

Desafíos del Software como Servicios: la respuesta de PLASTIC

Daniel Yankelevich
Área de Innovación – Pragma Consultores
dyankele@pragma.com.ar

Sumario

Este trabajo presenta algunos desafíos que introduce el nuevo modelo de software como servicios en un contexto de movilidad y ubicuidad, y la respuesta que el proyecto PLASTIC (un proyecto europeo de investigación) propone para estos desafíos, con una visión muy concreta. Adicionalmente, se presenta la participación de una organización local en un proyecto europeo con el objetivo de aprender y generar tecnología de última generación.

Palabras clave: servicios, computación móvil, calidad de servicio, acuerdos de nivel de servicio, QoS, SLA, confiabilidad

1. Introducción

Este trabajo tiene dos objetivos: en primer lugar presenta un análisis de los desafíos que introduce el nuevo modelo de Software como Servicios, intentando profundizar más allá de la mera opinión individual, recopilando distintas fuentes y considerando algunos escenarios reales para realizar un análisis. Este análisis permite mostrar la importancia de la introducción de redes B3G (beyond 3G) y la movilidad en este contexto, y los aspectos de importancia a considerar al implementar servicios en este contexto. En la segunda parte del trabajo, se presenta el proyecto PLASTIC en el cual nuestra organización está participando, los objetivos del mismo, y cómo este proyecto da respuesta a algunos de los desafíos planteados en un escenario de uso en particular.

Una contribución marginal de este trabajo es, por lo tanto, mostrar una experiencia de investigación aplicada a la industria, de una empresa argentina participando en un proyecto europeo de investigación dentro del marco del Sexto Programa Marco de la Sociedad de la Información.

El trabajo está organizado de la siguiente manera. En primer lugar, se presenta una visión del software como servicios insertado en la evolución de la ingeniería del software, para ubicar este modelo en un contexto. A partir de este punto, en la sección 3 se presentan los desafíos de este modelo, resumiendo en la sección 4 los principales desafíos. En la sección 5 se presentan unos ejemplos que permiten bajar los desafíos a temas concretos. En la sección 6 se discute la respuesta de PLASTIC a estos temas y en la siguiente sección se discute muy brevemente el rol de nuestra organización. Finalmente, se cierra con algunas conclusiones en la sección 8 y con una bibliografía con algunos comentarios agregados.

2. De hipótesis de mundo cerrado a servicios

Como suele ocurrir, la incorporación del modelo de software como servicios no es una ocurrencia aislada, se trata de una evolución paulatina en la forma en que hacemos software.

Tal como lo presenta con claridad Carlo Ghezzi [Gh07], el desarrollo de software ha evolucionado desde una hipótesis de mundo cerrado a un esquema de mundo abierto. En la ingeniería de software clásica, que de alguna manera ha sentado las bases de la forma en que desarrollamos o debemos desarrollar software, se parte de un universo donde los requerimientos se conocen a priori y el cambio es visto como algo que puede ocurrir pero que debemos intentar evitar (por ejemplo, especificando muy claramente y antes de iniciar la construcción el artefacto software que se construirá). En este universo, las interacciones del sistema con el entorno son conocidas, acotadas, claras y especificables, y los actores son conocidos e interactúan con el sistema en la forma especificada.

Esta visión dio origen a lo que podemos llamar la “ingeniería del software cerrada”: arquitecturas fijas, definidas a priori, una clara separación entre el ambiente de producción o “runtime” y desarrollo. Esta separación en realidad se da entre todos los ambientes (testeo, integración, etc.) y, ortogonalmente, entre los distintos tiempos: el tiempo de testeo es distinto al de desarrollo que a su vez es claramente distinto al de ejecución.

En la ingeniería del software cerrada no existe reestructuración de los sistemas, o bien esta reestructuración es mínima y planificada. Los cambios son centralizados y se coordinan a partir de un mandato central. El modelo típico de desarrollo que responde a este esquema es el Modelo Cascada, que durante tantos años fue tomado como base para múltiples metodologías, herramientas y contextos de desarrollo [HM01, Pr04].

Con los años, el modelo de desarrollo fue modificándose, y se pasó de un modelo estrictamente centralizado a un modelo distribuido. Es clara la influencia de la WEB en estos cambios de modelo y nociones de calidad [OI01], pero aún antes de la existencia de la WEB, se empezó a notar que el cambio es inevitable [Br95] y que incluso es necesario diseñar el software sabiendo que el mismo va a sufrir cambios, diseñar para el cambio [Pa78].

Esta evolución llevó a la explosión de técnicas tales como la orientación a objetos, entre cuyas principales motivaciones se contaba la necesidad de lidiar con el cambio [Me98], más allá de las mejoras en productividad. Poco a poco, el modelo cascada fue evolucionando hacia modelos que incluían prototipación, o modelos que directamente incorporaban el cambio como parte del mismo, como ser modelos en espiral, evolutivos o ágiles [La03].

3. Desafíos de SODA

SODA son las siglas de Service Oriented Development of Applications, desarrollo orientado a servicios. No hay dudas de que el desarrollo de software basado en modelos de servicios y la ingeniería de software orientada a dar sustento a estas actividades presenta interesantes desafíos.

Estos desafíos pueden clasificarse en tres dimensiones: aquellos relacionados con el producto, aquellos relacionados con el proceso de desarrollo, y finalmente, aquellos relacionados con el modelo de negocio del software.

En primer lugar, a nivel producto, a diferencia de lo que ocurría en el pasado, el software debe ser hecho para cambiar, pero no sólo eso: debe estar preparado para cambiar dinámicamente y en formas nunca vistas. La incorporación de un nuevo contexto en el cual estos artefactos de software deben posicionarse (incluyendo dispositivos móviles, y un contexto de ubicuidad) modifica el diseño de los mismos, así como el uso esperado. Por ejemplo, al concepto de sistemas distribuidos se agrega el de distribución de los dueños: en realidad, en muchos de estos sistemas no hay “un dueño” ya que se componen servicios de distintas fuentes para proveer a su vez un nuevo servicio que será usado por diferentes personas o sistemas en diferentes lugares. De hecho, la identificación de los “stakeholders”, personas con intereses en el sistema, cada vez resulta más compleja.

A nivel proceso se producen innovaciones que para la ingeniería de software representan un verdadero cambio en la forma de pensar, algunos lo identifican con un cambio de paradigma [Ha05]. Para dar un ejemplo, hoy la puesta en producción representa un hito mayor en el ciclo de vida de cualquier artefacto que incluya software. Existe una gestión de la configuración antes de la puesta en producción y otra después, que incluye versionado, planificación de “patches”, etc. [Ma05] Al trabajar con servicios, cuya vida se extiende a través de otros servicios que lo integran y que en muchos casos están ya activos (en producción) cuando accedemos a ellos para armar un nuevo servicio, la gestión de configuraciones debe extenderse a toda la vida útil del software, incluyendo sobre todo el runtime. Además, debemos estar preparados para cambios continuos en el mismo y poder gestionar estos cambios en forma coherente (no sería correcto decir “manteniendo el control”, ya que no queda claro quién tiene el control, ver la definición de [Pr04]). Este cambio de alcance en un área tradicional como SCM (software configuration management) afecta también las técnicas a usar, y posiblemente requiera el desarrollo de nuevas herramientas (algunas herramientas comerciales ya están considerando esto) o adaptar y mejorar las metodologías existentes. Esta visión no es futurista, es lo que está ocurriendo ahora: durante el año pasado, mientras participábamos en un proyecto en una organización local que requería integrar servicios externos, vimos el impacto que tuvo una modificación menor a un servicio externo (de validación de crédito) en todo el funcionamiento del proyecto. De hecho, la organización externa realizó la modificación sin pensar en el impacto que esa modificación pudiera tener en quienes usaban el servicio en forma integrada en otros sistemas, rompiendo la cadena de SCM.

Otro ejemplo, más relevante para el contexto de este trabajo, es el de validación y verificación. En líneas generales, se pasa de una etapa de validación o bien de una serie de tareas de validación distribuidas a lo largo de etapas concretas en el ciclo de vida a una necesidad de validación y verificación continuas. Las modificaciones “on line” a los servicios, el cambio de contexto o el uso del mismo servicio en diferentes contextos, entre otros factores, fuerza la necesidad de un esquema continuo de validación y verificación. Y este esquema debe ir aún más allá de la validación de comportamiento, funcional o de correctitud: es necesario saber si un servicio cumple lo que promete a nivel de especificación técnica (con respeto a los llamados requisitos no funcionales) [Pr04].

Por lo general, un servicio especificará, además de sus “funcionalidades”, un compromiso en cuanto a cómo el servicio se llevará a cabo. En la mayor parte de los casos, los principales compromisos tienen que ver con la confiabilidad y el tiempo de respuesta, aunque en muchos casos hay otros factores. Estos compromisos se especifican a través de acuerdos de nivel de servicios (ANS, o SLAs por sus siglas en inglés). Estos acuerdos especificarán en forma cuasi contractual la respuesta de los servicios y el compromiso de los mismos. La interfase de un servicio, al contrario de un tipo abstracto, objeto o módulo, no es sólo funcional: incluye una descripción de los aspectos no funcionales en forma de un SLA. El nivel de calidad (QoS), entonces, quedará definido por estos SLAs y las violaciones al SLA deberán detectarse y en muchos casos serán punibles.

Y esto nos lleva al tercer aspecto a tener en cuenta en lo que respecta a los desafíos: la dimensión de modelo de negocios. Al estar identificado con el modelo cascada y con el cierre monolítico (el momento de “entrega”), el modelo de negocio estándar para paquetes de software está asociado a un pago final (en el caso de llave en mano) o inicial (en el caso de paquetes). El pago es por la licencia, existe un momento de entrega (delivery) de un producto que se verifica y testea. Existe un “test de aprobación”. Luego, se inicia el mantenimiento. Un problema largamente estudiado y muy conocido en el ámbito de la industria del software es el de los altos costos de mantenimiento respecto de estas entregas cerradas. En gran medida, la hipótesis de mundo cerrado (el trabajo sobre ingeniería de requisitos y el énfasis en el trato de los cambios) fue motivado por este problema en el modelo de negocios. De hecho, existe un informe de una agencia americana [GAO94], que al estudiar las causas de los enormes desvíos presupuestarios en proyectos de software, identifica uno de los principales motivos en fallas en el trabajo de requisitos.

En el modelo cerrado, entonces, el énfasis está puesto en el licenciamiento y la entrega del producto. Existen diferentes modelos de negocio, pero siempre asociados a este esquema de entrega y pase a producción: por esfuerzo (horas/hombre), llave en mano, precio fijo, shared, etc. En el modelo de servicios, el énfasis debe estar puesto en el uso del software y en la calidad de servicio, en los acuerdos de nivel de servicio y en las violaciones a los mismos. El modelo general sería algo así como alquiler de software o pago por uso, pero es claro que se requiere un trabajo importante en la identificación de modelos viables. En general, el trabajo de investigadores y académicos se centra en los aspectos tecnológicos o de proceso, en pocos casos se orientan al modelo de negocio, en

donde muchas veces es necesario realizar un trabajo tan profundo como en los demás campos para poder entender y desarrollar una nueva forma de trabajo.

Este cambio de modelo, a nivel producto, proceso, y de modelo de negocio, nos lleva a un marco en el que el software se encontrará permanentemente en un estado “beta”. La (tal vez falsa) seguridad de contar con una separación férrea entre los ambientes de desarrollo, testing y ejecución/producción, irá perdiendo su sentido a partir de determinado punto, ya que posiblemente exista un ambiente de desarrollo donde se desarrollen y prueben “nuevos servicios”, pero los mismos requerirán interactuar con numerosos servicios existentes y a su vez evolucionarán en forma tal que posiblemente los pasajes deban ser muy rápidos y frecuentes.

Esto también responde a una clara necesidad de negocio: la aparición de la llamada “real time enterprise” o empresa en tiempo real, y la necesidad de que los cambios en los sistemas (que en muchos casos son la médula espinal de la organización, sobre todo en las organizaciones basadas en conocimiento) pasen de tomar meses a tomar días y tal vez horas. El pedido de cambio que debía pasar por un proceso de priorización, negociación, evaluación, formalización, análisis, diseño, programación, testeo, pasaje de ambientes, etc., que podía tomar tal vez un mes para un pedido pequeño, debe resolverse en cuestión de horas para responder más rápido que la competencia a un problema de negocio y para mantenerse en un mundo donde se debe trabajar bien y rápido si se quiere sobrevivir. Esta reacción a los cambios y necesidad de flexibilidad se suma a un cambio cultural, dado por la experiencia de los usuarios en el uso de la WEB y a la alfabetización tecnológica, y por un cambio de destinatario, ya que en general puede haber muchos usuarios distintos, en diferentes lugares geográficos y de diferente nivel, para el mismo servicio.

Es necesario mencionar, sin embargo, que de ninguna forma esto representa el fin de las actividades de aseguramiento de la calidad o de validación y verificación, sino un cambio en la forma de realizar las mismas. A modo de argumentación, es interesante señalar el caso de la rutina de búsqueda binaria, `java.util.Arrays`, de la biblioteca JDK, escrita por Joshua Bloch. Esta rutina es una sencilla rutina de búsqueda binaria, utilizada en numerosas aplicaciones, incluyendo rutinas del sistema de archivos del sistema operativo Solaris, y estuvo en producción durante casi 10 años. Adicionalmente, fue publicada en el libro de Programming Pearls [Be00] hace más de 20 años. Aún así, en 2006 se encontró un bug en la rutina [BUG06], producido por un overflow.

El principal punto de esta historia, tal como el mismo Joshua Bloch se encarga de destacar [BUG06], es que uno no puede asumir que el uso de una pieza de software (sea una rutina, componente o servicio) garantiza que la misma no tiene fallas. A veces, fallas que han quedado ocultas durante años salen a la luz cuando se encuentra un nuevo contexto de aplicación, cambia el marco de uso o las exigencias y aspectos no funcionales se vuelven más exigentes (estos aspectos son los que principalmente se consideran en el QoS). En este ejemplo en particular, la rutina fue utilizada durante mucho tiempo sin problemas, pero el error surge al usar la misma con arreglos de más de 2^{30} elementos (algo así como mil millones). En el pasado, uno no tenía arreglos de ese

tamaño: las estructuras de datos elegidas para esa cantidad de información eran otras. Pero al usar arreglos (o listas, el problema se produce igual) para organizar páginas WEB u otros componentes de información WEB, uno llega fácilmente a esos números. El permanente estado “beta”, entonces, no es una excusa para no realizar tareas de validación y verificación o para empeorar la calidad del software.

4. Principales desafíos

Para terminar esta sección, y como conclusiones sobre los nuevos desafíos presentados por esta modalidad o paradigma de software como servicios, se presentan las conclusiones de un workshop de expertos [SM05] en temas de servicios.

Según el workshop europeo citado, involucrando a 35 expertos participantes en 14 proyectos y 2 working groups del área, los principales desafíos de la ingeniería de software en el software orientado a servicios, pueden resumirse en los siguientes puntos:

- + ¿Cómo desarrollar servicios? ¿Qué es ingeniería de servicios?
- + ¿Cómo encontrar servicios? (service discovery)
- + ¿Cómo describirlos? (service description)
- + ¿Cómo verificar que hacen lo que dicen? (service monitoring)
- + ¿Cómo armar nuevos en base a existentes? (service composition)
- + Y, en particular “on the fly” para responder a necesidades (dynamic composition)

Adicionalmente, el tema de Interoperación también es mencionado (debido a la alta heterogeneidad de artefactos que puedan comunicarse, solicitar y proveer servicios).

Estos desafíos se potencian al considerar los servicios en un escenario concreto de uso: el escenario social en el cual se cuenta con una multiplicidad de usuarios en un contexto ubicuo y heterogéneo.

5. Escenarios ejemplo

En esta sección presentaremos brevemente algunos escenarios de servicios y los utilizaremos para analizar los desafíos presentados en forma más detallada. Este análisis servirá como introducción para presentar las respuestas propuestas por el modelo del proyecto PLASTIC.

Escenario I

Imprimir desde un dispositivo personal. Al dar la orden de impresión, se busca la impresora más cercana disponible – o la impresora con mejor relación costo/beneficio dentro de un determinado radio de cercanía. Si uno está en la oficina, puede elegir la impresora de acuerdo al lugar, si uno está en su casa utilizará automáticamente la hogareña, si uno está en tránsito podrá elegir una impresora en el lugar de destino o proponer servicios de impresión disponibles.

Escenario II

Reservar cine y pagar con tarjeta desde un dispositivo móvil.

Escenario III

Prender la luz desde un dispositivo móvil (¿tal vez un control remoto de gestión del hogar?). El mismo servicio podrá levantar las cortinas si es de día y están cerradas o prender una luz eléctrica, frente al mismo pedido, de acuerdo a la situación.

Escenario IV

Enviar mail (notar que esto aplica para cualquier aplicación de gestión de información) desde un dispositivo móvil. Si uno está en el subte, se conecta a la red WiFi del subterráneo. Si uno está en tránsito, se conecta a la red celular. Si uno está en la oficina, se conecta a WiFi de la oficina. Si uno está en su casa, al WiFi casero o por bluetooth a la computadora hogareña. Si uno está en un hotel, al cable o incluso al teléfono.

Estos escenarios presentan algunas características que son comunes a las aplicaciones que unen la organización por servicios en un contexto móvil. A saber:

a. Context Awareness (conciencia del entorno)

Esta es, posiblemente, la característica clave a considerar. Los servicios deben utilizar información sobre el entorno en el que se ejecutan para tomar determinadas decisiones. En algunos casos, el código a ejecutar puede ser distinto según el contexto: por ejemplo, según el dispositivo en el que se estén ejecutando, la red o conectividad disponible, los servicios disponibles para resolver el pedido, incluso la hora del día o lugar geográfico. En particular, es necesario contar con información cuantitativa en forma dinámica, puntualmente sobre características de los servicios disponibles para ser invocados.

b. Location Awareness

No en todos los casos, pero en muchos casos una parte importante de la conciencia del entorno es la identificación del lugar físico en el que se va a ejecutar el servicio. Este lugar puede ser geográfico (por ejemplo, si uno quiere resolver algo por cercanía, como imprimir o ir a retirar entradas en los escenarios de más arriba) o virtual (por ejemplo, dentro de una determinada red).

c. Binding Dinámico

En todos los casos mencionados, el binding de algunos servicios, en particular de un servicio con aquellos servicios que invoca para ser implementando, debe ocurrir en tiempo de ejecución y en forma dinámica. En algunos de los escenarios, es claro que es imposible realizar el binding en forma estática. Incluso realizando un binding estático que incluya todos los servicios que potencialmente podrían usarse, queda el problema de que se estaría invocando una versión antigua de algunos servicios.

El binding dinámico en un contexto móvil presenta algunos problemas técnicos concretos que deben resolverse. Por ejemplo, que la aplicación sea lo suficientemente robusta para mantenerse en funcionamiento si se pierde la conexión o si se cambia de red. Idealmente, debería ser posible migrar de un dispositivo a otro en el medio de una sesión sin perderla (por ejemplo, en el Escenario IV, si uno llega a su casa y quiere continuar el trabajo que estaba realizando en su dispositivo móvil en su máquina hogareña sin volver a comenzar).

La conjunción de binding dinámico, reconfiguración y context awareness tiene muchos puntos en común con la problemática de self healing encarada por autores como [CGSS02]. Si bien la visión desde el punto de vista técnico es un tanto distinta y las soluciones propuestas son diferentes, el análisis de las necesidades que estos escenarios generan resulta muy similar.

d. SLAs

En función de garantizar los requerimientos no funcionales, los servicios deben contar con información clara sobre los compromisos asumidos por los servicios que invoca. En algunos casos, la calidad de servicio es el principal parámetro a tener en cuenta al realizar el binding dinámico o elegir entre posibles servicios disponibles. Si esta información no está disponible en forma que pueda ser analizada en el momento de ejecución, y si no es monitoreada en forma efectiva, no es posible tomar las decisiones adecuadas. La estructura jerárquica de composición de servicios exigen que la descripción de las características no funcionales sea lo más formal posible. Adicionalmente, estos SLAs deben ser monitoreados: definir algo que nadie controla no sería demasiado útil. El monitoreo permanente, online, plantea un problema de eficiencia que debe ser atacado mediante nuevas técnicas [RSCE07].

Extrapolando, es sencillo encontrar escenarios con las mismas características en aplicaciones con un interés social mayor. Entre ellas, aplicaciones de eHealth (área de salud), gestión de emergencias, eVoting (aplicaciones de decisión distribuida, no necesariamente de voto en elecciones). Dentro del proyecto PLASTIC se desarrollarán aplicaciones en estas áreas a modo de ejemplo de aplicación.

6. ¿Qué es PLASTIC?

PLASTIC es el nombre de un proyecto de la UE, dentro del Sexto Programa Marco (IST). El coordinador del mismo es Valerie Issarny, del INRIA. El proyecto es ejecutado por un consorcio de 11 participantes. El proyecto se inició en Febrero del 2006 y terminará en Julio del 2008, con lo cual se ha finalizado una buena parte del trabajo conceptual, pero aún no se han iniciado todas las tareas de programación.

El nombre PLASTIC corresponde a las siglas de Providing Lightweight and Adaptable Service Technology for Pervasive Information and Communication (Proveyendo Tecnología de Servicios Liviana y Adaptable, para la Comunicación e Información de alta Difusión). El nombre de alguna manera refleja los principales aspectos que el

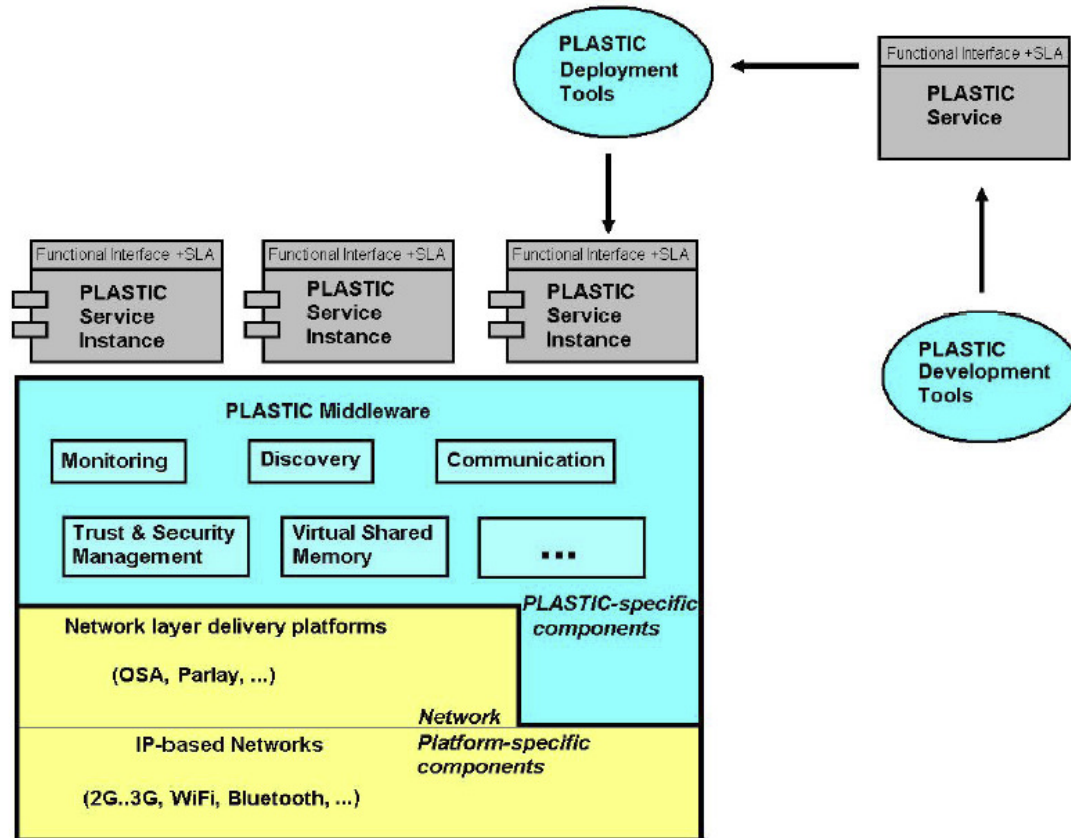
proyecto enfrenta y considera y que responden a algunas de las necesidades enunciadas en la primera parte de este trabajo.

Para empezar, el proyecto tiene su foco en Software Beyond 3G, es decir, software cuya plataforma de ejecución sean las redes de comunicaciones – y el uso que dichas redes van a generar – más allá de la introducción de las tecnologías 3G, que ya se dan por hecho. Estas redes suelen denominarse B3G y tienen como característica un alto grado de heterogeneidad: suelen permitir la incorporación y conexión de dispositivos muy diversos. PLASTIC se sitúa en el contexto presentado con una visión muy clara que representa el uso esperado de estas tecnologías en el contexto de movilidad, ubicuidad y flexibilidad/dinamismo: el software debe tener un alto grado de adaptación a cambios de contexto y debe interactuar con servicios brindados con calidad muy distinta. Esta calidad de servicio (QoS) debe, por lo tanto, ser especificada, medida y controlada; y a su vez la percepción de calidad del usuario es clave.

La otra característica significativa del proyecto es el nivel de confiabilidad que se espera de las aplicaciones que se construyan sobre la plataforma PLASTIC. La palabra clave aquí es “dependability”, ya que se espera que las aplicaciones se construyan con un grado de ingeniería tal que permitan confiar en su buen funcionamiento en distintos contextos. Una aplicación de ehealth, por ejemplo, no puede permitirse un bajo nivel de confiabilidad. Es necesario señalar que el alto nivel de confiabilidad requerido agrega una variable importante al problema, ya que exige un trabajo mayor sobre los aspectos no funcionales, tal como se mencionaba en la sección anterior.

La visión de PLASTIC, entonces, es que los usuarios en la era B3G contarán con una variedad de servicios a nivel de aplicación, estos servicios explotarán la riqueza de la red subyacente, con un alto grado de confiabilidad, sin necesidad de contar con una infraestructura de red integrada disponible en forma sistemática. La percepción de la QoS, que no es constante sino que varía según el usuario y el contexto de uso, requiere que a nivel de las aplicaciones se cuente con información para elegir entre diversos servicios y adaptar las soluciones al contexto. Esta visión está contrapuesta a la de “esconder” la riqueza de la red debajo de una “capa” común que debe ocuparse de todos los aspectos no funcionales en forma homogénea.

Este grafico, tomado de [BEII06], presenta el modelo conceptual sobre el que se sustenta el proyecto.



La Plataforma PLASTIC integrará métodos y herramientas de desarrollo, una capa de middleware que permitirá integrar los distintos aspectos en un ambiente abierto móvil.

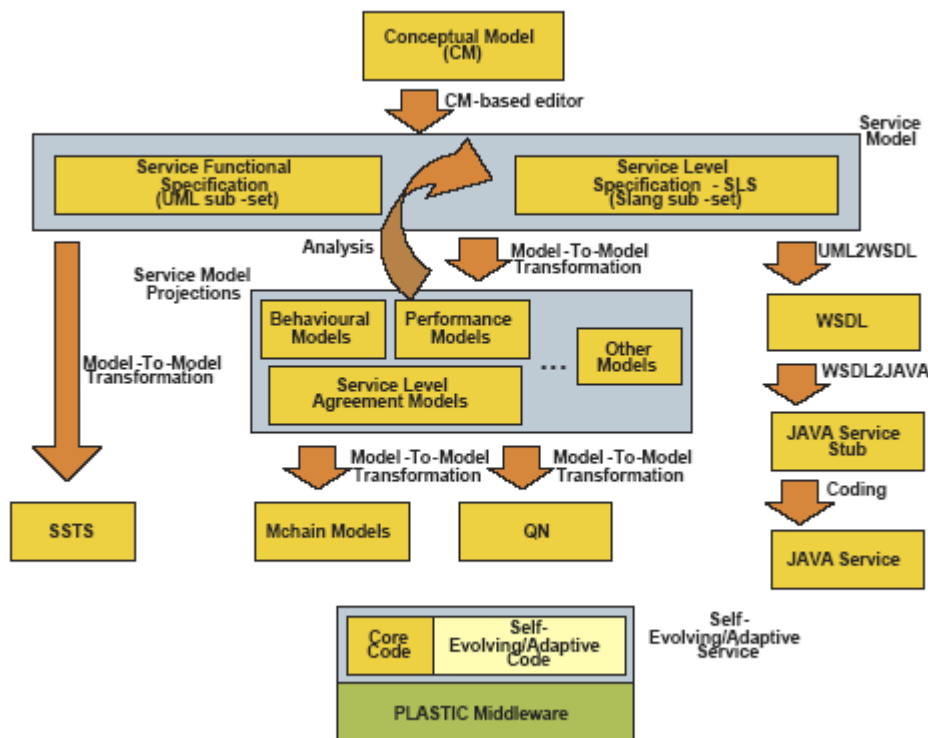
Como puede entenderse, por lo tanto, existen una serie de temas clave que orbitan en la problemática presentada en las secciones anteriores:

- a. context awareness (y resource awareness en particular), es decir, los servicios a nivel aplicación deben tener conciencia/información del contexto en el cual están operando, ya que en distintos contextos las respuestas deben ser distintas. Más aún: este contexto puede ser cambiante.
- b. Robustez + contexto dinámico, esto implica que los modelos para validación y verificación deben estar disponibles y el monitoreo debe realizarse durante la ejecución.
- c. QoS basado en SLA.

Si bien existen plataformas que soportan servicios en redes 2G y 3G y 3GPP [BEII06], estas plataformas se focalizan en network-layered services. La propuesta de PLASTIC debe ser más abarcativa para dar respuesta a los desafíos presentados, incluyendo servicios a nivel de la red, a nivel de middleware y también para cubrir aspectos del

desarrollo de software, desde el modelado hasta la instanciación del código para distintos dispositivos.

PLASTIC propone un modelo y un lenguaje para SLAs, herramientas (por ejemplo, un editor de SLAs) y esquemas de monitoreo, en particular técnicas para monitoreo on line del cumplimiento de SLAs. Claramente, el middleware, es una contribución fundamental. Es importante mencionar que todo el código que se genere en el proyecto será publicado como código abierto (<http://ist-PLASTIC.sourceforge.net>). A su vez, las herramientas tienen inserción en el proceso de desarrollo, ilustrado en el siguiente gráfico, tomado de [Inv07].



Como puede observarse, el proceso de desarrollo se orienta a la programación basada en modelos y en gran medida la programación es en sí una transformación de modelos. Esto no es nuevo, pero la inclusión de modelos de nivel de servicio y de performance al mismo nivel que los de comportamiento, sí lo es. Dada la hipótesis de que las decisiones sobre el uso de servicios son complejas y en muchos casos deben ser tomadas a nivel de la aplicación en un contexto heterogéneo, los aspectos funcionales (de comportamiento, en realidad) y los aspectos no funcionales (performance y SLA) deben estar al mismo nivel.

8. Nuestra experiencia

El rol de nuestra organización en el proyecto se focaliza en el desarrollo de una serie de aplicaciones centradas en aspectos de eHealth, pero también hemos tenido participación en el armado del modelo y en discusiones generales sobre SODA, participación en la difusión de resultados, etc.

La participación en el proyecto nos está permitiendo tener una visibilidad importante de tendencias en el área y del estado del arte en la ingeniería de servicios. Si bien vamos a poder contar con el software desarrollado, en particular con las herramientas de desarrollo, validación y verificación, en forma muy temprana, creemos que el principal aporte para nuestra organización será por el know-how obtenido en el área de servicios y de computación móvil.

8. Conclusiones

El modelo de software como servicios presenta algunos desafíos interesantes. Al sumarle a este modelo movilidad, ubicuidad y heterogeneidad, queda un cocktail complejo, que involucra no sólo la red y el “deployment” de servicios en sí, sino también el desarrollo, el monitoreo, y muchos otros aspectos. Los desafíos son variados y no es real pensar en un layer que uniformise todo y que resuelva los problemas “desde ahí para arriba”, como sería una arquitectura de red por layers. Por el contrario, la información de calidad de servicio y la riqueza de la red subyacente debe estar disponible a nivel de las aplicaciones. Por este factor, es necesario trabajar en soluciones que permitan “negociar” SLAs y permitan combinar servicios teniendo en cuenta la calidad de servicio (QoS). El proyecto PLASTIC, que se enmarca en una serie de iniciativas de la UE al respecto, ataca esta problemática.

9. Bibliografía

[Be00] Bentley, Jon, Programming Pearls, Addison-Wesley, 2000 (2nda edición).

[BEII06] Bertolino, Emmerich, Inverardi, Issarny "Softure: Adaptable, Reliable and Performing Software for the Future", Proc. FRCSS at ETAPS 2006, March 26th - Apr 2nd, Vienna, Austria.

Nota: todo el material de PLASTIC puede encontrarse en el site del proyecto, <http://www-c.inria.fr:9098/PLASTIC/dissemination> , incluyendo los artículos como el de más arriba.

[Br95] Brook, Frederick P. Jr.; The Mythical Man-Month, Essays on Software Engineering, Addison-Wesley, October 1995.

Edición aniversario del libro original, con capítulos agregados, comentarios y revisiones, incluyendo un capítulo llamado “Parnas was right, and I was wrong about information hiding” (pag 271)

[BUG06] SUN Developer Network (SDN), bug ID: 5045582. Ver en http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=5045582.

También es interesante la descripción en el Google Research BLOG realizada por Joshua Bloch, ver <http://googleresearch.blogspot.com/2006/06/extra-extra-read-all-about-it-nearly.html>

[CGSS02] Using Architectural Style as a Basis for Self-repair, Shang-Wen Cheng, David Garlan, Bradley Schmerl, João Pedro Sousa, Bridget Spitznagel, and Peter Steenkiste, *Software Architecture: System Design, Development, and Maintenance* (Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture) Jan Bosch, Morven Gentleman, Christine Hofmeister, Juha Kuusela (Eds), Kluwer Academic Publishers, August 25-31, 2002. pp. 45-59.

[GAO94] United States General Accounting Office, Report to the Committee on Armed Services, U.S. Senate, Stronger Management Practices are Needed to Improve DOD's Software-Intensive Weapon Acquisitions, March 2004, GAO 04-393.

Es un reporte más moderno, existen otros reportes anteriores del GAO sobre la misma problemática, iniciando con el reporte GAO/IMTEC Dec. 1992.

[Gh07] Ghezzi, Carlo, The Challenges of Open-World Software, Proceedings of the 6th International Workshop on Software and Performance, WOSP 2007, ACM Press

[Ha05] Harrison, William, AOSD and Service Oriented Software, in Proceedings AOSD-Europe Workshop (co-located with ECOOP), Glasgow, UK, July 2005

[HM01] Hamlet, Dick and Maybee Joe, The engineering of software: technical foundations for the individual, Addison-Wesley Longman, 2001.

[Inv07] Inverardi, Paola; Software of the Future is the Future of Software?; in Proceedings Trustworthy Global Computing 2006, IMT, Lucca, Italia, Noviembre 2006.

[La03] Larman, Craig; Agile and Iterative Development: A manager's Guide, Addison-Wesley Professional, 2003.

[Ma05] Maraia, Vicent; The Build Master: Microsoft's Software Configuration Management Best Practices (The Addison-Wesley Microsoft Technology Series), Addison-Wesley Professional, 2005.

[Me98] Meyer, Bertrand, Object-oriented Software Construction, International Series in Computer Science (C.A.R. Hoare Series Editor), Prentice Hall, 1988

[OI01] Olsina, Luis, Lafuente, Guillermo, y Rossi, Gustavo; Specifying Quality Characteristics and Attributes for Websites; Proceedings de la Web Engineering Conference, Lecture Notes in Computer Science 2016, Springer, 2001, pag 266-278

[Pa78] Parnas, David, Designing Software for Ease of extension and contraction, Proceedings of the 3rd international conference on Software engineering ICSE '78, IEEE Press, Mayo 1978

[Pr04] Pressman, Roger, Software Engineering: A Practitioner's Approach, McGraw-Hill Science/Engineering/Math, 2004 (6ta edicion)

[PTDL06] Mike P. Papazoglou and Paolo Traverso and Schahram Dustdar and Frank Leymann and Bernd J. Krämer, Service-Oriented Computing: A Research Roadmap, en Service Oriented Computing (Francisco Curbera and Bernd J. Krämer and Mike P. Papazoglou editors), Internationales Begegnungs und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, vol 05462, 2006.

[RSCE07] J. Skene, A. Skene, J. Crampton, and W. Emmerich, The Monitorability of Service-Level Agreements for Application-Service Provision. In Proc. of the 6th Int. Workshop on Software and Performance (WOSP), Buenos Aires, Argentina. ACM Press, February 2007. To appear.

[SM05] Sassen, Anne-Marie and MacMillan Charles, The Service Engineering Area: An overview of its current state and a vision of its future, European Comission, Information Society and Media – Directorate General, Software Technologies, Julio 2005.

Existe una segunda versión del documento de Noviembre del 2005, realizada con la ayuda de los expertos participantes en Future Services Engineering, Bruselas.