Cómo salvar la distancia entre los requisitos, la arquitectura y la implementación

Una solución de ingeniería de sistemas para el diseño basado en modelos

Marc Segelken
Ingeniería de aplicaciones
MathWorks
Ismaning (Alemania)
msegelke@mathworks.com

Resumen: ¿Cómo podemos realizar tareas de ingeniería de sistemas asociadas con estándares tales como ARP4754 e ISO 26262 parte 4, al tiempo que nos aseguramos de que los requisitos derivados clave se trazan hasta la implementación? El diseño y la actualización de sistemas a gran escala son tareas cada vez más complejas. La trazabilidad y la sincronización en todos los niveles de diseño resultan clave para agilizar el desarrollo de programas a gran escala. Sin embargo, a menudo falta el vínculo entre la ingeniería de sistemas y la implementación del diseño en enfoques "top-down". En este artículo, se demuestra cómo salvar la distancia entre la ingeniería de sistemas y la implementación en un proyecto de actualización de un sistema heredado siguiendo este enfoque "top-down". El ejemplo de caso práctico muestra cómo conseguir actualizar el sistema desde la arquitectura y los requisitos clave de alto nivel hasta la implementación de diseño requerida cuando hay modificaciones. Por último, se realiza un análisis de "trade-off" a nivel de sistema para realizar un análisis de impacto de esta actualización en todo el sistema. Palabras clave: ingeniería de sistemas, desglose de requisitos, modelado de arquitecturas, estereotipos, análisis de arquitecturas, análisis de "trade-off"

I. INTRODUCCIÓN

Debido al incremento constante del tamaño y complejidad de los sistemas; los requisitos que hay que identificar, mantener, derivar, asignar y cumplir; y las restricciones en cuanto a rendimiento, costes, tiempo de comercialización, consumo de energía, peso y otras áreas, la ingeniería de sistemas debe abordar este desafío y gestionar estos factores en la fase de diseño de la arquitectura del sistema. El resultado de este proceso suele ser un conjunto de puntos de partida para el diseño de los subcomponentes, con descripciones de las interfaces, restricciones secundarias y requisitos derivados.

A continuación, se presenta un enfoque de diseño de arquitectura "top-down" centrado en algunas actividades y aspectos clave que complementan el diseño basado en modelos con modelado de arquitectura basados en componentes estereotipados con propiedades para hacer análisis del sistema. Para poder centrarse en cada componente sin perder información crucial sobre el contexto del sistema, la trazabilidad de los requisitos a nivel de sistema y a nivel de componente (derivados), y el uso de vistas filtradas para gestionar la complejidad del sistema son fundamentales. Otros aspectos clave para el éxito son la transición fácil al desarrollo del sistema y la consistencia garantizada.

II. DESGLOSE DE REQUISITOS Y ASIGNACIÓN AL MODELO DE ARQUITECTURA

Un proyecto de ingeniería de sistemas generalmente comienza con requisitos de alto nivel y, opcionalmente, un sistema heredado que podría reutilizarse parcial o estructuralmente hasta cierto punto. La tarea principal es crear una arquitectura con subcomponentes, cada uno de ellos asignado a los requisitos derivados para cumplir con su parte de la funcionalidad total, incluyendo tantos niveles de jerarquía como sea necesario. Por lo tanto, este desglose estructural se acompaña de un desglose similar de los requisitos, de modo que las restricciones de cada subcomponente estén suficientemente definidas.

Debido a la naturaleza creativa de esta exploración del espacio de diseño, normalmente se necesitan muchas iteraciones y pasos de perfeccionamiento antes de llegar a una solución satisfactoria. Los estudios de viabilidad posteriores requieren información adicional considerable, especialmente sobre las restricciones no funcionales que deben cumplir los componentes y el sistema en general. Por consiguiente, este tipo de información también se debe descomponer cuidadosamente a lo largo del diseño de arquitectura. Normalmente, se producen varias arquitecturas, no solo una, y es necesario evaluarlas y compararlas con respecto al

rendimiento, el coste, el tiempo de comercialización y otros factores para elegir la solución de arquitectura final más adecuada.

En los siguientes apartados se describen los tipos de requisitos que se deben tener en cuenta.

A. Requisitos no funcionales

Muchos requisitos hacen referencia al ciclo de vida u otras restricciones no funcionales. Las posibles opciones tienen propiedades tales como peso, coste, fiabilidad, esfuerzo de desarrollo y otros datos de diseño específicos de su dominio que deben ajustarse a estos requisitos no funcionales, así como a sus composiciones, en cada uno de los niveles de jerarquía.

En consecuencia, debe definirse una jerarquía de estereotipos que represente cada uno de los tipos de subcomponentes, y capturen las propiedades según sea necesario, incluidos los requisitos no funcionales mencionados anteriormente. De esta manera, se pueden mantener las características correspondientes de los componentes elegidos, tanto si están disponibles comercialmente (commercial off the shelf, COTS) como si aún están por desarrollar. Para realizar análisis de "trade-off" con las distintas opciones de componentes y arquitecturas, cada solución debe analizarse con respecto a los requisitos no funcionales. Un ejemplo sencillo sería la determinación de la masa y, por lo tanto, el peso de una determinada solución de arquitectura. En este caso, el análisis consiste simplemente en sumar las propiedades de masa de todos los componentes para calcular la masa total. Otro ejemplo sencillo serían los costes de producción o los costes de desarrollo de un sistema, que se calcularían de la misma manera. Para sistemas más complejos, se necesita soporte de herramientas para obtener dichas cifras rápidamente mientras se exploran diferentes soluciones de arquitectura. Con estas herramientas, optimizar las arquitecturas en función de los resultados del análisis de "trade-off" requiere mucho menos esfuerzo.

B. Requisitos funcionales

Al margen de las restricciones temporales de rendimiento, los requisitos funcionales generalmente no se abordan específicamente en el nivel de arquitectura, aparte de desglosarlos en requisitos derivados en paralelo con el desglose del sistema. Es posible realizar un análisis completo en esta etapa inicial con los requisitos formalizados pero, debido a la dificultad de obtener un conjunto completo de requisitos y suposiciones, este razonamiento de tipo suposición-garantía se aplica muy raramente en la práctica y no se incluye en este enfoque metodológico. En su lugar, se propone la simulación a nivel de componente y arquitectura para validar la coherencia de los requisitos localmente, así como el comportamiento general del sistema.

Por lo tanto, se necesita tener la capacidad de simular el mismo modelo de arquitectura utilizado para definir los componentes con sus interfaces e interconexiones para evitar cualquier error causado por una brecha entre la ingeniería de sistemas y el flujo de diseño.

III. GESTIÓN DE LA COMPLEJIDAD

Por definición, la complejidad de los sistemas va más allá de una simple separación entre software y hardware, o de cualquier otra segmentación del sistema. Es necesario centrarse en partes específicas del sistema durante cualquier actividad de diseño para no perderse ni estancarse por la complejidad. Sin embargo, si falta información contextual importante sobre la función de un componente o su entorno dentro del sistema, los defectos de especificación o diseño son inevitables.

Por lo tanto, se debe configurar un subconjunto (vista) adecuado del sistema para entender el problema específico de diseño o análisis, incluyendo solo la información contextual mínima requerida; todo lo que no sea relevante para la tarea en cuestión debe ocultarse.

Aunque encontrar una vista adecuada que cumpla los criterios mencionados anteriormente no resulta fácil, normalmente no es suficiente tener una sola vista para una sub-parte del sistema. El enfoque de una vista universal no sirve aquí, ya que diferentes perspectivas a la hora de observar el sistema requiere diferentes vistas que se superpongan: dependencias funcionales, dependencias organizativas, cuellos de botella, consideraciones sobre consumo de energía, dependencias de proveedores, niveles de madurez, probabilidades de errores, secciones de nivel de integridad de seguridad, etc. La comprensión completa de un aspecto específico de diseño o análisis requiere la capacidad de cambiar rápidamente entre el gran número de diferentes agrupaciones y filtros necesarios en los (sub)sistemas. Dado que todas las vistas distintas de un sistema siempre deben ser consistentes, el soporte de herramientas es crucial para definir y utilizar esas vistas.

IV. SOPORTE DE HERRAMIENTAS

Debido al tamaño y la complejidad de los sistemas, los enfoques clásicos con herramientas de dibujo y hojas de cálculo para abordar las propiedades personalizadas y los análisis correspondientes ya no son apropiados. La probabilidad de que se den problemas de consistencia y problemas causados por datos desactualizados es demasiado alta si no hay un soporte de herramientas específico para que los datos se mantengan unidos y coherentes. Esto es aún más cierto con cualquier enfoque manual para crear algo parecido a una vista del sistema, centrándose solo en aspectos específicos y dejando fuera todo lo demás. Por tanto, son muy recomendables las herramientas de ingeniería de sistemas o los entornos de desarrollo de software y

hardware que proporcionan soluciones para los desafíos y las tareas descritos anteriormente.

Para utilizar la estructura de arquitectura, las definiciones de interfaz, los requisitos asignados, etc. en diseños posteriores de especificaciones de comportamiento, también se recomienda que esta funcionalidad de ingeniería de sistemas se integre con un entorno de desarrollo que permita la continuación fluida del trabajo a nivel de componente, así como la integración automática en el modelo de arquitectura, incluidas las prestaciones de simulación del sistema para la validación.

Para el diseño basado en modelos (diseñar sistemas mixtos de software y hardware con generación automática de código, incluidas las partes físicas y el entorno para fines de simulación), esta funcionalidad de ingeniería de sistemas está disponible con los productos System Composer y Simulink Requirements, que se integran con Simulink y lo amplían con todas las prestaciones descritas anteriormente.

V. CONCLUSIÓN

Este documento describe algunos aspectos clave de la ingeniería de sistemas: a) desglose de la arquitectura con desglose en paralelo y la asignación de requisitos, b) uso de estereotipos para los componentes que asignan valores de propiedades para todo tipo de información de ingeniería o requisitos no funcionales, incluido el análisis correspondiente de esta información a nivel de sistema, c) gestión de la complejidad inherente de los sistemas con un concepto de diferentes vistas que muestran solo la información contextual mínima requerida para la tarea en cuestión, y d) diseño fluido de las especificaciones de componentes según las definiciones de interfaz del modelo de arquitectura, sin riesgo alguno de pérdida o inconsistencia de información. Esto incluye la capacidad de simular el modelo de arquitectura con fines de validación según el comportamiento especificado para los componentes.

No se tratan otros aspectos como la función de un modelo de arquitectura para permitir la comunicación entre las diferentes partes interesadas.

Entre otras cosas, se recomienda que estas soluciones sean complementadas con herramientas de ingeniería de sistemas para eliminar el mantenimiento y el análisis de datos manuales, que son propensos a errores. System Composer y Simulink Requirements son extensiones de Simulink y el diseño basado en modelos que permiten a los usuarios generar descomposiciones jerárquicas de sistemas y software que ofrecen exactamente estas prestaciones.