Tecno Lógicas
ISSN: 0123-7799
tecnologicas@itm.edu.co
Instituto Tecnológico Metropolitano
Colombia

Serna, Sergio
Desarrollo de un Robot Interactivo para la Distribución de Información
Tecno Lógicas, núm. 18, julio, 2007, pp. 137-169
Instituto Tecnológico Metropolitano
Medellín, Colombia

Disponible en: http://www.redalyc.org/articulo.oa?id=344234311007

Desarrollo de un robot interactivo para la distribución de información

Sergio Serna[1]

Resumen

Este artículo presenta el desarrollo de un robot para la distribución de información. El diseño busca un robot simple e intuitivo, con buen desempeño, que proporcione información actualizada. Se utiliza la metodología de trabajo UN-MÉTODO, la cual nos permite especificar el sistema desde los requisitos del usuario, pasando por sus requisitos no funcionales hasta combinarlos con los objetivos de la organización. Se parte de una descripción verbal, la cual se va refinando poco a poco por medio de un esquema preconceptual, un diagrama de procesos, un diagrama de objetivos y una caracterización de los requisitos no funcionales, hasta llegar a los diagramas que utilizan UML. De esta manera, se logra una especificación semiformal del sistema, que garantice que los requisitos deseados por el usuario y la organización se cumplan. La combinación de técnicas de modelado, permite tomar lo mejor de ellas y usarlas con el fin de hacer más completa la especificación para garantizar un sistema más cercano al deseado. La metodología implementada permite reducir los riesgos inherentes a cualquier proyecto, y facilita la labor de coordinación de los procesos con el fin de optimizar los recursos y cumplir los plazos de entrega de los productos. Al final se obtienen ejecutables para la plataforma específica del sistema, y en el futuro se espera caracterizar los diferentes elementos que hacen parte del robot.

[1] Ingeniero Electricista y Electrónico. Candidato a Magíster en Ingeniería de Sistemas. Líder del grupo de Investigación Biotelectrónica del ITM.
sergioserna@itm.edu.co

Abstract

This paper displays the development of a robot for information distribution. The design looks for a simple and intuitive robot, with good performance, capable to provide updated information. "UN-MÉTODO" methodology is used, which allows specify the system from user requirements, going trough its nonfunctional requirements until combining them with the organizational objectives. The design initializes with a verbal description, which is been refined step by step by means of a preconceptual scheme, a processes diagram, an goals diagram and the characterization of nonfunctional requirements, until UML diagramming is reached. This way a semiformal specification can be reached, is such a way, both, user and organizational requirements can be fulfilled, resources can be optimized and development schedules can be accomplished. At the end of the process executable files can be obtained for the system. In short time is expected to characterize those different elements than constitute the robot.

1. Introducción

Los robots facilitan la labor en tareas que están muy mecanizadas, que son altamente peligrosas o que requieren una gran precisión y velocidad en su desempeño. La distribución de información es una actividad que puede implementarse de una manera muy ventajosa utilizando estas máquinas, ya que puede lograrse un medio de información móvil, interactivo y actualizado.

El objetivo de este trabajo[2] es el desarrollo de un sistema robotizado de reparto de información, impresa, visual y auditiva, que se desplaza en un entorno limitado permitiéndole a una comunidad determinada elegir lo que desea consultar y, al mismo tiempo, retroalimentar el sistema con sus comentarios.

El trabajo hace uso de técnicas de modelado utilizadas en ingeniería de software (Booch: 1999), que nos permiten especificar a cualquier nivel de detalle el sistema a construir. Los estereotipos son ampliamente utilizados en los diagramas con el fin de modelar los componentes hardware del sistema, ya que la interacción software hardware es muy estrecha.

La metodología de desarrollo implementada (Jacobson: 2000) nos permite cuantificar el avance del proyecto con el fin de tomar decisiones acerca de la viabilidad del mismo. Este proceso de desarrollo está soportado en UML, pero también utiliza técnicas de elicitación de requisitos en ingeniería de software como diagrama de objetivos, diagrama de procesos y modelo preconceptual (Arango: 2006).

Este artículo está compuesto inicialmente por un marco teórico que nos permite identificar los aspectos más relevantes en la robótica, seguido por el desarrollo metodológico basado en el proceso de desarrollo unificado y en UN-MÉTODO soportado por UML, posteriormente se muestra el modelo según los resultados

---

[2] Este artículo es parte del proyecto de investigación "Desarrollo de un prototipo robótico para la entrega de información" del grupo de investigación BIOTELECTRÓNICA del ITM.

producidos por la captura de requisitos, allí se muestra también parte de la implementación del software de bajo nivel. Por último, se presentan las conclusiones.

2. Marco Teórico

Son varias las áreas en las cuales se direcciona la investigación robótica en la actualidad: la que relaciona al robot con su entorno, la conductual, la cognitiva, la de desarrollo, la evolutiva y la biológica (Moriello: 2007).

Son varias las actividades que una máquina de éstas realiza: sensa, actúa, se mueve, calcula, toma decisiones y controla, entre otras. Lo que exige una gran interdisciplinariedad del equipo de desarrollo del sistema, ingenieros mecánicos, electricistas, electrónicos, informáticos, físicos, etc.

Qué tan inteligente[3] parezca un robot, está fuertemente ligado a qué tanto y qué tan rápido pueda sensar (Knight: 2000). Es necesario tener claro que los sensores de un robot son muy diferentes de los sensores biológicos que los seres humanos poseemos, y por ende las cosas se perciben de forma también diferente (Toko: 2000). Lo que se sensa está claramente definido por las tareas que se deseen realizar y las tareas que se pretenden, podrán realizarse de una manera más fácil si el estado del robot y del mundo exterior puede o no determinarse, léase sensarse.

Las actuaciones del robot se realizan a través de sus actuadores, estos son muy diferentes de sus pares biológicos, sin embargo, ambos son utilizados para locomoción o manipulación. Con este doble uso de los actuadores podemos pensar en dos líneas dentro de la robótica: los robots móviles y los robots manipuladores (Knight: 2000). En ambas líneas podemos lograr movimientos en una, dos o tres dimensiones, ya sea por medio de llantas u orugas, para el caso de robots móviles; o el desplazamiento de un brazo, para el caso de los manipuladores.

---

[3] No nos interesa en este artículo la discusión de si verdaderamente los robots son inteligentes o no.

Al haber movimiento en el robot podemos esperar algo de autonomía. Entiéndase por autonomía la capacidad de tomar sus propias decisiones y actuar según ellas. Un robot puede ser completamente autónomo o parcialmente autónomo. La autonomía parcial está presente, regularmente, en aquellas máquinas que son operadas remotamente.

Por otra parte, el control, indudablemente, siempre debe estar presente, o sea la coordinación de las acciones de sensado y actuación. Hay muchas formas de enfocar el control de una máquina de éstas: control reactivo, control basado en comportamiento, control deliberativo y control híbrido. Los sistemas reactivos no "piensan"[4], simplemente reaccionan a un estímulo que es sensado, esto los hace bastante rápidos y muy simples en su implementación, ya que no requieren gran cantidad de memoria, ni técnicas de aprendizaje, ni tener un modelo del mundo exterior.

Los demás enfoques de control utilizan técnicas que exigen, en algún grado, al robot "pensar". Pero "pensar" tiene algunas desventajas, como hacer al sistema más lento, y esto en algunos procesos puede ser peligroso y requerir modelos del mundo que le permitan obtener información exacta, lo que hace su implementación más compleja. No obstante, tiene también sus ventajas. Si la máquina "piensa" es posible planear cuidadosamente las acciones a seguir, minimizando los posibles errores. Los sistemas híbridos "piensan" y actúan independientemente, en forma paralela.

3. Metodología

La especificación, el diseño y la implementación de un robot es una tarea que incluye tanto o más componente hardware que software, lo que hace necesario utilizar técnicas que permitan especificar de la forma más detallada posible las características

[4] El proceso de pensar, en el contexto de la Inteligencia Artificial (IA), incluye la captura de datos y el procesamiento de estos utilizando un modelo del entorno y técnicas de IA como lógica difusa, redes neuronales, sistemas expertos, etc. Con el fin de tomar decisiones sobre las salidas.

de estos elementos. Sin embargo, la elicitación[5] de requisitos es una tarea que no debe escapar al diseño de ningún sistema, y para ello se utilizó un método de desarrollo propuesto por el grupo de investigación en ingeniería de software de la Escuela de Sistema de la Universidad Nacional de Colombia, UN-MÉTODO (Arango: 2006).

El método consiste de una serie de entregables que se deben elaborar para realizar el proceso completo de elicitación de requisitos de una pieza de software, lo cual implica la captura de los requisitos del interesado y su procesamiento hasta convertirlos en especificaciones formales o semiformales de la solución. Algunos de los artefactos que se emplean son comunes a muchos métodos de desarrollo: diagrama de procesos, diagrama de objetivos, diagrama causa efecto, diagrama de casos de uso, diagrama de clases, etc. y se pueden considerar estándares de modelamiento.

Son varias las características de UN-MÉTODO que lo hacen una gran alternativa para la elicitación de requisitos (Arango: 2006): Se genera la documentación necesaria para mantener la trazabilidad de la pieza de software desde su concepción hasta la solución final; el método es reiterativo, ya que a medida que se gana conocimiento acerca del dominio del problema se van perfeccionando los artefactos que previamente se habían generado; mantiene una estrecha relación con el estándar de facto de la industria del software, UML; a diferencia de varios métodos de la industria del software la solución es el punto de llegada, no de partida; el método tiene varios entregables que se refieren al estudio de la organización, el problema, la solución y el esquema conceptual de la solución; se utiliza lógica de predicados de primer orden con la cual se representan las restricciones, las derivaciones, las precondiciones y las postcondiciones del sistema; este método parte de artefactos muy cercanos al usuario[6], los cuales paulatinamente se van acercando al lenguaje de la máquina.

---

[5] En el contexto del proceso de desarrollo de software, el término elicitación de requisitos, hace alusión a la captura de los requisitos que el interesado tiene sobre el sistema, así como su procesamiento hasta convertirlos en especificaciones formales o semiformales de la solución

[6] Lenguaje natural

UN-MÉTODO tiene cuatro grandes hitos: el contexto del sistema, el análisis del problema, las propuestas de solución y el esquema conceptual.

En el contexto del sistema se pretende dar la descripción del dominio del problema, identificar los actores, obtener un modelo preconceptual y determinar el modelo del dominio. De estos cuatro artefactos el único que no se utilizó fue el modelo del dominio.

El análisis del problema habla de los procesos del sistema, los objetivos de la organización y sus problemas y causas. Los dos primeros artefactos permiten conocer la forma como el sistema satisface los objetivos y cuáles son los objetivos a satisfacer; el tercer artefacto, relaciona los problemas y las soluciones que se pueden dar a estos primeros. Nuestro sistema es una propuesta para hacer uso de la tecnología en una actividad rutinaria, y no parte de la identificación de un problema en particular, por esta razón no tenemos un diagrama causa-efecto.

La propuesta de solución evalúa los cambios que sufre la organización debido a las propuestas de solución que surjan, así mismo realiza una primera aproximación a las interfaces del sistema en forma de casos de uso y, por último, evalúa el impacto de las decisiones en la solución a los problemas previamente identificados. De este hito se hace uso del diagrama de casos de uso, ya que es paso inicial para la utilización de UML en la especificación.

El esquema conceptual establece las características estructurales y comportamentales del sistema, las cuales se ven reflejadas en los diagramas de clases, los diagramas de secuencias y los diagramas de transición de estados, los cuales son típicos en el lenguaje de modelado estándar, UML. Adicional a estos diagramas, se especifican las restricciones y derivaciones utilizando lógica de predicados y las consultas y transacciones del sistema.

La mayoría de los diagramas utilizados por UN-MÉTODO son de uso estándar en la industria del software y en otras disciplinas (Dardenne: 1993, Ishikawa: 1986, OMG: 2007, Oracle: 2000). Sin embargo, existen algunos que han surgido como resultado del proceso de investigación del grupo.

Un diagrama de gran valor a la hora de validar la especificación con el cliente es el esquema preconceptual. Este esquema está un paso más adelante de las especificaciones textuales, donde ya se han resuelto las ambigüedades introducidas por el lenguaje natural. Este artefacto es una representación gráfica que hace uso de una serie normalizada de verbos, los cuales representan relaciones estructurales y dinámicas, así como sustantivos que representan conceptos, y otra serie de elementos como condicionales, implicaciones y conexiones (Arango: 2006).

El proceso que se siguió para la especificación de nuestro sistema incluye: especificación verbal, esquema preconceptual, diagrama de procesos, diagrama de objetivos, diagrama de casos de uso, diagrama de despliegue y diagrama de clases. En lo que resta de este trabajo se mostrarán cada uno de los diagramas arrojados por la especificación del sistema.

4. Modelo del sistema

Es necesario delimitar claramente el sistema[7] con el fin de especificar el alcance del mismo, esto permitirá establecer límites bien definidos al proyecto, así sea en un lenguaje tan impreciso como el natural. La especificación detallada se debe plasmar en un modelo verbal, que nos indique todas las actividades que se presentan durante el flujo normal, así como las variantes, en un estricto orden cronológico.

4.1. Especificación verbal

La descripción del problema es un resumen preliminar de necesidades que nos sirve de punto de partida para comprender los requisitos que QuestBot debe reunir. Esta descripción, que es una simulación de una descripción dada por un cliente[8], evolucionará con el modelo de requisitos hasta lograr la especificación final del

---

[7] En adelante lo denominaremos QuestBot

[8] En este caso el cliente es el Instituto Tecnológico Metropolitano –ITM-

sistema. Con seguridad esta descripción es incompleta y obviamente informal, lo que la hace posiblemente "incorrecta".

La descripción de QuestBot, que denominaremos modelo verbal, es la siguiente:

El robot informativo QuestBot, inicialmente podrá ser dejado en cualquier lugar, luego de encenderse lo primero que hará QuestBot es un diagnóstico de su operación interna, debe conocer el estado de su batería y verificar la comunicación con el sistema central de información, a continuación debe identificar el área donde se encuentra y validar que ésta es para la cual está programado; por último, verificará que la hora y fecha actual sea de lunes a viernes entre las 6:00 AM y las 10:00 PM, y sábados entre las 6:00 AM y 6:00 PM, si alguna de estas condiciones no se cumple el sistema avisará, si puede hacerlo, de la anomalía y no realizará función alguna ni responderá a estímulos. Validadas las condiciones previas, QuestBot se ubicará, determinando las coordenadas exactas, dentro del área de movilidad por medio del sistema de sensores para ello instalado.

Mientras no se le solicite algún servicio y no detecte la presencia de personas frente a él, QuestBot se desplazará por el área de movilidad durante 15 segundos y se detendrá durante 15 minutos, siempre cuidando de no acercarse demasiado a los límites del área de movilidad y actualizando permanentemente, con la ayuda del sistema de navegación, su posición exacta. El tiempo de espera debe pasarlo en un área de su región de movilidad que facilite el encuentro con los usuarios y en modo de bajo consumo, pero con el monitor encendido.

Siempre que se detecte un obstáculo, no una persona, QuestBot lo esquivará y continuará su camino cuidando no salirse del área de movilidad ya determinada. Cuando se detenga una persona frente a QuestBot éste esperará que el usuario inicie un diálogo por medio de su interfaz o que permanezca allí el tiempo suficiente, para QuestBot tomar la iniciativa. Mientras QuestBot se desplaza deberá controlar su velocidad, de tal manera que esto le permita reaccionar a obstáculos, personas o límites del área.

QuestBot tendrá una interfaz de usuario muy intuitiva de tal manera que toda persona que desee utilizar sus servicios podrá iniciar una interacción sólo presionando un botón, o simplemente parándose frente a él por espacio de tres segundos, momento en el cual QuestBot iniciará la interacción.

Iniciado el diálogo con el usuario, QuestBot saludará de acuerdo con la hora del día y presentará las opciones de información que tiene al momento, con el fin de permitirle elegir la que más le interese a quien interactúa con él. A medida que se despliega la información, ésta debe ir acompañada por palabras aclaratorias pronunciadas por QuestBot; no debe leerse exactamente lo que en la pantalla se muestra. Si la sesión inicia y el usuario no interactúa con QuestBot, éste optará por preguntarle al usuario si desea continuar y si éste no contesta decidirá, después de transcurrido un tiempo significativo, dar por finalizada la sesión. Cualquiera que sea la manera de finalizar la comunicación con el usuario, QuestBot debe despedirse cordialmente y agradecer.

La información almacenada en el sistema de QuestBot debe ser de interés general y muy concreta, y a ella debe ser fácil acceder por medio de la interfaz. Después de cinco minutos de interacción con un usuario QuestBot debe dar por finalizada la sesión, no sin antes terminar de entregar la información ya iniciada, aclarándole que esto es necesario con el fin de permitir el uso de su sistema a más personas.

QuestBot tendrá un dispensador de volantes los cuales repartirá uno por uno a solicitud, disculpándose si en algún momento se quedó sin información. La solicitud del volante la hará el usuario simplemente presionando un botón.

Por fuera de las horas y días hábiles, QuestBot debe permanecer en modo de bajo consumo con el monitor apagado, con lo cual se preservará la batería. Los días hábiles de trabajo QuestBot debe despertar alrededor de las 6:00 AM en forma automática para prestar sus servicios y ponerse en modo de bajo consumo con el monitor apagado alrededor de las 10:00 PM. Cuando su batería se encuentre baja, pero aún funcional, QuestBot se ubicará estratégicamente

para interactuar con los usuarios. Mientras tanto QuestBot emitirá una alarma visual permanente y enviará un mensaje por correo electrónico, al administrador del sistema, cada 15 minutos hasta que sea recargada o cambiada su batería.

Si QuestBot es levantado, éste debe informar, visual y auditivamente, acerca de su condición indicando lo desventajoso de este procedimiento. Para retirar a QuestBot de su área de movilidad se recomienda apagarlo.

La información que portará QuestBot puede ser actualizada a través de la red institucional, por medio de su interfaz inalámbrica. Mientras QuestBot se encuentre interactuando con un usuario la información no puede ser actualizada, lo que se le informará al sistema de actualización. Luego de finalizada la sesión con el usuario, QuestBot iniciará la actualización de información mostrando un mensaje en pantalla acerca de la actividad que lleva a cabo e inhabilitando todos los controles de inicio de sesión. La actualización de información debe hacerse de manera segura.

Es importante tener en cuenta lo informal y limitada de esta descripción, la cual se irá refinando a medida que se avance en la especificación del sistema. Dentro de toda esta descripción se destacarán los requisitos que el sistema debe cumplir, con el fin de lograr un producto de calidad. Se enfatizará en "tener un sistema con buen desempeño y que entregue información exacta". Claramente la información exacta es un requisito deseable para cualquier sistema de información, mientras que el buen desempeño trata de la máxima atención de usuarios en el menor tiempo.

4.2. Esquema preconceptual

La especificación verbal descrita en la sección 4.1 puede ser refinada por una representación gráfica en la cual se han identificado una serie de conceptos y relaciones que eliminan, en cierto grado, las ambigüedades propias del lenguaje natural. Esta descripción es muy cercana al modelo verbal, pero limitada en el uso de verbos que implican relaciones en el sistema. La Figura 1 representa la primera aproximación de nuestro sistema

a una especificación que lo delimita. Los conceptos mostrados en el esquema, rectángulos, surgen del nivel de detalle que se le quiere dar a éste. Por conocimiento previo de los modeladores, se pueden identificar varios elementos hardware en QuestBot que serán claves en su especificación e implementación, como son los sensores, los motores y la fuente de energía. Igualmente, es posible definir dos grandes nodos con memoria y capacidad de procesamiento, como son el sistema de información y el sistema de control. Algunas acciones que QuestBot debe realizar también, están claramente especificadas, se muestra como el sistema de control interactúa con los sensores, indicadores, motores y fuente de poder. Igualmente, la transición entre los diferentes estados de QuestBot se dan a través del sistema de control, el cual también identifica los obstáculos y las personas. Los usuarios también están representados e interactúan con el sistema de información, lo mismo que el administrador del sistema.

Las relaciones, en óvalos, son claramente establecidas por una delimitación en los verbos utilizados; se identifican los verbos *ser* y *tener* como los más comunes, los cuales establecen las relaciones estructurales del sistema. Los demás verbos representan relaciones dinámicas entre conceptos del esquema. Por ejemplo, es posible identificar a *controlar* y *detectar* como actividades que debe realizar el sistema de control.

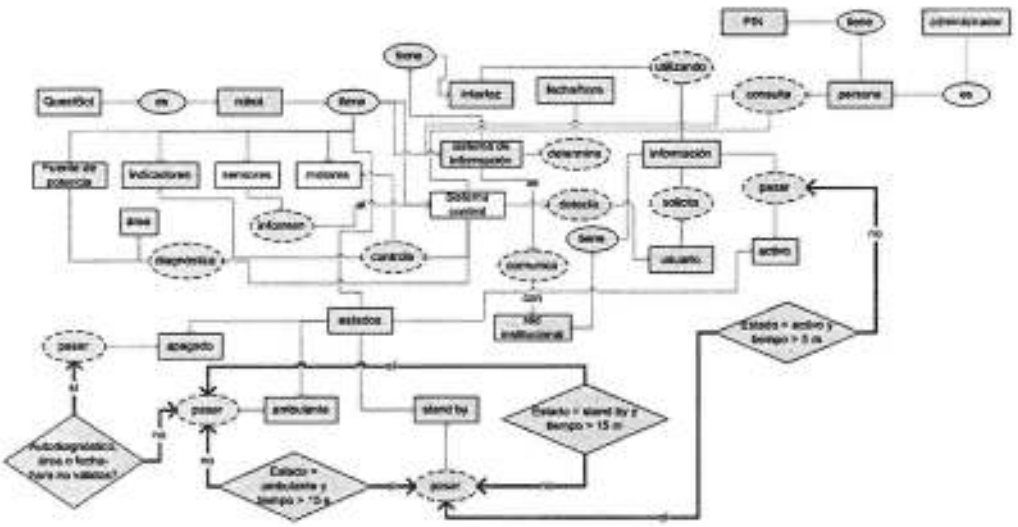
Igualmente, se muestran unos disparadores, condicionales representados por un rombo, que nos permiten establecer algunas reglas del negocio que se deben cumplir. Por ejemplo, es necesario, para que QuestBot cambie de estado, que se cumpla una condición de tiempo y que se encuentre en un estado previo conocido.

Regresando a la Figura 1 podemos observar que QuestBot es un robot que consta de un sistema de información, un sistema de control, motores, sensores, indicadores, fuente de energía y estados. Nótese que los conceptos no tienen que ser dispositivos físicos, pueden ser también condiciones del sistema. Los sensores le informan al sistema de control, y éste controla los motores y las luces indicadores, a la vez que diagnostica la fuente de energía y el

área donde se encuentra ubicado QuestBot. De esta forma se puede hacer un seguimiento al sistema, identificando sus elementos claves y el comportamiento que cada uno de ellos tiene y que contribuye al comportamiento general del sistema.

Es de gran interés los conceptos sistema de control, motores y sensores, ya que actuando sobre ellos será posible reunir algunos de los requisitos no funcionales que van surgiendo en este sistema.

FIGURA 1. ESQUEMA PRECONCEPTUAL QUESTBOT



4.3. Diagrama de procesos

Partiendo del esquema preconceptual de la Figura 1 obtenemos el diagrama de procesos de la Figura 2, allí podemos identificar los flujos de los diferentes procesos y quienes intervienen en cada uno. Questbot, el usuario del sistema, el administrador del mismo y la red Institucional son quienes intervienen en las distintas labores que involucra todo el proceso de información.

Inicialmente Questbot hará un diagnóstico de su sistema, y comprobará las condiciones de su entorno. Esto le permite decidir

el inicio de actividades, dependiendo de si se encuentra en el área apropiada[9] para movilizarse y si la fecha y hora del día son de actividad en el Instituto. Si no se tienen las condiciones ideales de funcionamiento, el administrador del sistema recibirá un aviso, ya sea señal luminosa o correo electrónico, para que tome acciones que permitan corregir la anomalía.

El usuario del sistema, producirá un evento que le indicará a Questbot el requerimiento de sus servicios, esto desencadenará una serie de acciones que apunten a satisfacer las necesidades de información de la comunidad institucional.

Para Questbot es posible detectar el movimiento del usuario y seguirlo por toda la zona de desplazamiento, evitando objetos y otras personas, mientras no se rebasen los límites bien definidos que esta área tiene.

La red institucional puede iniciar el evento de actualización del sistema, el cual será atendido por Questbot sólo si en el momento no se encuentra interactuando con un usuario, si éste no es el caso, terminará su tarea actual y se dispondrá a realizar el proceso de actualización.

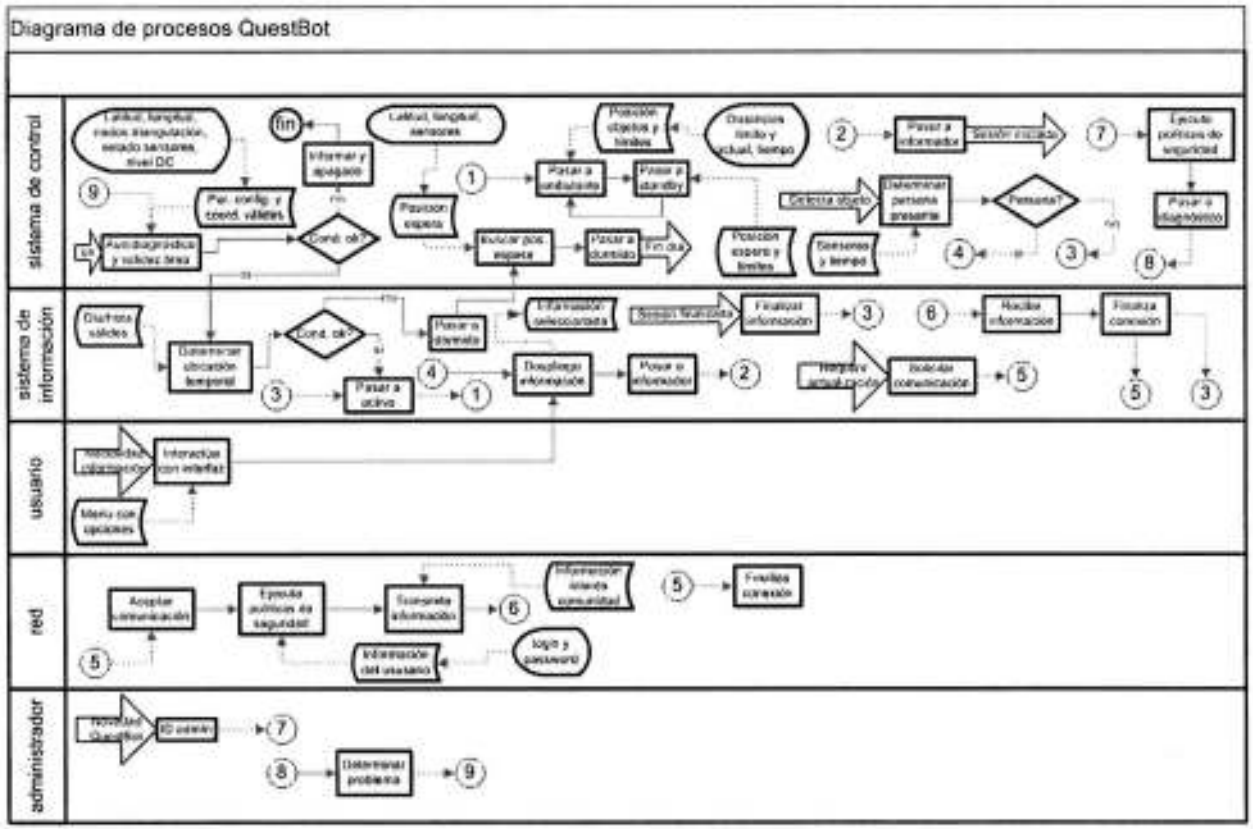
El diagrama de procesos nos permite, desde un punto de vista de muy alto nivel, identificar las tareas, el flujo de éstas y quienes intervienen en su desarrollo. Es un paso inicial que facilitará las siguientes etapas del desarrollo del modelo, donde nos podremos introducir en la especificación de los requerimientos del sistema.

4.4. Diagrama de objetivos

Inicialmente estableceremos los objetivos de alto nivel de la organización (véase la Figura 3), con el fin de subrogarlos a objetivos más especializados. Este diagrama surge a partir de la misión y la visión del Instituto Tecnológico Metropolitano, llevándonos hasta comprender la importancia de distribuir la información de

[9] Implica la comunicación con los sensores que le permitirán determinar el área de desplazamiento

FIGURA 2. DIAGRAMA DE PROCESOS DE QUESTBOT



manera interactiva y siempre actualizada. En este punto es donde entra a jugar un papel muy importante QuestBot (véase objetivo O62 en la Figura 3).

El requisito general para QuestBot es que sea un sistema de alta calidad (véase objetivo O71 en la Figura 4), lo que hace necesario mantener alto desempeño y publicar información exacta. Estos requisitos, por sus características, entran a ser parte de los llamados requisitos no funcionales (NFRs)[10], los cuales pueden tratarse dentro de un marco de trabajo denominado Framework NFR (Chung: 2000).

Es posible subrogar el softgoal 081 de la siguiente forma:

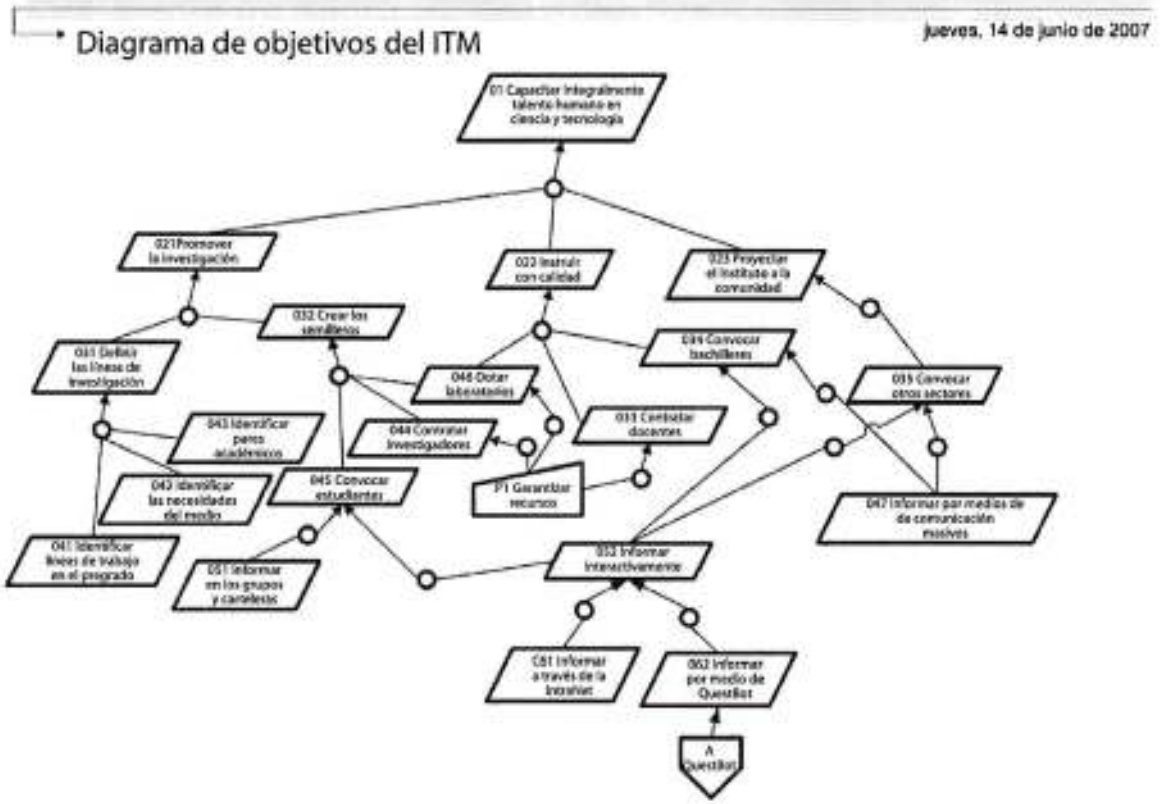
ResponseTime[QuestBot] AND Throughput[QuestBot] AND Space[QuestBot] AND TechnicalServiceTime[QuestBot] SATISFICE Performance[QuestBot]

Esta declaración debe leerse de la siguiente forma: el desempeño de QuestBot se maximizará si se cumple que el tiempo de respuesta es bajo, el número de operaciones por unidad de tiempo es alto, el espacio de almacenamiento utilizado es bajo y el tiempo en servicio técnico es bajo (softgoals O91, O92, O93 y O94). Estas cuatro restricciones están conectadas por un operador AND, esto quiere decir que es obligatorio que todas se cumplan. Sin embargo, puede verse que el softgoal minimizar almacenamiento secundario (O102) entra en conflicto con maximizar almacenamiento secundario (O122), los cuales contribuyen a satisfacer los softgoal de espacio de almacenamiento bajo y número de operaciones por unidad de tiempo alto, respectivamente. Este conflicto debe ser resuelto priorizando los softgoals, y aquí está parte del poder del Framework NFR (Chung: 2000). Este conflicto podemos representarlo de la siguiente forma:

O93 HELP O81 AND O92 HELP O81
O102 HELP O93 AND O121 BREAK O93
O121 HELP O92

[10] Cuando se habla de NFRs los objetivos se denominan softgoal.

FIGURA 3. DIAGRAMA DE OBJETIVOS DE LA ORGANIZACIÓN



Lo anterior debe leerse que los softgoals O92 y O93 contribuyen positivamente a la obtención del softgoal O81 (palabra clave HELP), mientras que el softgoal O93 se ve positivamente influenciado por el softgoal O102, pero definitivamente el softgoal O121 no permite satisfacer el softgoal O93 (palabra clave BREAK), algo no deseado por el softgoal O81. Sin embargo, el softgoal O121 contribuye positivamente a alcanzar el softgoal O92, que es lo que espera el softgoal O81. Claramente aquí hay una contradicción, y es necesario decidir cuál de los dos softgoals (O102, O121) es más importante para el sistema. QuestBot tiene como prioridad el softgoal O121, lo que permite satisfacer O92, pero no permite satisfacer O93, lo que produce una satisfacción parcial del softgoal O81 (véase Figuras 4 y 5). Un análisis similar puede hacerse con el softgoal exactitud (Accuracy), en este caso satisfacer el softgoal será posible si se cumple lo siguiente:

(Certification[RedInstitucional] OR ConsistencyChecking[Info])
AND Validation[RedInstitucional]
SATISFICE Accuracy[QuestBot]

Las contradicciones se presentan debido a que la validación se opone al alto throughput pero contribuye a una buena Accuracy, de la siguiente forma se puede representar:

Validation[RedInstitucional] HELP Accuracy[QuestBot]
Validation[RedInstitucional] HURT Throughput[QuestBot]
Performance[QuestBot] AND Accuracy[QuestBot]
SATISFICE Calidad[QuestBot]

La contribución positiva de Validation a Accuracy se representa con HELP, mientras que la contribución negativa de Validation a Throughput utiliza la palabra HURT. Pero claramente el softgoal alta calidad de QuestBot, será sólo parcialmente satisfecho.

FIGURA 4. PRIMERA PARTE DEL DIAGRAMA DE OBJETIVOS DE QUESTBOT INTEGRADO CON LOS DE LA ORGANIZACIÓN

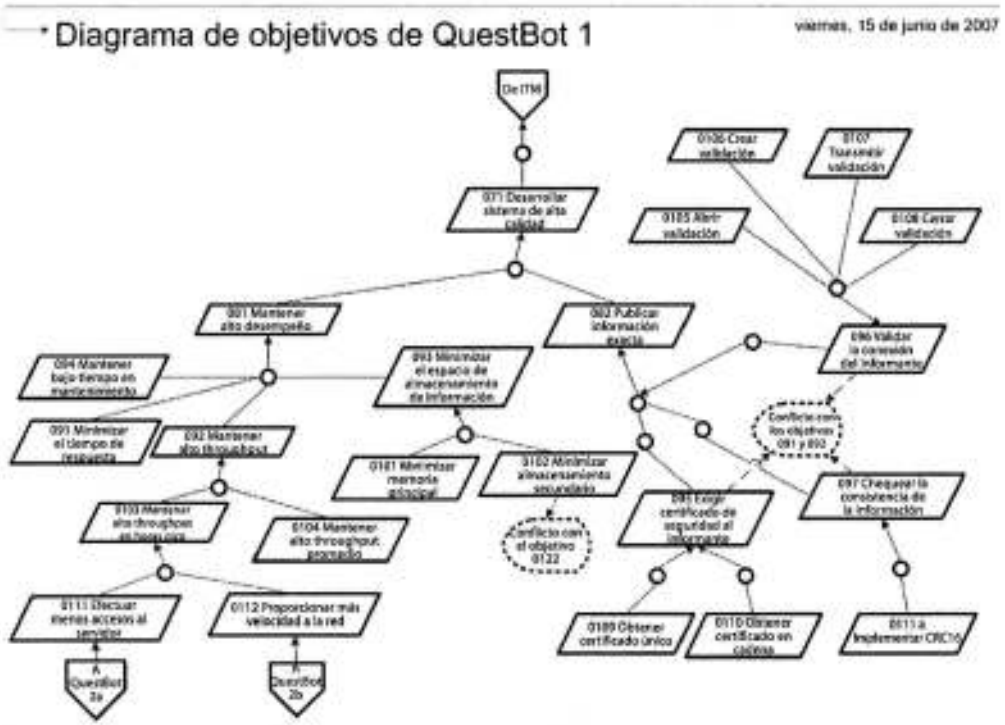
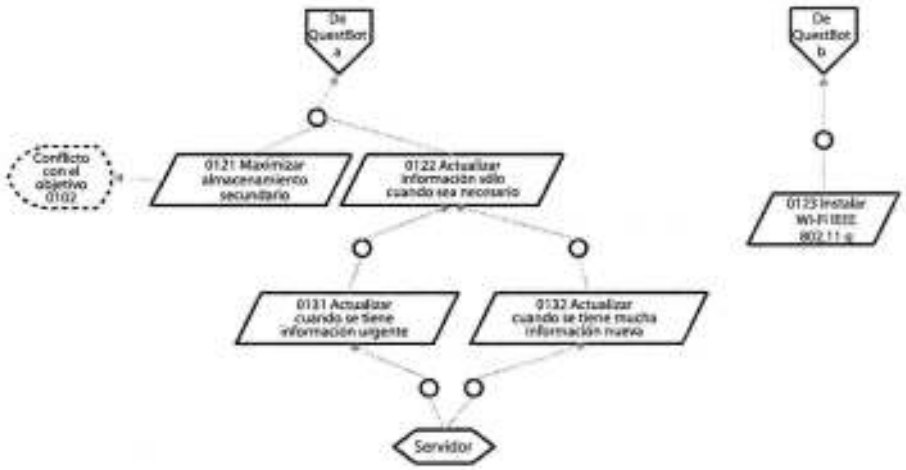


FIGURA 5. SEGUNDA PARTE DEL DIAGRAMA DE OBJETIVOS DE QUESTBOT



4.5. Modelo de casos de uso

La forma como funciona un proceso o como se quiere que funcione hace parte de la especificación de los requisitos, los casos de uso son la herramienta más utilizada, para este fin por los modeladores que trabajan con UML (Booch: 1999). Esta parte de la especificación es, regularmente, el punto de partida para el análisis del sistema. A pesar de que el paradigma orientado a objetos es lo que se imponen en las técnicas actuales de la implementación de sistemas software, los casos de uso son más una forma de modelar procesos que una aproximación a la orientación a objetos.

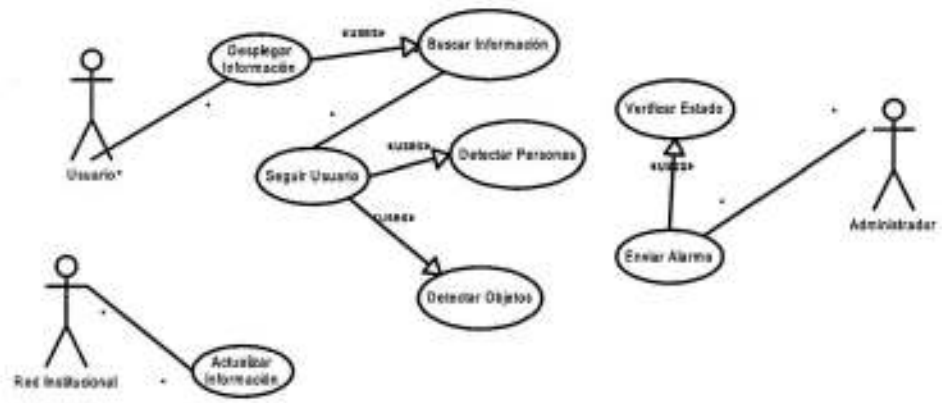
Los conceptos claves asociados con los casos de uso son actores, casos de uso y sujetos (Rumbaugh: 2000). El sujeto es el sistema bajo consideración al cual aplica el caso de uso. Los usuarios y cualquier otro sistema que pueda interactuar con el sujeto son representados como actores. Los actores siempre modelan entidades fuera del sistema. El comportamiento requerido por el sujeto es especificado por uno o más casos de uso, los cuales son definidos de acuerdo con las necesidades de los actores.

La Figura 6 muestra una representación simplificada, utilizando un diagrama de casos de uso, de Questbot. Allí pueden identificarse quiénes interactúan con el sistema, ya sean personas u otros sistemas. Los casos de uso son representaciones de acciones que le agregan valor a un sistema, desde el punto de vista de un actor. Sin embargo, es necesario realizar una especificación más detallada al momento de querer implementar el sistema.

Algunos casos de uso utilizan a otros con el fin de cumplir su tarea, esto se representa con la etiqueta *<<uses>>* en la relación, por ejemplo como para enviar una alarma al administrador es necesario previamente verificar el estado del sistema y determinar si alguna falla existe en el mismo, o la batería se descargó.

Algunas tareas representadas en el diagrama de procesos, no se muestran explícitamente en este diagrama de casos de uso ya que un caso de uso puede contener varios procesos, y sólo cuando se quiere refinar al máximo el modelo se detalla más el diagrama.

Figura 6. Modelo de casos de uso de Questbot



4.6. Diagrama de despliegue del sistema

La disposición de los nodos que forman el sistema y los componentes que residen en ellos, representan el diagrama de despliegue. En Questbot se pueden identificar claramente dos subsistemas, uno es el de alto nivel con el cual el usuario tiene contacto directo y el otro es el de bajo nivel que se encarga de controlar la parte robotizada. La Figura 7 muestra el despliegue del sistema, donde se pueden observar los nodos más importantes tanto del sistema embebido (Bentham: 2002), como son los microcontroladores, los motores, las interfaces seriales, los sensores, etc., y el sistema de alto nivel, compuesto básicamente por un computador y la interfaz serial RS-232 para comunicarse con el sistema embebido.

Se pueden identificar tres procesadores en nuestro sistema embebido, uno encargado de los motores de Questbot, otro de los aspectos propios de la comunicación que permite realizar la triangulación con el fin de lograr la ubicación dentro del área delimitada, y por último un procesador central encargado de la detección de objetos y personas, la comunicación con el sistema de alto nivel y el control de la comunicación serial síncrona con los demás procesadores. Este procesador, adicionalmente se encargará de tomar las decisiones que involucren información capturada por los tres cerebros, ya que es el nodo que controla la comunicación serial de alta velocidad.

Que el procesador central sea un microprocesador o un microcontrolador, es una decisión de diseño que quizás no deba tomarse en este punto del desarrollo, pero dada la experiencia y disponibilidad en el mercado la elección fue casi obvia, de hecho el estereotipo con el cual se marcaron los tres nodos se refiere a los nombres que el fabricante le da a cada uno de ellos.

Un diagrama que nos permite identificar las clases y los nodos en las que residen puede observarse en la Figura 8, esta vista nos permite observar el software de bajo nivel asociado a cada uno de los procesadores.

Se puede observar que algunas clases se utilizan en varios nodos, esto no indica que el sistema sea distribuido o la clase emigre entre nodos, sino que en cada uno de los nodos existe una réplica de cada clase para que pueda ser utilizada en forma local.

FIGURA 7. DIAGRAMA DE DESPLIEGUE DE QUESTBOT

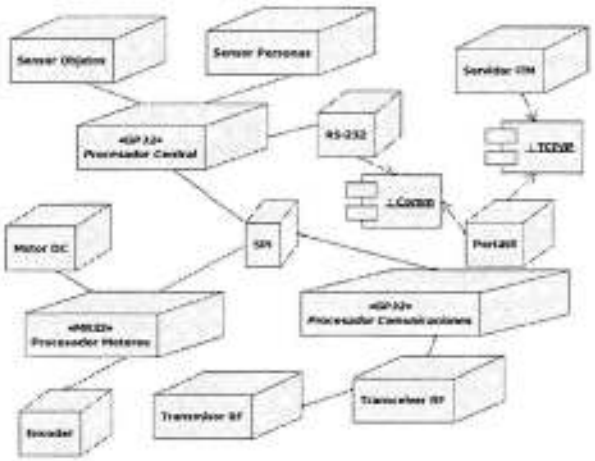
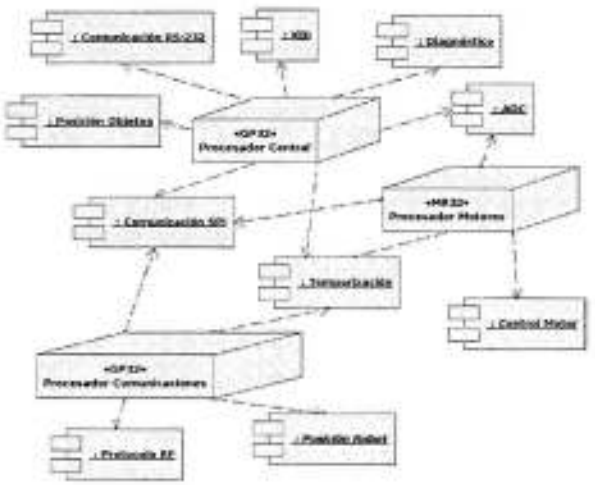
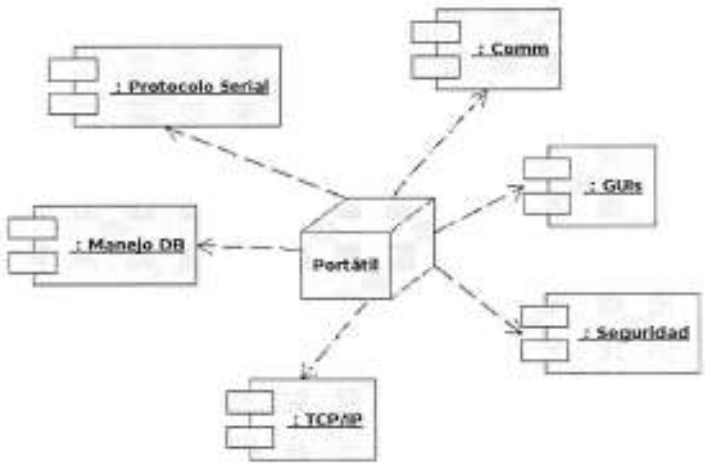


FIGURA 8. DIAGRAMA DE DESPLIEGUE DEL SISTEMA EMBEBIDO DE QUESTBOT, DONDE SE RELACIONAN LOS NODOS Y LAS CLASES DEL SISTEMA



Para el sistema de alto nivel se identifican varios paquetes que nos permitirán lograr el funcionamiento deseado (véase Figura 9), aunque algunas de las clases allí representadas, dependiendo de la herramienta de desarrollo y de la plataforma de ejecución que se utilice, estarán disponibles como parte del sistema operativo o dentro de la jerarquía de clases propia del lenguaje de desarrollo.

FIGURA 9. DIAGRAMA DE DESPLIEGUE DEL SISTEMA DE ALTO NIVEL DE QUESTBOT, DONDE SE RELACIONAN LOS NODOS Y LAS CLASES DEL SISTEMA



4.7. Diagrama de clases

Los diagramas de clases son quizás los más utilizados e importantes en un modelo de un sistema orientado a objetos cuya especificación se realiza con UML (Fowler: 2003). Es tan importante, porque allí se pueden representar los clasificadores[11] que son el alma de la filosofía de diseño orientada a objetos, las

[11] Elemento del modelo que describe características estructurales y de comportamiento. Las clases, actores, componentes, tipos de datos, interfaces, nodos, señales, subsistemas y casos de uso son tipos de clasificadores.

clases. Una clase es una abstracción de un grupo de objetos que cuentan con unas propiedades y unos métodos comunes, lo que permite que sean agrupados dentro de un elemento que los acoja a todos.

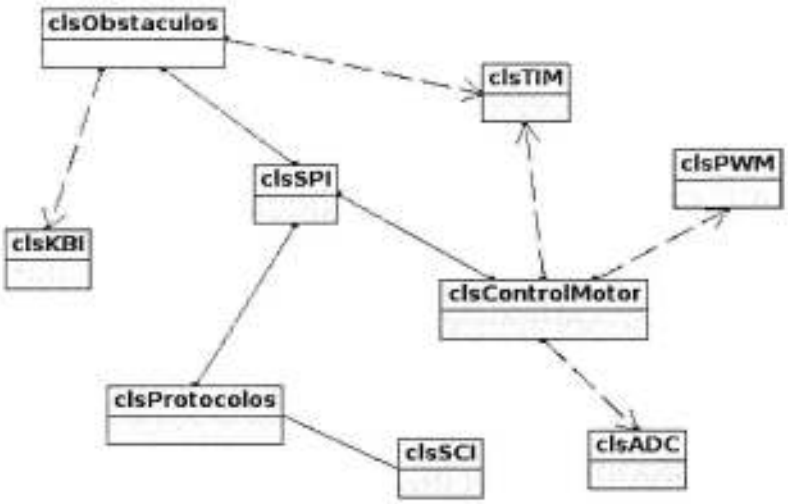
El diagrama de clases es, como casi todo en UML, una representación gráfica de la vista estática del sistema, que muestra una colección de elementos del modelo y sus relaciones. Esta vista modela los conceptos del dominio de la aplicación, así como los conceptos internos inventados como parte de la implementación de la aplicación. Esta visión es estática porque no describe el comportamiento del sistema como una función del tiempo. La Figura 10 muestra una representación de muy alto nivel del diagrama de clases del sistema embebido de Questbot. Si bien no se describen las propiedades y métodos de cada clase, sí pueden identificarse las más importantes, y por su nombre tan descriptivo intuir su función dentro del sistema. Las relaciones que allí se muestran son básicamente de asociación y de dependencia, donde las clases que detectan obstáculos, controlan los motores y manejan los protocolos de comunicación dependen de las otras para cumplir su labor.

4.8. Implementación del software del sistema de bajo nivel

El modelado con UML está orientado al desarrollo del sistema con un lenguaje orientado a objetos, o sea una plataforma de desarrollo que permita la representación directa, en términos del lenguaje, del concepto de clase, así como las diferentes características propias de estos sistemas, como herencia y polimorfismo.

Los procesadores para sistemas embebidos, típicamente poseen recursos de procesamiento y memoria limitados, lo que exige algoritmos bastante eficientes con el fin de no sobrecargar el sistema (Lavagno: 2003). Esto, hasta hace poco, exigía el uso del lenguaje ensamblador como único medio de programar el sistema, lo que en definitiva resultaba en un programa poco amigable y difícil de mantener, ya que cada fabricante tiene un lenguaje ensamblador para sus dispositivos.

FIGURA 10. DIAGRAMA DE CLASES SIMPLIFICADO DEL SISTEMA EMBEBIDO



En el último tiempo, la popularidad y el poder de C han motivado a diversas compañías a construir compiladores basados en este lenguaje para diferentes clases de microprocesadores y microcontroladores. Freescale tiene un IDE para sus microcontroladores de la familia 08 (CPU08: 2001), que son los procesadores de nuestro sistema embebido. Esta suite, permite utilizar todo el poder de C para representar las clases modeladas con UML. Si bien es cierto que C no soporta directamente el concepto de clases, se pueden crear estructuras de datos y asociar a ellas funciones, de tal forma que los métodos y las operaciones, propias de una clase, queden así representadas (Deitel: 1995). La Figura 11 muestra las funciones de la clase SCI, la cual maneja el puerto serial del microcontrolador GP32 de Freescale (GP32: 2002).

FIGURA 11. MÉTODOS DE LA CLASE SCI, PARA EL MANEJO DEL PUERTO SERIAL ASÍNCRONO

```
// Archivo con las rutinas de inicializaci n , transmisi n  y recepci n
// de caracteres por polling e interrupci n del m d  ulo SCI
#include <hidef.h>
#include <MC68HC908GP32.h>
#include <sci.h>

/*****************************************************************************
* Nombre: SetupSCI                                                           *
* ParÆmetros deentrada: velocidad y paridad                                  *
* Valor retornado: ninguno                                                   *
* Autor: Sergio Sema                                                         *
* Fecha: Julio/2005                                                          *
* versi n: 1.0                                                               *
* Descripci n : Configura el m du lo SCI de acuerdo a los parÆmetros         *
* de entrada.          Utilizando un cristal de 4.9152 MHz y la fuente de    *
* reloj para el m  dulo SCI igual a la frecuencia del bus                    *
* (Fbus = 1.2288 MHz), se pueden configurar velocidades de 19200,            *
* 9600, 4800, 2400, 1200, 600, 300 y 150 baudios. Si la frecuencia           *
* del reloj del SCI es el cristal, las velocidades posibles son              *
* 76800, 38400, 19200, 9600, 4800, 1200 y 600 baudios. El m dulo             *
* puede transmitir con paridad par (E), impar (O) o ninguna (N).             *
* Las configuraciones predeterminadas de esta funci n son m dulo             *
* SCI habilitado, bits por dato 8, interrupciones deshabilitadas             *
* y Tx y Rx habilitados                                                      *
*****************************************************************************/

void SetupSCI(uint baudios,uchar paridad) {
 uchar bd;
 ulong preescaler;
 SCC1 = 0x42;                          //Configura el módulo SCI de acuerdo
 SCC2 = 0x0C;                          //a los parámetros de entrada. Utilizando
 switch(paridad) {                     //un cristal de 4.9152 MHz y la fuente
  case 'E':                            //de reloj para el módulo SCI igual a la
   SCC1_PTY = 0;                       //frecuencia del bus (Fbus = 1.2288 MHz),
  break;                               //se pueden configurar velocidades de
  case 'O':                            //19200, 9600, 4800, 2400, 1200, 600, 300
   SCC1_PTY = 1;                       //y 150 baudios. Si la frecuencia del reloj
  break;                               //del SCI es el cristal, las velocidades
  default:                             //posibles son 76800, 38400, 19200, 9600,
   SCC1_PEN = FALSE;                   //4800, 1200 y 600 baudios. El módulo
  break;                               //puede transmitir con paridad par, impar
 }                                     //o ninguna.
 SCBR_SCP = 0;                         //Las configuraciones que esta función no
  if(CONFIG2_SCIBDSRC) {               //cambia son que el módulo SCI inicia
   preescaler = CLOCKS_PER_SEC/256;
   preescaler /= baudios;              //habilitado, los bits por dato son 8,
  }                                    //las interrupciones están deshabilitadas,
  bd = (uchar)preescaler;              //y los módulos de Tx y Rx habilitados
```

```
  switch(bd) {
   case 1:
    SCBR_SCR = 0;
   break;
   case 2:
    SCBR_SCR = 1;
   break;
   case 4:
    SCBR_SCR = 2;
   break;
   case 8:
    SCBR_SCR = 3;
   break;
   case 16:
    SCBR_SCR = 4;
   break;
   case 32:
    SCBR_SCR = 5;
   break;
   case 64:
    SCBR_SCR = 6;
   break;
   case 128:
    SCBR_SCR = 7;
   break;
   default:
    SCBR_SCR = 0;
   break;
  }
}

/*****************************************************************
* Nombre: EnableSCI                                              *
* Parámetros de entrada: verdadero o falso                       *
* Valor retornado: ninguno                                       *
* Autor: Sergio Serna                                            *
* Fecha: Julio/2005                                              *
* versión: 1.0                                                   *
* Descripción: Habilita o deshabilita el módulo SCI              *
*****************************************************************/
void EnableSCI(Bool ena_dis) {
 if(ena_dis) {                          //Habilita o deshabilita el módulo SCI
  SCC1_ENSCI = TRUE;
 }else {

 // Archivo con las rutinas de inicialización, transmisión y recepción
 // de caracteres por polling e interrupción del módulo SCI
 #include <hidef.h>
 #include <MC68HC908GP32.h>
 #include <sci.h>
```

```
  switch(bd) {
   case 1:
     SCBR_SCR = 0;
   break;
   case 2:
     SCBR_SCR = 1;
   break;
   case 4:
     SCBR_SCR = 2;
   break;
   case 8:
     SCBR_SCR = 3;
   break;
   case 16:
     SCBR_SCR = 4;
   break;
   case 32:
     SCBR_SCR = 5;
   break;
   case 64:
     SCBR_SCR = 6;
   break;
   case 128:
     SCBR_SCR = 7;
   break;
   default:
     SCBR_SCR = 0;
   break;
  }
}

/*****************************************************************
* Nombre: EnableSCI                                              *
* Parámetros de entrada: verdadero o falso                       *
* Valor retornado: ninguno                                       *
* Autor: Sergio Serna                                            *
* Fecha: Julio/2005                                              *
* versión: 1.0                                                   *
* Descripción: Habilita o deshabilita el módulo SCI              *
*****************************************************************/
void EnableSCI(Bool ena_dis) {
  if(ena_dis) {                          //Habilita o deshabilita el módulo SCI
    SCC1_ENSCI = TRUE;
  }else {

  // Archivo con las rutinas de inicialización, transmisión y recepción
  // de caracteres por polling e interrupción del módulo SCI
  #include <hidef.h>
  #include <MC68HC908GP32.h>
  #include <sci.h>
```

Allí pueden verse los métodos de la clase, se identifican los de configuración y los de transmisión y recepción. Un método será una función pública, que tiene un prototipo determinado por sus parámetros de entrada y su valor de retorno. Las propiedades podrían representarse como una estructura de datos donde la visibilidad privada[12] (Stroustrup: 1997), sería difícil de implementar.

Los dos primeros archivos incluidos son propios del microcontrolador, el tercero es el archivo de cabecera que permite definir los prototipos de función y algunas variables útiles para la clase [Deitel, 1995].

5. Conclusiones

Questbot presenta varios retos que el grupo de investigación Biotelectrónica no había enfrentado en sus proyectos anteriores. Inicialmente, la complejidad del sistema exige un desarrollo apegado de manera rigurosa a una metodología de diseño e implementación, que facilite la administración de recursos y una planeación disciplinada de las tareas y procesos, minimizando al máximo los riesgos inherentes a todo proyecto. Desde el punto de vista tecnológico, se presentan muchos retos que involucran desde el mismo software del sistema, pasando por aspectos de telecomunicaciones, hasta llegar a elementos mecánicos.

Una motivación adicional, es la proyección del sistema a otro tipo de mercados diferente al de las instituciones académicas, ya que la información y la publicidad son herramientas utilizadas por todo tipo de industrias para llegarles a sus clientes. Y Questbot, es una alternativa novedosa, fácil de utilizar y fácil de administrar.

El estado actual del proyecto, como mandan los procesos de desarrollo de grandes proyectos, pasa por varias etapas: captura de requisitos, diseño e implementación. Poco a poco se están

[12] Principio de mínimo privilegio, parte de la filosofía de diseño en los sistemas orientados a objetos.

obteniendo los requisitos verdaderos del sistema para sus usuarios más importantes, los cuales cada vez se van refinando más. Con esto, se obtienen modelos que mejoran el diseño del sistema completo. Y la implementación, gracias a la modularidad que la filosofía de diseño nos da, se puede realizar paso a paso permitiéndonos posteriormente utilizar todos los módulos creados e integrarlos dentro de la aplicación final.

Estos nuevos retos que trajo Questbot al grupo, implica la apropiación de una serie de herramientas que mejorarán la productividad. Desde el uso de lenguajes de modelado, hasta el diseño de estructuras mecánicas mucho más grandes y robustas que las de proyectos anteriores, pasando por lenguajes de programación de alto nivel, hacen necesaria una fuerte tarea de capacitación para los integrantes del grupo, tratando de identificar allí a quienes ya poseen conocimientos previos o muestran facilidad de apropiación para los nuevos.

La experiencia nos dice que el posicionamiento del robot dentro del área delimitada es un gran reto, ya que la ubicación por medio de dispositivos RF en sectores tan reducidos, no se facilita debido a los principios de operación de éstos (Raisanen: 2003). Esto quizás nos obligue a pensar en otro tipo de alternativas, como el uso de guías para la ubicación de la máquina.

Faltan varios aspectos sobre los cuales trabajar, el software del sistema de alto nivel aún se encuentra sin especificar, el diseño del sistema de alimentación presenta serias dificultades por el tamaño y tipo de estructura que se propone, ya que debe darle por suficiente tiempo autonomía energética a Questbot, y la selección de los motores, con la potencia y torques necesarios para moverlos aproximadamente 60 kg de la estructura.

Bibliografía

ARANGO, Fernando y ZAPATA, Carlos Mario. UN-MÉTODO para la Elicitación de Requisitos de Software. Universidad Nacional de Colombia, 2006.

BENTHAM, Jeremy. TCP/IP Lean, Second Edition. CMP Books, Lawrence, Kansas, USA, 2002.

BOOCH, Grady, RUMBAUGH, James y JACOBSON, Ivar. El Lenguaje Unificado de Modelado. Addisson-Wesley, Madrid 1999.

CHUNG, L., NIXON, B., YU, E. and MYLOPOULOS, J. NonFunctional Requirements in Software Engineering. Kluwer Academic Publisher, Boston, 2000.

CPU08 Central Processor Unit, Reference Manual, Rev. 3. Freescale Semiconductor, Inc. Motorola Literature Distribution, Denver, Colorado, USA, 2001.

DARDENNE, A., VAN, Lamsweerde, A. y FICKAS, S. Goal-Directed Requirements Acquisition. Science of Computer Programming, Vol. 20, 1993. Pp. 3-50.

DEITEL, H. M. y DEITEL, P. J. Como Programar en C/C++, Segunda Edición. Prentice Hall, México, 1995.

FOWLER, Martin. UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition (Paperback). Addisson-Wesley, Pearson Education 2003.

ISHIKAWA, K. Guide to quality control. Asian Productivity Organization, Tokio 1986.

JACOBSON, Ivar, BOOCH, Grady y RUMBAUGH, James. El Proceso Unificado de Desarrollo de Software. Addisson-Wesley, Madrid 2000.

KNIGHT, Terry. Modular Mobile Robotics Equipment. Dearborn, MI, USA: Society of Manufacturing Engineers, 2000. p 1.

LAVAGNO, Luciano. UML for Real : Design of Embedded Real-Time Systems. Secaucus, NJ, USA: Kluwer Academic Publishers, 2003. p i.

MC68HC908GP32, Technical Data, Rev. 6. Freescale Semiconductor, Inc. Motorola Literature Distribution, Denver, Colorado, USA, 2002.

MORIELLO, S. Los Robots Inteligentes Autónomos son la nueva generación. http://www.tendencias21.net. Citado el 15 de Mayo de 2007.

Object Management Group. Unified Modeling Language: Superstructure. Version 2.1.1. February 2007.

Oracle® Corporation. Oracle MethodSM CDM Quick Tour. Oracle Corporation, Redwood City, 2000.

RAISANEN, Antti. Radio Engineering for Wireless Communication and Sensor Applications. Norwood, MA, USA: Artech House, Incorporated, 2003. p i.

RUMBAUGH, James, JACOBSON, Ivar y BOOCH, Grady. El Lenguaje Unificado de Modelado Manual de Referencia. Addsison-Wesley, Madrid 2000.

STROUSTRUP, Bjarne. The C++ Programming Language, Third Edition. Addison Wesley, Reading, Massachusetts, USA, 1997.

TOKO, Kiyoshi. Biomimetic Sensor Technology. West Nyack, NY, USA: Cambridge University Press, 2000. p i.