tiempo además se configuró el ordenador de trabajo con el sistema operativo Debian Linux y los programas y librerías necesarias para el desarrollo.

Las fases de configuración inicial del sistema y estudio de los diagramas MSC y del lenguaje WebL se ha realizado principalmente en Septiembre y Octubre de 2003, llevando aproximadamente 200 horas.

El estudio de las tecnologías relacionadas con XML se centró en los meses de Diciembre de 2003 y Enero de 2004, dedicando para esta tarea aproximadamente 200 horas.

El aprendizaje del lenguaje Python y PyGTK ocupó los meses de Marzo y Abril de 2004, dedicando para esta tarea aproximadamente 200 horas.

Por tanto, las tareas de documentación inicial, configuración del sistema y aprendizaje de los distintos lenguajes, centradas en los meses de Septiembre, Octubre y Diciembre de 2003 y Enero, Marzo y Abril de 2004, llevaron aproximadamente 600 horas.

#### 6.1.2. Desarrollo inicial con DIA

Una vez iniciado el proyecto, hubo un periodo de tiempo en que se siguió una línea de investigación y desarrollo que se abandonó posteriormente. Inicialmente se pensó en desarrollar la aplicación empleando el paquete software DIA[10] para dibujo de diagramas de todo tipo, incluyendo UML, muy similar a MSC. Se estudió su código fuente, escrito en C y GTK+ para intentar adaptarlo a la tarea de dibujar diagramas

MSC. Se invirtió aproximadamente el mes de Noviembre en seguir esta línea de investigación que, aunque fue abandonada en poco tiempo debido a su dificultad, aportó la idea de basar la arquitectura de la aplicación en XML por completo, al ser este el lenguaje empleado por DIA para representación interna de los diagramas. El total de horas invertidas en esta fase fue de 50 horas ya que la dedicación al proyecto durante esta fase no fue exclusiva.

### **6.1.3. Desarrollo de la arquitectura XML**

Se emplearon los meses de Enero y Febrero de 2004 para desarrollar la arquitectura XML de la aplicación. Se definió el Schema XML para representar el subconjunto de la norma MSC necesario para la representación de las tareas Web (que se delimitó en las fases de documentación inicial). La dedicación media a estas tareas fue de 5 horas/día laborable, obteniendo un total de 200 horas.

### **6.1.4. Desarrollo de la interfaz gráfica de usuario**

Esta fase ocupó el periodo de tiempo abarcado desde Abril a Junio de 2004 invirtiendo para ello un total de 300 horas (5 horas/día laborable). Si bien hubo que corregir abundantes errores y problemas en la aplicación y en el modelo empleado, estas tareas quedan incluidas en las fase de prueba y depuración y de corrección del proyecto.

### **6.1.5. Pruebas y depuración de la aplicación**

Durante esta fase se probó intensivamente la aplicación MSC2WEBL y los schemas y plantillas XSLT definidos. Varios fueron los cambios que hubieron de introducirse para corregir los fallos que surgían de un uso exhaustivo del programa. Así mismo, durante esta fase, se desarrolló el script de instalación de la aplicación.

Durante esta fase, desarrollada en el mes de Julio de 2004, se emplearon 80 horas.

### **6.1.6. Confección de la batería de ejemplos**

En esta fase, se empleó la aplicación MSC2WEBL para la confección de una batería de ejemplos, incluida en la presente memoria. Se empleo el mes de Agosto de 2004 con una dedicación de 4 horas día/laborable con un total de 80 horas.

### **6.1.7. Redacción de la documentación del proyecto**

En esta fase se redactó la memoria y presentación del proyecto, el manual de la aplicación y se creó una página Web del proyecto de forma que se hiciera accesible la aplicación. Esta tarea ocupó los meses de Septiembre y Noviembre de 2004, empleando para ello un total de 150 horas (100 en el mes de Septiembre y unas 50 en el mes de Noviembre para incluir todas las correcciones hechas en esa fase).

### **6.1.8. Corrección y depuración final**

Una vez finalizado el proyecto se sometió a corrección por parte del tutor Dr. Vicente Luque, siendo necesario incluir cambios tanto en la aplicación como en la memoria. Esta fase se extendió durante los meses de Octubre y Noviembre de 2004, empleándose 150 horas.

### **6.1.9. Calendario**

Se incluye a continuación un calendario del proyecto en que quedan reflejadas las distintas fases del proyecto.

## **6.2. Presupuesto**

Se va a desglosar en dos apartados: uno referente al coste material en el que se incluirán tanto gastos debidos al software utilizado, como al hardware o a las instalaciones empleadas; en el otro apartado se comentará el coste derivado del salario del personal involucrado en este proyecto.

SEP-03	OCT-03	NOV-03	DIC-03	ENE-04
Documentación inicial: MSC y lenguaje WebL		Desarrollo DIA	Documentación XML	Desarrollo XML
FEB-04	MAR-04	ABR-04	MAY-04	JUN-04
Desarrollo XML	Documentación Python PyGTK	Desarrollo GUI		
JUL-04	AGO-04	SEP-04	OCT-04	NOV-04
Pruebas y Depuración	Ejemplos	Memoria Proyecto	Corrección	Corrección Memoria

Figura 6.1: Calendario del proyecto

### 6.2.1. Costes de material

En la realización de este proyecto se han empleado los siguientes materiales:

- Un ordenador valorado en 1500 euros. Como dicho ordenador ha sido y será reutilizado su coste real aproximado será de 500 euros.
- Un lugar de trabajo debidamente adecuado y acondicionado que conlleva unos gastos de aproximadamente 150 euros/mes. Como está utilizado por una sola persona, su uso es exclusivo durante la duración del proyecto (unos 15 meses), el coste final por este punto será de 2250 euros.
- Conexión a Internet, necesaria para la obtención de información. Está valorada en 42 euros/mes que multiplicado por la duración del proyecto, da un coste aproximado de 630 euros.
- Costes varios, tales como fotocopias, impresión de documentos, encuadernaciones, etc., con un coste aproximado de 500 euros.
- Los programas empleados para la realización de las pruebas fueron todos de libre distribución o con licencias que permiten su uso en investigación, de manera que el coste debido a este factor será considerado como nulo.

Con los costes indicados anteriormente se puede obtener que el coste final debido al material empleado en este proyecto según muestra en la siguiente tabla:

Coste Material	Precio (euros)
Ordenador	500
Lugar de trabajo	2250
Conexión a Internet	630
Costes varios	500
Software	0
Total	3880

Cuadro 6.1: Costes de material

### 6.2.2. Costes de personal

En el coste de personal debe tenerse en cuenta tanto los honorarios derivados del desarrollo del proyecto, como los derivados de la dirección del mismo. Para obtener los honorarios correspondientes a un ingeniero, se visitó la página Web oficial del Colegio de Ingenieros de Telecomunicaciones[52], según la cual el coste de una hora laboral de un ingeniero de telecomunicaciones asciende (para el 2004) a 70 euros/hora. No se han encontrado los honorarios de un Ingeniero Técnico, así que se ha supuesto que estos son de 45 euros/hora.

Para el desarrollo de este proyecto se han invertido un total aproximado de 1530 horas en el desarrollo del proyecto. Como esta cifra supera el rango de las 1080 horas, según el 'Baremo de honorarios orientativos para trabajos profesionales en 2004' que figura en la referencia anterior, se debe aplicar un factor corrector de 0.40, lo que finalmente fija el coste de personal en 27.540 euros.

La dirección del proyecto fue llevada a cabo por el Dr. Vicente Luque Centeno, que empleó aproximadamente unas 100 horas en llevar a cabo esta tarea, lo que supone un coste de 4.500 euros.

Teniendo en cuenta estos valores, el coste final debido al personal es el indicado en la tabla siguiente:

Coste del personal	Precio (euros)
Desarrollo y realización	27.540
Dirección	4.500
Total	32.040

Cuadro 6.2: Costes de Personal

### 6.2.3. Presupuesto total

El presupuesto total de este proyecto asciende a treinta y cinco mil novecientos veinte euros.

Leganés a 22 de Noviembre de 2004

El ingeniero técnico proyectista

Fdo. Daniel García Jones

# Apéndice A: Manual de la aplicación

## A-1. Introducción

Cada vez es más común el uso de aplicaciones remotamente accesibles desde la Web para diversos propósitos, desde la compra de artículos a la consulta de saldos o movimientos bancarios. Al mismo tiempo, la cantidad de información disponible en la WWW crece día a día. Por todo ello aumenta el número de usuarios que acceden a la red para el envío y recepción de información. En muchos de los casos es necesario un alto grado de interacción por parte del usuario con la red empleando algún navegador (navegación manual). La heterogeneidad de datos en la Web, cada vez mayor, es otro de los factores que incrementa el tiempo de procesamiento manual de esta información. Por todo ello sería de interés disponer de sistemas de navegación automática capaces de realizar estos procesos Web por sí solos, ahorrando a los usuarios tiempo y esfuerzo.

MSC2WEBL es una aplicación para el desarrollo rápido de clientes Web, empleando el lenguaje *WebL* y basado en los diagramas MSC, *Gráficos de secuencias de mensajes*, para la especificación de requisitos de la aplicación. Está desarrollada en *Python* y emplea los módulos *PyGTK*, *Ft*, *jaxml* y *XSV* entre otros. Las aplicaciones que podemos desarrollar van desde simples scripts para consulta de buscadores o de las cabeceras de los principales periódicos, a agregadores de contenido o aplicaciones más complejas que impliquen un manejo intensivo de formularios.

La aplicación ha sido desarrollada y probada en varios sistemas basados en Debian Linux<sup>1</sup>, aunque debería funcionar en cualquier otro sistema Linux con las librerías adecuadas instaladas y un intérprete *Python* (mínimo la versión 2.0).

Este manual describe el funcionamiento de la aplicación y el proceso de instalación.

## A-2. La aplicación MSC2WEBL

La aplicación MSC2WEBL consta de una ventana principal con 5 subventanas accesibles mediante pestañas, desde las que acceder a todas las funciones de la aplicación: **Vista Proyecto**, **Vista MSC**, **Editor Latex**, **Vista WebL** y **Explorador de marcado Web**. A continuación se explica el funcionamiento de cada una de estas subventanas y de los elementos que contienen.

### A-2.1. Vista Propiedades del Proyecto

Desde esta vista (figura A-2) podremos crear un nuevo proyecto vacío, abrir uno existente en disco, cambiar las propiedades del proyecto actual o visualizar los ficheros de registro del sistema<sup>2</sup>.

Las propiedades que definen un proyecto son:

- **Nombre del proyecto.**
- **Autor del proyecto.**
- **Descripción breve del proyecto.**
- **Nombre del fichero del proyecto** (con la extensión .prj).
- **Nombre del fichero MSC** (con la extensión .msc).
- **Nombre del fichero Latex** se usa para dibujar los diagramas MSC por pantalla (extensión .tex).

---

<sup>1</sup>La aplicación ha sido probada en Debian versión SID, DeMudi y Ubuntu.

<sup>2</sup>Estos logs muestran información de validez de los documentos XML de trabajo y otras cuestiones internas.

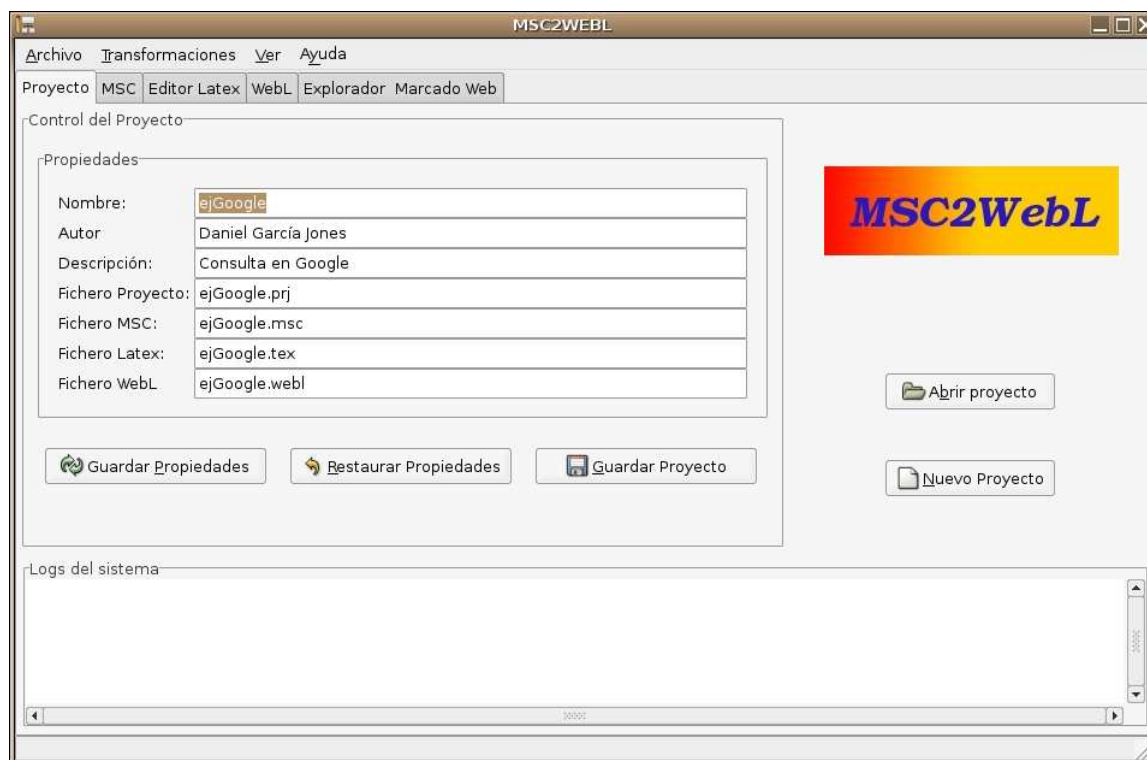


Figura A-2: Ventana de propiedades de proyecto

- **Nombre del fichero WebL** donde se guardará el código generado (extensión .WebL).

Tanto el nombre del proyecto como el nombre del fichero de proyecto son campos obligatorios. Para el resto de ficheros, si no se especifica lo contrario, el nombre es el mismo que el del proyecto seguido de la extensión adecuada.

### A-2.2. Vista Editor MSC

Es la ventana más importante de las que componen la aplicación. Consta de un editor para crear los documentos y diagramas MSC. Para ello se incluyen una serie de botones que permiten crear de forma sencilla los distintos diagramas. También se ofrece la posibilidad de explorar el código (generado siguiendo un esquema XML definido a propósito) o incluso crear los distintos sistemas empleando esta sintaxis textual.

Además incluye una serie de controles para validar los documentos XML, visualizar la representación escrita en forma gráfica, y generar los programas WebL. El aspecto de esta ventana es el de las figuras A-3 y A-4

### A-2.3. Vista Latex

En esta ventana podemos ver y editar el código  $\text{\LaTeX}$  generado para representar gráficamente los diagramas y documentos MSC. Este código se genera a partir de una plantilla XSLT con una serie de valores predefinidos. Se puede modificar dicha plantilla para que el resultado de la transformación MSC a Latex sea diferente<sup>3</sup>.

Existen tres tamaños predeterminados para los diagramas: *pequeño*, *mediano* y *grande*, que pueden elegirse directamente en la ventana de edición MSC. y mediante el editor incluido se pueden añadir todos los textos o elementos que se quiera.

Además en esta vista tenemos una serie de controles para guardar a disco el fichero  $\text{\LaTeX}$  y generar la documentación en formato PDF. El aspecto de esta venta aparece en la figura A-5.

<sup>3</sup>No conviene hacerlo si no se tienen conocimientos de  $\text{\LaTeX}$  y del paquete para dibujar diagramas MSC.



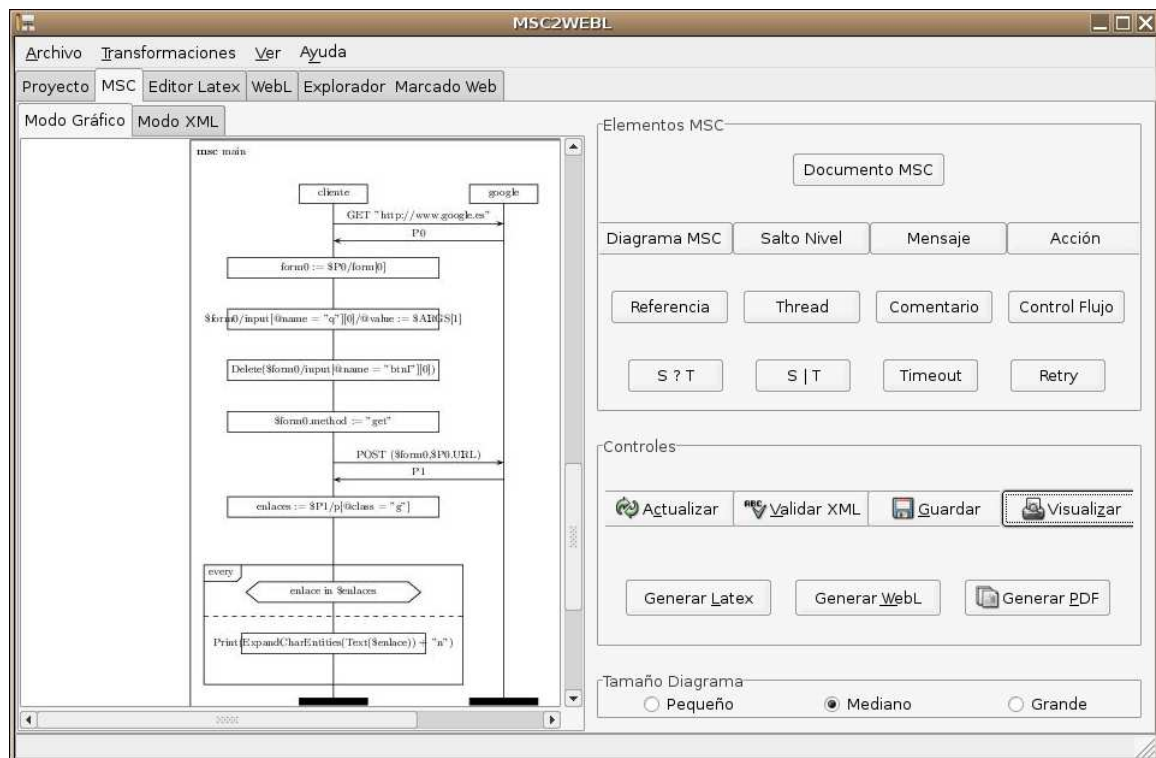


Figura A-3: Ventana del editor MSC (Representación Gráfica)

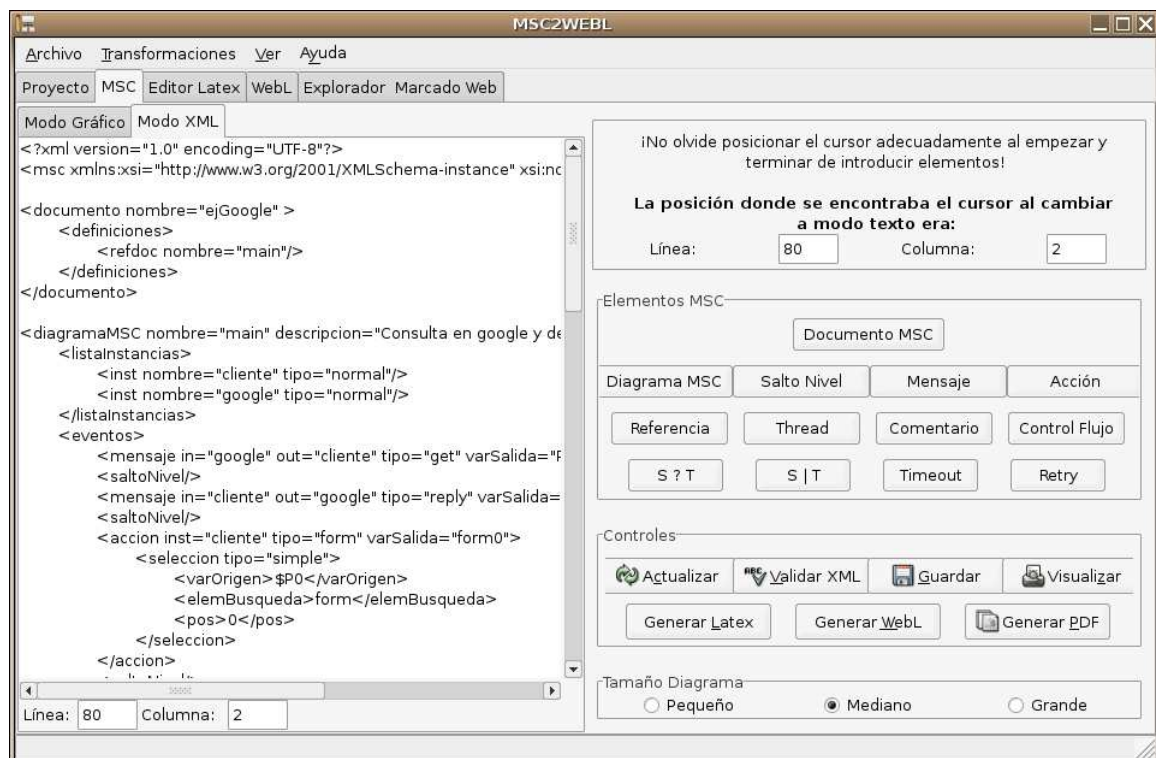


Figura A-4: Ventana del editor MSC (Representación XML)

### A-2.4. Vista WebL

Esta ventana contiene los elementos necesarios para la edición y ejecución del código WebL generado. Por un lado, incluye un sencillo editor donde podemos ver el código generado automáticamente e incluir modificaciones si fuera necesario; por el otro, incluye un entorno de ejecución de los programas WebL generados.

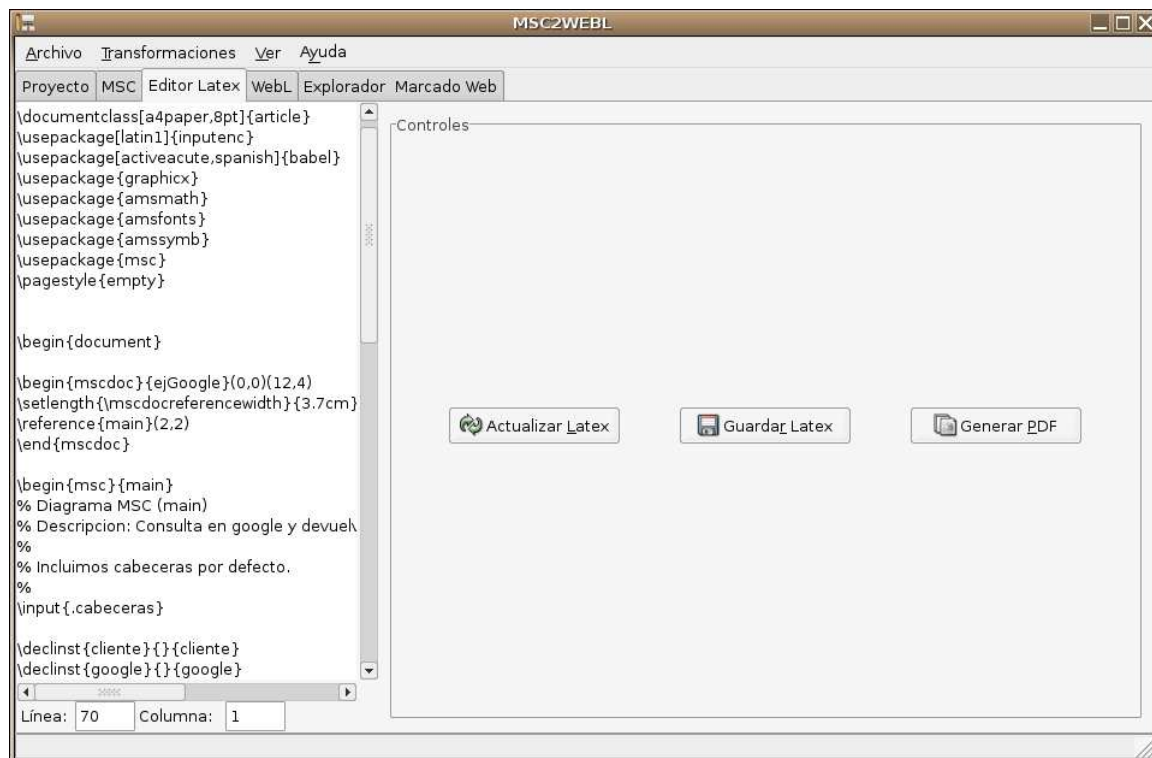


Figura A-5: Ventana del editor LaTeX

El aspecto de esta vista es el de la figura A-6.

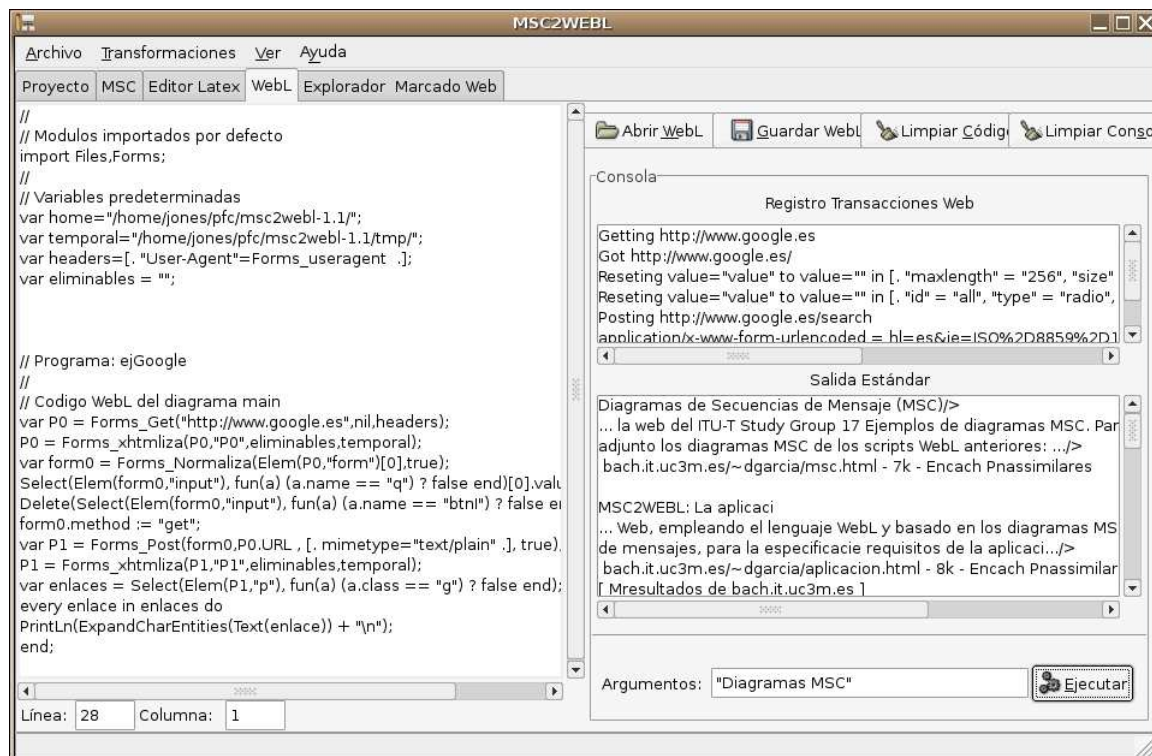


Figura A-6: Ventana WebL (edición y ejecución)

El código generado automáticamente mediante una transformación XSLT incluye una serie de módulos y variables predefinidas y hace uso de una librería para tratar el envío y recepción de formularios y páginas.

Esta librería, `Forms.Web1`, se encuentra en el directorio `bin`.

La consola de ejecución de programas WebL esta compuesta de 3 elementos principales:

- **Registro de transacciones Web** En esta venta vemos las transacciones HTTP que van surgiendo en tiempo de ejecución y algunos mensajes de error del intérprete WebL.
- **Salida estándar** La salida por pantalla del script ejecutado aparece en esta vista de texto.
- **Argumentos de ejecución** Dependiendo del programa que estemos generando puede ser necesario introducir argumentos (por ejemplo login y password). Dentro de los diagramas MSC y del código WebL las variables que representan a los argumentos se denominan como en el lenguaje C (`ARGS[1]`, `ARGS[2]`,...).

Para ejecutar los programas basta con introducir los argumentos si fueran necesarios en la caja de texto de argumentos y presionar el botón ejecutar. Si el programa fuera interactivo (requiriera entrada de teclado) no sería posible ejecutarlo en la consola integrada.

### A-2.5. Vista Explorador de Marcado Web

Uno de los aspectos que más hay que estudiar a la hora de programar para la Web es el marcado interno de las páginas que se quieren procesar. A menudo las páginas siguen estilos muy diferentes y el código HTML para dos páginas en apariencia similar puede llegar a tener marcados muy diferentes. Para hacer más fácil la inspección del marcado de las páginas, se ha incluido un navegador Web, que permite ver el código de la página indicada mediante una URL. Además, al igual que se puede hacer en los programas WebL, es posible eliminar una serie de etiquetas del código recibido antes de mostrarlo, de tal forma que podemos eliminar elementos de estilo o etiquetas que no contengan la información que nos interesa. El programa encargado de obtener las páginas y eliminar las etiquetas necesarias es un script WebL incluido con la aplicación<sup>4</sup>. El aspecto de la ventana de navegación es el de la figura A-7.

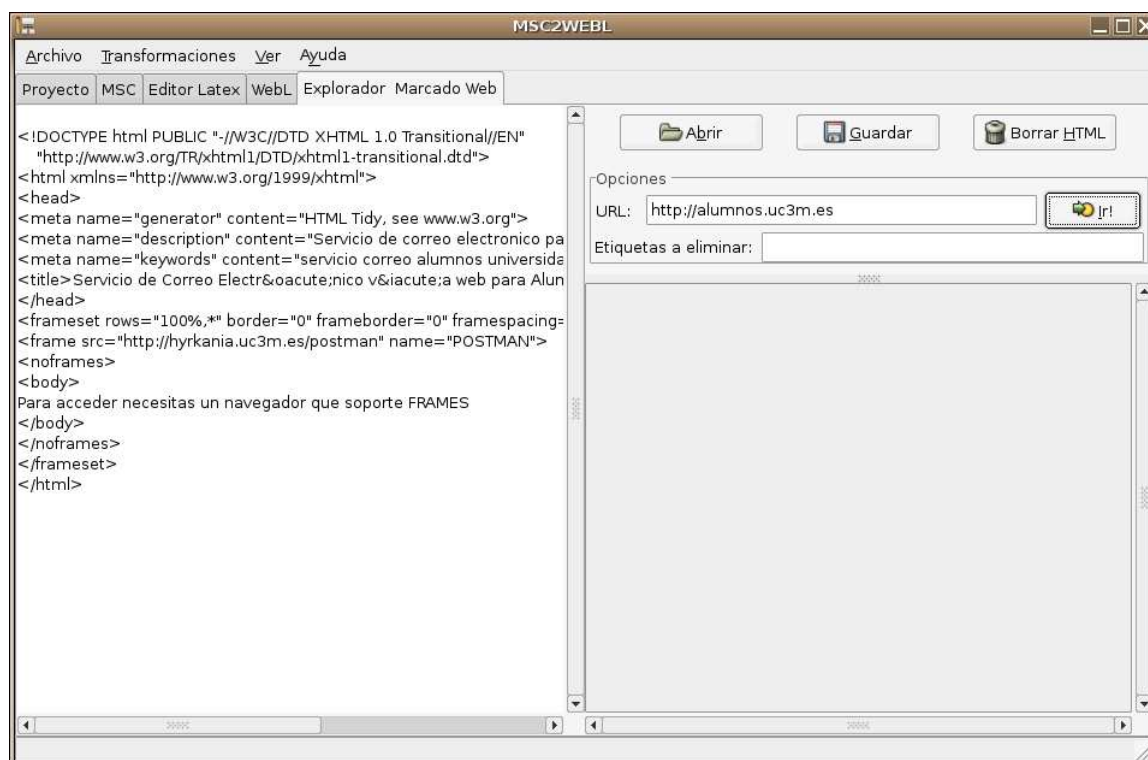


Figura A-7: Ventana Navegador

<sup>4</sup>Para entender el mecanismo de servicios de WebL se puede ojear este código. Se encuentra en `bin/descargaPagina.Web1`.

### A-3. Descripción de tareas Web mediante MSC

El objetivo de la aplicación es ayudar a desarrollar y mantener de forma sencilla scripts de navegado automático de la red. Para ello es vital entender en profundidad cómo una tarea Web típica puede describirse mediante diagramas MSC y cómo estos se editan con la aplicación MSC2WEBL.

Estructuralmente descomponemos todo proyecto en un documento MSC y una colección de diagramas MSC que combinados de la forma descrita en el documento MSC dan lugar al programa final. Vemos en detalle cada uno de los posibles elementos.

#### A-3.1. Documentos MSC

Definen de forma clara y sencilla la estructura de un proyecto MSC2WEBL. Cada documento MSC tiene asociado una colección de diagramas MSC básicos. Estos describen de una forma modular el comportamiento del programa que se quiere generar. Los datos que almacena un documento MSC son:

- Nombre del documento;
- Opcionalmente puede incluirse una descripción de la funcionalidad del programa;
- Un cuerpo del documento que se descompone en dos partes: **parte de definiciones** y **parte de utilidades**.

La parte de definiciones incluye una referencia por cada uno de los diagramas MSC que definen el programa principal. A la hora de generar el código WebL a partir de los diagramas MSC referenciados en la parte de definiciones, se ensamblan uno a continuación del otro, por orden de aparición en el documento los diagramas referenciados en la parte de definiciones.

La parte de utilidades incluye referencias a los módulos (otros diagramas MSC's) que son llamados desde dentro de las referencias incluidas en la parte de definiciones. Esto permite la definición de funciones auxiliares que pueden ser llamadas en cualquier punto del programa principal.

En la figura A-8 se representa un ejemplo de documento MSC. La parte de definiciones incluye únicamente a *fileDownloading*. *connection*, *fileRequest*, *fileSearch* y *fileDownload* componen la parte de utilidades.

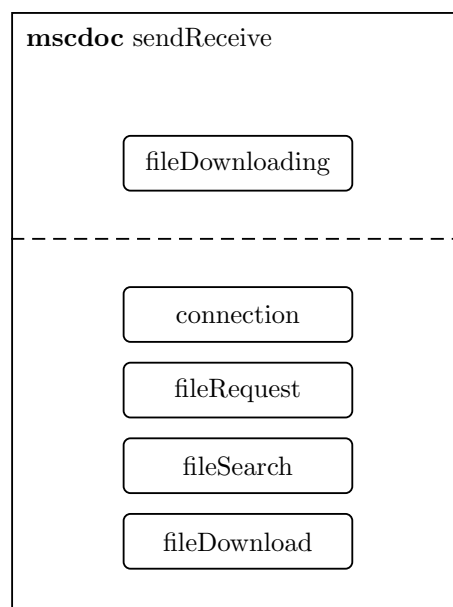


Figura A-8: Documento MSC fileDownloading

El diálogo para crear un nuevo documento MSC desde el editor MSC tiene el aspecto de la figura A-9. Para incluir en el documento una parte de definición o utilidades, basta con indicar los nombres de las referencias que queremos crear en la casilla de texto correspondiente del diálogo de creación de Documentos.

Se puede crear una función que devuelva un valor de salida, típicamente en la parte de utilidades. Para ello, habrá que indicar los parámetros, entre paréntesis y separados por comas, y la posible variable de salida. Las distintas referencias, tanto en la parte de definiciones como de utilidades se introducen separadas por espacios.



Figura A-9: Dialogo Nuevo Documento MSC

### A-3.2. Diagramas MSC

Un diagrama MSC representa un conjunto de tareas Web entre los distintos componentes del sistema, representados por **entidades**. Las tareas que se pueden realizar son el intercambio de ficheros y páginas Web, envío de formularios, procesamiento de los datos, definir condiciones de flujo, combinadores de servicio, etc.

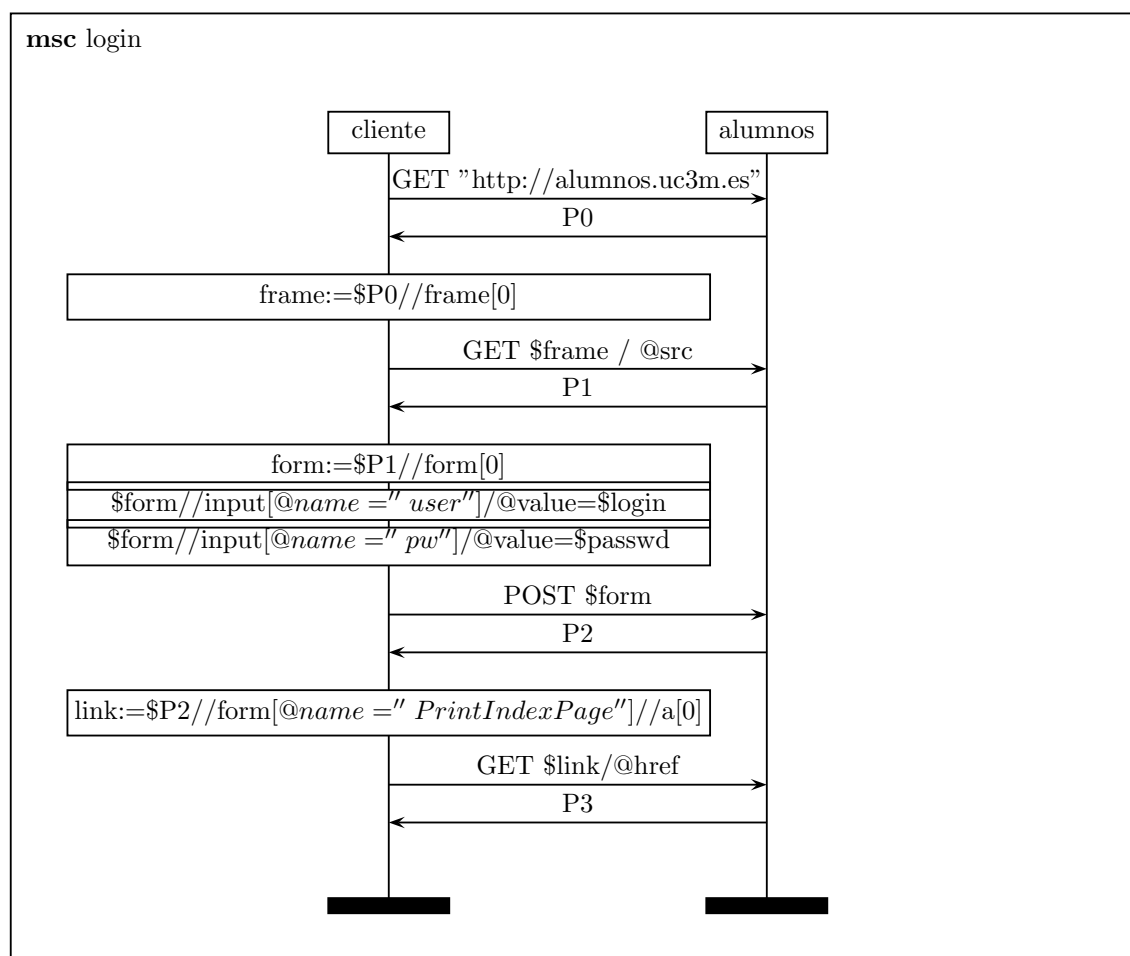


Figura A-10: Ejemplo de diagrama MSC

No pueden existir dentro de un mismo proyecto diagramas MSC de mismo nombre. Opcionalmente tam-

bién se puede incluir datos sobre el autor del diagrama y una pequeña descripción del mismo.



Figura A-11: Dialogo Nuevo Diagrama MSC

Un ejemplo de diagrama MSC es el de la figura A-10. El diálogo para crear un nuevo diagrama MSC está representado en la figura A-11.

Para crear un nuevo diagrama, bastará con indicar su nombre y las entidades presentes en dicho diagrama. El resto de parámetros son opcionales. Al crear el diagrama habrá que indicar el nombre de todas las entidades presentes en el orden en que queramos que aparezcan dibujadas en el diagrama. Una entidad dinámica (creada por un Thread de ejecución) se antecede de \*.

### A-3.3. Entidades

Una entidad representa a uno de los elementos presentes en el sistema. Generalmente, en todo desarrollo que hagamos, habrá obligatoriamente una entidad, a menudo denominada *Cliente*, que representará nuestro ordenador. Este ordenador cliente interactuará con otra u otras entidades para resolver una tarea dada. El símbolo de entidad aparece en la figura A-12.

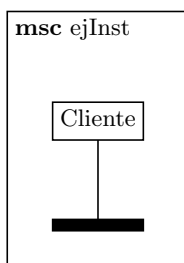


Figura A-12: Ejemplo Entidad

Las entidades se definen al crear el diagrama MSC en que aparecen, empleando para ello el propio diálogo de creación de diagramas MSC, separadas todas ellas por espacios. Una vez creada la entidad, su nombre se introduce en una lista de entidades con las que trabaja el proyecto actual. Esto es así por que para toda acción posible hay siempre que indicar a que entidad o entidades está asociada dicha acción. El eje vertical de cada entidad representa la evolución temporal, de forma que eventos dibujados gráficamente más hacia abajo ocurren después que los situados por encima. Distinguimos entre entidades normales y dinámicas (creadas en tiempo de ejecución). Estas últimas se definen antecediendo de \* el nombre de la entidad.

### A-3.4. Mensajes

La operación más básica que tendremos entre un par de entidades es el envío y recepción de mensajes. Estos mensajes pueden representar el envío de un formulario Web, o la devolución de una página con resultados de una búsqueda, por ejemplo. Existen los siguientes tipos de mensaje:

- get (obtiene una página o fichero, sin almacenarlo en el disco duro).
- post (envía un formulario).

- **head** (obtiene las cabeceras HTTP).
- **getfile** (obtiene una página o fichero y lo guarda en el disco duro).

Por cada mensaje enviado, sea del tipo que sea, automáticamente se genera una respuesta en sentido contrario. La aplicación genera esta respuesta de forma automática, con lo que no es necesario hacerlo manualmente. Los parámetros para todos los tipos de mensaje son:

- **Origen** Entidad que envía el mensaje;
- **Destino** Entidad receptora del mensaje;
- **URL** del objeto que se quiere enviar/recibir;
- **Tipo** del mensaje a enviar;
- **Variable de salida** Nombre de la variable donde se almacenará el código de las páginas descargadas o recibidas;
- **Formulario** Para el caso de mensaje tipo *post* es necesario indicar el formulario que se va enviar. Previamente se habrán introducido los valores adecuados en el mismo;

Un ejemplo de mensaje:

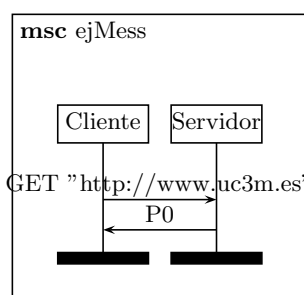


Figura A-13: Ejemplo mensaje

Figura A-14: Dialogo Nuevo Mensaje

### A-3.5. Referencias

Estas referencias pueden devolver una variable de salida y tomar una serie de parámetros de entrada (separados por comas). Un ejemplo de diagrama MSC que incluye una referencia aparece en la figura A-15. La sintaxis empleada es `var_salida = referencia(par_1,par_2,...,par_n)`.



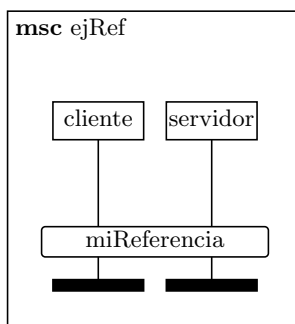


Figura A-15: Ejemplo de referencia MSC



Figura A-16: Dialogo Nueva Referencia

### A-3.6. Control de flujo

En un proyecto MSC2WEBL típico podrán aparecer las siguientes expresiones de control de flujo:

- If-Then
- If-Then-Else
- Repeat-Until
- While
- Every

Para expresarlas en MSC se emplean *expresiones inline*. Como vemos en la figura A-18 las expresiones de control de flujo pueden anidarse. La última expresión, *Every*, es muy útil en computación Web, donde recorrer iterativamente todos los elementos de un determinado tipo en una página (enlaces, imágenes,...) es una operación muy común.

Para crear una nueva expresión de control de flujo empleamos el diálogo de Control de Flujo (figura A-17), donde, una vez seleccionado el tipo de expresión, habremos de indicar la condición a evaluar y la entidad afectada por la expresión y, una vez hecho esto, especificar las operaciones que se quieren realizar dentro de cada argumento. Para ello, habrá que presionar el botón "Comenzar", antes de introducir las operaciones dentro de cada bucle. Una vez no podamos introducir más operaciones, porque ya hayamos acabado con todos los argumentos, los distintos controles dejarán de ser activos, para evitar errores. Una vez presionado el botón "Aceptar", se dibujará la expresión por pantalla. Todas las expresiones tienen un único argumento salvo *If-Then-Else* que tiene dos.

Las condiciones a evaluar serán en todo caso expresiones que devuelvan un valor booleano. En el caso de la expresión *Every*, la condición a evaluar será siempre del tipo **A in B**, que quiere significar que toma uno a uno los elementos de B y que en cada ciclo del bucle el elemento seleccionado (sobre el que se opera) se identifica como A.

### A-3.7. Acciones

Además del intercambio de mensajes, pueden definirse acciones en los MSC que representan las operaciones posibles sobre los datos obtenidos mediante ese intercambio de mensajes. El lenguaje de especificación



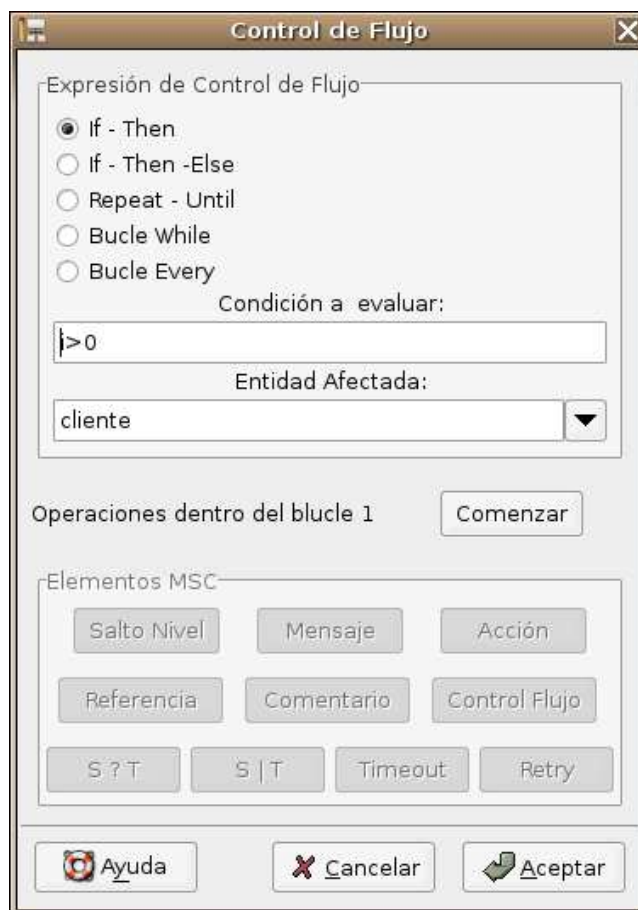


Figura A-17: Dialogo Nueva Expresión de Control de Flujo

que se emplea para expresar las acciones es *XPath* ya que permite de forma sencilla referenciar cualquier elemento dentro de un documento XML (las páginas obtenidas se convierten automáticamente a XHTML y se corrige su sintaxis al ser recibidas). No es necesario tener un conocimiento profundo de XPath puesto que empleando el editor estas expresiones XPath se construyen automáticamente. Además existe la posibilidad de escribir directamente código WebL y se han incluido en el modelo una serie de funciones útiles para la programación Web (ver Funciones Misceláneas).

En MSC el símbolo de acción es una caja de texto conectada al eje de la entidad (ver figuras A-10 y A-18).

Hemos definido 7 tipos básicos de acciones MSC2WEBL:

- Procesamiento de formulario.
- Creación de nueva variable.
- Asignación de valor a una variable ya existente.
- Borrado de algún elemento (por ejemplo un campo de un formulario).
- Impresión por pantalla de una cadena o una variable.
- Salvar a disco.
- Expresión WebL.

Muchas de las acciones<sup>5</sup> requieren seleccionar elementos dentro de una página. Podremos hacer 3 tipos de selecciones, independientemente de lo que se quiera hacer con los elementos seleccionados. Estos son:

<sup>5</sup>Conviene a la hora de procesar acciones en la aplicación tener ciertos conocimientos sobre las posibles etiquetas XHTML que pueden existir en una página. Para ayudar a los desarrollos se puede emplear la vista **Navegador Web** para inspeccionar el código de las páginas que quieren procesarse para identificar los elementos de interés.

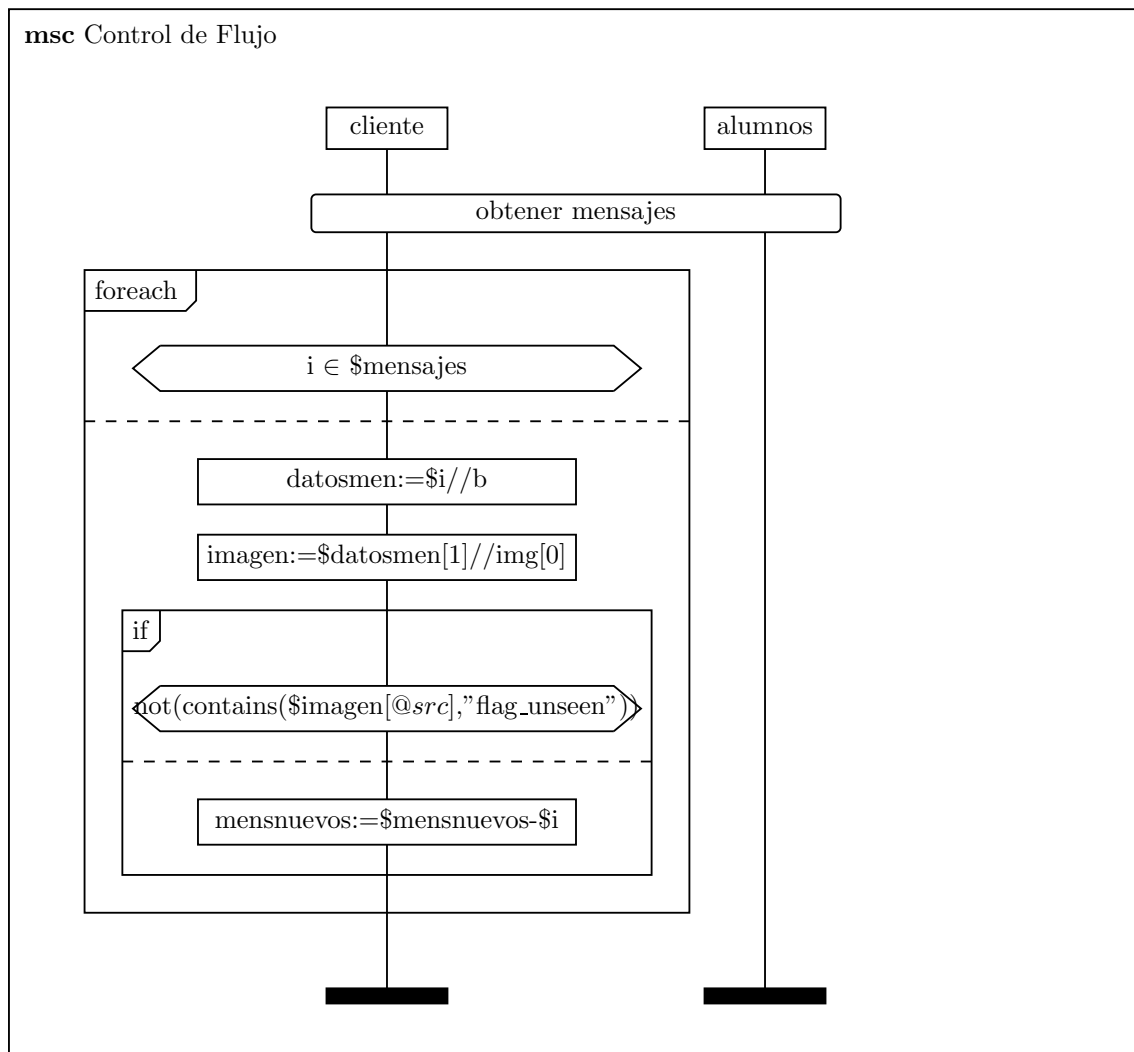


Figura A-18: Ejemplo de expresiones de control de flujo anidadas

#### ■ Identificador de variable

- **Selección simple** Permite seleccionar un elemento de un tipo determinado (identificado por el nombre de su etiqueta) en una posición dada. Si no se especifica posición, se seleccionan todos los elementos del tipo dado que aparezcan en la página (o variable origen). Por ejemplo: la variable de origen puede ser "P0" (la última página recibida) y los elementos de búsqueda pueden ser las imágenes, marcadas con la etiqueta "img". La expresión XPath del ejemplo anterior es: `$P0/img/*`.
- **Selección compleja** podremos afinar más la búsqueda de elementos seleccionando aquellos de cierto tipo que tengan un atributo con un valor determinado. Por ejemplo podremos seleccionar dentro de un formulario una casilla *input* de nombre `login` para posteriormente introducir en ella el valor apropiado. La expresión XPath de este ejemplo es `$form0/input[@name="login"]`.

Además de seleccionar elementos de una página también habremos de introducir ciertos valores en las distintas variables que nos podamos ir encontrando. La asignación de valores se hace de forma parecida a la de selección. Tendremos 4 tipos de asignación:

- **Cadena de texto** que ha de ir SIEMPRE entrecomillada<sup>6</sup>.
- **Identificador de variable** (sin comillas).
- **Selección simple**.
- **Selección compleja**.

<sup>6</sup>MSC2WEBL emplea comillas simples para notar las cadenas de texto.

En función del tipo de acción que queramos definir los campos habilitados del diálogo de creación de acciones serán unos u otros. Vemos esto con mayor profundidad.

#### A-3.7.1. Selección de formularios

Una de las tareas típicas a la hora de establecer una sesión Web consiste en rellenar formularios dentro de páginas Web para enviarlos de vuelta al servidor con los datos necesarios para obtener una respuesta adecuada en función de los mismos. Seleccionar un formulario en WebL<sup>7</sup> implica una serie de pasos, muchos de los cuales los hace automáticamente la aplicación. Otros sin embargo requieren de algunos parámetros que el usuario ha de introducir. Para seleccionar un formulario dentro de una página ya recibida empleamos el diálogo de procesado de acciones (seleccionando acción tipo **Form**). El diálogo tiene el aspecto de la figura A-19.

Figura A-19: Dialogo Nueva Acción de selección de formulario

Los únicos campos editables, que coinciden con aquellos parámetros necesarios para la selección son:

- **Entidad** a la que se asocia la acción (normalmente del lado del cliente).
- **Variable de salida** Nombre con que nos referiremos posteriormente en el programa al formulario seleccionado.
- **Campos de selección** Podremos elegir entre selección por identificador de variable, selección simple o selección compleja.

#### A-3.7.2. Creación de una nueva variable

Se puede crear nuevas variables que contengan partes de interés dentro de una página, por ejemplo el conjunto de enlaces de la página, la tercera imagen del interior de una tabla etc. Para ello hay que definir el nombre de salida de la variable a crear y asignarle un valor. La casilla de asignación de valor (en el GUI identificada como *Valor*) permite asignar una cadena de texto (entre comillas), una variable (identificador sin comillas), o una selección de tipo simple o complejo.

<sup>7</sup>Esto es así porque se emplea una biblioteca especial, *Forms.Webl* para normalizado y procesado de formularios.

Constructor de acciones

Tipos de Acciones:

☐ Form ☒ Nueva Variable ☐ Asignación ☐ Borrado ☐ Print ☐ Salvar Disco ☐ WebL

Entidad: cliente

Var. Salida: cadena

Selección:

☐ Variable ☐ Selección Simple ☒ Selección Compleja

Identificador Var:

Var. Origen:

Elem. Búsqueda:

Atributo Búsqueda:

Valor Atributo:

Posición:

Parámetros Asignación/Borrado:

Atributo a modificar/borrar:

Salvar Disco:

Nombre Fichero:

Valor:

☒ Variable o Cadena de texto ☐ Selección Simple ☐ Selección Compleja

Variable o Texto: 'Posicion' + &i

Var. Origen:

Elem. Búsqueda: head

Atributo Búsqueda: name

Valor Atributo:

Posición:

WebL:

Expresión WebL:

Ayuda Cancelar Aceptar

Figura A-20: Dialogo Nueva Acción de creación de variable

### A-3.7.3. Asignación de valor a una variable

Una vez creada, podremos variar el valor de una variable o incluso de uno de sus atributos. Esto permite entre otras cosas introducir el login o el password dentro del campo correcto de un formulario. En caso de que se quiera variar el valor de un atributo de una variable seleccionada lo haremos indicando el nombre del atributo en el campo apropiado y asignando el valor adecuado. Si no se especifica atributo, se cambia el valor de la variable. El siguiente ejemplo, representado en la figura A-22 contiene varias acciones de procesamiento de formularios (una selección y una asignación).

### A-3.7.4. Borrado de elementos

Se pueden borrar atributos de algún elemento, por ejemplo para desactivar una casilla de verificación en un formulario. Simplemente hay que seleccionar que variable es la afectada e indicar el atributo a ser eliminado.

### A-3.7.5. Impresión por pantalla

Podremos imprimir por pantalla tanto cadenas de texto, como el contenido de variables o selecciones. Para imprimir el código de una página descargada guardada en la variable P0 (por ejemplo), lo que pondremos en la casilla *Variable o Texto* será Markup(P0). El diálogo de acción tipo impresión aparece en la figura A-23.

### A-3.7.6. Salvar a disco

Podremos salvar a disco el código (o parte de él) de las páginas descargadas, para ello al seleccionar la opción *Salvar Disco* habrá que seleccionar el elemento que se quiere guardar a disco y el nombre con el que lo salvaremos. Normalmente se guardan a disco las páginas descargadas o parte de ellas, aunque también se puede guardar texto o incluso imágenes.

### A-3.7.7. Expresión WebL

Podremos introducir directamente una expresión WebL. Para ello basta con teclear la opción *WebL* y escribir la expresión en la caja de texto editable.

Figura A-21: Dialogo Nueva Acción de asignación

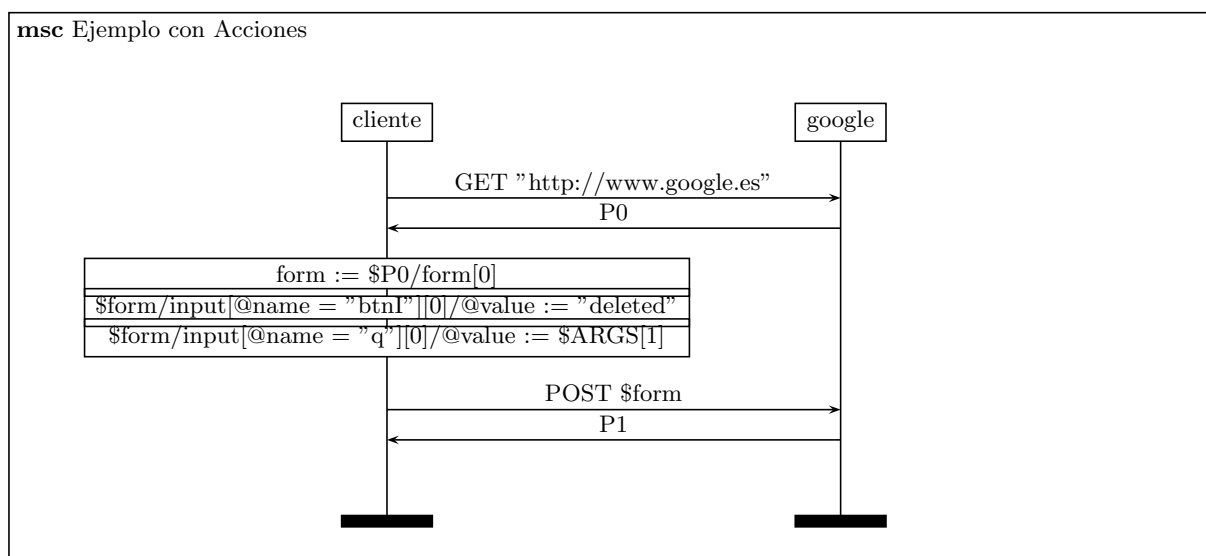


Figura A-22: Ejemplo de acciones

### A-3.8. Descripción de combinadores de servicio

Con el fin de reproducir el comportamiento humano al navegar a través de la Web, es necesario tener en cuenta sus reacciones ante tasas de transmisión bajas y caídas de ciertos enlaces. Cardelli y Davies definieron en 1996 un formalismo para tener en cuenta los conceptos de fallo y tasa de transmisión, los *Combinadores de Servicio*, que pueden ser incorporados a los distintos modelos de programación para la Web, basados en el protocolo HTTP.

El servicio básico con el que trabajaremos será una petición o envío de una pagina Web desde o hacia una URL. Este servicio puede realizarse satisfactoriamente o fallar. La idea es combinar distintos servicios que puedan estar sujetos a fallos para obtener "servicios virtuales" más fiables. Los combinadores de servicio incluidos en MSC2WEBL son:

Figura A-23: Diálogo Nueva Acción de impresión por pantalla

- Combinador secuencial.
- Combinador concurrente.
- Timeout.
- Retry.

#### A-3.8.1. Combinador Secuencial ( $S_1?S_2$ )

Dados dos servicios  $S_1$  y  $S_2$ , primero ejecutamos  $S_1$  y si falla entonces  $S_2$ . En caso de que los dos servicios fallen, la combinación también falla. Un ejemplo de combinador secuencial y su código XML aparece en las figuras A-24 y A-26.

Para introducir un combinador secuencial (e igual para el resto de combinadores), simplemente basta con indicar las entidades afectadas por el combinador (los elementos que intervienen en la comunicación) y el nombre de la variable de salida fruto del servicio con el que nos referiremos a la página obtenida en el resto del programa. Para indicar las entidades afectadas se introduce el nombre de la primera y la última de ellas separadas una coma<sup>8</sup>. Esto hace que sea importante el orden de definición de las entidades en el programa, ya que toma como afectadas por el combinador de servicio a todas las entidades entre la primera y la última indicada. Una vez definidos los distintos parámetros, introduciremos las operaciones de cada uno de los servicios de la misma forma que hicimos para el caso de expresiones de control de flujo, indicando el cambio de operando pulsando el botón a tal efecto.

Las operaciones permitidas dentro de un operando de combinador de servicio son:

- Intercambio de mensajes.
- Otros combinadores de servicio (anidados).
- Stall (sólo presente en el interior de combinadores Timeout).
- Salto de nivel.
- Comentario.

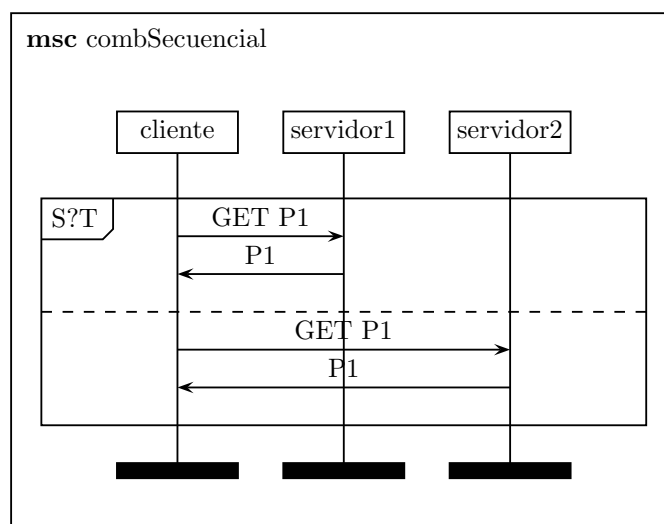


Figura A-24: Ejemplo de combinador secuencial



Figura A-25: Diálogo Nuevo combinador secuencial de servicio

Como se aprecia en la figura ??, el símbolo para un combinador de servicio es el mismo que el de las expresiones de control de flujo, únicamente cambia la leyenda que identifica el tipo de expresión de que se trate. El funcionamiento del resto de combinadores es similar, únicamente variará el número de operandos de cada combinador.

### A-3.8.2. Combinador Concurrente ( $S_1|S_2$ )

Se ejecutan  $S_1$  y  $S_2$  concurrentemente y se devuelve el resultado del que primero termina. Si los dos fallan la combinación también.

### A-3.8.3. Timeout( $t,S$ )

El combinador funciona como el servicio  $S$  excepto que falla si después de  $t$  segundos  $S$  no ha finalizado. Al contrario que para los combinadores secuenciales y concurrente, el combinador *Timeout* se asocia únicamente a una entidad, la de origen de los mensajes y tiene un único operando. En la figura A-28 vemos la representación gráfica del combinador *Timeout* diferente de la del resto de combinadores. El diálogo de creación de combinadores tipo *Timeout* es el de la figura A-29. Dentro de un combinador *Timeout* puede aparecer una operación de tipo *Stall*, en que la entidad afectada se queda congelada (sin hacer operaciones) hasta que el temporizador venza.

<sup>8</sup>Para evitar errores se nos muestra por pantalla una lista de las entidades presentes en todo momento en cada diagrama.

```

<?xml version="1.0" encoding="UTF-8"?>
<msc xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="../xml/msc.xsd">

  <documento nombre="google" >
    <definiciones>
      <refdoc nombre="main"/>
    </definiciones>
  </documento>

  <diagramaMSC nombre="main" >
    <listaInstancias>
      <inst nombre="cliente" tipo="normal"/>
      <inst nombre="google" tipo="normal"/>
      <inst nombre="google2" tipo="normal"/>
    </listaInstancias>
    <eventos>
      <combseq varSalida="P1">
        <listaInst>
          <inst nombre="cliente" tipo="normal"/>
          <inst nombre="google2" tipo="normal"/>
        </listaInst>
        <operando1>
          <mensaje out="cliente" in="google" tipo="get" varSalida="P1">
            'http://www.google.es'
          </mensaje>
          <saltoNivel/>
          <mensaje out="google" in="cliente" tipo="reply" varSalida="P1">P1</mensaje>
        </operando1>
        <operando2>
          <mensaje out="cliente" in="google2" tipo="get" varSalida="P1">
            'http://www.google.com'
          </mensaje>
          <saltoNivel/>
          <mensaje out="google2" in="cliente" tipo="reply" varSalida="P1">P1</mensaje>
        </operando2>
      </combseq>
    </eventos>
  </diagramaMSC>
</msc>

```

Figura A-26: Código XML de un combinador de servicio

#### A-3.8.4. Repetición

Se repite el servicio S hasta que finaliza correctamente. En la figura A-30 vemos el diálogo de creación de combinadores de tipo *Repetición*. La figura A-31 representa un ejemplo de este combinador.

Los combinadores pueden generar fallos, es decir, las páginas seleccionadas pueden no llegar a descargarse, con lo que la ejecución del diagrama MSC debe finalizar en el momento en que falla alguno de los servicios combinados. En la aplicación MSC2WEBL no se tiene en cuenta esta circunstancia, se supone que todos los servicios finalizan correctamente, si esto no fuera así habría que emplear el mecanismo de excepciones interno de Webl, lo que no está contemplado en la aplicación.

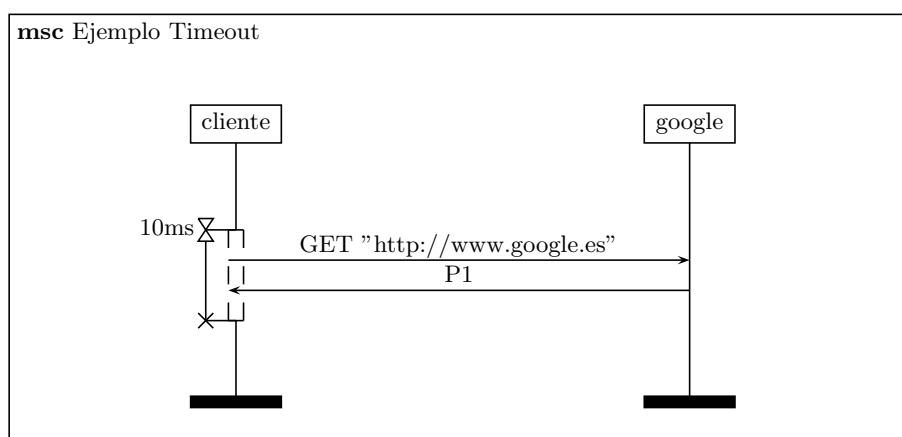
#### A-3.9. Threads

Se permiten estructuras concurrentes de ejecución de la forma representada en la figura A-32. Es importante notar la diferencia entre el combinador concurrente y los threads, ya que en un combinador concurrente el servicio que finaliza con éxito cancela al otro, mientras que en caso de ejecución de dos threads paralelos,





Figura A-27: Diálogo Nuevo combinador concurrente de servicio

Figura A-28: Ejemplo de *Timeout*

ambos se ejecutan hasta el final y el que termina primero, no sólo no aborta el otro, si no que espera a que termine su ejecución. Además dentro de un thread podremos introducir cualquier código, combinadores de servicio y acciones incluidos. El diálogo de creación de threads aparece en la figura A-33.

### A-3.10. Comentarios

Podremos introducir comentarios en los MSC con el fin de aclarar partes del código o por cuestiones de documentación.

### A-3.11. Variables y tipos de datos

El modelo no hace un tratamiento específico de las variables y los tipos de datos posibles son aquellos presentes en WebL. Una variable en XPath y por tanto en el modelo propuesto, consta de un identificador único. La referencia a estas variables se representa antecediendo el símbolo "\$" al identificador de la misma. Sin embargo en todos los diálogos de la aplicación en los campos de identificador de variable, no será necesario teclear el símbolo \$, ya que la aplicación lo inserta automáticamente en los diagramas. Los tipos de datos posibles específicos para la programación Web, además de los propios de lenguajes de programación tradicionales (int, bool, real, char, string, list, set) presentes en WebL son:

- **Página:** representa una página descargada. Incluye tanto su código HTML como una serie de propiedades entre las que se incluye su URL;
- **Parte:** identifica una subregión contigua de una página;



Figura A-29: Diálogo Nuevo combinador de servicio timeout



Figura A-30: Diálogo Nuevo combinador de servicio repetición

- **Conjunto de partes:** una colección de partes;

Para acceder a alguna de las propiedades de una variable de los tipos anteriores se hace indicando el nombre del atributo a continuación del identificador de variable y precedido por un punto. Por ejemplo, `pag.URL` devuelve dicho atributo de la variable `pag`.

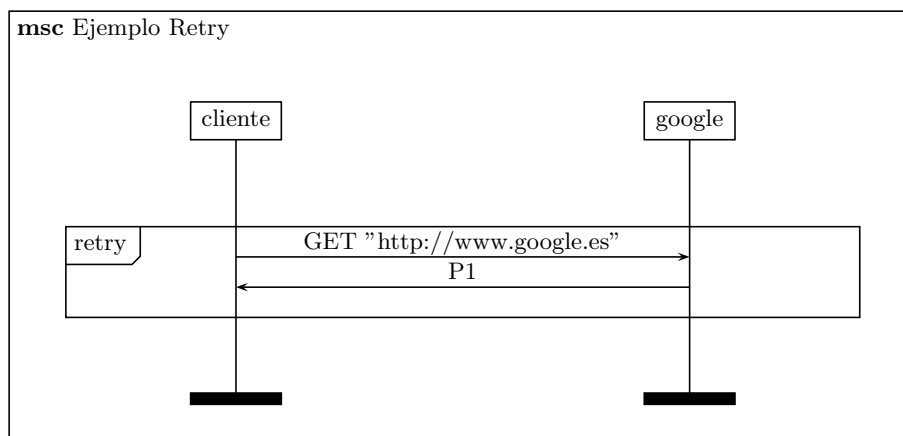
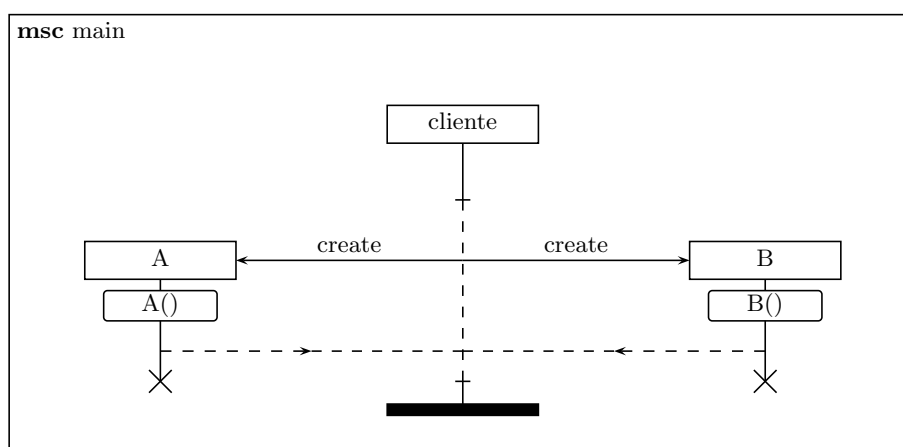
Figura A-31: Ejemplo de combinador *Repetición*

Figura A-32: Ejemplo de Threads



Figura A-33: Diálogo Nuevo Thread de ejecución



Figura A-34: Diálogo Nuevo comentario

### A-3.12. Funciones Misceláneas

Para la automatización de tareas en la Web es necesario incluir varias funciones para ayudar al procesamiento de la información. Conviene destacar dos funciones útiles del lenguaje WebL incluidas en el

modelo.

#### A-3.12.1. Markup

Bien sea para construir una nueva página, o para analizar el código HTML de una página descargada o de parte de ella, resulta apropiada la definición de una función que sea capaz de devolver el código HTML de una variable dada. WebL incluye la función *Markup()*, de forma que es sencillo calcular este marcado.

#### A-3.12.2. Text

Una página HTML o una parte de ella, contiene abundante información textual embebida, que en muchos de los casos ha de ser accedida y procesada. La función *Text()* devuelve todo el texto contenido en la variable argumento, que ha de ser de tipo página, parte o conjunto de partes.

### A-4. Opciones de configuración de la aplicación

Varios son los parámetros de configuración de MSC2WEBL. Estos se guardan en un fichero XML en el directorio de la aplicación, y pueden ser cambiados en todo momento por el usuario desde la barra de menú (*Ver -> Opciones*) tal y como se muestra en la siguiente figura.



Figura A-35: Diálogo opciones de la aplicación

Las distintas opciones presentes son:

- **Directorio de la aplicación MSC2WEBL** (por defecto es `/msc2webl-1.1`).
- **Directorio almacén de proyectos.**
- **Directorio temporal** (para guardar logs, y ficheros temporales).
- **Eliminables** indica las etiquetas XHTML que han de ser eliminadas directamente de toda página descargada. La lista de las etiquetas debe estar separada por espacios. Por ejemplo "tr td table center font".
- **Módulos** WebL empleados en el proyecto actual. Los módulos cargados por defecto son Forms y Files.
- **Cabeceras** con que se identifica WebL al acceder a cada página. Su uso permite identificar a nuestro script como un navegador existente a la hora de acceder a un servidor en busca de datos.
- **Autodibujo** es un valor booleano que indica si se dibuja de nuevo un diagrama al introducir cada elemento o sólo bajo petición explícita del usuario.

## A-5. Creación de proyectos MSC2WEBL

Los pasos que hay que seguir para crear un proyecto MSC2WEBL son los siguientes:

1. Crear un nuevo proyecto (en la vista *Propiedades Proyecto*).
2. Crear un nuevo Documento MSC desde la ventana del editor MSC.
3. Crear una parte de definiciones y una de utilidades, si fuera necesario.
4. Definir las referencias que sean necesarias dentro del documento MSC.
5. Generar los diagramas MSC básicos de cada una de las referencias incluidas en el documento MSC.
6. Opcionalmente generar la documentación PDF en disco.
7. Transformar a código WebL.
8. Ejecutar el código generado (puede hacerse desde la consola incluida en el sistema, o desde un terminal cualquiera).

## A-6. Guía Instalación

MSC2WEBL depende de un gran número de paquetes y bibliotecas Linux. Estos programas son:

- Python (>=2.0)
- Python-dev
- Python-GTK2
- Libglade-2
- Python-Glade2
- Jaxml
- 4Suite
- LtXML
- PyLTXML
- XSV
- Latex
- Latex MSC package
- LibMagick
- Imagemagick
- jre1-2-2
- Webl.jar
- gcc
- gv
- csh

Existen tres paquetes disponibles de la aplicación<sup>9</sup>:

1. `msc2Webl-1.1.tgz`
2. `msc2Webl-1.1-nodoc.tgz`
3. `msc2Webl-1.1-con_paquetes.tgz`

---

<sup>9</sup>Se pueden descargar en <http://bach.it.uc3m.es/dgarcia/archivos/>

El primer paquete incluye únicamente la aplicación PyGTK, las plantillas XSLT, la documentación y el código de ejemplo. Para su funcionamiento requiere que todas las dependencias estén satisfechas y la creación de dos enlaces simbólicos a los directorios donde se encuentren java (jre1-2-2) y WebL.jar. Para crearlos, una vez descomprimido el paquete hay que ejecutar los siguientes comandos:

```
> cd msc2Webl/bin  
> ln -s path_a_jre1-2-2 jre1-2-2  
> ln -s path_a_Webl.jar Webl
```

El segundo paquete es igual al primero salvo que no incluye la documentación de la aplicación y del presente proyecto.

El tercer paquete contiene además todos los paquetes necesarios que no están ya incluidos en las fuentes oficiales de Debian Sid. Estos paquetes son:

- 4Suite
- LTXML
- PyLTXML
- XSV
- tidy
- Latex MSC package

Incluye además un script de instalación interactivo que consulta una a una todas las dependencias, instala los paquetes y crea los enlaces simbólicos necesarios.

Para cualquier problema con el proceso de instalación no dude en consultar al autor por correo electrónico en [dgarcia@it.uc3m.es](mailto:dgarcia@it.uc3m.es) o [danielgj@gmail.com](mailto:danielgj@gmail.com) indicando [msc2Webl] en el subject del mensaje.

# Apéndice B: Guía de desarrollo e instalación

Muchos son los programas y librerías empleadas para el desarrollo de este proyecto. En este apéndice se indican la mayor parte de ellos y se explica, brevemente, el proceso de obtención e instalación de los paquetes más importantes implicados en el desarrollo.

## B-1. El sistema operativo Linux: Debian SID

El sistema operativo elegido para el desarrollo del proyecto ha sido Debian GNU/Linux[53], concretamente en su versión SID. Las ventajas que aportan los sistemas operativos basados en GNU/Linux son muchas, pero cabe destacar la amplia variedad de software, generalmente libre, y la cantidad de listas de usuarios y foros de discusión que ayudan a solventar casi cualquier problema que surja con la ayuda de la amplia comunidad de usuarios de software libre.

El proceso de adquisición e instalación de Debian, además de complejo, escapa al alcance de este proyecto, sin embargo comentaré el proceso de instalación de paquetes en sistemas Debian, para entender cómo se instalan el resto de los programas. El mecanismo de instalación de paquetes Debian está basado en la aplicación APT, que permite consultar la lista de paquetes disponibles e instalarlos, bien sea desde Internet, cdrom o disco duro. Para ello cuenta con un fichero<sup>10</sup> donde se incluyen las fuentes donde están disponibles los distintos paquetes. El contenido de este fichero empleado para mantener el sistema de trabajo aparece en la figura B-1.

```
# Debian.es
deb ftp://ftp.es.debian.org/debian/ testing main non-free contrib
deb-src ftp://ftp.es.debian.org/debian/ testing main non-free contrib
deb http://security.debian.org/ testing/updates main contrib non-free
deb http://non-us.debian.org/debian-non-US unstable/non-US main contrib non-free
deb-src http://non-us.debian.org/debian-non-US unstable/non-US main contrib non-free

#JAVA
deb ftp://ftp.cica.es/pub/java-linux/debian unstable non-free main
```

Figura B-1: Fuentes oficiales de Debian Sid en rediris.es

Empleando el comando `apt-cache search patron_de_búsqueda` se consultan todos los paquetes que coinciden con el patrón de búsqueda. Con el comando `apt-get install nombre_paquete` se instala el programa deseado. Algunos de los programas y librerías necesarios no se pueden instalar de este modo al no existir paquetes exclusivos de esta distribución. Salvo que se indique lo contrario, el mecanismo de instalación de los programas descritos a continuación es empleando apt.

## B-2. Python

Se ha empleado el lenguaje Python[12][54] en su versión 2.3.4 para el desarrollo del interfaz gráfico. Python es un lenguaje de programación interpretado y de propósito múltiple que cuenta con una amplia variedad de módulos para todo tipo de tareas, como por ejemplo, programación Web (tanto del lado cliente, como del servidor), desarrollo de aplicaciones gráficas empleando varias librerías (Qt, GTK, WxWindows, etc), tareas

---

<sup>10</sup>La ruta completa a este fichero es `/etc/apt/sources.list`.

multimedia, procesamiento de documentos XML, etc. Python cuenta con una sintaxis muy sencilla y potente, así como un gran mecanismo de ayuda, incorporado en todos los módulos, funciones y tipos implementados, lo que facilita enormemente el desarrollo de los programas.

Python se ha aplicado en el proyecto para el desarrollo de la parte gráfica de la interfaz y el procesamiento de los documentos XML. Los paquetes relacionados exclusivamente con Python (independientemente de los aplicados a las tareas anteriormente indicadas) y que están presentes en las fuentes de Debian son:

- python-2.3
- python-2.3-dev

## B-3. Procesado XML con Python

Varias son las librerías y programas para el procesamiento de documentos XML empleando Python. Las más importantes aparecen detalladas a continuación.

### B-3.1. Jaxml

Jaxml[55] es un módulo Python para creación de documentos XML. Está incluido en las fuentes oficiales de Debian, por lo que su instalación es bien sencilla. Puede consultar su sitio Web para más información sobre este paquete.

### B-3.2. 4Suite

4Suite[56] es una librería que permite a los usuarios trabajar con un amplio conjunto de tecnologías XML estandarizadas para desarrollo e integración de aplicaciones basadas en el Web. Está implementada en Python y C.

El núcleo de 4Suite es una librería de herramientas integradas (incluyendo aplicaciones de línea de comandos) para procesamiento XML, incluyendo tecnologías como DOM, RDF, XSLT, XInclude, XPointer, XLink, XPath, XUpdate, RELAX NG, and XML/SGML Catalogs. En el proyecto se ha empleado esta librería para la aplicación de las distintas transformaciones XSLT sobre los documentos XML de trabajo.

El proceso de instalación de 4Suite requiere bajar el código fuente del paquete del sitio Web y compilarlo siguiendo las instrucciones incluidas en el fichero, para lo que se requiere tener instalados los paquetes Python. Este proceso de instalación está automatizado en el script de instalación de la aplicación MSC2WEBL.

### B-3.3. XSV

El mecanismo de validación de los documentos XML contra los schemas correspondientes se hace empleando la librería XSV (XML Schema Validator)[57]. El proceso de instalación de este paquete, automatizado también en el script de instalación de la aplicación, es muy sencillo e incluso, se permite la posibilidad de validar los documentos a través de un interfaz Web, lo que en el caso de esta aplicación no es útil por razones obvias. Para instalar este paquete es necesario tener instalado el interprete Python y el paquete python-dev.

Esta librería depende de otras dos para su instalación y funcionamiento. Estas son LtXML y PyLTXML[58], incluidas en el paquete de la aplicación MSC2WEBL y que pueden descargarse de su sitio Web.

## B-4. GTK y Glade

La librería gráfica empleada para el desarrollo de la interfaz es GTK+[59] y su puente a Python (PyGTK [13]). GTK+ es un conjunto de herramientas multiplataforma para la creación de interfaces gráficas de usuario que ofrece un conjunto completo de widgets gráficos y es apropiado para todo tipo de proyectos de cualquier tamaño. GTK+ es parte del proyecto GNU y su licencia, GNU LGPL, permite que sea usada por todo tipo de programadores, incluso para el desarrollo de software propietario. Muchas son las aplicaciones desarrolladas usando GTK+, de las cuáles las más conocidas son el escritorio GNOME y el programa de gráficos GIMP, de quien GTK toma sus siglas (Gimp ToolKit).

Sin embargo el uso de GTK directamente es una tarea complicada. Para evitar ésto se emplea además Glade[60] que es una ayuda para la construcción de interfaces basados en GTK que permite la construcción



de código C, C++, Ada, Perl y Python de manera sencilla. El funcionamiento de esta aplicación es similar a una herramienta de dibujo que permite crear las distintas ventanas y elementos de nuestro GUI, de forma que esta especificación se realiza de forma muy sencilla generando un fichero XML que es analizado en tiempo de ejecución y convertido al código necesario, en este caso PyGTK.

Todos los paquetes necesarios para el desarrollo GTK+ con PyGTK y Glade están incluidos en las fuentes oficiales. Los nombres de los paquetes necesarios son:

- libgtk2.0-0
- libgtk2.0-dev
- libgtk2.0-doc
- glade-2
- libglade2-0
- libglade2-dev
- python2.3-gtk2
- python2.3-glade2

## B-5. Dibujo de los diagramas

El dibujo de los diagramas, como ya se ha explicado anteriormente, se hace empleando el sistema de documentación  $\text{\LaTeX}$  apoyado en una macro[45] diseñada para el dibujo de diagramas MSC de todo tipo, incluyendo documentos MSC, diagramas MSC y hMSC. Latex es un paquete estandar en las distribuciones Linux, no así dicha macro. Para la instalación de la macro, basta con copiar los ficheros en el directorio por defecto de Latex que, en el caso particular de Debian, se encuentra en `/usr/share/texmf/tex/latex/` y ejecutar el comando `texhash`.

Además, para la conversión de los diagramas generados por latex en formato dvi, es necesario el paquete *dviutils* para la conversión a formato pdf de los diagramas generados. Se añade también un último paso de conversión de los diagramas a formato jpg empleando la librería *libmagick*[61]. Estos dos últimos paquetes se encuentran incluidos en las fuentes oficiales de Debian.

## B-6. WebL

Web Lenguaje o WebL[2], como se le conoce comúnmente, es un lenguaje de programación interpretado específico para automatizar tareas en el Web desarrollado en 1998 por Hannes Marais en los laboratorios de *Compaq's Research Center*. El intérprete WebL esta escrito en Java[62] por lo que requiere de éste para su ejecución.

No es necesario hacer nada especial para instalar WebL una vez se disponga de un entorno de ejecución de Java. En las primeras versiones de WebL era necesario, en plataformas Linux, emplear Java Runtime Environment 1.2.2, sin embargo ésto ya ha sido corregido en las últimas versiones.

Estos dos paquetes son los únicos empleados en todo el desarrollo que no cuentan con una licencia libre, es por ello que no se distribuyen junto con la aplicación, pero pueden ser descargados de sus respectivos sitios Web.



# Bibliografía

- [1] **R. ZAKON.** *Hobbes' Internet Timeline v7.0.*  
[[www.zakon.org/robert/internet/timeline/](http://www.zakon.org/robert/internet/timeline/)].
- [2] **Compaq's Research Center.** *Web Language Site.*  
[[research.compaq.com/SRC/WebL/](http://research.compaq.com/SRC/WebL/)].
- [3] **H. MARAIS.** *WebL - A Programing Language for the Web (Reference Manual).* Compaq Systems Research Center, 1997.
- [4] **T. KISTLER y H. MARAIS.** *WebL - A Programing Language for the Web.* Technical report, Digital Systems Research Center, 1998.
- [5] **ITU-T.** *Formal Description Techniques - Message Sequence Chart.* RFC Z.120, Noviembre 1999.
- [6] **ITU-T.** *Corrigendum 1 to Recommendation Z.120.* RFC, Diciembre 2001.
- [7] **L. CARDELLI y R. DAVIES.** *Service Combinators for Web Computing.* Technical report, Mayo 1999.  
[[www.hippo.cis.strath.ac.uk/papers/service\\_combinators\\_and\\_webl.html](http://www.hippo.cis.strath.ac.uk/papers/service_combinators_and_webl.html)].
- [8] **W3C.** *Extensible Markup Language (XML).* RFC, Febrero 98.  
[[www.w3.org/XML/](http://www.w3.org/XML/)].
- [9] **W3C.** *XHTML 1.0: A Reformulation of HTML 4 in XML 1.0.* RFC, Agosto 2002.  
[[www.w3.org/TR/xhtml1/](http://www.w3.org/TR/xhtml1/)].
- [10] **DIA.**  
[[www.gnome.org/projects/dia/](http://www.gnome.org/projects/dia/)].
- [11] **W3C.** *XSL Transformations (XSLT) Versión 1.0.* RFC, Noviembre 1999.  
[[www.w3.org/TR/xslt/](http://www.w3.org/TR/xslt/)].
- [12] **Python.org.**  
[[www.python.org](http://www.python.org)].
- [13] **PyGTK.**  
[[www.pygtk.org](http://www.pygtk.org)].
- [14] **V. LUQUE.** *Automatización de Tareas en el Web: Una Propuesta Basada en Estándares.* Tesis Doctoral, Universidad Carlos III de Madrid, Junio 2003.
- [15] **V. CRESCENZI, G. MECCA y P. MERIALDO.** *RoadRunner: Towards Automatic Data Extraction from Large Websites.* Technical report, 2001.
- [16] **P. BREUER, C. DELGADO, J.A. HERRAIZ, V. LUQUE y L. SANCHEZ.** *MSC-based language for Specifying Automated Web Clients.* Technical report, Eighth IEEE Symposium on Computers and Communications, 2003. ISCC 2003, ISSN 1530-1346, Kemer, Antalya, Turkey, June 30-July 3.
- [17] **R. CHATLEY, J. KRAMER, J. MAGEE y S. UCHITEL.** *Visual Methods for Web Application Design.* Technical report, Dpt. of Computing, Imperial College, London, 2003.
- [18] **W3C.** *HTML page.*  
[[www.w3.org/MarkUp/](http://www.w3.org/MarkUp/)].
- [19] **W3C.** *HTML 4.01 Specification.* RFC, Diciembre 99.  
[[www.w3.org/TR/html4/](http://www.w3.org/TR/html4/)].

- [20] **W3C**. *Cascading Style Sheets Level 1*. RFC, Enero 99.  
[[www.w3.org/Style/CSS/](http://www.w3.org/Style/CSS/)].
- [21] **W3C**. *Sitio Web del World Wide Web Consistorium*.  
[[www.w3.org/](http://www.w3.org/)].
- [22] **W3C**. *Web Accessibility Initiative*. RFC, Mayo 99.  
[[www.w3.org/WAI/](http://www.w3.org/WAI/)].
- [23] **W3C**. *Semantic Web*. RFC, Febrero 2004.  
[[www.w3.org/2001/sw/](http://www.w3.org/2001/sw/)].
- [24] **W3C**. *Resource Description Framework (RDF)*. RFC, Febrero 2004.  
[[www.w3.org/RDF/](http://www.w3.org/RDF/)].
- [25] *Barrapunto*.  
[[www.barrapunto.com](http://www.barrapunto.com)].
- [26] *NewsForge*.  
[[www.newsforge.com](http://www.newsforge.com)].
- [27] *FlashGet*.  
[[www.amazesoft.com/](http://www.amazesoft.com/)].
- [28] *Getright*.  
[[www.getright.com/](http://www.getright.com/)].
- [29] **J.A. HERRAIZ**. *Desarrollo de un portal Web integrador de servicios de correo electrónico basado en tecnología XML*. Proyecto Fin de Carrera, Universidad Carlos III de Madrid, Febrero 2004.
- [30] **M.A. GONZALO**. *Desarrollo de un portal Web integrador de información basada en subastas online*. Proyecto Fin de Carrera, Universidad Carlos III de Madrid, Marzo 2003.
- [31] **S. FENOLL**. *Desarrollo de un portal Web integrador de contenidos basados en viajes online*. Proyecto Fin de Carrera, Universidad Carlos III de Madrid, Noviembre 2003.
- [32] **W3C**. *XML Schema Definition Language*. RFC, Mayo 2001.  
[[www.w3.org/XML/Schema](http://www.w3.org/XML/Schema)].
- [33] **W3C**. *XML Path Language (XPath) Version 1.0*. RFC, Noviembre 99.  
[[www.w3.org/TR/xpath](http://www.w3.org/TR/xpath)].
- [34] **W3C**. *XML Path Language (XPath) 2.0*. Borrador de Trabajo, Julio 2004.  
[[www.w3.org/TR/xpath20/](http://www.w3.org/TR/xpath20/)].
- [35] **ITU-T**. *Specification and Description Language*. RFC, Octubre 2002.
- [36] **R. BRAEK y A. MEISINGSET**. *The ITU-T Languages in a Nutshell*. *Teletronikk*, (4), 2000.
- [37] **OMG**. *Unified Modeling Language*. RFC, Marzo 2003.
- [38] **P.B. LADKIN y S. LEUE**. *Implementing and Verifying MSC Specifications using PROMELA/XSPIN*. Technical report, 1996.
- [39] **H. BEN-ABDALLAH y S. LEUE**. *MESA: Support for Scenario-Based Design of Concurrent Systems*. Technical report, 1998.  
[[tele.informatik.uni-freiburg.de/leue/publications.files/tacas98.ps.Z](http://tele.informatik.uni-freiburg.de/leue/publications.files/tacas98.ps.Z)].
- [40] **J. MAGEE y J. KRAMMER**. *LTSA y plugin para MSC*.  
[[www.doc.ic.ac.uk/ltsa/](http://www.doc.ic.ac.uk/ltsa/)].
- [41] **BELL LABS**. *Ubet*.  
[[cm.bell-labs.com/cm/cs/what/ubet/](http://cm.bell-labs.com/cm/cs/what/ubet/)].
- [42] *Tidy*.  
[[www.w3.org/People/Raggett/tidy/](http://www.w3.org/People/Raggett/tidy/)].
- [43] *GNU Public License*.  
[[www.gnu.org/licenses/gpl-howto.html](http://www.gnu.org/licenses/gpl-howto.html)].

- [44] *Buscador Online de Códigos Postales.*  
[[www.meguias.com/cp.php](http://www.meguias.com/cp.php)].
- [45] **M. SJOUGE y V. BOS.** *Drawing Message Sequence Charts with L<sup>A</sup>T<sub>E</sub>X.* TUGBoat Vol. 22, Marzo/Junio 2001.  
[<ftp://ftp.win.tue.nl/pub/techreports/sjouke/tbmsc.ps.Z>].
- [46] **K. INDERMARK.** *MSC Tools Index Page.*  
[[www-i2.informatik.rwth-aachen.de/Research/AG/MCS/MSC/index.html](http://www-i2.informatik.rwth-aachen.de/Research/AG/MCS/MSC/index.html)].
- [47] *Xalan.*  
[[xml.apache.org/xalan-j/](http://xml.apache.org/xalan-j/)].
- [48] *Sitio Web del grupo de noticias es.comp.linux.*  
[[es.comp.linux.org](http://es.comp.linux.org)].
- [49] *Tira Ecol.*  
[[tira.es.comp.linux.org](http://tira.es.comp.linux.org)].
- [50] *Interfaz Web Correo Alumnos UC3M.*  
[[alumnos.uc3m.es](http://alumnos.uc3m.es)].
- [51] **D. GARCIA y V. LUQUE.** *Modelado de Aplicaciones Web a partir de Especificaciones MSC.* III Taller sobre Ingeniería del Software Orientada al Web (Web Engineering) WebE'2003 Alicante, Noviembre 2003.  
[[simpl68.si.ehu.es/webe03/Papers/5Garcia.pdf](http://simpl68.si.ehu.es/webe03/Papers/5Garcia.pdf)].
- [52] *Página Web del Colegio Oficial de Ingenieros de Telecomunicación.*  
[[www.coit.es](http://www.coit.es)].
- [53] *Debian.org.*  
[[www.debian.org](http://www.debian.org)].
- [54] *Documentación Oficial Python 2.3.4.*  
[[www.python.org/doc/2.3.4/](http://www.python.org/doc/2.3.4/)].
- [55] *Jaxml.*  
[[www.librelogiciel.com/software/jaxml/action.Presentation](http://www.librelogiciel.com/software/jaxml/action.Presentation)].
- [56] *4Suite.*  
[[4suite.org](http://4suite.org)].
- [57] *Current Status of XSV.*  
[[www.ltg.ed.ac.uk/~ht/xsv-status.html](http://www.ltg.ed.ac.uk/~ht/xsv-status.html)].
- [58] *LT XML.*  
[[www.ltg.ed.ac.uk/software/xml/](http://www.ltg.ed.ac.uk/software/xml/)].
- [59] *GTK+.*  
[[www.gtk.org](http://www.gtk.org)].
- [60] *GLADE.*  
[[glade.gnome.org/](http://glade.gnome.org/)].
- [61] *ImageMagic.org.*  
[[www.imagemagick.org/](http://www.imagemagick.org/)].
- [62] *Sitio Web de Java.*  
[[java.sun.com](http://java.sun.com)].
- [63] *Google.*  
[[www.google.es](http://www.google.es)].
- [64] *Sitio Web de MSC2WEBL.*  
[[bach.it.uc3m.es/~dgarcia](http://bach.it.uc3m.es/~dgarcia)].
- [65] **Defense Information System Agency.** *Commercial Telecommunication Standards.*  
[[www-comm.itsi.disa.mil/itu/r\\_z0.html](http://www-comm.itsi.disa.mil/itu/r_z0.html)].

- [66] **A. MOLLER y M.I. SCHWARTZBACH.** *The XML Revolution: Technologies for the Future Web.* [[www.brics.dk/~amoeller/XML/index.html](http://www.brics.dk/~amoeller/XML/index.html)] Última Revisión Octubre 2003.
- [67] **Sax.** [[www.saxproject.org/](http://www.saxproject.org/)].
- [68] **J. FINLEY.** *PyGTK 2.0 Tutorial.* [[www.pygtk.org/pygtk2tutorial/](http://www.pygtk.org/pygtk2tutorial/)].
- [69] **PyGTK FAQ Index.** [[www.async.com.br/faq/pygtk/index.py?req=index](http://www.async.com.br/faq/pygtk/index.py?req=index)].
- [70] **T. GALE y I. MAIN.** *GTK+ 2.0 Tutorial.* [[www.gtk.org/tutorial/](http://www.gtk.org/tutorial/)].
- [71] **W3C.** *The Extensible Stylesheet Language Family (XSL).* RFC. [[www.w3.org/Style/XSL/](http://www.w3.org/Style/XSL/)].
- [72] **W3C.** *XML Linking Language (XLink) Version 1.0.* RFC, Junio 2001. [[www.w3.org/TR/xlink/](http://www.w3.org/TR/xlink/)].
- [73] **W3C.** *XML Pointer Language (XPointer).* Borrador de Trabajo, Agosto 2002. [[www.w3.org/TR/xptr/](http://www.w3.org/TR/xptr/)].
- [74] **W3C.** *XML Query.* RFC, Agosto 2004. [[www.w3.org/XML/Query](http://www.w3.org/XML/Query)].
- [75] **W3C.** *Document Object Model.* RFC, Noviembre 2000. [[www.w3.org/DOM/](http://www.w3.org/DOM/)].
- [76] **D. TIDWELL.** *XSLT.* O'Reilly, 2001.
- [77] **E. HARLOW.** *Desarrollo de aplicaciones Linux con GTK+ y GDK: guía avanzada.* Prentice Hall, 1999.
- [78] **S. UCHITEL y J. MEGEE.** *Synthesis of Behavioral Models from Scenarios.* *TRANSACTIONS ON SOFTWARE ENGINEERING* Vol. 29, (2), Febrero 2003.
- [79] **D. PELED.** *Specification and Verification of Message Sequence Charts.* Technical report, BELL-LABS, 2002.
- [80] **H. BEN-ABDALLAH y S. LEUE.** *Syntactic Analysis of Message Sequence Chart Specifications.* Technical report, University of Waterloo, Noviembre 1996.
- [81] **H. BEN-ABDALLAH y S. LEUE.** *Architecture of a Requirements and Design Tool Based on Message Sequence Charts.* Technical report, University of Waterloo, Octubre 1996.
- [82] **S. LEUE.** *Research on Message Sequence Charts and Message Flow Graphs.* Technical report, University of Waterloo, 1998. [[tele.informatik.uni-freiburg.de/~leue/msc.html](http://tele.informatik.uni-freiburg.de/~leue/msc.html)].
- [83] **S. UCHITEL, R. CHATLEY, J. KRAMER y J. MAGEE.** *LTSA-MS: Tool Support for Behaviour Model Elaboration Using Implied Scenarios.* Technical report, Imperial College, London, Julio 2003.
- [84] **P.B. LADKIN y S. LEUE.** *What Do Message Sequence Charts Mean?* Technical report, Institute of Informatics and Applied Mathematics, University of Berne, 1992.
- [85] **A. TAN, C. TAN y L. VIJJAPPU.** *Web Structure Analysis for Information Mining.* Technical report.
- [86] **J. MYLLYMAKI.** *Effective Web Data Extraction with Standard XML Technologies.* Technical report, IBM AlmadenResearch Center, 2001.
- [87] **J.L. CARRASCO.** *XML.* Technical report, Dto. Ingeniería Telemática, Abril 2000.