Una Aproximación a la Gestión Automática de Alternativas de Diseño [*]

Octavio Martín, Antonio Ruiz y Miguel Toro

Dpto. de Lenguajes y Sistemas Informáticos
Facultad de Informática y Estadística, Universidad de Sevilla
41.012 Sevilla, Spain
{octavio,aruiz,mtoro}@lsi.us.es

Resumen La obtención, evaluación y selección de alternativas durante la fase de diseño de cualquier sistema informático son problemas muy conocidos pero, no por ello, menos complejos. Habitualmente, a la hora de realizar este tipo de tareas hemos seguido técnicas *ad-hoc*, que varían de un proyecto a otro según las necesidades. Por otro lado, la necesidad de obtener un producto final de calidad requiere un gran esfuerzo, por lo que la posibilidad de automatizar estos procesos es de máximo interés. Basándonos en los resultados positivos obtenidos referentes a la fase de elicitación de requisitos de calidad, en este trabajo, quizás un poco prematuramente, presentamos unos esquemas o aproximaciones a una fase de diseño que también sea sensible a la calidad mediante la búsqueda de la mejor alternativa, con vistas a una futura automatización de su gestión.

Palabras Claves: Ingeniería del *Software*, Alternativas de Diseño, Calidad del *Software*.

1 Introducción

La obtención, evaluación y selección de alternativas de diseño son problemas a los que nos enfrentamos los ingenieros durante la fase de desarrollo de cualquier tipo de proyecto. En general, su complejidad se ve incrementada cuando el producto resultante debe cumplir un conjunto de requisitos de calidad que, en su momento, fueron indicados por el cliente y que, por regla general, presentarán ciertas incompatibilidades entre sí. Tradicionalmente, encontramos muchos modelos de procesos o ciclos de vida en los que se proponen la obtención y evaluación de diseños alternativos y la selección de uno de ellos [3, 7, 12, 15–17, 23, 28].

Desde un punto de vista específico de la arquitectura *software* existen propuestas que tienen en cuenta de una manera inherente los requisitos de calidad [1, 11, 13, 14, 16, 24–26], también conocidos como no funcionales. Entre todos ellos destacamos aquí el trabajo de [4], que propone un diseño arquitectónico del *software* como una actividad con múltiples objetivos donde el ingeniero de *software* tiene que poner en equilibrio los distintos requisitos necesarios, dando un enfoque especial a los requisitos no funcionales a través de un proceso iterativo de evaluación y transformación. Para ello,

[*] Este trabajo está subvencionado por el proyecto "GEOZOCO" de la CICYT TIC2000-1106-C02-01, Comisión Interministerial de Ciencia y Tecnología.

identifica varias categorías de transformaciones arquitectónicas, cada una de las cuales implica unos costes y puede afectar tanto positiva como negativamente al grado de cumplimiento de los atributos de calidad. En este sentido, respecto a la posible y más que segura conflictividad que surge entre los requisitos, el trabajo de [2] presenta una herramienta que facilita la detección de estos problemas mediante la definición de una base de reglas que contiene las relaciones entre los participantes en estos procesos, sus intereses respecto a los atributos primarios y secundarios de calidad e, igualmente, el efecto de los estilos arquitectónicos que se pueden aplicar.

Entre los estudios con un punto de vista formal, en la propuesta de [27] encontramos un modelo conceptual para resolver problemas de Ingeniera del *Software* basado precisamente en la selección de alternativas. Destaca el hecho de que, en este modelo, los requisitos no se expresan en términos cuantificables, más aún, se asevera que los factores de calidad no se pueden cuantificar y, por tanto, no se pueden utilizar para obtener, evaluar y seleccionar alternativas de diseño. Su opinión se basa en la constatación de que, en Ingeniería del *Software*, el concepto de calidad es implícito y sin medidas, mientras que en las Ingenierías con mayor madurez ya es explícito y matemáticamente definido. Por ello, la evaluación de alternativas no están formalizadas y se basan en requisitos de calidad cualitativos.

En el presente trabajo, describiremos en primer lugar, en la sección 2, algunos resultados positivos que ya hemos obtenido [8, 20–22] referentes a la especificación y soporte automático de requisitos de calidad en los que mostramos que **(1)** la calidad del *software* sí puede ser explícita y matemáticamente definida, **(2)** una alternativa se puede caracterizar mediante atributos de calidad y **(3)** una alternativa puede ser medida de manera automática. Con ello, podemos abordar de manera cuantitativa todos estos problemas, mientras que [2, 27] sólo tienen un punto de vista cualitativo. A continuación, y basándonos en los resultados anteriores, en la sección 3 fijaremos nuestro objetivo, mostrando los esquemas de un modelo que nos permitirá caracterizar una alternativa de diseño mediante requisitos de calidad formalizados de manera que nos facilitará la tarea de automatizar la obtención, evaluación y selección de alternativas. Finalmente, en la sección 4 concluiremos con las líneas de trabajo futuro.

2 Automatización del Catálogo de Requisitos de Calidad

En la sección 1 ya vimos que los problemas planteados por [27] se basan principalmente en la dificultad existente en la transformación de los requisitos de calidad, expresados habitualmente en lenguaje natural, en una descripción formal que facilite su procesado automático. Para resolverlos, hemos definido un lenguaje de especificación formal QRL (*Quality Requirements Language*), presentado en [20–22] y basado en [5, 9, 10]. La aportación de QRL es que **(1)** permite precisamente dicha transformación, obteniendo restricciones matemáticas equivalentes que son viables para un razonamiento cuantitativo y **(2)** facilita la automatización de la elicitación de requisitos, incluyendo la evaluación y resolución de conflictos. Para ello, QRL utiliza patrones lingüísticos [8] (patrones-L) que son frases normalizadas que representan el conjunto completo de los requisitos expresables con dicho patrón y tienen una semántica precisa porque se asocian a una restricción matemática (restricciones-Q) predeterminada.

```
catalogue q2.lsi.us.es {
  ...
  category Reliability {
    ...
    Availability {
      description: "Readiness to use";
      domain: increment numeric percentage;
      pattern:"The system will be ready to use the <percentage> of its lifetime" = "Availabity >= <percentage>";
    }
    ...
  }
  ...
  category Perfomance {
    ...
    Latency {
      description: "Time to response";
      domain: decreasing numeric ms;
      pattern:"The maximum time to response will be <time> at most" = "Latency.max <= <time>";
    }
    ...
  }
  ...
}
```

a) Ontology

```
using q2.lsi.us.es ;

conditions for ITrailersBank {
  requires {
    c1: Availability >= 95 %;
    c2: Latency.max <= 75 ms;
  }
  weights {
    Availability = 0.75;
    Latency.max = 0.25;
  }
}
```

b) Conditions

Figura1. Ejemplos de catálogos y definición de requisitos de calidad.

En la figura 1.b mostramos brevemente un ejemplo de la definición de los requisitos de calidad para una interfaz en un sistema cualquiera, cada uno de las cuales tiene asociado un peso que mide la importancia dada a dicha restricción, relativa al atributo de calidad correspondiente. Estas condiciones se basan en una ontología, definida en la figura 1.a, que no es sino una metadescripción formada por categorías de atributos que, a su vez, contienen la formalización de los atributos de calidad, cada uno los cuales está bien definido con su patrón de utilización y su restricción asociada. Durante la fase de diseño de sistemas que siguen este modelo, estas condiciones se confrontan frente a las ofertas realizadas por las diversas alternativas disponibles, como se verá en la sección 3, y se escogerá la mejor de ellas. También estamos construyendo una plataforma para que los sistemas puedan realizar todos estos procesos en tiempo de ejecución [18].

La necesidad de utilizar estos catálogos viene dada por la falta de uniformidad en las diferentes taxonomías existentes [5, 14, 15] .De esta forma, se intenta describir un vocabulario universal que actúe como modelo común de referencia y que facilite la automatización de las tareas relacionadas con los atributos de calidad.

3 Esquemas para la Automatización de la Gestión de Alternativas de Diseño

Nuestro principal objetivo es ofrecer un soporte automático al proceso de obtención, evaluación y selección de alternativas de diseño y que sea sensible a la calidad. Basándonos en lo expuesto en la sección 2, en los siguientes apartados caracterizaremos estos problemas y discutiremos algunas propuestas para resolverlos.

3.1 Caracterización del Problema

En la figura 2 mostramos el esquema lineal completo con sus procesos principales: **(1)** se obtienen las diferentes alternativas de diseño según los requisitos funcionales y de calidad expresados por el cliente y un repositorio de transformaciones arquitectónicas, **(2)** se evalúan cada una de ellas y **(3)** se selecciona la alternativa óptima, es decir, aquella que se aproxime mejor a lo marcado por los requisitos de calidad.

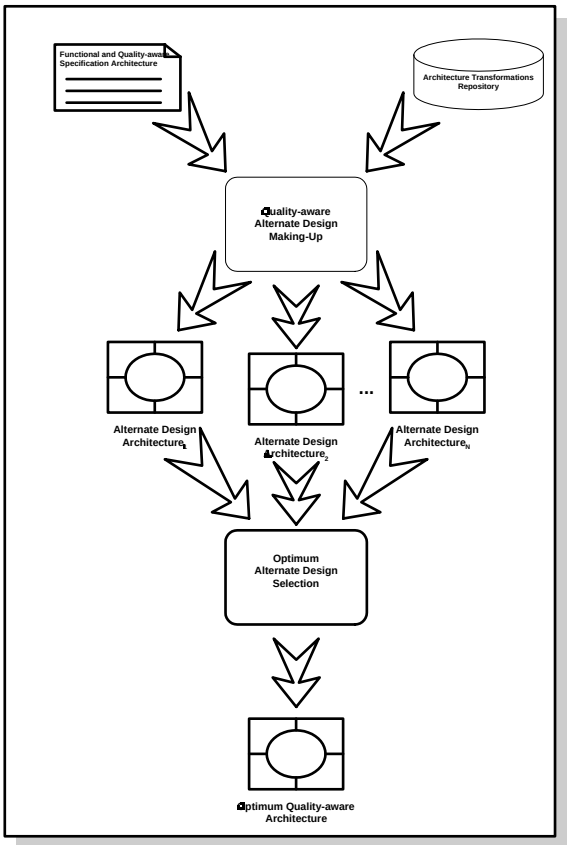

Figura2. Esquema general (lineal) de gestión de las alternativas de diseño.

Se hace evidente que la tarea más costosa es la generación de todas las posibles alternativas. Afortunadamente, podemos cambiar este punto de vista para obtener directamente la alternativa óptima mediante un esquema incremental, tal y como aparece en la figura 3. Este esquema se basa en el modelo de procesos propuesto por [4] donde se describe un desarrollo iterativo e incremental en la fase de diseño guiado por los requisitos de calidad que hay que satisfacer.

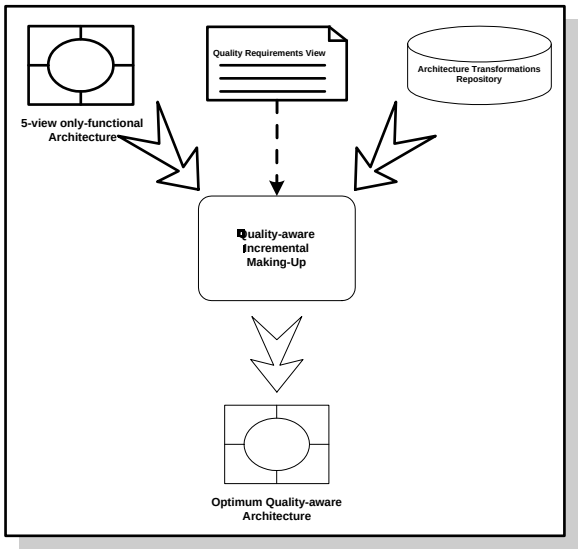

Figura3. Esquema incremental de gestión de las alternativas de diseño.

De esta forma, se obtiene directamente una única alternativa que cumple los requisitos de calidad sin necesidad de evaluar distintas alternativas. Sin embargo, el problema persiste desde otra perspectiva: la evaluación de las diferentes transformaciones arquitectónicas que podemos aplicar a la arquitectura del sistema en cada ciclo con vistas conseguir el mismo objetivo, es decir, el cumplimiento de los requisitos de calidad propuestos por el cliente, es un problema de búsqueda de la solución óptima que es NP-completo. En la figura 5 mostramos un metamodelo abreviado del espacio de búsqueda que describiremos en el apartado 3.3.

Una primera e interesante propuesta para resolver (con una complejidad algorítmica razonable) este tipo de problemas la encontramos en [6], que describe una serie de búsquedas metaheurísticas (tales como algoritmos genéticos, de templado simulado, búsqueda tabú y otras) que se han utilizado de manera satisfactoria en multitud de problemas de Ingeniería aunque no se han aplicado mucho a la Ingeniería del *Software* debido, en gran medida, a la falta de formalización y la dificultad del espacio de búsqueda. Las dos operaciones básicas a definir son, como es habitual, la función objetivo para medir cada una de las soluciones candidatas y la eliminación de aquellas alternativas que no cumplan las restricciones impuestas.

Pero, sin ningún género de dudas, la solución más elegante, una vez que hemos definido formalmente los requisitos de calidad y tenemos un metamodelo del espacio de búsqueda, es la caracterización de cada una de las alternativas arquitectónicas por su grado de satisfacción de los requisitos de calidad y, a partir de aquí, plantear un sistema de restricciones a resolver, obteniendo una solución óptima. En este sentido, en [20–22] ya hemos tratado con problemas de este tipo para la búsqueda y resolución de conflictividad entre requisitos de calidad en la propia fase de elicitación. Las dificultades reales comienzan cuando se incluyen restricciones no lineales y/o temporales porque el árbol de búsqueda que se obtiene es demasiado grande.

3.2 Un Ejemplo Ilustrativo: la Disponibilidad

Para dar una idea intuitiva del problema mostramos el siguiente ejemplo: en la figura 4 se presentan varios diagramas de despliegue (de UML) correspondientes a varias alternativas para un sistema cliente/servidor con arquitectura en 3 capas. La figura 4.a muestra una alternativa donde aún no se han tenido en cuenta los requisitos de calidad, por lo que el grado de cumplimiento de estos requisitos (en principio) no tendrían por qué corresponder al requerido en la vista de calidad.

En la figura 1.b mostramos los requisitos de calidad que debe cumplir este sistema, donde ya se indica que el mayor peso recae sobre la disponibilidad del servicio prestado. En general, se aplican diversas técnicas para obtener una disponibilidad muy alta y una de las más conocidas es el *clustering*, consistente en la disposición conjunta de varias máquinas que operan de cara al usuario como si fueran una sóla. Por ejemplo, imagínese una máquina realmente "mala" con una tasa de disponibilidad de sólo el 95% el primer año, es decir, que no estará disponible aproximadamente durante 1 hora y 12 minutos al día. Si añadiésemos otra máquina idéntica con el mismo coste, la disponibilidad del *cluster* resultante se incrementaría hasta el 99,75%, así que únicamente no estaría disponible durante 3 minutos y 36 segundos al día. El incremento del coste será ligeramente superior al doble porque también se cuenta el *software* de *clustering* que se debe instalar. Todo esto corresponde a la primera alternativa en la figura 4.b, aunque concluimos que el rendimiento final de este diseño no es suficiente y no cumple con los requisitos exigidos.

Si queremos conseguir un gran rendimiento del sistema, podríamos optar por olvidar la técnica del *clustering* e invertir en la adquisición de una máquina servidora de grandes prestaciones, entre ellas una altísima disponibilidad y un inmejorable rendimiento como vemos en la figura 4.d pero, desgraciadamente, estas máquinas son muy caras y esto hace que tengamos que desechar esta alternativa, técnicamente quizás la mejor, por su alto coste.

Volviendo a la técnica del *clustering*, podemos mejorar su rendimiento si utilizamos algún tipo de algoritmo de balanceo de carga que reparta el trabajo entre las distintas máquinas, tal y como se muestra en la figura 4.c, a un costo razonable dentro de los márgenes indicados en la vista de calidad. De esta manera, tras evaluar todas estas alternativas optaremos por esta última que es la que se acerca más al objetivo fijado en dicha vista de calidad.

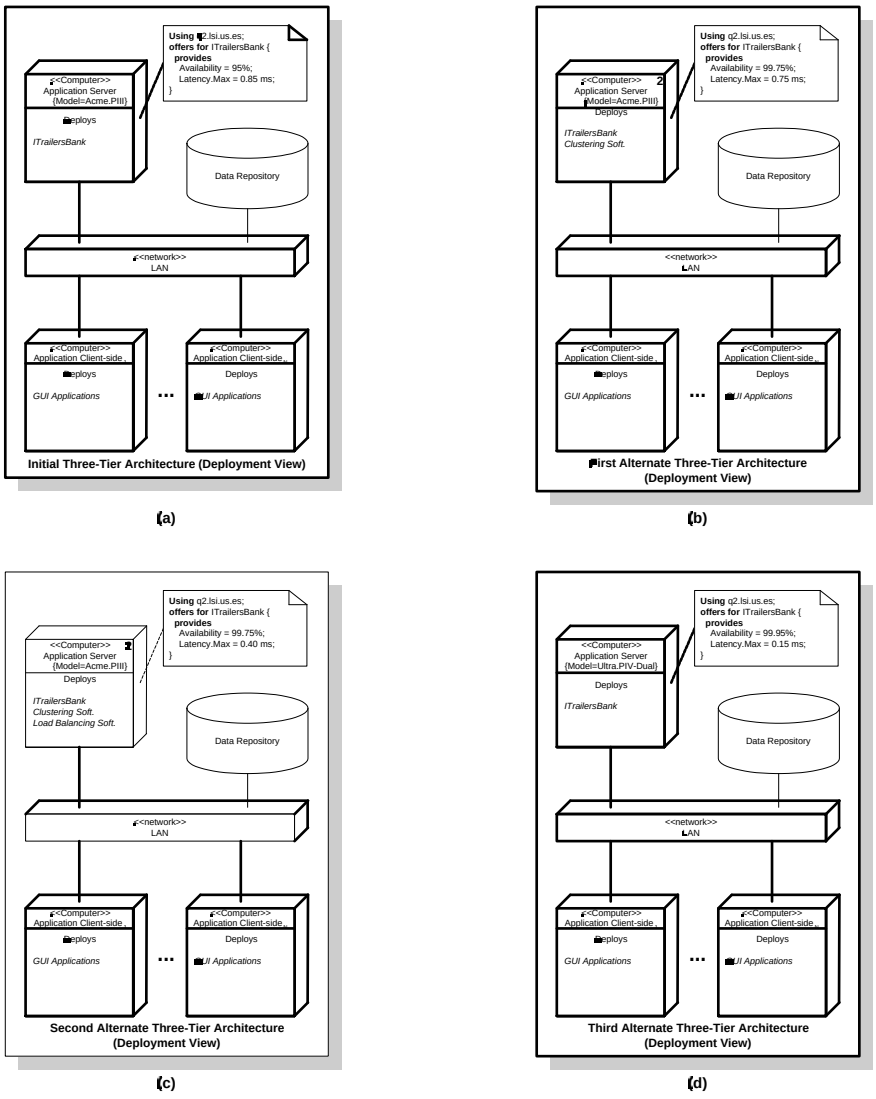

Figura4. Ejemplo ilustrativo de selección de alternativas.

3.3 Metamodelo del Espacio de Búsqueda

Como hemos mecionado anteriormente, la automatización de todas estas tareas requiere de forma imprescindible la generación de un modelo de cada estructura de datos que está implicada, entre las que podemos encontrar **(1)** la especificación del futuro sistema incluyendo sus requisitos funcionales y su vista de calidad junto a los catálogos utilizados (todo ello obtenido en la fase de elicitación de requisitos anterior), **(2)** los repositorios de transformaciones arquitectónicas y **(3)** las diversas alternativas de diseño junto al grado de satisfacción de los requisitos de calidad. En la figura 5 mostramos un esbozo del metamodelo inicial que incluye las relaciones existentes entre cada una de estas estructuras, que pasamos a describir brevemente a continuación.

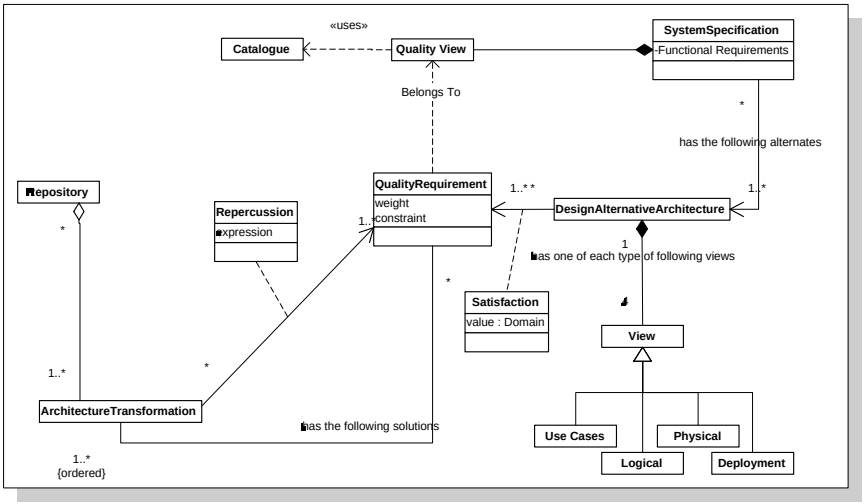

Figura5. Metamodelo abreviado del espacio de búsqueda.

La especificación de un sistema tiene una parte funcional y una vista de calidad donde se encuentran cada uno de los requisitos de calidad que expresan las condiciones que debe cumplir el sistema y que utilizan un catálogo de atributos de calidad, tal y como se ha descrito en la sección 2. Además, cada especificación tiene un conjunto de arquitecturas alternativas de diseño, cada una de las cuales viene definida con cuatro vistas [19], basadas en [13], a saber la vista de casos de uso, la vista lógica, la vista física y la vista de despliegue. Por otro lado, una alternativa puede satisfacer (o no satisfacer) los requisitos de su vista de calidad, ofreciendo unos servicios con un grado cuantificable de calidad.

A su vez, cada requisito de calidad tiene asociado una lista de transformaciones arquitectónicas que pueden utilizarse para mejorar el grado de cumplimiento de cualquier arquitectura respecto a dicho requisito. Dicha lista debe estar ordenada de mayor a menor impacto para facilitar la búsqueda de la mejor posibilidad. A su vez, cada transformación arquitectónica repercute de una u otra forma sobre el grado de cumplimiento

del resto de requisitos de calidad. El conjunto de transformaciones arquitectónicas forman un repositorio, al estilo de [2], que puede utilizarse por parte de muchas otras arquitecturas. Estas repercusiones vienen dadas en forma de expresiones regulares que contienen una función. Dichas funciones pueden llegar a ser muy complejas, puesto que dependen de factores tales como el dominio del atributo de calidad en cuestión y de los elementos que participan en la transformación, y se va a necesitar del concurso de expertos en cada una de las categorías de calidad involucradas para obtenerlas.

Para ilustrar este metamodelo, en la figura 6 mostramos un ejemplo de una estructura que corresponde a los requisitos de calidad especificados en la figura 1.b anterior. Cada uno de ellos tiene asociada una lista ordenada de transformaciones aplicables para mejorar su grado de cumplimiento. En nuestro ejemplo, un experto ha indicado que el orden de impacto de las transformaciones del repositorio para el requisito de la disponibilidad, con un peso del 75% sobre la latencia, viene encabezado por el establecimiento de un *cluster*, seguido por la utilización de unidades de potencia ininterrumpibles y, por último, el despliegue de una máquina de altas prestaciones. El impacto o repercusión de cada una de estas transformaciones viene dado por una expresión en el propio dominio del atributo de calidad. Así pues, en el ejemplo vemos que la expresión del nuevo grado de disponibilidad en un *cluster* que utiliza dos máquinas idénticas viene dada por $D' = D^2 + 2(1 - D)$, siendo D' el grado de disponibilidad posterior a la aplicación de la transformación, y D el anterior. A su vez, cada una de estas transformaciones va a afectar en mayor o menor medida a la satisfacción de otros requisitos de calidad, por lo que habrá que buscar cuál es la mejor combinación de transformaciones y su orden de aplicación (la aplicación de una transformación no es conmutativa) para encontrar la alternativa óptima.

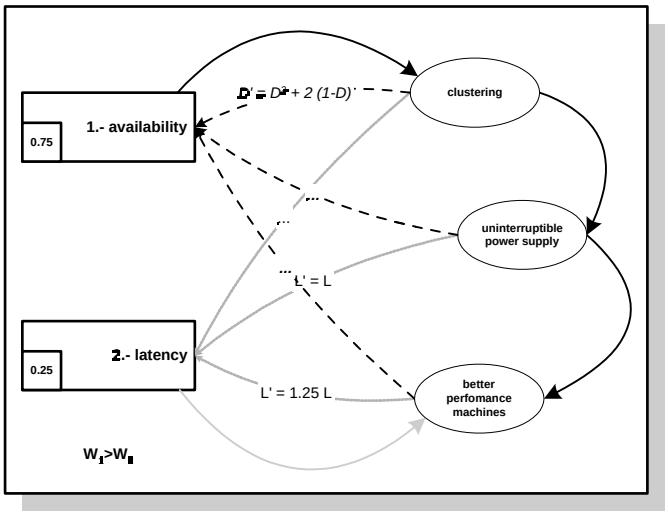

Figura6. Ejemplo ilustrativo de una instancia del metamodelo.

4 Conclusiones y Trabajo Futuro

Hay que decir que respecto a la gestión automática de requisitos de calidad, sobre la que estamos basando nuestro trabajo, los resultados son muy positivos: la descripción tanto sintáctica como semántica de QRL, cuya expresividad nos ha permitido conseguir definiciones formales (y cuantitativas) de los requisitos de calidad que ya ha llevado al desarrollo de herramientas CASE para automatizar la fase de la elicitación y, además, la implementación de diversos componentes básicos para una plataforma de ejecución de servicios Web sensibles a la calidad [18].

Por otro lado, se hace efectivamente prematuro el vislumbrar resultados en esta fase de nuestra investigación, de la que sólo hemos presentado unos esquemas para la gestión automática de alternativas de diseño. Hasta este momento, nuestros resultados han sido la caracterización del problema y la obtención de un metamodelo de cada una de las estructuras involucradas. Sin embargo, el futuro se hace prometedor puesto que, una vez conseguido un metamodelo, completo estamos en disposición de abordar cada una de las posibles soluciones (que sólo hemos esbozados brevemente) y poder realizar comparaciones entre ellas para, finalmente, incluirlas todas juntas en una futura herramienta que ayude al diseñador a producir "la mejor" solución para el sistema que está desarrollando.

Referencias

1. B. Boehm. Engineering context for software architecture. In *Invited talk, 1st Intl. Workshop on Architecture for Software Systems*, April 1995.
2. B. Boehm and H. In. Identifying Quality–Requirements Conflicts. *IEEE Software*, 12(6):25–35, March 1996.
3. G. Booch. *Object–Oriented Analysis and Design with Applications.* Benjamin/Cummings, 2 edition, 1994.
4. J. Bosch and P. Molin. Software Architecture Design: Evaluation and Transformation. In *Proc. of the IEEE Engineering of Computer Based Systems Symposium, ECBS99*, December 1999.
5. L. Chung, B. Nixon, E. Yu, and J. Mylopoulos. *Non-functional Requirements in Software Engineering.* Kluwer Academics Publishers, 2000.
6. J. Clarke, M. Harman, R. Hierons, B. Jones, M. Lumkin, K. Rees, M. Roper, and M. Shepperd. The application of Metaheuristic Search Techniques to Problems in Software Engineering. Technical report, Brunel University, August 2000. SEMINAL-TR-01-2000.
7. Consejo Superior de Informática. Métrica v3: Metodología de Planificación, Desarrollo y Mantenimiento de Sistemas de Información. Technical report, Ministerio de Administraciones Públicas, 2001. http://www.map.es/csi/metrica3.
8. A. Duran. *Un Entorno Metodológico de Ingeniería de Requisitos para Sistemas de Información.* Phd. thesis, University de Sevilla, 2000.
9. X. Franch and P. Botella. Putting non-functional requirements into software architecture. In *Proc. of the IXth Intl. Workshop on Software Specification and Design*, Ise-Shima (Isobe), Japan, April 1998.

10. S. Frolund and J. Koistinen. Quality-of-Service Aware Distributed Object Systems. Technical Report HPL-98-142, Hewlett-Packard Laboratories, July 1998.
11. D. Garlan and D.E. Perry. Introduction to the Special Issue on Software Architecture. *IEEE Transactions on Software Engineering, Special Issue on Software Architecture*, 21(4):269–274, April 1995.
12. I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object–Oriented Software Engineering: A Use Case Driven Approach.* Addison–Wesley, 4ª edition, 1993.
13. P. Kruchten. The 4+1 View Model of Architecture. *IEEE Software*, 12(6):42–50, 1995.
14. P. Clements L. Bass and R. Kazman. *Software Architecture in Practice*. Addison–Wesley, 1998.
15. B. Meyer's. *Object–Oriented Software Construction.* Prentice Hall, 1997.
16. D. Perry and A. Wolf. "Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4), 1992.
17. R.S. Pressman and D. Ince. *Software Engineering: A Practitioner's Approach (European Adaptation).* MacGraw Hill, 2000.
18. A. Ruiz, D. Benavides, M. Gómez, and M. Ramos. Gestión de Catálogos de Requisitos de Calidad e Intermediario de Calidad. Technical report, Dpto. Lenguajes y Sistemas Informáticos, Universidad de Sevilla, 2001. http://zipi.lsi.us.es/qtrader.
19. A. Ruiz, R. Corchuelo, A. Durán, O. Martín, and J.A. Pérez. Prototipado Arquitectónico de Sistemas Distribuidos Abiertos. In *Proc. of the V Jornadas de Trabajo MENHIR*, pages 87–96, Granada, Spain, March 2000.
20. A. Ruiz, R. Corchuelo, A. Durán, and M. Toro. Automated Support for Quality Requirements in Web-services-based Systems. In *To be published in proc. of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems FTDCS'2001*, Bologna, Italia, November 2001.
21. A. Ruiz, A. Durán, R. Corchuelo, and M. Toro. Especificación de Requisitos de Calidad en Sistemas Multiorganizacionales basados en Servicios Web. In *To be published in proc. of the VI Jornadas de Ingeniería del Software y Bases de Datos JisBd'01*, Almagro, Spain, November 2001.
22. A. Ruiz, A. Durán, R. Corchuelo, and M. Toro. Tratamiento Automático de Requisitos de Calidad en Sistemas Multiorganizacionales Basados en la Web. In *Actas de las Jornadas de Ingeniería de Requisitos Aplicada JIRA 2001*, Sevilla, Spain, June 2001.
23. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice–Hall, 1991.
24. M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
25. D. Soni, R.Ñord, and C. Hofmeister. Software architecture in industrial applications. In *Proc. of the XVIIth Intl. Conference on Software Engineering. ICSE' 95*, pages 196–210, Seattle, WA, April 1995. ACM.
26. C. Szyperski. *Component Software*. Addison–Wesley, 1998.
27. B. Tekinerdogan. *A Formal Approach to Software Architecture*. Phd. thesis, University of Twente, 2000.
28. Yourdon Inc. *Yourdon Systems Method: Model–Driven Systems Development*. Prentice–Hall, 1993.