UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA:

INGENIERÍA ELECTRÓNICA

Trabajo de titulación previo a la obtención del título de:

INGENIERO ELECTRÓNICO

TEMA:

DISEÑO E IMPLEMENTACIÓN DE UN MRAAS, MANIPULADOR ROBÓTICO TIPO SCARA COMO SERVICIO EN LA NUBE.

AUTOR:

EDISON MAURICIO PILATASIG CHANATASIG

TUTOR:

CARLOS GERMÁN PILLAJO ANGOS

Quito, diciembre del 2015

CESIÓN DE DERECHOS DE AUTOR

Yo, Edison Mauricio Pilatasig Chanatasig, con documento de identificación N° 1718599606, manifiesto mi voluntad y cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del trabajo de grado/titulación intitulado: DISEÑO E IMPLEMENTACIÓN DE UN MRAAS, MANIPULADOR ROBÓTICO TIPO SCARA COMO SERVICIO EN LA NUBE, mismo que ha sido desarrollado para optar por el título de: Ingeniero Electrónico, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en mi condición de autor me reservo los derechos morales de la obra antes citada. En concordancia, suscribo este documento en el momento que hago entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Politécnica Salesiana.

.....

Nombre: Edison Mauricio Pilatasig Chanatasig

Cédula: 1718599606

Fecha: Quito, diciembre del 2015

DECLARATORIA DE COAUTORÍA DEL DOCENTE TUTOR

Yo, declaro que bajo mi dirección y asesoría fue desarrollado el proyecto técnico, DISEÑO E IMPLEMENTACIÓN DE UN MRAAS, MANIPULADOR ROBÓTICO TIPO SCARA COMO SERVICIO EN LA NUBE, realizado por Edison Mauricio Pilatasig Chanatasig, obteniendo un producto que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana, para ser considerados como trabajo final de titulación.

Quito, diciembre del 2015

Carlos Germán Pillajo Angos

C.I.: 1709255119

DEDICATORIA

Dedico este trabajo a mi familia, que con su apoyo incondicional han sabido darme fuerzas para continuar el día a día a lo largo de la formación para mi carrera profesional, a mi padre y madre que supieron enseñarme el valor de la responsabilidad para la vida, y sobre todo el valor de la humildad para con los demás, agradeciendo también infinitamente a Dios por llevarme lejos en cada momento, darme fuerzas y por permitirme realizar el presente trabajo.

AGRADECIMIENTO

Con profunda gratitud a la Universidad Politécnica Salesiana que a través de su misión y visión guían a los futuros profesionales, además al Ing. Carlos Pillajo por la labor que desempeñó como director en la realización del presente trabajo.

ÍNDICE

INTRODUCCIÓN 1

CAPÍTULO 1 4

MARCO CONCEPTUAL 4

1.1 Robot como servicio de Cloud Computing 4

1.1.1 Directorio de aplicaciones RAAS en el Cloud Computing 5

1.1.2 Laboratorios como Servicio (LaaS) 7

1.2 La teleoperación y telerrobótica 9

1.2.1 Teleoperación. 9

1.2.2 Telerrobótica 10

1.2.3 Autonomía local 12

1.2.4 Autonomía supervisada 13

1.3 Retardos de comunicación 15

1.4 Herramientas de Software 15

1.4.1 La plataforma .NET 16

1.4.2 Entorno Visual Studio 18

1.4.3 Arquitectura .NET Framework 18

1.4.4 Lenguaje C# 19

1.4.5 Windows Presentation Foundation – WPF 20

1.4.6 Windows Communication Foundation (WCF) 21

1.4.7 Simple Object Access Protocol - SOAP 23

1.4.8 Hosting Internet Information Services –IIS 25

1.4.9 Microsoft SQL Server 26

1.4.10 Entity Framework 27

1.4.11 Kinect SDK 28

1.4.12 OpenWRT 30

1.4.13 Python 31
1.4.14 Arduino IDE 32
1.5 Herramientas de Hardware 32
1.5.1 Arduino Board Mega 2560 33
1.5.2 Dragino - Shield Arduino Yún 33
1.5.3 Robot SCARA Rhino XR series 35
1.5.4 Sensor Sharp GP2Y0A41SK0F 36
1.5.5 Potenciómetro lineal 37
1.5.6 Driver controlador L298n 37
1.5.7 Cámara IP C30 SD PTZ. 38
CAPÍTULO 2 39
DISEÑO 39
2.1 Diseño Conceptual 39
2.1.1 Requerimientos de los usuarios 39
2.1.2 Especificaciones técnicas 40
2.1.3 Selección de la alternativa 41
2.1.4 Análisis de la mejor alternativa 43
2.2 Arquitectura del sistema 44
2.2.1 Caso de uso de la arquitectura general 46
2.2.2 Caso de Uso del Módulo del Operador 47
2.2.3 Caso de Uso del Módulo de Trabajo 48
2.2.4 Diagrama de interacción en secuencia de la arquitectura general 50
2.3 Modelo de la Base de Datos 55
2.4 Procesos en el sistema MRaaS 58
2.4.1 Proceso de conexión con módulo de trabajo 59
2.4.2 Proceso de inserción de movimientos 60
2.4.3 Proceso de petición de movimientos 62

2.4.4 Proceso de fin de sesión 63
CAPITULO 3 65
IMPLEMENTACIÓN 65
3.1 Servicio MRaaS 65
3.1.1 Capa Interfaz del Servicio 67
3.1.2 Capa de lógica de negocio 68
3.1.3 Capa de acceso de datos 70
3.1.4 Capa entidades 74
3.1.5 Base de datos 74
3.1.6 Hosting 76
3.2 Cliente módulo de operador 79
3.2.1 Interfaz de autenticación 81
3.2.2 Interfaz de control 82
3.2.3 Librería Kinect 83
3.2.4 Librería de Conexión Servicio WEB 83
3.2.5 Control de movimientos 84
3.2.6 Administrador de conexión 85
3.3 Cliente módulo de trabajo 86
3.3.1 Interfaz de estados 87
3.3.2 Librería de Conexión de Servicio Web 87
3.3.3 Controlador de eventos 88
3.3.4 Algoritmo de Posición de Coordenadas 89
3.3.5 Hardware Controlador 89
3.3.6 Firmware Controlador 90
3.3.7 Brazo robótico Scara 90
3.3.8 Circuito propuesto 92
PRUEBAS Y RESULTADOS 95

4.1 Consumo del servicio web usando SOAPUI 95
4.1.1 Sesión de clientes 96
4.1.2 Descubrimiento de módulos de trabajo 100
4.1.3 Conexión con el módulo de trabajo 101
4.1.4 Consulta de componentes manipulables 103
4.1.5 Consulta de tiempo de uso 104
4.1.6 Inserción de registros de movimientos 105
4.1.7 Fin de conexión 107
4.2 Manipulación remota 108
CONCLUSIONES 117
RECOMENDACIONES 119
LISTA DE REFERENCIAS 120

ÍNDICE DE FIGURAS

Figura 1. Directorio RaaS..... 6

Figura 2. Esquema de un RaaS en la nube. 7

Figura 3. Sistema de teleoperación general..... 10

Figura 4. Sistema telerrobótico general..... 12

Figura 5. Autonomía local..... 13

Figura 6. Autonomía supervisada..... 14

Figura 7. Evolución de .NET 17

Figura 8. Visual Studio 2012..... 18

Figura 9. Arquitectura .NET. 19

Figura 10. Ejemplo de comunicación simplex 22

Figura 11. Ejemplo de comunicación envío - respuesta..... 23

Figura 12. Ejemplo de comunicación dúplex..... 23

Figura 13. Estructura de un mensaje SOAP 25

Figura 14. Página principal de IIS 8.5– Windows 8.1 26

Figura 15. Arquitecturas de alto nivel de Entity Framework. 28

Figura 16. Kinect..... 29

Figura 17. Arduino Mega 2560. 33

Figura 18. Módulos Dragino Yún Shield. 34

Figura 19. Placa Dragino Yún Shield..... 35

Figura 20. Selective Compliance Assembly Robot Arm. 35

Figura 21. Sensor Sharp. 36

Figura 22. Potenciómetro lineal. 37

Figura 23. Módulo L298n. 38

Figura 24. Cámara IP C30 SD PTZ..... 38

Figura 25. Arquitectura general del sistema MRaaS..... 44

Figura 26. Caso de uso del Sistema MRaaS..... 46
Figura 27. Módulo del Operador..... 47
Figura 28. Módulo de Trabajo..... 49
Figura 29. Diagrama de interacción en secuencia de la arquitectura general. 52
Figura 30. Modelo de la base de datos..... 57
Figura 31. Proceso de conexión con módulo de trabajo. 59
Figura 32. Proceso para insertar movimiento en el Módulo del Operador. ... 61
Figura 33. Proceso de petición de movimientos. 62
Figura 34. Proceso de fin de sesión en el sistema MRaaS. 63
Figura 35. Arquitectura de capas del servicio MRaaS. 66
Figura 36. Capa Interfaz del Servicio..... 68
Figura 37. Capa Lógica de Negocios. 69
Figura 38. Capa acceso de datos S. 71
Figura 39. Contexto en el sistema MRaaS. 75
Figura 40. Hosting MRaaS..... 76
Figura 41. Versión MS SQL en pruebas locales. 77
Figura 42. Versión MS SQL en la nube. 77
Figura 43. Creación de la base de datos. 77
Figura 44. Base de datos en la nube. 78
Figura 45. Servicio MRaaS hacia la nube. 78
Figura 46. Servicio MRaaS en la nube..... 78
Figura 47. Módulo del operador..... 80
Figura 48. Interfaz de autenticación. 81
Figura 49. Interfaz de control. 83
Figura 50. Referencia de servicio..... 84
Figura 51. Componentes en el módulo del operador. 85
Figura 52. Módulo de trabajo. 86

Figura 53. Interfaz de estados. 87
Figura 54. Dispositivos de entrada. 88
Figura 55. Hardware controlador. 90
Figura 56. Brazo robótico Scara modificado. 91
Figura 57. Pines de conexión del brazo robótico Scara. 91
Figura 58. Diagrama de bloques del circuito propuesto. 92
Figura 59. Módulo de trabajo. 94
Figura 60. Nuevo proyecto SOAP. 95
Figura 61. Panorama general del servicio MRaaS en SoapUI. 96
Figura 62. Sesión del operador. 97
Figura 63. Registros de sesión del operador. 98
Figura 64. Sesión del brazo robótico SCARA. 98
Figura 65. Registros de sesión del brazo SCARA. 99
Figura 66. Acción no permitida de inicio de sesión. 100
Figura 67. Descubrir módulos de trabajo. 101
Figura 68. Conexión con el módulo de trabajo. 102
Figura 69. Consulta de componentes manipulables. 104
Figura 70. Consulta de tiempo de uso. 105
Figura 71. Inserción de nuevo registro de movimiento. 106
Figura 72. Fin de conexión. 107
Figura 73. Asignación de IP dinámica en la red local. 108
Figura 74. Configuración de crontab. 108
Figura 75. Inicio manual del proceso. 109
Figura 76. Presentación del módulo de trabajo. 109
Figura 77. Estados en el sistema módulo de trabajo. 110
Figura 78. Configuración de movimiento del brazo robótico SCARA. 111
Figura 79. Aplicación de control y autenticación. 111

Figura 80. Registros en la base de datos. 114

ÍNDICE DE TABLAS

Tabla 1. Requerimientos del usuario operador 39

Tabla 2. Especificaciones técnicas 57

Tabla 3. Alternativa A 58

Tabla 4. Alternativa B 41

Tabla 5. Ventajas y desventajas de la alternativa A 59

Tabla 6. Ventajas y desventajas de la alternativa B 60

Tabla 7. Códigos de estado del sistema MRaaS 70

Tabla 8. Campos de la Base de Datos 74

Tabla 9. Descripción de capa lógica de negocios 86

Tabla 10. Descripción de capa acceso de datos 88

Tabla 11. Conexión de pines 110

RESUMEN

Hoy en día existen muchos diseños y aplicaciones que implican comunicación, control y computación. Estas aplicaciones generan sistemas de control de red, que son sistemas que consisten en una multitud de entidades integradas, nodos sensores acoplados a través de una red de comunicación común para la manipulación y el control de los procesos físicos, como también de dispositivos robóticos, generando aplicaciones que se están volviendo omnipresentes y utilizadas en varias áreas como la salud, la gestión de desastres, fabricación flexible, la educación, etc., pero su aplicación todavía se enfrenta a varios retos, como el costo y la eficiencia de los recursos. Este trabajo se centra en la implementación de un servicio en la nube para el control remoto de un manipulador robótico, generando en su desarrollo una infraestructura (IaaS) para el brazo robótico cuyo servicio se programa en C # (PaaS) que da acceso a los usuarios remotos (SaaS) para crear una unidad MRaaS.

Se diseña una base de datos relacional con las tablas y campos necesarios para cubrir con: gestión de usuarios, gestión de estaciones, permisos y jerarquías de uso de la plataforma, gestión de sesiones, conexiones y registros, se utiliza sistemas modulares en el lado de la estación de trabajo y además capacidades de soporte para diversos actuadores robóticos que pueden ser manipulados simultáneamente, además del uso del sensor Kinect para la detección de movimientos deseados.

ABSTRACT

Today there are many designs and applications involving communication, control and computing. These applications generate network control systems, which are systems composed of a lot of integrated entities, sensor nodes coupled through a common communication network for manipulation and physical control processes, as well as robotic devices, network control systems are generating applications that are becoming ubiquitous and useful in several areas such as health care, disaster management, flexible manufacturing, education, etc., but network control systems implementation still faces several challenges such as cost and efficiency of resources. This work focuses on the implementation of a cloud service for remote control of a robotic manipulator, that develops an infrastructure (IaaS) for the robotic arm, a service is programmed in C # (PaaS) that gives access to remote users (SaaS) to create a MRaaS unit.

A relational data base with tables and fields needed is created to cover with: user management, station management, permissions and hierarchies for using the platform, session management, connection and records, modular systems are used on the side of the workstation and also capabilities to support various robotic actuators in order to be manipulated simultaneously, and the use of Kinect sensor for detecting the desired movements.

INTRODUCCIÓN

Investigaciones previas de sistemas de Cloud Computing orientados a aprovechar recursos físicos, fueron descritos, como “Laboratorio como Servicio” (Tawfik, y otros, 2014)

Donde se estructura por primera vez un modelo de desarrollo, implementación y manipulación remota de un tacómetro usando *LabVIEW-RESTFul Web Service*, como aproximación a un laboratorio físico real, cabe mencionar que dicha investigación fue fruto de un anterior trabajo que daba a conocer las etapas de diseño en el desarrollo de laboratorios remotos para electrónica industrial, y disciplinas de educación, investigación realizada por el mismo autor (Tawfik, Sancristobal, Martin, Diaz, & Castro, 2012)

Aproximándose a la explotación de servicios en la nube para la enseñanza, una investigación para infraestructuras virtuales compuestas por routers, switches y máquinas virtuales bajo demanda (Dinita, Wilson, Winckles, Cirstea, & Jones, 2012).

Se concluyó que el entorno remoto virtualizado, automatizado y auto-mantenido satisface las necesidades de los estudiantes, con muy poca administración continua. En base a estas exposiciones se presentó un modelo de gestión de servicios en la nube para la manipulación de sistemas robóticos para dispositivos móviles, planteando como conclusión general que resulta útil considerar arquitecturas de cómputo en la nube que, mediante servicios web, habiliten la integración de usuario-robot y servidor (Guzmán Luna, Torres Pardo, & Galeano-Hincapié, 2014)

Estos trabajos van aproximándose a lo que sería el concepto de un Manipular Robótico, pero tomando la característica de PaaS (Platform as a Service), o plataforma como servicio (Hunziker, Gajamohan, Waibel, & D' Andrea, 2013)

Se obtuvo el conocimiento de una reciente aplicación llamada “Rapyuta”, que es en sí un repositorio de datos del proyecto “RoboEarth”, que permitiría a robots automatizados reducir la carga computacional, y acceder a datos almacenados en servidores para la realización de tareas y convivencia con otros robots dentro de un mismo ecosistema.

Todos, estos conceptos se van enfocando en la explotación de servicios web y posterior consumo por parte de usuarios, los cuales podrán manipular diversos objetos del mundo real, que son accesibles por medio de la nube, ahora bien, un manipulador en la robótica, es un dispositivo que bajo el control humano permite manipular objetos físicos sin un contacto directo. Lo que se propone es implementar un MRaaS (Manipulador Robótico como Servicio), lo cual define una interrogante, siendo posible utilizar las prestaciones de manipuladores robóticos a través de la red y usando tecnología WCF (Windows Communication Foundation) (Liu, 2012)

¿Qué ventajas y desventajas aportará la implementación de un Manipulador Robótico como servicio en la nube, usando la plataforma WCF?, orientando el proyecto hacia la investigación para aplicaciones educativas, esperando obtener los resultados que la implementación de un sistema como este pueda brindar, y el cómo impulsaría a la virtualización de manipuladores robóticos en la nube, para las redes de telecomunicación y robótica.

En el capítulo referente al Marco Conceptual se hará énfasis en base a las aplicaciones robóticas orientadas a servicios web en la nube, así como estructuras y teorías planteadas de diferentes autores que se orientan a la teleoperación y telerrobótica, lo cual sirve para definir las herramientas hardware y software necesarias para desarrollar el presente trabajo, así se procede al capítulo referente al Diseño del sistema MRaaS, donde se definen arquitecturas de funcionamiento del sistema entre los diferentes actores y procesos necesarios que deben cumplirse en el sistema, se definen los actores que intervienen en el sistema y se deduce la lógica de programación del sistema, para proceder a la Implementación en donde se aplica el análisis realizado y se plasma las ideas en código de programación en las diferentes plataformas existentes, se definen los módulos a usarse en cada cliente así como el diseño de circuitos e interfaces gráficas necesarias, finalmente se realizan pruebas del sistemas tomando en cuenta varios escenarios, y se registran y analizan estos resultados, con el fin de concluir si lo obtenido es en realidad lo planteado en los objetivos del trabajo.

CAPÍTULO 1
MARCO CONCEPTUAL

1.1 Robot como servicio de Cloud Computing

El avance de la arquitectura orientada a servicios y la computación en nube se están convirtiendo en un paradigma de computación dominante. Donde la robótica y la robótica de servicios orientada a la informática comienzan a tomar parte en los últimos años y planean presentarse en gran escala.

En el artículo *Robot as a Service in Cloud Computing*, se presenta una investigación en robótica de servicios orientada a la informática así como también, el diseño, implementación y evaluación de un RaaS (Robot as a Service) (Chen, Du, & García Acosta, 2012).

Para calificar totalmente el RaaS como una unidad de computación en la nube se consideran los estándares de servicio común, plataformas de desarrollo e infraestructura de ejecución, el RaaS soportará lenguajes de programación utilizados para la computación orientada a servicios tales como C # (*C Sharp*), SOA (Service Oriented Architecture), la computación en la nube se caracteriza típicamente por: Software como servicio SaaS (Software as a Service), Plataforma como servicio PaaS (Platform as a service), Infraestructura como Servicio IaaS (Infrastructure as a Service), y Hardware como servicio HaaS (Hardware as a Service).

La computación en la nube (Cloud Computing) permite el desarrollo basado en Web y aplicación: Los desarrolladores utilizan una plataforma Web a través de un navegador Web, desarrollan software, almacenan datos en la Web, configuran el

hardware, la infraestructura (procesamiento, capacidad de memoria, ancho de banda), y ejecutan aplicaciones en la Web. SOA y *Cloud Computing* se consideran la base de aplicaciones basadas en la Web de hoy en día, aunque se han limitado a sistemas de comercio electrónico y computación empresarial. La aplicación hacia dispositivos físicos y sistemas embebidos es un desafío y una fuente de investigación (Chen, Du, & García Acosta, 2012)

El uso de SOA para la robótica brinda ventajas sobre métodos tradicionales en la construcción de un robot. El beneficio principal es tener una capa de servicios comunes con interfaces estándar. Cada servicio representa un componente, como sensores, actuadores, o efector. Las invocaciones de servicios son independientes del dispositivo, implementando la misma aplicación para diferentes plataformas robóticas. Los idiomas utilizados en la escritura de la aplicación y los servicios pueden ser diferentes, permitiendo aplicaciones de robótica utilizando lenguajes de alto nivel como Java y C #. Dependiendo de los controladores disponibles se utilizan diferentes lenguajes, como C, C ++ y lenguajes ensambladores para los programas de control de dispositivo antes de envolverlos en los servicios.

1.1.1 Directorio de aplicaciones RAAS en el Cloud Computing

Por definición un robot es un agente artificial mecánico o virtual, pero existen ciertas condiciones que se proponen para considerarse al RAAS en el Cloud Computing (Chen, Du, & García Acosta, 2012)

En primer lugar una unidad RaaS en la nube es un proveedor de servicios: Cada unidad alberga un repositorio de servicios precargados. Un desarrollador o un cliente pueden desplegar nuevos servicios dentro o eliminar el servicio de un robot, además los servicios pueden ser compartidos con otros robots. En segundo lugar una unidad RaaS en la nube contiene un conjunto de aplicaciones desplegadas: Un desarrollador o cliente pueden componer una nueva aplicación en base a los servicios disponibles en la unidad y fuera de la unidad. En tercer lugar una unidad RaaS es un corredor de servicio: Un cliente puede buscar y descubrir las aplicaciones y servicios desplegados en el robot al navegar por el directorio. Servicios y aplicaciones pueden estar organizados en jerarquía de clases para facilitar el descubrimiento (véase Figura 1).

Directorio RaaS

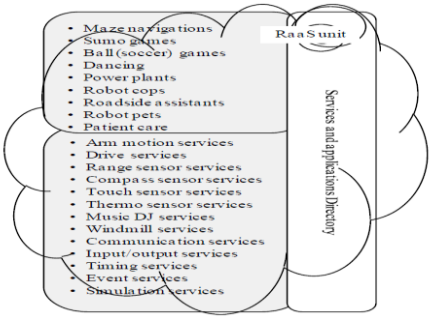


Figura 1. Directorio de servicios y aplicaciones de un RaaS.

Fuente: (Chen, Du, & García Acosta, 2012)

Las unidades RaaS (véase Figura 2) encajan en el Cloud Computing como un software que se comunicará con componentes del sistema operativo, dispositivos y otros componentes de hardware. Las unidades RaaS localmente podrían comunicarse directamente a través del protocolo 802.11 definido por el *Institute of Electrical and Electronics Engineers* (IEEE) o Wi-Fi (Wireless Fidelity), incluso podrán

comunicarse usando tecnologías inalámbricas Bluetooth si dos unidades están muy cerca uno del otro. Pero la comunicación entre un RaaS y otros servicios en la nube son a través de WSDL (Web Services Description Language) (Christensen, Curbera, Meredith, & Weerawarana, 2001)

El protocolo de comunicación que soporta las invocaciones es SOAP (Simple Object Access Protocol) (Box, y otros, 2000). O REST (Representational State Transfer) (Hawke, 2011)

Esquema de un RaaS en la nube

Figura 2. Unidades RaaS conectadas a través de la nube.

Fuente: (Chen, Du, & García Acosta, 2012)

Tras analizar los Robots en *Cloud Computing* se puede inferir que se abre las puertas para muchas aplicaciones tales como: virtualización sobre robots físicos, acceso a recursos de la web por parte de los robots, reducción del costo de hardware en robótica, creación de robots "inteligentes", baratos y más ligeros, funcionalidades como el reconocimiento de objetos y servicios de voz bajo demanda, implementación de una "base de conocimiento compartido" para los robots, ya que los resultados de habilidades aprendidas pueden ser publicados o compartidos entre los robots.

1.1.2 Laboratorios como Servicio (LaaS)

En los últimos años, ha habido un creciente interés en el desarrollo de tecnologías que ofrecen todo tipo de servicios públicos como servicios.

Algunos de los servicios más utilizados en IaaS son: Amazon Elastic Compute Cloud. Servicios en PaaS como: Google App Engine. Servicios en SaaS como: Google Docs. Estas tecnologías permiten una fácil creación, despliegue y uso de infraestructuras, plataformas de desarrollo, o software, respectivamente. Además, un sistema de gestión de laboratorio remoto es necesario para permitir el acceso a equipos remotos, a fin de proporcionar a usuarios remotos la capacidad de utilizar laboratorios de una manera similar a como si estuvieran en la misma ubicación física que el laboratorio. Esta es una herramienta esencial para la educación a distancia, sobre todo en el campo de la ingeniería, donde la experiencia práctica en el uso de equipo es uno de los principales activos de los estudiantes, una vez que terminen su grado y comienzan sus carreras profesionales.

De acuerdo al trabajo titulado *Deconstructing remote laboratories to create Laboratories as a Service (LaaS)*, la creación de laboratorios como servicio (LaaS), presentarían una utilidad basada en servicios para ser consumidos por terceros de una manera versátil y personalizable (Caminero, Robles-Gomez, Ros, & Tobarra, 2014) Así los usuarios de laboratorios remotos pueden crear experimentos que se ajusten a sus necesidades, y evitar funcionalidades innecesarias que puedan poner en riesgo la adquisición de conocimientos por los estudiantes.

Un mismo laboratorio puede ser adaptado a las necesidades de sus usuarios, lo que aumenta la utilidad del sistema y también su desarrollo. Un LaaS brindará beneficios tales como: creación versátil, evitando la necesidad de utilizar todos los servicios, y así mantener a los estudiantes lejos de la complejidad innecesaria que puede poner en peligro su proceso de aprendizaje (Caminero, Robles-Gomez, Ros, & Tobarra, 2014)

Por ejemplo, un laboratorio remoto se puede utilizar para experimentos en un curso básico y también en cursos avanzados, creados mediante el uso de un conjunto más amplio de servicios que permitan a los estudiantes tener más influencia y control sobre los experimentos y sus resultados. Así los mismos laboratorios se utilizan en una amplia serie de experimentos (haciendo visibles los servicios para cada caso), aumentando los niveles de uso.

La creación del LaaS incluye: En primer lugar, los laboratorios remotos deben ser desmontados, esto es, sus funcionalidades deben ser independientes y en un servicio-base. En segundo lugar, los clientes que consumen los servicios de laboratorios se deben crear, de manera similar. Por último, un servidor debe ser elegido con el fin de mantener a estos clientes y crear experimentos personalizables (Belqasmi, Glitho, & Chunyan, 2011)

1.2 La teleoperación y telerrobótica

1.2.1 Teleoperación.

Se entiende por teleoperación a la extensión de las capacidades sensoriales y destreza humana que afecta el espacio en una localización remota. La intervención del operador puede producirse en muchas formas diferentes, por ejemplo: telemanipulación cuando el operador controla un dispositivo manipulable que puede ser un brazo articulado con una pinza localizado remotamente, y teleguiado cuando el operador realiza una tarea como la de guiado de un vehículo sin conductor. También existen términos como teleactuación para referirse a la generación de

comandos para los actuadores remotos, así también se tiene el término de telesensorización cuando se refiere a la adquisición de datos y visualización de información sensorial del medio remoto.

La intervención del operador puede producirse en muchas formas diferentes, desde la teleoperación directa de los actuadores de los grados de libertad de un brazo robótico, hasta una simple especificación de movimientos, o incluso de tareas las cuales se realizan de forma automática empleando para ello lenguajes de programación para el control en la estación de teleoperación y en la localización remota.

En un sistema de teleoperación, el operador actúa a través de las interfaces de entrada generando consignas de movimiento que se transmiten desde el puesto del operador (sistema maestro) hasta los mecanismos actuadores del sistema remoto o esclavo, que obedecen las órdenes del operador (véase Figura 3).

Sistema de teleoperación general

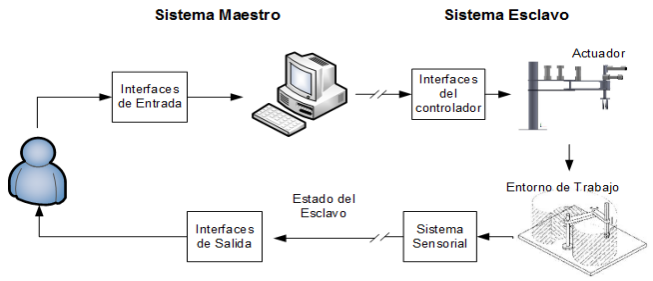


Figura 3. Esquema general de un sistema de teleoperación.

Fuente: Carlos Pillajo

1.2.2 Telerrobótica

La telerrobótica puede considerarse como una forma evolucionada de teleoperación caracterizada por un aumento de autonomía (capacidad de decisión y actuación) en el sistema remoto, manteniendo una intervención significativa del operador humano para supervisión o teleoperación directa. En telerrobótica la comunicación entre el operador humano y la computadora de control se realiza a un nivel más elevado que en la teleoperación convencional, el término teleoperación está asociado al control manual en donde todos los movimientos del robot son especificados por el operador humano sin intervención de la PC. En el control supervisor los movimientos del robot son causados por el operador humano o intervención del PC, en el control automático todos los movimientos del robot son causados por un computador.

Los avances en robótica hacen posible el desarrollo de sistemas más capaces, dotados de sensores para actuar de manera autónoma en un gran número de aplicaciones, surgen por lo tanto técnicas telerrobóticas que estudian la comunicación entre el operador y un robot, que permiten al ser humano dirigir a los robots para que ejecuten las tareas de forma más rápida, más segura y con mayor calidad. La telerrobótica pretende integrar habilidades humanas con los robots aumentando la capacidad de alguno de ellos (Gómez de Gabriel, Ollero, & García , 2006)

Técnicas que pretenden superar las limitaciones humanas, la implantación de las técnicas telerrobóticas no debe limitarse a una reproducción de los movimientos realizados por el operador como en la teleoperación clásica, debe aprovechar la capacidad sensorial y computacional local del robot para aumentar la fiabilidad de las operaciones, facilitar su operación y reducir el tiempo de ejecución, de forma que se tenga un control compartido en el que se combinan las órdenes del operador con

un sistema de control automático. Un sistema pone en comunicación al operador con el entorno, y el sistema telerobótico no es sino un medio que transporta la capacidad de acción y de percepción del operador salvando las barreras de espacio y de tiempo. En este caso el operador puede suministrar órdenes de movimiento de alto nivel, que son transformadas en la estación del operador o el sistema robótico remoto, el robot posee un sistema de percepción que utiliza la información sensorial del entorno para el control del propio o para el envío a la estación del operador (véase Figura 4).

Sistema telerrobótico general

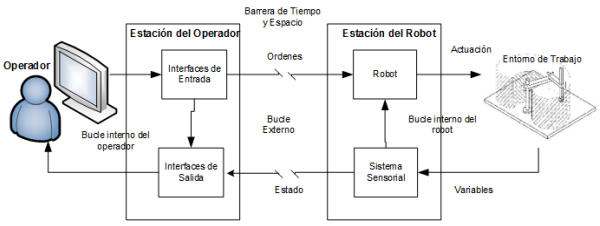


Figura 4. Esquema general de un sistema telerrobótico.

Fuente: Carlos Pillajo

En la práctica los sistemas telerrobóticos combinan la autonomía con la teleoperación. Entre las técnicas más usuales para reducir los problemas ocasionados por el retardo de comunicación están: simulaciones predictivas, desacoplo de tiempo y espacio, control supervisor y teleprogramación

1.2.3 Autonomía local

Cuando existen retardos en la transmisión se emplean técnicas que no utilizan realimentación de esfuerzos sino que emplean solamente realimentación visual. Si es necesario dotar al manipulador esclavo de una cierta acomodación, ésta se efectúa sólo localmente (véase Figura 5).

Autonomía local

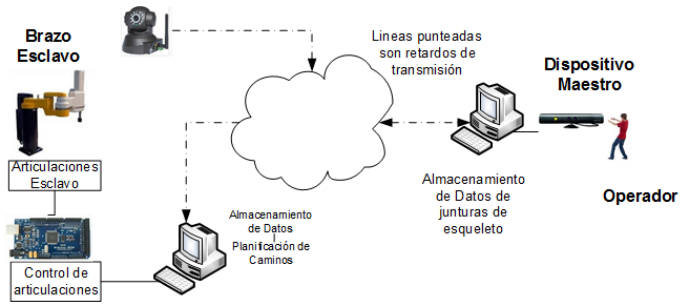


Figura 5. Control telerrobótico con autonomía local.

Fuente: Carlos Pillajo

1.2.4 Autonomía supervisada

En la autonomía supervisada (véase Figura 6), las órdenes de ejecución autónoma son generadas por el operador humano y enviadas para su ejecución en el robot.

Una instrucción puede ser enviada inmediatamente o ser simulada y modificada iterativamente antes de enviarla para su ejecución en el robot real (Gómez de Gabriel, Ollero, & García , 2006)

También las órdenes individuales pueden ser descripciones completas del movimiento o instrucciones para un módulo que especifiquen sólo modificaciones de control o monitoricen un módulo específico del sistema remoto.

Mediante esta técnica, el operador humano, en lugar de estar dentro del bucle de control, puede comunicar al robot remoto, a través del canal de comunicación. Dado que el robot es un sistema dotado de una computadora capaz de recibir, almacenar y ejecutar órdenes utilizando sus propios sensores y actuadores, no hay retardo significativo en su bucle de control local, por lo que no hay inestabilidad causada por el retardo de comunicación. No obstante puede existir un retardo en el bucle

supervisor, este retardo en la confirmación de los resultados deseados al supervisor es aceptable si: el retardo es menor que el tiempo de ejecución de la tarea, la instrucción es una gran parte de la tarea completa, los aspectos no predecibles del entorno no cambian de forma muy rápida, el sistema automático remoto es fiable.

Autonomía supervisada

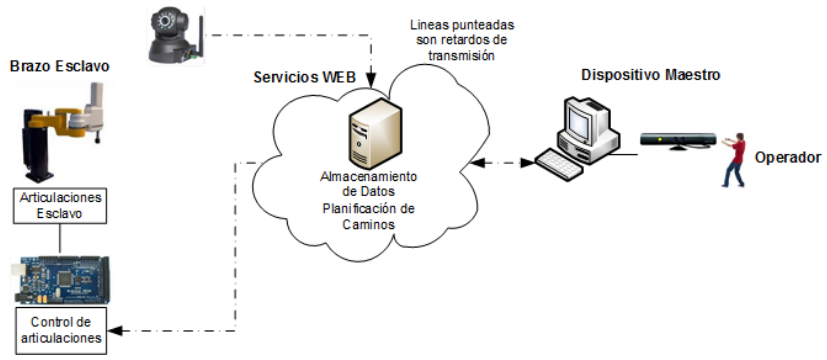


Figura 6. Control Telerrobótico con Servicios Web y autonomía Supervisada.

Fuente: Carlos Pillajo

Un operador humano puede adoptar una estrategia “mover y esperar” para evitar la inestabilidad. Esto significa que el operador puede efectuar sólo un pequeño cambio de posición incremental en bucle abierto y esperar ver cómo el esclavo lo haya reproducido, se considera que el rendimiento de la tele operación es función predecible del retardo, la relación entre distancia de movimiento y precisión requerida, y otros aspectos de la realimentación (Gómez de Gabriel, Ollero, & García , 2006)

También se debe tomar en cuenta el tiempo de ejecución de una tarea se altera no sólo por el tiempo de retardo, sino por el número de grados de libertad del robot.

1.3 Retardos de comunicación

Se toman en cuenta ciertas consideraciones y ejemplos acerca del retardo de comunicación en el bucle de control ya que es un factor importante porque puede causar inestabilidad (Gómez de Gabriel, Ollero, & García , 2006)

La magnitud de retardo condiciona el tipo de técnicas necesarias para teleoperar de manera segura. En la comunicación entre el sistema local y el sistema remoto pueden darse tiempo del orden de varios segundos.

En algunas aplicaciones (control remoto a través de internet) el retardo de tiempo es con frecuencia pequeño, pero no determinista y ocasionalmente demasiado grande para algunas aplicaciones. El retardo en la obtención de la posición y la información sensorial de los robots remotos hace que la teleoperación de tareas que interactúan con el entorno sea extremadamente difícil. Los retardos de más de un segundo pueden producir un problema clásico en la ingeniería de sistemas de control. A partir de los 0.25 segundos, la teleoperación tradicional basada en la realimentación de esfuerzos se hace más difícil, y cuando este retardo es del orden de los 0.40 segundos se considera un retardo significativo, que hace necesaria la aplicación de técnicas específicas. Los métodos propuestos para la ejecución de tareas telerrobóticas con retardos mayores pasan por dotar de una mayor autonomía al robot y para trabajar con instrucciones de mayor nivel.

1.4 Herramientas de Software

En la presente sección se hace referencia a las herramientas de software utilizadas.

Empezando por el lado del servicio web del MRaaS al definir la plataforma .NET utilizada así como también explicar el lenguaje de programación C# a usarse en WCF Y WPF, para pasar a especificar el tipo de *hosting* a utilizarse, así mismo se define el gestor y modelo conceptual para la bases de datos. Posteriormente en el lado del usuario operador se define el uso del Kinect SDK como componente en la aplicación de interfaz gráfica, por otra parte para el usuario de módulo de trabajo se necesitará un sistema operativo embebido, el cual es OpenWrt donde se ejecutará un programa en Python para consumir el servicio web del sistema MRaaS, finalmente se enviará comandos de movimientos a ser interpretados por un programa realizado en Arduino IDE.

1.4.1 La plataforma .NET

La plataforma .NET fue introducida por Microsoft en el año 2000 para mantenerse en la industria del software, y principalmente competir con Java de Sun, .NET es una plataforma de desarrollo, de múltiples tecnologías como: ASP (Active Server Pages), ADO (ActiveX Data Objects), LINQ (Language-Integrated Query), WPF (Windows Presentation Foundation), WCF (Windows Communication Foundation), Silverlight, etc. (Conesa, Gavidia, Ceballos, & Gañán, 2010)

.NET Incorpora un entorno integrado de desarrollo llamado Visual Studio, permite el desarrollo de aplicaciones de: línea de comandos, servicios, para escritorio con Windows Forms o WPF, web con ASP.NET o Silverlight, SOA para servicios web, Windows Mobile (Conesa, Gavidia, Ceballos, & Gañán, 2010, pág. 15)

.NET principalmente es para sistemas operativos Windows, aunque en las nuevas generaciones se están incluyendo MAC OS X, Android, iOS (Microsoft Visual Studio, 2015)

Para las demás plataformas el proyecto Mono brinda una implementación de código abierto de .NET, con soporte completo para Linux, Windows, y MAC OS X (Cross platform, open source .NET framework, 2015)

Desde la aparición de la primera versión estable de .NET en el año 2002, se consideran las versiones más destacadas (véase Figura 7).

Evolución de .NET

.NET Framework 1.0		
Primera versión de .NET Framework.	Plantea un nuevo entorno de desarrollo.	2002.

.NET Framework 1.1		
Visual Studio .NET 2003 y Windows Server 2003.	.NET Compact Framework, para dispositivos móviles.	2003.

.NET Framework 2.0		
Visual Studio 2005 y SQL Server 2005.	Segunda versión del .NET Compact Framework.	2005.

.NET Framework 3.0		
Windows Vista.	Tecnologías como WPF, WCF, WWF, CardSpace.	2006.

.NET Framework 3.5		
Visual Studio 2008, Windows Server 2008, y SQL Server 2008 .	Lenguaje LINQ y el Entity Framework.	2007.

.NET Framework 4		
Visual Studio 2010.	Opciones Parallel LINQ, y Task Parallel Library.	2010.

.NET Framework 4.5		
Visual Studio 2012.	Incluye Modern UI Runtime, y Task-Based Async Model.	2012.

Figura 7. Evolución de la plataforma .NET

Elaborado por: Edison Pilatasig

1.4.2 Entorno Visual Studio

Microsoft Visual Studio es considerado un IDE o Integrated Development Environment por sus siglas en inglés, se comparte y es necesario para todos los lenguajes de .NET (Conesa, Gavidia, Ceballos, & Gañán, 2010)

Para el desarrollo del presente trabajo se ha optado por utilizar el IDE Visual Studio Ultimate 2012 en su versión 11, junto con Microsoft .NET Framework 4.5 (véase Figura 8), debido a que a la fecha de realización del trabajo esta era la versión más actual y estable, destacando que en Mayo de 2015 se liberó la versión Visual Studio 2015, la cual se está empezando a adoptar por los desarrolladores de software.

Visual Studio 2012

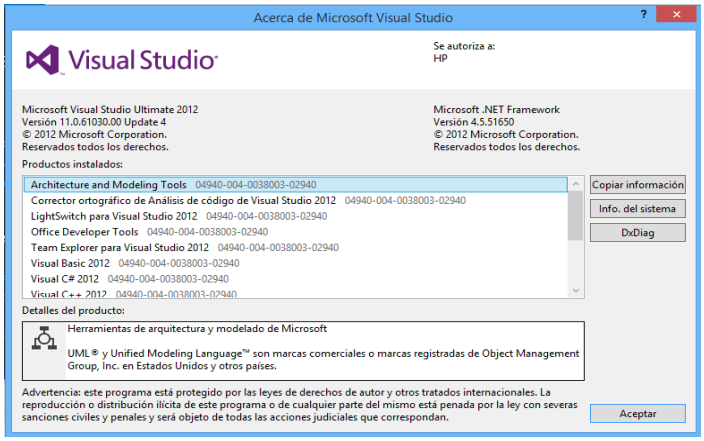


Figura 8. Información del software Visual Studio utilizado

Elaborado por: Edison Pilatasig

1.4.3 Arquitectura .NET Framework

El CLR (Common Language Runtime) posee una máquina virtual análoga a la máquina virtual de Java como entorno de ejecución de .NET.

Proporciona el entorno donde las aplicaciones .NET son ejecutadas de esta manera activa los objetos, los ejecuta, gestiona la memoria, desempeña comprobaciones de seguridad. (Common Language Runtime (CLR), 2014)

Luego del nivel de CLR se sitúa el Framework Class Library, tiene más de 4000 clases siendo de las mayores bibliotecas de clases. Posteriormente están las clases para el acceso a datos usando ADO.NET y finalmente están las tecnologías para creación de aplicaciones (véase Figura 9) (ADO.NET, 2014)

Arquitectura .NET

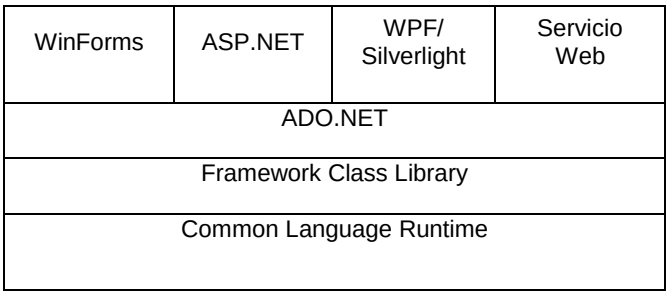


Figura 9. Arquitectura .NET Framework.

Fuente: (Conesa, Gavidia, Ceballos, & Gañán, 2010, pág. 23)

1.4.4 Lenguaje C#

Visual Studio provee un entorno de programación en múltiples lenguajes (versiones más actuales incluso proveen soporte multiplataforma), entre los más representativos se encuentran visual C# y visual Basic, se opta por trabajar con lenguaje C# dado que es el más utilizado en .NET y de tal manera el que más se ha venido desarrollando, se elige C# ya que dota al usuario programador el poder trabajar en un entorno orientado a objetos.

C# es un lenguaje orientado a objetos, se especifican un conjunto de operaciones para definir el comportamiento sean funciones o métodos, se especifican miembros de datos sean variables o propiedades que describen el estado del objeto, las clases al ser instanciadas o referenciadas crean un objeto de la misma, objeto que permitirá acceder a las propiedades de la clase (Ceballos, 2013)

Para una mejor comprensión sobre objetos, propiedades, métodos, constructores, herencia, interfaces, excepciones y tipos genéricos, temas que serán constantemente utilizados en el desarrollo del presente trabajo se recomienda leer la información que ofrecen (Conesa, Gavidia, Ceballos, & Gañán, 2010) y (Ceballos, 2013).

1.4.5 Windows Presentation Foundation – WPF

WPF es una tecnología basada en XML, para concebir interfaces gráficas de usuario mucho mejores que al usar Windows Forms (Windows Presentation Foundation, 2015)

WPF apareció en el año 2006 formando parte de .NET Framework 3.0 y Windows Vista. WPF es la evolución de Windows Forms, existen variantes de WPF para entornos WEB como: XBAP1 o WPF Browser (Hasta Visual Studio 2010) y Silverlight (Silverlight, 2015)

WPF usa una variación de XML, llamada XAML (eXtensible Application Markup Language) (¿Qué es XAML?, 2015)

El cual permite separar la interfaz gráfica, de la lógica de aplicación, lo cual permite modificar la visualización de la interfaz sin afectar a la lógica de la aplicación

(código de programa) y viceversa, se dispone de varias herramientas visuales ofrecidas en WPF (Conesa, Gavidia, Ceballos, & Gañán, 2010).

Se usa la tecnología WPF en el desarrollo del presente trabajo para desarrollar una aplicación con interfaz gráfica para el usuario operador del sistema, aplicación que consumirá el servicio web del sistema MRaaS.

1.4.6 Windows Communication Foundation (WCF)

WCF es considerada una interfaz de desarrollo que busca unificar tecnologías de desarrollo distribuido como: COM+, o servicios Web, y que permita inter operar con otras tecnologías de desarrollo de aplicaciones distribuidas de terceros (COM+ (Component Services), 2015)

WCF utiliza comúnmente el protocolo SOAP para comunicarse con un servicio Web, aunque puede configurarse para usar otros formatos de mensaje diferentes. Además usando especificaciones WS* se implementan tecnologías de servicios Web como WSAddressing, WS-Policy, WS-Security o WS-Coordination, las cuales definen una serie de protocolos que dotan a la arquitectura de servicios web de funcionalidades adicionales como direccionamiento, coordinación, seguridad o transacciones distribuidas (Web Services Specifications Index Page, 2015)

Debido a que el servicio web del sistema MRaaS se ha diseñado para trabajar con comunicaciones petición/respuesta usando SOAP, se hace factible usar WCF como plataforma de desarrollo. La funcionalidad principal de WCF se encuentra dentro del

ensamblado System.ServiceModel.dll cuya referencia es necesario importar en el proyecto antes de empezar ya que contiene modelos de servicios web, sin olvidar añadir la librería con la correspondiente instrucción *using* al inicio del programa.

Definido como un conjunto de información enviado desde un origen a un destino determinado, WCF toma una comunicación basada en mensajes, un mensaje SOAP está formado básicamente por una cabecera (Header) y un cuerpo de mensaje (Body), en dichas partes del mensaje SOAP se incorpora el contenido del mensaje del usuario. Los clientes y servicios de WCF intercambian mensajes SOAP siguiendo una serie de patrones (Box, y otros, 2000)

En el patrón simplex también llamado *One-way*, la comunicación se produce del cliente al servicio y no existe una respuesta por parte del servicio (véase Figura 10).

Patrón de Comunicación Simplex

Figura 10. Ejemplo de comunicación simplex

Fuente: (Conesa, Gavidia, Ceballos, & Gañán, 2010, pág. 159)

En el patrón envío - respuesta llamado también *Request-Reply*, la comunicación se produce del cliente al servicio y el servicio devuelve una respuesta al cliente, por tal razón es una comunicación asociada a servicios bajo demanda (véase Figura 11).

Patrón de Comunicación Envío-respuesta

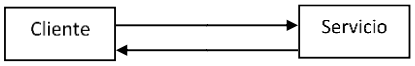


Figura 11. Ejemplo de comunicación envío - respuesta.

Fuente: (Conesa, Gavidia, Ceballos, & Gañán, 2010, pág. 159)

En el patrón dúplex se dan comunicaciones en ambos sentidos, tanto cliente-servidor, como servidor-cliente, además pueden ser tanto one-way como request-reply (véase Figura 12).

Patrón de Comunicación Dúplex

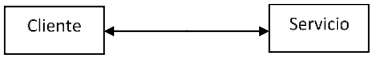


Figura 12. Ejemplo de comunicación dúplex.

Fuente: (Conesa, Gavidia, Ceballos, & Gañán, 2010, pág. 160)

El patrón de comunicaciones definido entre el cliente y servicio MRaaS es envío-respuesta o solicitud-respuesta, debido a la necesidad de enviar primero peticiones al servicio web, el cual realizará las operaciones necesarias, y posteriormente devolverá un mensaje de respuesta conteniendo el resultado de la petición.

1.4.7 Simple Object Access Protocol - SOAP

Se plantea el uso de un modelo de comunicaciones solicitud-respuesta con SOAP y WCF como plataforma de programación del servicio web para el sistema MRaaS, SOAP es un mecanismo sencillo y ligero para el intercambio de información estructurada entre puntos en un entorno descentralizado y distribuido mediante XML.

SOAP no define ninguna semántica de aplicación, como lo hace un modelo de programación (Box, y otros, 2000).

Más bien SOAP define un mecanismo simple para expresar la semántica de la aplicación, creando un modelo de empaquetamiento modular y codificación de los mecanismos de codificación de datos dentro de los módulos, lo que permite a SOAP ser utilizado en una gran variedad de sistemas (Marshall, 1999).

Las tres partes que conforman SOAP: la envoltura SOAP (SOAP envelope) define un marco general para expresar lo que está en un mensaje, quien debe tratar con él, y si es opcional u obligatoria (Box, y otros, 2000)

Las reglas de codificación de SOAP (SOAP encoding rules) definen un mecanismo de serialización, que se puede utilizar para intercambiar instancias de tipos de datos definidos por la aplicación (Serialización (C# y Visual Basic), 2015)

La representación SOAP RPC (SOAP RPC representation) define una convención que se puede utilizar para representar llamadas a procedimientos remotos y respuestas.

Los mensajes SOAP son fundamentalmente transmisiones de un solo sentido desde un emisor a un receptor, mensajes SOAP se combinan para aplicar patrones tales como solicitud/respuesta. Las implementaciones de SOAP se pueden optimizar para explotar las características únicas de los sistemas de red. Por ejemplo, el enlace o *binding* HTTP ofrece a los mensajes de respuesta SOAP ser entregados como respuestas usando HTTP, utilizando la misma conexión que la petición de entrada.

La envoltura SOAP (SOAP Envelope), consta del encabezado SOAP y el cuerpo del mensaje SOAP, un elemento en la cabecera (SOAP Header) es opcional, el *header* es

un mecanismo de extensión para proporcionar una manera de pasar información en mensajes SOAP que no son datos de la aplicación. El cuerpo (SOAP Body) es un elemento obligatorio, es aquí donde la principal información de extremo a extremo transmitida en un mensaje SOAP debe incluirse (véase Figura 13).

Mensaje SOAP

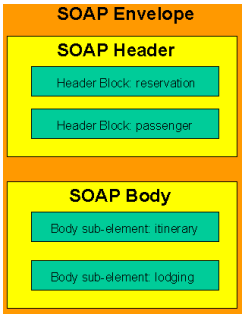


Figura 13. Estructura de un mensaje SOAP

Fuente: (Box, y otros, 2000)

SOAP puede utilizarse en varios protocolos, uno de ellos y el que se ha tomado en cuenta para la realización de este trabajo, es SOAP en HTTP con el patrón de intercambio de mensajes Petición-Respuesta. Teniendo en cuenta que el uso de este patrón de intercambio de mensajes implica el intercambio de datos XML generales o RPC (Remote Procedure Call) encapsulados en mensajes SOAP. Información específica acerca del uso de SOAP, semánticas, escenarios, modelos, y ejemplos pueden encontrase en (Box, y otros, 2000).

1.4.8 Hosting Internet Information Services –IIS

Igual que los servicios Web, los servicios WCF pueden implementarse utilizando IIS (véase Figura 14), una vez implementado el servicio se puede compilar y ejecutar de

forma análoga a los servicios Web, Visual Studio incorpora un tipo de proyecto específico para crear servicios WCF alojados en IIS llamado: WCF *Service Application*, es un tipo de proyecto dentro de la categoría *Web*.

IIS 8.5.

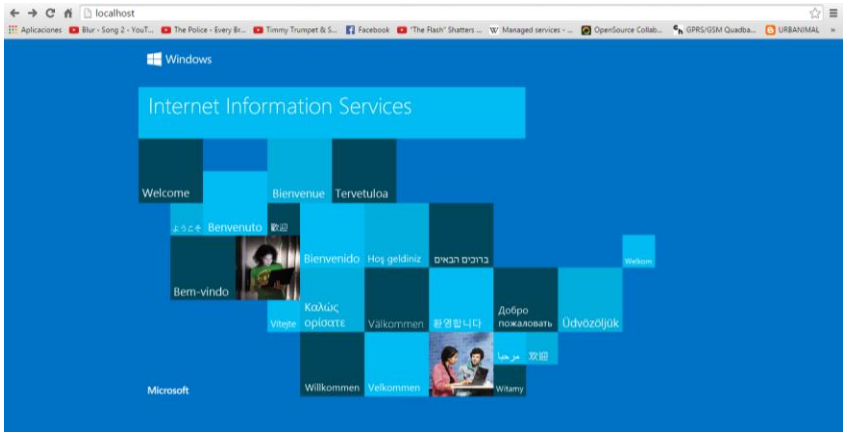


Figura 14. Página principal de IIS 8.5– Windows 8.1

Elaborado por: Edison Pilatasig.

En el proyecto WCF se crean varios ficheros, que son de vital importancia (Conesa, Gavidia, Ceballos, & Gañán, 2010, pág. 180)

1.4.9 Microsoft SQL Server

"Sistema de administración y análisis de bases de datos relacionales de Microsoft, desarrollado para soluciones de comercio electrónico, línea de negocio y almacenamiento de datos" (Microsoft SQL Server, 2015)

La integración que ofrece Microsoft desde Visual Studio a través del paquete Entity Framework hacia MS SQL (Microsoft Structured Query Language) (Liu, 2012), hace posible el desarrollo del presente trabajo. Se utiliza la versión MS SQL 2012 R2,

aunque actualmente se ha liberado la versión 2014, la cual aún está empezando a adoptarse por los servicios de hosting.

MS SQL Server incluye varias tecnologías de gestión y análisis de datos, de las cuales solo se utiliza el motor de base de datos, el cual es el servicio principal que permite almacenar, procesar y proteger los datos, proporciona acceso controlado y procesamiento rápido de transacciones para satisfacer los requerimientos de escritura de datos de nuevos registros del sistema MRaaS.

1.4.10 Entity Framework

ADO.NET EF (Entity Framework) permite a los desarrolladores crear aplicaciones de acceso de datos por programación en un modelo de aplicación conceptual, en lugar de la programación de un esquema de almacenamiento relacional, con el fin de disminuir la cantidad de código y mantenimiento requerido para las aplicaciones orientadas a datos, esta es la característica por la cual se eligió Entity Framework como aplicación para el acceso a la base de datos del sistema MRaaS.

Las aplicaciones pueden trabajar con tipos con herencia, miembros complejos y relaciones. Los mapeos entre el modelo conceptual y el esquema de almacenamiento específico pueden cambiar sin cambiar el código de la aplicación. Entity Framework permite trabajar con un modelo de objetos de aplicación coherente que se puede asignar a varios esquemas de almacenamiento, posiblemente ejecutados en diferentes sistemas de gestión de bases de datos, ya que modelos conceptuales múltiples se pueden asignar a un solo esquema de almacenamiento. Por último LINQ proporciona

la validación de sintaxis en tiempo de compilación de consultas en un modelo conceptual.

Entity Framework

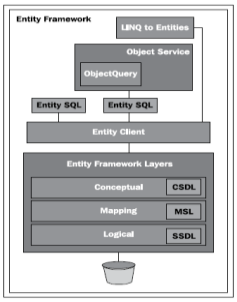


Figura 15. Arquitecturas de alto nivel de Entity Framework.

Fuente: (Liu, 2012, pág. 170)

El modelo de datos es conceptual, llamado EDM (Entity DataModel), en lugar de bases de datos subyacentes. El esquema conceptual del modelo de datos se expresa en la CSDL (Conceptual Schema Definition Language), el modelo de almacenamiento actual se expresa en el SSDL (Storage Schema Definition Language), y el mapeo en el medio se expresa en el MSL (Mapping Schema Language). Un nuevo proveedor de acceso de datos se crea, el llamado *EntityClient*, para este nuevo *framework*, pero bajo la cubierta los proveedores de datos de ADO.NET aún están siendo usados para la comunicación con la base de datos (véase Figura 15).

1.4.11 Kinect SDK

Originalmente llamado proyecto Natal, el Kinect fue lanzado en Noviembre del año 2010 como un dispositivo para la XBOX 360 (XBOX, 2015), Kinect es un

dispositivo basado en la ubicación de cámaras, que permite al usuario manejar la Xbox sin la necesidad de usar un control remoto, a través de una interfaz de usuario natural usando gestos y comandos hablados (Cordella, 2011, pág. 23)

El Kinect interpreta la información 3D de una escena desde una luz infrarroja que se proyecta de forma continua. Este escáner 3D se llama “Light Coding”, que implementa una variante de la reconstrucción 3D basada en imágenes (Cordella, 2011, pág. 23)

Kinect ha venido evolucionando para adaptarse a las nuevas tecnologías, en el presente trabajo se utiliza la versión 1.8 del hardware Kinect y así mismo la versión 1.8 del Kinect SDK, aunque en el mercado ya se dispone de la versión 2.0 tanto en Kinect como el SDK, las herramientas aún continúan en desarrollo.

Un sensor Kinect (véase Figura 16) posee: cámara de color, emisor infrarrojo IR (Infrared), sensores infrarrojos de profundidad, motor de inclinación, arreglo de micrófonos, Led, y para la versión de desarrollo para Windows se dispone de un adaptador USB (Abhijit, 2012, pág. 9).

Kinect

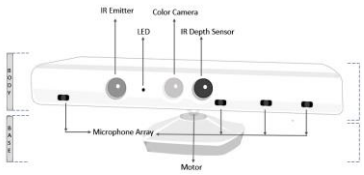


Figura 16. Componentes de un sensor Kinect

Fuente: (Abhijit, 2012, pág. 9)

Para el desarrollo de la aplicación gráfica para el usuario operador con el sensor Kinect, se hace necesario instalar en la computadora tanto la plataforma de desarrollo SDK (Software Development Kit v.1.8), y además el conjunto de herramientas o *Toolkit*, con el fin de simplificar la programación referente a la detección de movimientos.

1.4.12 OpenWRT

Debido a la posibilidad de implementación del sistema MRaaS como parte de un LaaS se hace necesario utilizar sistemas embebidos con el fin de reducir costes en equipos de computación y con el fin de tener un sistema modular.

OpenWrt es una distribución de GNU / Linux altamente extensible para dispositivos embebidos (enrutadores inalámbricos), OpenWrt es un sistema operativo con funciones fácilmente modificables, al poseer un núcleo de Linux (OpenWrt Wireless Freedom, 2015)

El desarrollo de OpenWrt empezó en enero del 2004, las primeras versiones se basaban en fuentes libres GPL (General Public License) de Linksys. En el 2005 se lanzaron versiones experimentales de OpenWrt, con fuentes oficiales del *kernel* GNU/Linux donde se agrega parches al sistema, drivers para interfaces de red, gestión de paquetes opcionales para adaptarse a cualquier aplicación. Para el sistema MRaaS se elige este sistema operativo debido a la personalización completa que podemos llegar a tener.

OpenWrt no es una distribución lista para su uso, en el presente trabajo se utiliza una variación realizada por la empresa Dragino, cabe destacar que los paquetes de software que existen desarrollados para OpenWrt (bordean los 2000) o se pueden crear. Al ser un sistema basado en Linux, OpenWrt ofrece varias capacidades y características, aunque algunas variaran de acuerdo al dispositivo en el que se haya instalado, sean estas, limitaciones en memoria, interfaces, periféricos, etc.

1.4.13 Python

Python es considerado como un lenguaje de programación con estructuras de datos de alto nivel y un enfoque simple pero eficaz hacia la programación orientada a objetos (The Python Tutorial, 2015)

El intérprete de Python y la extensa biblioteca estándar están disponibles gratuitamente en formato fuente o binario para todas las plataformas desde el sitio Web de Python.

Habiendo planteado la necesidad de utilizar sistemas operativos embebidos para el sistema MRaaS, se selecciona el lenguaje de programación Python para consumir el servicio Web por medio del cual se podrá realizar consultas al servicio web y manejar su respuesta como objetos en la programación. El intérprete de Python se extiende fácilmente con nuevas funciones y tipos de datos implementados en C o C ++ (u otros idiomas derivados de C). Python también es adecuado como un lenguaje de extensión para aplicaciones personalizables. Existen varias Funciones, estructuras y tipos de datos (The Python Tutorial, 2015).

1.4.14 Arduino IDE

El software Arduino IDE es de código abierto y se ejecuta en Windows, Mac OS X y Linux, el entorno está escrito en Java, este software brinda la posibilidad de subir el código a cualquier placa Arduino, posee un conjunto de drivers para las placas disponibles, así como también ejemplos para la mayoría de dispositivos desarrollados para Arduino. El IDE de Arduino está disponible para su edición al ser de código abierto (Arduino, 2015). Las versiones actualmente usadas son 1.0.6, para las versiones más comerciales de las placas, y la versión beta 1.5.8, la que se utilizó en el presente trabajo, donde se incluyen más modelos de placas como Arduino Yún y Arduino Due.

1.5 Herramientas de Hardware

En la presente sección se hace referencia a las herramientas de hardware necesarias para la implementación del sistema MRaaS, en el lado del usuario de módulo de trabajo al plantear un diseño modular y económico se hace uso de la plataforma Arduino Mega 2560 junto con la placa Dragino Yún Shield, la cual posee un sistema operativo embebido OpenWrt, donde se consumirá el servicio web y se obtendrá datos de movimientos para el brazo robótico SCARA que constará de un control electrónico basado en sistemas PWM (Pulse Width Modulation), y sensores analógicos donde constan potenciómetros e infrarrojos SHARP. En el lado del usuario operador se utiliza un sensor Kinect y al especificar un modelo por realimentación visual se hace necesario incorporar una cámara web IP para la verificación del movimiento del brazo robótico SCARA.

1.5.1 Arduino Board Mega 2560

De forma general Arduino es una plataforma de computación de código abierto basada en una tarjeta con un microcontrolador simple y un ambiente de desarrollo para escribir el software a la tarjeta. Es usado para desarrollar objetos interactivos al conectarse de forma fácil al computador para aplicaciones multimedia. El Arduino Mega 2560 (véase Figura 17) es una placa electrónica basada en el chip ATmega2560, utiliza un chip ATmega16U2 para la comunicación con el computador (Arduino, 2015)

Arduino Mega 2560

Figura 17. Fotografía de la Placa Arduino Mega 2560.

Fuente: (Arduino, 2015)

Se selecciona la placa Arduino Mega 2560 para el desarrollo del presente proyecto, debido a la cantidad de salidas PWM que son suficientes para controlar los motores del brazo robótico SCARA, además la cantidad de pines de entrada/salida se acopla perfectamente a las necesidades de este trabajo, las características eléctricas y detalles de interfaces se muestran en (Arduino, 2015).

1.5.2 Dragino - Shield Arduino Yún

El Yún shield de la empresa Dragino está diseñado para aplicaciones de conectividad a internet con un sistema operativo OpenWrt, es compatible con el Arduino IDE

v1.5.4 o superior, Yún shield Dragino es considerado la unión de un Yún Shield Original con un Arduino Leonardo, pero con la capacidad de poder adaptarse a cualquier placa electrónica de Arduino (Linux, WiFi Shield for Arduino, 2015)

Dada la capacidad del procesador del Dragino Yún Shield a 400MHz, memoria RAM a 64 MB, interfaces de red WiFi 802.11 b/g/n y RJ 45 10M/100M, y bajo consumo de potencia, se acopla perfectamente para realizar el consumo del servicio Web del sistema MRaaS utilizando Python.

El módulo Dragino HE (véase Figura 18), es el núcleo del módulo Yún shield, requiere alrededor de 200 mA de corriente cuando está a plena carga, los módulos (véase Figura 2.19) RJ45, WiFi, USB Host, están conectados al módulo Dragino HE directamente, además el módulo Dragino HE usa comunicación SPI y UART con la placa Arduino, donde la interfaz SPI es usado para subir los sketches (código de programas) del Arduino IDE sea vía LAN o WiFi, la interfaz SPI puede ser también usada para conectar otros dispositivos esclavos, la interfaz UART por su lugar es usada por la clase *Bridge* en Arduino, para realizar operaciones sobre el módulo Dragino Yún shield.

Módulos Dragino Yún Shield

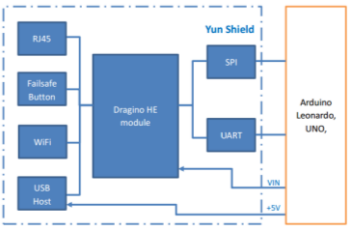


Figura 18. Arquitectura de módulos Dragino Yún Shield.

Fuente: (Yun Shield Quick Start Guide, 2014, pág. 4)

Placa Dragino Yún Shield

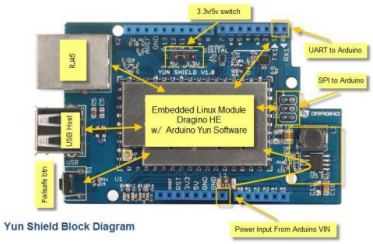


Figura 19. Arquitectura física Dragino Yún Shield.

Fuente: (Yun Shield Quick Start Guide, 2014, pág. 4)

1.5.3 Robot SCARA Rhino XR series

El Rhino SCARA (Selective Compliance Assembly Robot Arm) es un brazo robot de cuatro ejes con una pinza de accionamiento eléctrico (SCARA, 2015)

Puede ser operado ya sea con un controlador Mark III o el Mark IV. El brazo tiene un eje "Z" ajustable en incrementos de una pulgada de altura, los Robots SCARA se utilizan principalmente en operaciones de montaje automatizado de alta precisión. El brazo robótico SCARA utilizado se encuentra descontinuado y ante la escasez de repuestos se realizó modificaciones mecánicas y eléctricas (véase Figura 20).

Selective Compliance Assembly Robot Arm

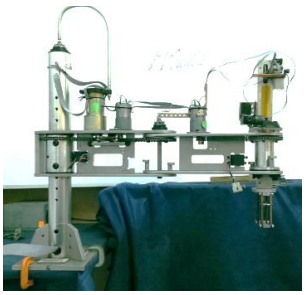


Figura 20. SCARA Modificado.

Elaborado por: Edison Pilatasig

El SCARA utilizado posee 5 motores PMDC (Permanent Magnet Direct Current) con cajas reductoras, para el control de posición de las extremidades de hombro, codo, altura se adaptaron potenciómetros, mientras que el control de posición de giro y el gripper se realiza mediante temporizadores. Se permite el manejo individual y simultáneo de los componentes, la velocidad del gripper es de un segundo (abrir o cerrar). La envoltura de movimiento es 22.86 centímetros dentro del radio, con rotación motor de Hombro hasta 180 grados, rotación motor de Codo hasta 180 grados, rotación motor de Muñeca hasta 180 grados, motor vertical hasta 65 milímetros.

1.5.4 Sensor Sharp GP2Y0A41SK0F

Dado que se ha rehabilitado el brazo robótico SCARA y que componentes controladores para la altura del gripper o pinza no se venden se hace necesario utilizar un sensor de distancia análogo (véase Figura 21) en el rango de cuatro centímetros a treinta centímetros con una alta resolución en la medición, características físicas y eléctricas pueden ser encontradas en (Pololu Robotics, 2015).

Sensor SHARP

Figura 21. Sensor de distancia análogo Sharp.

Fuente: (Pololu Robotics, 2015)

1.5.5 Potenciómetro lineal

Se ve en la necesidad de utilizar potenciómetros lineales (véase Figura 22) para controlar la posición de dos de las extremidades del brazo robótico SCARA, existen diferentes modelos (Micropik, 2015).

Potenciómetro lineal

Figura 22. Potenciómetro lineal utilizado.

Fuente: (Micropik, 2015)

1.5.6 Driver controlador L298n

El módulo L298N (véase Figura 23) se utiliza para el control de giro y velocidad de motores de corriente directa DC del brazo robótico SCARA. Dispone de cuatro terminales para la conexión de dos motores de DC y cuatro pines para las señales de control en este caso PWM. Posee bornes para la alimentación de los motores (Vcc) y para la parte lógica (5V), características físicas y eléctricas específicas también se describen (Modulo L298n Driver, 2015).

Para el presente trabajo se usaron tres módulos L298n, dado que el número de máximo de motores DC a controlar es de cinco, tomando en cuenta que por cada módulo se puede controlar dos motores DC.

Módulo L298n

Figura 23. Módulo L298n utilizado.

Fuente: (Modulo L298n Driver, 2015)

1.5.7 Cámara IP C30 SD PTZ.

Para la implementación de un sistema telerrobótico con realimentación supervisada en el lazo de control del cliente, se opta por utilizar una cámara IP C30 SD PTZ (véase Figura 24) que permitirá que al usuario operador observar la respuesta de los movimientos solicitados hacia el brazo robótico SCARA al utilizar la aplicación gráfica. Las características generales y manuales de operación pueden ser encontradas en la página oficial de Tenda en (C30 SD PTZ IP Camera , 2015).

Cámara IP C30 SD PTZ

Figura 24. Cámara IP C30 SD PTZ utilizada.

Fuente: (C30 SD PTZ IP Camera , 2015)

CAPÍTULO 2
DISEÑO

En la presente sección se detalla el proceso mediante el cual se diseña tanto en hardware como en software el sistema en su conjunto, se modela el funcionamiento general del sistema MRaaS usando el lenguaje UML, finalmente se diseña la base de datos según los requerimientos para el presente trabajo.

2.1 Diseño Conceptual

Se describe a continuación el diseño conceptual realizado para el sistema MRaaS, para ello se analiza los requerimientos del usuario, para pasar a definir las especificaciones técnicas que servirán para definir alternativas para el sistema, las ventajas y desventajas de las alternativas propuestas serán finalmente analizadas para determinar que alternativa es viable para el desarrollo del proyecto..

2.1.1 Requerimientos de los usuarios

El usuario operador es la persona que realizará la manipulación remota (desde su ubicación actual) del brazo robótico SCARA (ubicada en un laboratorio de control) en el sistema MRaaS a través de una aplicación de computador, estos requerimientos se encuentran descritos en la tabla 1.

Tabla 1.
Requerimientos del usuario operador

No.	Voz del usuario
1	Disponibilidad 24 horas del día.
2	Usuarios independientes y con jerarquía.

3	Fácil manejo de la aplicación.
4	Bajo costo.
5	Uso para personas con capacidades especiales.
6	Control de tiempo de conexión.
7	Uso de múltiples robots SCARA.
8	Visualización del robot en movimiento.
9	Registro de movimientos.
10	Aplicación interactiva.

Nota: Descripción de los requerimientos del usuario operador.

Elaborado por: Edison Pilatasig

2.1.2 Especificaciones técnicas

En base a lo expuesto en la tabla 1, se consideran las especificaciones técnicas con las cuales se puede dar cumplimiento a lo solicitado como se muestra en la tabla 2.

Tabla 2.

Especificaciones técnicas

No.	Voz del usuario operador	Voz del ingeniero	Unidades
1	Disponibilidad 24 horas del día.	Alojamiento del servicio web MRaaS	Disponibilidad > 99%
2	Usuarios independientes y con jerarquía.	Control de acceso por autenticación.	5 ≤ Accesos ≤ 50
3	Fácil manejo de la aplicación.	Interfaz gráfica intuitiva y simple.	16’’ ≤ pantalla ≤ 20’’
4	Bajo costo.	Aplicación considera recursos óptimos y económicos.	Costo proyecto ≤ 1500$
5	Uso para personas con capacidades especiales.	Uso del sensor de movimiento	Sensor movimiento = 1
6	Control de tiempo de conexión.	Asignación de tiempo de uso del sistema en una base de datos.	1 h ≤ tiempo ≤ 2h
7	Uso de múltiples robots SCARA.	Asignación de registros para múltiples robots en una base de datos.	1 ≤ Robot ≤ 15
8	Visualización del robot en movimiento.	Realimentación visual desde el laboratorio.	1 ≤ Cámara IP ≤ 2
9	Registro de movimientos.	Almacenamiento en registros por usuario en una base de datos.	1 ≤ Registros ≤ 1000
10	Aplicación interactiva.	Despliegue de mensajes de estado ante las acciones realizadas por el operador en la aplicación o fallos del sistema.	1 ≤ Mensajes en pantalla ≤ 2

Nota: Descripción de las especificaciones técnicas.

Elaborado por: Edison Pilatasig

2.1.3 Selección de la alternativa

Se elaboran dos alternativas para dar cumplimiento con los requerimientos del operador expuestos en la tabla 1, la alternativa A se detalla tanto en software como en hardware en la tabla 3.

Tabla 3.

Alternativa A

a)

No.	Elemento	Descripción
1	Computador de escritorio	Procesador Intel core i3, RAM 4 GB, Disco 500GB, puerto USB, conector 1x10M/100M RJ45, pantalla y periféricos.
2	SCARA	Modelo XR Series
3	Driver controlador del brazo SCARA	Puentes H L298, sensor infrarrojo GP2Y0A41SK0F, potenciómetro lineal.
4	Arduino	Mega 2560
5	Sensor Kinect	Kinect v 1.8
6	Cámara IP	Tenda C30 SD PTZ
7	Access Point	Huawei Hg530

b)

No.	Elemento	Descripción
1	Microsoft Visual Studio	Ultimate V. 2012
2	Kinect SDK	SDKV.1.8 y Toolkit
3	Arduino IDE	V. 1.5.8
4	Filezilla	V. 3.11.0.2

Nota: Descripción de la alternativa A; a) Hardware, b) Software.

Elaborado por: Edison Pilatasig

La alternativa B se detalla tanto en software como en hardware en la tabla 4.

Tabla 4.

Alternativa B

a)

No.	Elemento	Descripción
1	Micro dispositivo Dragino	Procesador 400 MHz, RAM 64 MB, 150Mbps Wifi, conector 1x10M/100M RJ45.
2	SCARA	XR Series
3	Controlador del brazo SCARA	Puentes H L298, sensor infrarrojo GP2Y0A41SK0F, potenciómetro lineal.
4	LCD	2*16
5	Arduino	Mega 2560

6	Periféricos	Pulsadores, Leds, terminales alimentación
7	Sensor Kinect	Kinect v 1.8
8	Cámara IP	Tenda C30 SD PTZ
9	Access Point	Huawei Hg530

b)

No.	Elemento	Descripción
1	Python	V2.7.10
2	Kinect SDK	SDKV.1.8 y Toolkit
3	Arduino IDE	V. 1.5.8
4	Filezilla	V 3.11.0.2
5	WinSCP	V. 5.7.2
6	Putty	V. 0.63

Nota: Descripción de la alternativa A; a) Hardware, b) Software.

Elaborado por: Edison Pilatasig

En base a lo presentado en la tabla 3 se detallan las ventajas y desventajas que esta alternativa presenta para la solución del sistema y se muestra en la tabla 5.

Tabla 5.

Ventajas y desventajas de la alternativa A

Ventajas	Desventajas
Menor cantidad de software utilizado.	El sistema posee un computador que actúa como puente con interfaz gráfica entre la comunicación entre Módulo de trabajo y el servicio web en la nube.
Menor cantidad de hardware en el controlador del brazo SCARA.	El controlador del brazo SCARA usa una conexión USB hacia el computador de escritorio y este a su vez conectado directamente por cable de red RJ45 a la red local.
La visualización de estado del sistema en el módulo de trabajo es amplia al poseer una pantalla gráfica de escritorio.	El sistema no es modular y ocupa gran espacio.
Brinda la opción para manejo simple por parte de personas con capacidades especiales.	No se puede administrar el sistema remotamente.
	Alto consumo de energía al considerar el computador siempre prendido en la estación de trabajo.
	Necesidad de gran espacio en el laboratorio al implementar más de cinco módulos robóticos.
	El sistema necesita ser inicializado manualmente en el módulo de trabajo.

Nota: Descripción de las ventajas y desventajas de la alternativa A.

Elaborado por: Edison Pilatasig

En base a lo presentado en la tabla 4 se detallan las ventajas y desventajas que esta alternativa presenta para la solución del sistema y se muestra en la tabla 6.

Tabla 6.

Ventajas y desventajas de la alternativa B

Ventajas	Desventajas
El sistema posee un micro dispositivo con sistema operativo OpenWrt (distribución Linux) que actúa como puente entre la comunicación entre Módulo de trabajo y el servicio web en la nube.	Mayor cantidad de software utilizado.
El controlador del brazo SCARA usa una conexión serial hacia el módulo Dragino y este a su vez conectado vía WiFi a la red local.	Mayor cantidad de hardware en el controlador del brazo SCARA.
El sistema es modular, transportable y de bajo costo.	El sistema necesita ser armado físicamente en una nueva estructura (caja de plástico).
El sistema es administrable vía SSH.	La administración del sistema se da por consola de comandos.
El consumo de energía es reducido en el módulo de trabajo.	Visualización del estado del sistema en el módulo de trabajo es limitada al usar un LCD
Tamaño del proyecto en el módulo de trabajo es pequeño.	
Características Plug & Play en el módulo de trabajo.	
Brinda la opción para manejo simple por parte de personas con capacidades especiales.	
Necesidad de poco espacio en el laboratorio al implementar más de cinco módulos robóticos.	

Nota: Descripción de las ventajas y desventajas de la alternativa B.

Elaborado por: Edison Pilatasig

2.1.4 Análisis de la mejor alternativa

En base a lo expuesto en las tablas 5 y 6, se opta por elegir la alternativa B, porque permite ampliar las capacidades del sistema en cuanto a reducción de costos, administración de espacio, bajo consumo energético, y en su futuro la capacidad de implementar un gran número de unidades robóticas que se puedan manipular remotamente.

2.2 Arquitectura del sistema

Para el modelado del sistema MRaaS, se opta por usar un método telerrobótico basado en autonomía supervisada, se considera al usuario operador fuera del bucle de control local del brazo robótico SCARA, sin embargo las órdenes de ejecución autónoma realizadas a distancia por el operador son enviadas a través de la nube de Internet para su ejecución local en el brazo robótico SCARA (véase Figura 25).

Arquitectura general del sistema MRaaS

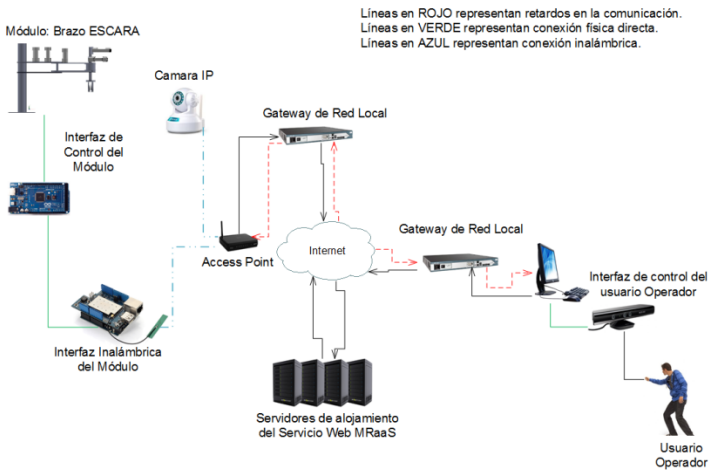


Figura 25. Modelo de arquitectura general del sistema MRaaS.

Elaborado por: Edison Pilatasig

La Figura 25 nos muestra el modelo de la arquitectura general del sistema MRaaS a utilizarse, el usuario operador a través de una interfaz gráfica podrá manipular remotamente al brazo robótico ESCARA, la interfaz gráfica podrá ser utilizada manualmente utilizando los dispositivos periféricos (mouse, teclado) o a través de dispositivos de detección del movimiento natural humano como es el Kinect, el uso de Kinect extiende las aplicaciones a usuarios quizá limitados físicamente de alguna manera pero que disponen de movimiento en los brazos, la interfaz del operador es la

responsable de consumir (realizar peticiones) el servicio web donde se realizaran peticiones de descubrimiento disponibilidad del módulo de trabajo, en este caso el brazo robótico SCARA, o a su vez otros módulos robóticos. Una vez realizada la conexión con el módulo de trabajo seleccionado, se realizará una petición de descubrimiento de restricciones de movimientos o movimientos aceptados, solo entonces se habilitará la interfaz de control del manipulador robótico.

Una vez completado el proceso para operaciones de inicio de sesión de usuario y conexión con el módulo de trabajo, entonces el usuario es capaz de realizar operaciones sobre registros, esto es el proceso mediante el cual el usuario operador podrá manipular el módulo de trabajo, o brazo robótico SCARA. El usuario operador confirmará que el comando de movimiento se ejecutó correctamente al realizar realimentación del sistema a través de una cámara IP instalada en el laboratorio físico donde se encuentra el sistema MRaaS.

El siguiente paso es definir el modelo necesario de funcionamiento del sistema MRaaS, los modelados en UML (Unified Modeling Language), inventados en 1997 (Kimmel, 2008)

Permiten la definición formal del funcionamiento de un sistema antes de su implementación, es considerado un lenguaje más con la diferencia de que se utilizan símbolos (cuadrados, círculos, líneas) para transmitir significados, se logra así el entendimiento del funcionamiento del sistema por cualquier persona, no solo desarrolladores de software, UML ofrece diagramas como: casos de uso, actividades, clases, interacción, estado, componentes, entre otros, que son usados para diferentes necesidades.

2.2.1 Caso de uso de la arquitectura general

Obsérvese la Figura 26, se describe el caso de uso para el modelado general del sistema MRaaS, en UML los diagramas de caso de uso sirven para documentar los requisitos generales del sistema, para ello utiliza símbolos fáciles de entender por cualquier persona común (Kimmel, 2008, pág. 1)

Se tienen dos actores (véase Figura 26) y cuatro casos de uso en el sistema MRaaS. El actor "Módulo del operador" representa a la persona física que opera la interfaz gráfica de control del brazo robótico SCARA. El actor "Módulo de trabajo" representa al módulo físico a ser teleoperado, en este caso el brazo robótico SCARA aunque se puede albergar un ecosistema variado de manipuladores robóticos.

Caso de uso del Sistema MRaaS

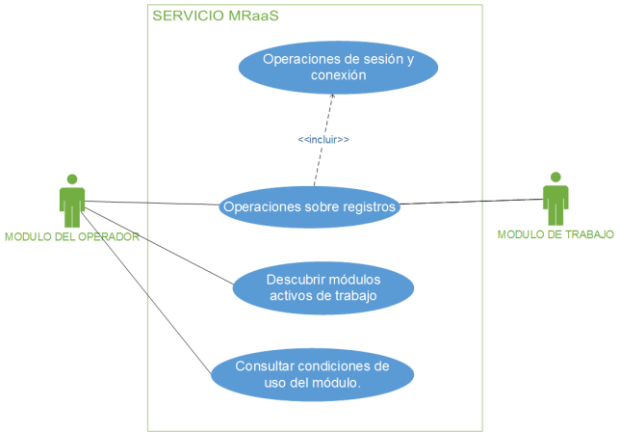


Figura 26. Caso de uso del funcionamiento general del sistema MRaaS.

Elaborado por: Edison Pilatasig

Los actores: módulo del operador y módulo de trabajo, se asocian con el caso de uso "Operaciones sobre registros" que depende del caso de uso "Operaciones de sesión y conexión", es decir primero se tendrán que cumplir las actividades referentes a inicio

de sesión de usuarios, inicio de conexión entre estaciones, para entonces si tener acceso a la escritura o consulta de registros en la base de datos. El actor modulo del operador, se asocia con el caso de uso “Descubrir módulos activos de trabajo” que hace referencia al proceso que realiza el sistema MRaaS para dar a conocer al “usuario operador” los “módulos de trabajo” que están activos y disponibles para ser usados en este caso el brazo robótico SCARA. El actor módulo del operador se asocia con el caso de uso “Consultar condiciones de uso del módulo” que hace referencia al proceso que realiza el sistema MRaaS luego del inicio de una nueva conexión para dar a conocer en el módulo del operador las características de movimientos permitidos del brazo robótico SCARA y así mismo detalles de conexión como por ejemplo el tiempo de uso asignado.

2.2.2 Caso de Uso del Módulo del Operador

El actor “Módulo del Operador” es a su vez un sistema más grande como se muestra en la figura 27.

Módulo del Operador

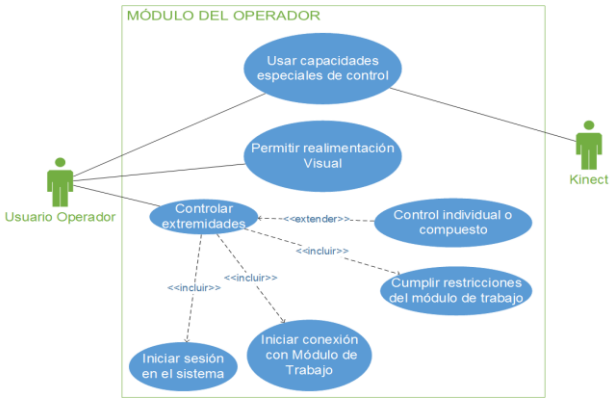


Figura 27. Descomposición del sistema “Módulo del Operador”.

Elaborado por: Edison Pilatasig

En la figura 27 el actor “Usuario Operador” hace referencia a la persona física que utiliza del manipulador robótico, se asocia con el caso de uso “Usar capacidades especial de control” que a su vez se encuentra asociado a un actor llamado “Kinect”, que representa al hardware Kinect con el cual se podrá gestionar el control por gestos humanos en el sistema del manipulador robótico.

El usuario operador se asocia con el caso de uso “Permitir realimentación visual” para especificar que el sistema debe ofrecer de alguna manera un mecanismo de realimentación de forma que el usuario operador pueda verificar que los movimientos se estén ejecutando en el lado remoto, esta es la razón por la cual el sistema es considerado de autonomía supervisada.

Finalmente el usuario operador se asocia con el caso de uso “Controlar extremidades”, lo que ofrecerá capacidades como control individual de extremidades o un control compuesto de todas las extremidades al mismo tiempo, además para poder realizar tal tarea se debe incluir acciones como: inicio de sesión en el sistema MRaaS, descubrir e iniciar una conexión con un módulo de trabajo que en este caso es el brazo robótico SCARA, y cumplir restricciones del módulo de trabajo que pueden ser extremidades no funcionales o tiempo de conexión asignado.

2.2.3 Caso de Uso del Módulo de Trabajo

El actor “Módulo de Trabajo” es a su vez un sistema más grande como se muestra en la figura 28.

Módulo de Trabajo

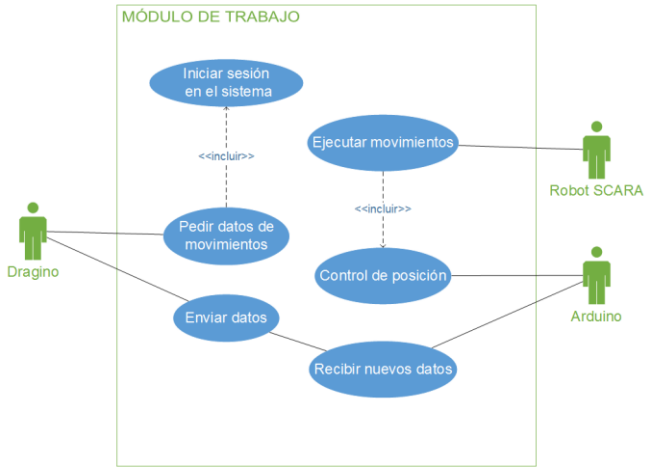


Figura 28. Descomposición del sistema "Módulo de Trabajo".

Elaborado por: Edison Pilatasig

En la figura 28 el actor "Dragino" se asocia con el caso de uso "Pedir datos de movimientos" en donde se realizan operaciones de consultas de nuevos movimientos de extremidades para el robot SCARA a través de internet, para esto se debe incluir un proceso de inicio de sesión en el sistema MRaaS, posteriormente se procede a "enviar datos" hacia el driver controlador que en este caso se representa como el actor llamado "Arduino", en donde se realiza la tarea "recibir nuevos datos" en donde se interpretan y se procede a realizar un "control de posición" que tomando en cuenta las lecturas de sensores de posición para las diferentes extremidades realizará o no un nuevo ajuste de valores para los actuadores (motores).

Esto es viable ya que el actor "Robot SCARA" para "ejecutar movimientos" debe estar conectado a una placa electrónica (driver) por medio de la cual se puedan leer datos de posición y posicionar los actuadores del mismo.

2.2.4 Diagrama de interacción en secuencia de la arquitectura general

Al usar casos de uso para modelar el sistema MRaaS se puede identificar el conjunto de capacidades o características que se deben ofrecer, adicionalmente se procede a especificar el funcionamiento general del sistema usando diagramas de interacción en secuencia, que también pertenecen al modelado UML, para dar a conocer el orden lógico en el cual funciona el sistema y los mensajes que se intercambian entre cada actor en diferentes procesos.

Se tiene en la figura 29 el diagrama de interacción en secuencia de la arquitectura general del sistema MRaaS, se tienen tres líneas de vida representadas por los tres actores principales de la arquitectura general del sistema (véase figura 26): módulo del operador, módulo de trabajo, y servicio MRaaS. Ambos actores: modulo del operador y módulo de trabajo deben iniciar una sesión en el sistema para lo cual envían un mensaje síncrono (flechas con relleno en el esquema) “iniciar sesión” con credenciales (usuario, dirección MAC o contraseña), dicho mensaje es procesado en el servicio MRaaS y si es un usuario valido se procede a devolver el *token* o identificador único que usarán los módulos (operador y trabajo) de aquí en adelante para realizar todas las tareas, es un mensaje síncrono ya que el proceso termina cuando se reciba una respuesta del servicio MRaaS pero no antes.

Posteriormente el actor modulo del operador es el encargado de enviar un mensaje síncrono “descubrir módulos de trabajo” con el fin de que el servicio MRaaS realice operaciones de consulta en la base de datos de módulos de trabajo conectados y activos, con lo que se procede a enviar un mensaje con un listado de módulos de

trabajo disponibles, con esto es posible enviar un mensaje síncrono “iniciar conexión con módulo” que el servicio MRaaS procesará con el fin de determinar si el módulo de trabajo en cuestión está aceptando nuevas conexiones, si es posible entonces se crea una nueva conexión y se envía un mensaje conteniendo la ID de conexión creada, solo entonces el actor modulo del operador es capaz de enviar un mensaje síncrono “descubrir características”, mensaje que tiene por tarea pedir al servicio MRaaS un listado de características del sistema como: extremidades habilitadas para su uso, restricciones del rango de movimiento, y el tiempo de conexión.

Con toda la información recolectada hasta el momento sobre el módulo de trabajo, en el módulo del operador se pueden intercambiar mensajes asíncronos de “enviar datos de movimientos” con el fin de que el servicio MRaaS los procese (verifique) y almacene, pueden ser mensajes asíncronos (fechas sin relleno en el esquema) ya que se pueden obviar las respuestas del servicio MRaaS con el fin de disminuir el tiempo entre envió de datos al esperar alguna confirmación, tiempo de envió entre datos que puede ser crítico en el desempeño del manipulador robótico. Simultáneamente se está ejecutando un proceso por parte del actor módulo de trabajo en donde se envían mensajes síncronos “consultar datos de movimientos” con el fin de obtener un listado con los nuevos datos de movimientos a ejecutar por parte del módulo del operador, dichos datos serán procesados en el módulo de trabajo y se posicionará el brazo robótico SCARA a través de un bucle de control local en el lado del actor módulo de trabajo, este proceso se ejecutará en un bucle mientras el usuario este enviando datos, recordando que los datos podrán ser enviados mientras no se salga del sistema o tenga el suficiente tiempo de conexión asignado, el mensaje “salir” es considerado síncrono con el fin de obtener una respuesta por parte del servicio MRaaS y asegurar

el fin de sesión y conexión de ambas partes (módulo del operador y módulo de trabajo). Se debe tener en cuenta que los actores: módulo del operador, módulo de trabajo, y servicio MRaaS son sistemas grandes que se pueden analizar en partes más pequeñas, pero hasta este punto, el fin del diagrama de secuencia es explicar el funcionamiento lógico del sistema y no restringir el funcionamiento a una plataforma de desarrollo específica.

Diagrama de interacción en secuencia de la arquitectura general

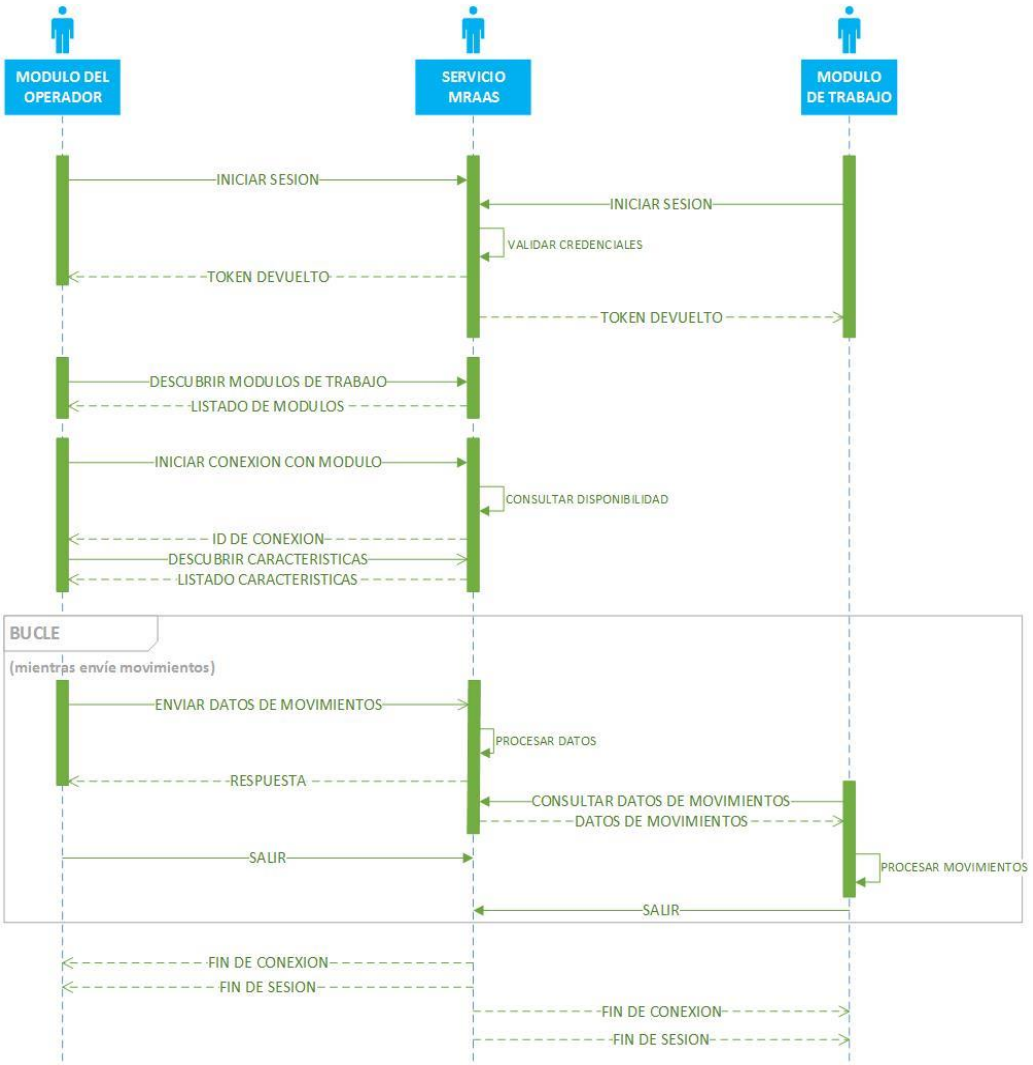


Figura 29. Diagrama de interacción en secuencia de la arquitectura general del sistema MRaaS.

Elaborado por: Edison Pilatasig

El diagrama de interacción en secuencia nos muestra los mensajes intercambiados entre los diferentes actores, en analogía al protocolo HTTP, se han denominado a estos mensajes como códigos de estado del sistema MRaaS, y se muestran detallados en la tabla 7.

Tabla 7.

Códigos de estado del sistema MRaaS

CÓDIGO	EVENTO	DESCRIPCIÓN	
		PETICIONES CORRECTAS -200	
200	*OK*		
200.1	OK.	Mensaje general para acciones de consulta ejecutadas correctamente, que devuelven el recurso solicitado, tal como "pedir la información de Usuario".	
201	*Creado*		
201.1	Creado.	Mensaje general para acciones de escritura ejecutadas correctamente, que devuelven el identificador del elemento creado, tal como "agregar una nueva sesión".	
204	*Sin contenido*		
204.1	Desconectado.	Mensaje generado para acciones de fin de conexión ejecutadas correctamente entre operador-modulo, que no devuelven un recurso, tal como "finalizar una conexión actual".	
204.2	Eliminado.	Mensaje generado para acciones de escritura ejecutadas correctamente en la base de datos, que no devuelven un recurso, tal como "Eliminar un usuario".	
204.3	Caducado.	Mensaje generado para acciones de control de uso ejecutadas correctamente en la base de datos, que no devuelven un recurso, tal como "indicar que el usuario ya no puede usar el sistema".	
204.4	Finalizado.	Mensaje generado para acciones de fin de sesión ejecutadas correctamente por el operador, que no devuelven un recurso, tal como "finalizar una sesión actual".	
		ERRORES DEL CLIENTE - 400	
400	*SOLICITUD INCORRECTA*		
400.1	Solicitud incorrecta.	Mensaje de error generado, en acciones en donde se ha enviado un campo vacío al servidor en la solicitud SOAP.	Verificar que se esté enviando todos los datos requeridos en la solicitud Soap hacia el servidor.
401	*No autorizado*		
401.1	Nivel Administrador requerido.	Mensaje de error generado, en acciones no permitidas para un usuario común.	Verificar que se usen credenciales de administrador, o solicite una al departamento administrativo.
401.2	Cuenta bloqueada.	Mensaje de error generado, cuando un usuario intenta iniciar sesión en el sistema, y tiene cuenta bloqueada referente a tiempo de conexión agotado o estado deshabilitado de usuario.	Actualice el estado de su cuenta en el departamento administrativo.
403	*Prohibido*		
403.1	Conexión prohibida.	Mensaje de error generado, ante un fallo en el intento de conexión, entre operador - módulo.	Verifique que la contra parte no sea de su mismo tipo (operador, módulo), y que el módulo se

			encuentre actualmente activo.
403.2	Acción no permitida.	Mensaje de error generado, ante intentos de ejecutar métodos con derechos de usuario no válidos o intentos de inicio doble de sesión.	Compruebe su llave token única, caso contrario contáctese con un administrador, quien eliminará su sesión o conexión previa.
403.3	Credenciales incorrectas.	Mensaje de error generado, ante el intento de inicio de sesión en el sistema con credenciales de acceso incorrectas o desactualizadas.	Revise sus credenciales (alias, contraseña, MAC), o contáctese con un administrador, para la actualización o creación de nuevas credenciales.
403.4	Elemento no eliminado.	Mensaje de error generado, en el intento de eliminar un elemento en la base de datos que corresponde a su actual sesión o conexión.	Comprobar que el elemento que intenta eliminar no corresponda, a su mismo usuario, a su misma sesión, o a su misma conexión activa.
403.5	Elemento duplicado	Mensaje de error generado, en el intento de escribir información que ya se encuentra en la base de datos.	Modificar la información, caso contrario contacte con un administrador, si la información es la correcta.
404	*No encontrado*		
404.1	Tipo módulo no encontrado.	Mensaje de error generado, cuando un elemento especificado por el usuario no existe en la tabla *TipoModulo* de la base de datos.	Contáctese con un administrador para agregar un nuevo tipo de módulo.
404.2	Componente no encontrado.	Mensaje de error generado, cuando un elemento especificado por el usuario no existe en la tabla *Componente* de la base de datos.	Contáctese con un administrador para agregar un nuevo componente.
404.3	Componente de Modulo No encontrado.	Mensaje de error generado, cuando un elemento especificado por el usuario no existe en la tabla *ComponenteModulo* de la base de datos.	Contáctese con un administrador para agregar una nueva relación para un componente con un módulo específico.
404.4	Rango usuario no encontrado.	Mensaje de error generado, cuando un elemento especificado por el usuario no existe en la tabla *Rango* de la base de datos.	Contáctese con un administrador para agregar un nuevo rango de usuario, si así el caso lo requiere.
404.5	Contrato no encontrado.	Mensaje de error generado, cuando un elemento especificado por el usuario no existe en la tabla *Contrato* de la base de datos.	Contáctese con un administrador para agregar un nuevo contrato de uso del sistema.
404.6	Módulo no encontrado.	Mensaje de error generado, cuando un elemento especificado por el usuario no existe en la tabla *Módulo* de la base de datos.	Contáctese con un administrador para agregar un nuevo módulo.
404.7	Usuario no encontrado.	Mensaje de error generado, cuando un elemento especificado por el usuario no existe en la tabla *Usuario* de la base de datos.	Contáctese con un administrador para agregar un nuevo usuario.
404.8	Sesión no encontrada.	Mensaje de error generado, cuando un elemento especificado por el usuario no existe en las tablas *SesionModulo/SesionUsuario* de la base de datos.	Verificar que se disponga de una sesión activa, ambos operador-modulo.
404.9	Conexión no encontrada.	Mensaje de error generado, cuando un elemento especificado por el usuario no existe en la tabla *Conexión* de la base de datos.	Verificar que se disponga de una conexión establecida.
404.10	Registro no encontrado.	Mensaje de error generado, cuando un elemento especificado por el usuario no existe en la tabla *Registro* de la base de datos.	Verificar que se disponga de un elemento en la tabla *Registro* en la base de datos.
406	*No Aceptado*		
406.1	Datos no válidos.	Mensaje de error generado, cuando el tipo de dato de un campo, no corresponde con el permitido en el servidor.	Compruebe que se ingrese el correcto tipo de datos, para cada campo en la solicitud Soap.
ERRORES DEL SERVIDOR - 500			
500	*Error Interno*		
500.1	Error Interno no controlado.	Mensaje de error generado ante un fallo en el lado del servidor.	Contacte con el administrador de sistema, indicando el mensaje de error presentado.

Nota: Detalle de códigos de estado intercambiados en el sistema MRaaS.

Elaborado por: Edison Pilatasig

2.3 Modelo de la Base de Datos

Hasta el momento se ha definido el funcionamiento del sistema MRaaS como conjunto de características mostrado en la figura 26 y como lógica mostrada en la figura 29, para lograr tal fin es necesario tener una colección de datos accesible y modificable donde administrar los registros necesarios.

Se opta por usar un gestor de bases de datos relacionales, aunque existen varios gestores más, se selecciona MS SQL por la versatilidad y facilidad de manejar la aplicación de acceso de datos por programación a través de un contexto de aplicación conceptual en código usando el paquete Entity Framework de Microsoft Visual Studio 2012, en pocas palabras se permitirá el manejo de la base de datos a través de líneas de programación en el servicio MRaaS.

Para diseñar un modelo de base de datos relacional del sistema MRaaS se sigue un proceso de normalización, respetando las tres primeras formas normales con el fin de evitar la redundancia de datos, evitar problemas de actualización de los datos en las tablas, y para proteger la integridad de los datos. Para la primera forma normal se establece dominios atómicos o indivisibles, es decir en cada tabla se identifican columnas repetidas y se las traslada a una nueva tabla, como se puede ver en la figura 30, como ejemplo nótese la tabla “Modulo” y “Usuario” aunque se las puede unir en una sola tabla, no es lo recomendado ya que “Modulo” tiene características diferentes o adicionales que de igual manera se relacionan mediante claves foráneas hacia otra tablas para evitar escribir la misma información en cada fila de la tabla, este

procedimiento lleva a la creación de tablas independientes para el manejo de sesiones “SesionModulo” y “SesionUsuario”.

La segunda forma normal establece que se cumpla de forma obligatoria la primera forma normal y además que cada tabla dependa de una clave principal que la identifique plenamente, esto se logra identificando cada fila en las tablas de la base de datos por medio de un ID que sencillamente es un numero entero que se incrementa automáticamente en orden ascendente (aunque puede hacerse por código), llamado campo *Identity,* y que nos asegura un control total por parte de MS SQL sin tener que realizar comprobaciones en el código de programa de los campo ID.

Para la tercera forma normal, y habiendo cumplido la segunda forma normal, se hace especial énfasis en establecer las relaciones entre tablas, estas relaciones se han establecido con el fin de ampliar las prestaciones del sistema MRaaS para crear un sistema manipulador robótico que se adapte a la enseñanza, para lo cual se toma en cuenta el control de uso por tiempo, se agrega tablas para usuarios operadores administradores y comunes, se agrega tablas para la aceptación de más de un módulo robótico que de igual manera pueden tener extremidades o articulaciones extra y se puedan controlar, un claro ejemplo de esto es la tabla “ComponenteModulo” en la figura 30, dicha tabla contiene una relación varios a varios, ya que para varios módulos robóticos pueden haber varias extremidades o componentes habilitados.

Modelo de la base de datos

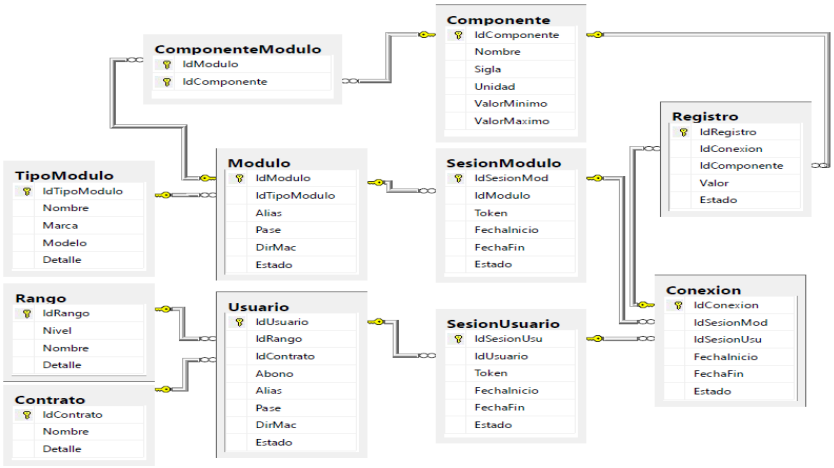


Figura 30. Esquema del modelo de la base de datos utilizada en el sistema MRaaS.

Elaborado por: Edison Pilatasig

Los atributos o campos de cada tabla en la base de datos se muestran detallados en la tabla 8.

Tabla 8.

Campos de la Base de Datos

NOMBRE	TIPO	DESCRIPCION
USUARIO *Almacenamiento de datos del usuario operador.*		
IdUsuario	Int	Clave principal dela tabla.
IdRango	Int	Id de rango asociado.
IdContrato	Int	Id de contrato asociado.
Abono	Int	Cantidad de tiempo de uso disponible para conexión.
Alias	Varchar(30)	Nombre de usuario operador.
Pase	Varchar(25)	Contraseña de usuario operador.
DirMac	Varchar(17)	Dirección MAC de usuario operador.
Estado	Bool	Estado habilitado o deshabilitado.
RANGO *Almacenamiento de rangos o jerarquías de usuarios operadores.*		
IdRango	Int	Clave principal dela tabla.
Nivel	Int	Nivel de rango o jerarquía.
Nombre	Varchar(20)	Nombre del rango o jerarquía.
Detalle	Varchar(250)	Detalle de la jerarquía agregado.
CONTRATO *Almacenamiento de tipos de contrato para el uso del sistema MRaaS.*		
IdContrato	Int	Clave principal dela tabla.
Nombre	Varchar(20)	Nombre del contrato.
Detalle	Varchar(250)	Detalle del contrato agregado.
SESIONUSUARIO *Almacenamiento de datos de sesiones de los usuarios operadores.*		
IdSesionUsu	Int	Clave principal dela tabla.
IdUsuario	Int	Id de usuario.
Token	Int	Token asignado al usuario para usar el sistema MRaaS
FechaInicio	Varchar(30)	Fecha de inicio de sesión.
FechaFin	Varchar(30)	Fecha de fin de sesión.

Estado	Bool	Estado sesión activa o finalizada.
CONEXIÓN *Almacenamiento de datos de conexiones realizadas en el sistema MRaaS.*		
IdConexion	Int	Clave principal dela tabla.
IdSesionMod	Int	Id de sesión de módulo conectado.
IdSesionUsu	Int	Id de sesión de usuario conectado.
FechaInicio	Varchar(30)	Fecha de inicio de conexión.
FechaFin	Varchar(30)	Fecha de fin de conexión.
Estado	Bool	Estado conexión activa o finalizada.
MODULO *Almacenamiento de datos del módulo de trabajo.*		
IdModulo	Int	Clave principal dela tabla.
IdTipoModulo	Int	Id de tipo de módulo.
Alias	Varchar(30)	Nombre del módulo
Pase	Varchar(25)	Contraseña para el módulo.
DirMac	Varchar(17)	Dirección MAC del módulo.
Estado	Bool	Estado habilitado o deshabilitado.
TIPOMODULO *Almacenamiento de datos para tipo de módulos robóticos existentes.*		
IdTipoModulo	Int	Clave principal dela tabla.
Nombre	Varchar(50)	Nombre del tipo de módulo.
Marca	Varchar(25)	Marca del fabricante del módulo.
Modelo	Varchar(50)	Modelo o serie del módulo.
Detalle	Varchar(250)	Detalle del módulo descrito.
COMPONENTE *Almacenamiento de datos para tipos de componentes o extremidades existentes.*		
IdComponente	Int	Clave principal dela tabla.
Nombre	Varchar(30)	Nombre del componente o extremidad.
Sigla	Varchar(3)	Sigla para el componente.
Unidad	Varchar(15)	Unidad de medida de movimiento.
ValorMinimo	Int	Valor mínimo para movimiento.
ValorMaximo	Int	Valor máximo para movimiento.
COMPONENTEMODULO *Almacenamiento de datos para relacionar los módulos de trabajo con componentes individuales.*		
IdModulo	Int	Claves principales de tabla para formar relación: varios a varios, se especifica de que componentes dispone cada módulo existente.
IdComponente	Int	
SESIONMODULO *Almacenamiento de datos de sesiones de los módulos de trabajo.*		
IdSesionMod	Int	Clave principal dela tabla.
IdModulo	Int	Id de módulo.
Token	Int	Token asignado al usuario para usar el sistema MRaaS.
FechaInicio	Varchar(30)	Fecha de inicio de sesión.
FechaFin	Varchar(30)	Fecha de fin de sesión.
Estado	Bool	Estado sesión activa o finalizada.
REGISTRO *Almacenamiento de datos de los movimientos realizados para los diferentes módulos registrados.*		
IdRegistro	Int	Clave principal dela tabla.
IdConexion	Int	Id de conexión.
IdComponente	Int	Id de componente manipulado.
Valor	Int	Valor de movimiento de componente.
Estado	Bool	Estado leído o no leído.

Nota: Descripción de los campos de la Base de Datos para el sistema MRaaS.

Elaborado por: Edison Pilatasig

2.4 Procesos en el sistema MRaaS

Se definen a continuación los diagramas de actividades en UML, similar a los diagramas de flujo, necesarios para lograr una mejor comprensión del

funcionamiento de los dos procesos más relevantes que se van a ejecutar en el sistema MRaaS: proceso de conexión con módulo de trabajo, proceso de inserción de movimientos, proceso de petición de movimientos y el proceso de fin de sesión.

2.4.1 Proceso de conexión con módulo de trabajo

En el lado del módulo del operador, se plantea el desarrollo de una aplicación de interfaz gráfica basada en WPF, tecnología de Microsoft Visual Studio, en lenguaje c#, con el fin de consumir el servicio web MRaaS, obtener un resultado de interfaz gráfica agradable y total compatibilidad con el sensor Kinect.

Proceso de conexión con módulo de trabajo

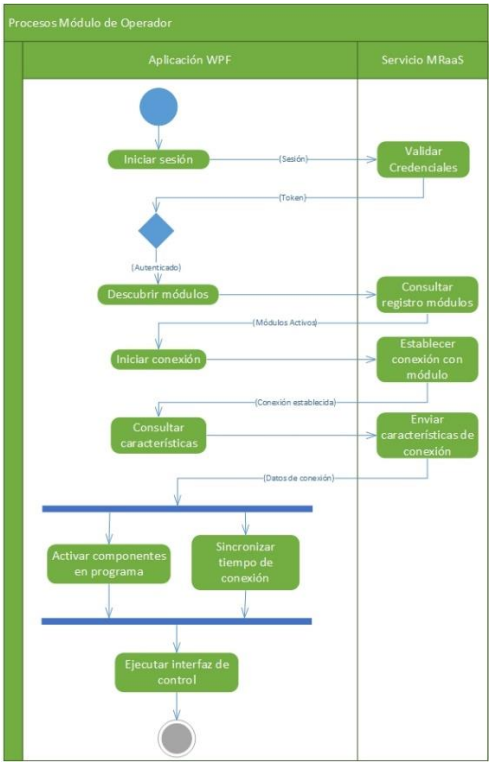


Figura 31. Diagrama de actividades UML representando los procesos en el Módulo del Operador para realizar una conexión con el Módulo de Trabajo.

Elaborado por: Edison Pilatasig

Se describe en la figura 31 una diagramación de características como procesos con diagramas de actividades de UML, el proceso empieza al ejecutar la aplicación, en ese momento el usuario ingresará las credenciales para ser validadas en el servicio MRaaS, credenciales que de ser válidas permitirán al usuario obtener un *token*, término que simboliza el identificador único que se dará a los usuarios conectados al MRaaS para realizar operaciones sobre el mismo, de esta manera el usuario necesita poder descubrir los módulos de trabajo conectados, siendo este caso el brazo robótico SCARA, de esta forma usando el token se envían peticiones de descubrimiento y el servicio MRaaS devolverá un listado de módulos de trabajo conectados, los cuales están a disposición del usuario operador para iniciar una conexión que luego de ser validada por el servicio MRaaS, permitirá al usuario operador consultar las características de conexión, características como: tiempo de conexión disponible, extremidades habilitadas y rango de movimiento de las mismas, posteriormente esta información es interpretada por la aplicación WCF, con lo que se procede a activar las extremidades válidas en la interfaz gráfica y al mismo tiempo a sincronizar el tiempo de conexión disponible, finalmente la aplicación para la manipulación del brazo robótico SCARA inicia.

2.4.2 Proceso de inserción de movimientos

Terminado el proceso de conexión, ante acciones de manipulación del robot SCARA por parte del usuario operador se genera un proceso de petición de inserción de nuevos movimientos en el servicio MRaaS, este proceso se detalla en la figura 32.

Proceso para insertar movimiento en el Módulo del Operador

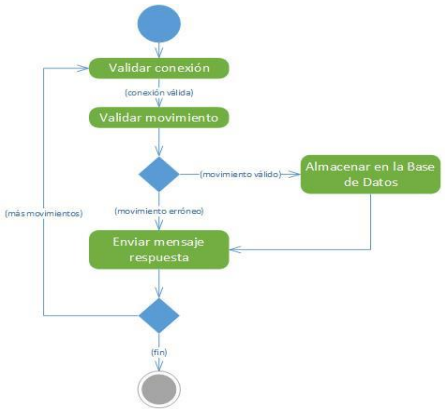


Figura 32. Diagrama de actividades UML representando el proceso de inserción de movimientos.

Elaborado por: Edison Pilatasig

El proceso se realiza en el servicio MRaaS, luego de la petición de inserción de nuevos movimientos por parte del usuario operador tomando en cuenta el token de usuario, para comprobar si es una conexión válida, posteriormente se realiza una verificación del movimiento realizado tomando en cuenta si la extremidad a moverse está habilitada o si el movimiento está dentro del rango permitido. Si se cumplen estas condiciones entonces el servicio MRaaS procede a almacenar la información referente al movimiento en la base de datos relacional creada para tal objetivo y retorna un mensaje de respuesta de información exitosa o de errores que pueden surgir, si existen más peticiones el proceso se repite, destacando que le funcionamiento del sistema es petición/respuesta el servidor actuará ante peticiones efectuadas.

2.4.3 Proceso de petición de movimientos

Por otra parte un módulo de trabajo está conectado al sistema MRaaS, se detallan los procesos principales necesarios en el módulo de trabajo en la figura 33.

Proceso de petición de movimientos

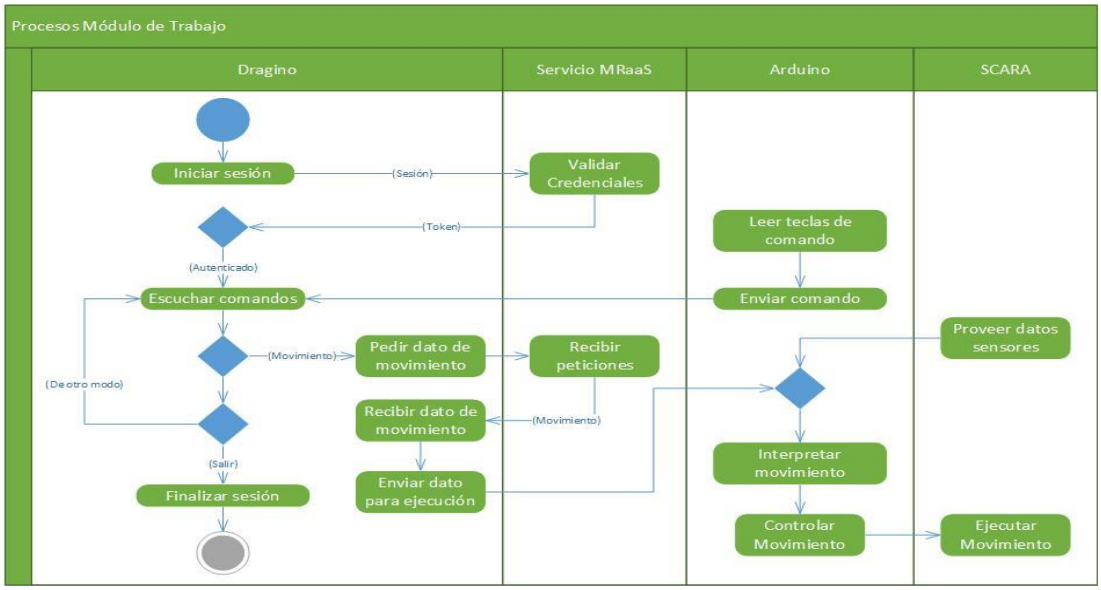


Figura 33. Diagrama de actividades UML representando los procesos en el Módulo del Trabajo para realizar peticiones de movimientos.

Elaborado por: Edison Pilatasig

De igual manera que en el módulo del operador, en el módulo de trabajo se realiza un proceso de autenticación de modulo llevado a cabo por el módulo Dragino a través de una aplicación en Python hacia el servicio MRaaS, estas credenciales de autenticación incluyen la dirección MAC del módulo con el fin de añadir una capa extra de seguridad, posteriormente y una vez obtenido el token el módulo Dragino se mantiene a la escucha de comandos, comandos que son enviados por la placa Arduino que posee una botonera por medio de la cual se puede administrar los procesos (inicio, fin de sesión o conexión y petición de datos) manualmente o

automáticamente, la aplicación en Python debe ser capaz de diferenciar estos comandos. Ante la petición de datos de movimientos el servicio MRaaS realiza una consulta en la base de datos tomando en cuenta el token de módulo de trabajo y así discriminar entre varios módulos que pudieran estar conectados a la vez, los datos obtenidos con tratados con el fin de armar una trama que es enviada a la placa Arduino, la cual habiendo realizado un proceso de lectura de sensores, determina la posición actual de brazo robótico SCARA y dependiendo del nuevo dato de movimiento activa sus salidas lógicamente para interactuar con el hardware de control del brazo robótico SCARA y lograr su movimiento bajo demanda de extremidad requerida o abriendo las puertas a interpretación de movimientos bajo algoritmos dado.

2.4.4 Proceso de fin de sesión

A modo general el proceso para finalizar una sesión en el sistema MRaaS se representa en la figura 34.

Proceso de fin de sesión en el sistema MRaaS

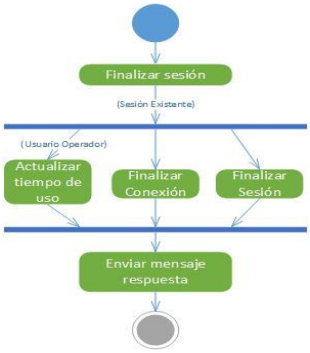


Figura 34. Diagrama de actividades UML representando el proceso en que se finaliza una sesión en el sistema MRaaS.

Elaborado por: Edison Pilatasig

Ante la petición de “finalizar sesión” ya sea por parte del módulo de trabajo o por parte del módulo del operador, el servicio MRaaS procede a verificar si efectivamente se encuentra activa esa sesión, para ello hace uso del token asignado, posteriormente se finaliza una conexión si hubiere, si es un usuario operador se actualizará el tiempo restante de uso o abono, y finalmente se dará fin a la sesión, registrando en cada caso la información en la base de datos.

CAPITULO 3
IMPLEMENTACIÓN

Se presenta el desarrollo del servicio MRaaS bajo la plataforma WCF, la aplicación del módulo del operador y del módulo de trabajo, finalmente la implementación del hardware en sí. En el capítulo 2 se había modelado el funcionamiento del sistema MRaaS mediante lenguaje UML, dicho análisis permite establecer criterios para crear las clases necesarias en el servicio MRaaS.

3.1 Servicio MRaaS

Se programa el servicio MRaaS utilizando la plataforma WCF de Microsoft Visual Studio, se utiliza mensajes SOAP para hacer que el sistema MRaaS funcione bajo un patrón de comunicación Envío-Respuesta, WCF es un modelo unificado de programación para el desarrollo de aplicaciones orientadas a servicios, permite crear servicios bajo la plataforma .NET y está diseñado para ofrecer un enfoque hacia la computación distribuida, amplia interoperabilidad y apoyo directo a la orientación de servicios.

WCF es compatible con muchos estilos de desarrollo de aplicaciones distribuidas al proporcionar una arquitectura de capas. La arquitectura de canal WCF proporciona mensajes asíncronos primitivos sin tipo, comunicación a través de estándares de servicios web, interoperabilidad con otras plataformas que soporten SOAP, o se puede configurar WCF para trabajar con mensajes simples en formato XML.

Arquitectura de capas del servicio MRaaS

Figura 35. Descripción del modelo de arquitectura de capas del servicio MRaaS.

Elaborado por: Edison Pilatasig

Un aspecto importante del diseño SOA es que los límites de servicios deben ser explícitos, lo que significa que se debe ocultar todos los detalles de la implementación detrás del límite del servicio (Liu, 2012, pág. 82)

Esto incluye no revelar la tecnología usada, dentro de la implementación de un servicio, el código responsable de la manipulación de datos debe ser separado del código responsable de la lógica de negocio. Tomando estas consideraciones en cuenta para el sistema MRaaS como buena práctica se implementa un servicio WCF en cuatro capas, véase Figura 35. Las cuatro capas son la capa de interfaz de servicio, la capa de lógica de negocio, la capa de acceso de datos y capa entidades. Logrando así una mayor seguridad, organización y comprensión del código.

El modelo usado consiste en MVC (modelo – vista – controlador), consiste en la interfaz del servicio (vista), la capa que maneja los eventos o acciones de los usuarios al invocar peticiones al servicio (controlador), y las capas de acceso a datos y lógica de negocio del servicio (modelo).

La capa interfaz del servicio contendrá todos los contratos de servicio y operaciones de contrato usados para definir las interfaces de servicios que serán expuestas a los

usuarios, los contratos de datos definen el paso de datos de entrada o salida del servicio.

La capa lógica de negocios aplica las operaciones del servicio donde se revisan las condiciones de cada operación, como por ejemplo operaciones para la administración de la sesión de usuarios, se realizan actividades de negocio y se retornan a la interfaz de servicio algunos valores de ser necesario, estas operaciones contienen llamadas a métodos presentes en la capa acceso de datos, donde se realizan las tareas necesarias de acceso a bases de datos subyacentes, conexiones, procesamiento de transacciones y control de concurrencia, se utiliza algún adaptador para realizar consultas y actualizaciones a las bases de datos, en este caso a través de un contexto con el paquete Entity Framework hacia la capa Entidades por medio de la cual se puede acceder a la base de datos mediante código de programa para operaciones de consulta, modificación o eliminación.

3.1.1 Capa Interfaz del Servicio

La capa de interfaz del servicio contendrá los contratos de servicio a los cuales el usuario está habilitado de acceder, esta capa contiene la clase IServicioMRaaS destinada a exponer las operaciones mostradas en la figura 36, contiene además la clase ServicioMRaaS que implementa la interface de contrato de servicios, es decir es aquí en donde se puede agregar código.

Capa Interfaz del Servicio

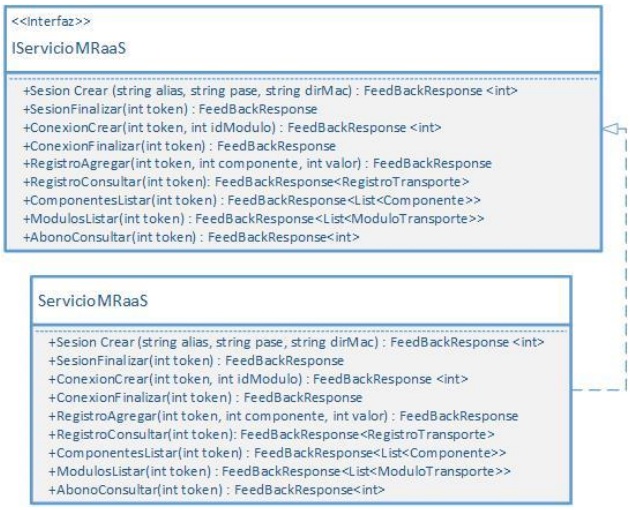


Figura 36. Diagrama de clases UML de la Capa Interfaz del Servicio del servicio MRaaS.

Elaborado por Edison Pilatasig

El servicio web es descrito a través de WSDL (Web Services Description Language) usando un formato XML donde se describen: los tipos de datos usados en los mensajes, los elementos del mensaje o sus parte lógicas, los tipos de puerto para definir operaciones permitidas y mensajes intercambiados, los enlaces donde se definen los protocolos de comunicación, y los servicios definiendo un conjunto de puertos y dirección de los mismos. Es así que un cliente “lee” el archivo wsdl para conocer la forma en la que se debe interactuar con el servicio web. La interfaz define las operaciones permitidas y estas se implementan en forma de métodos, los cuales realizan llamadas a la capa lógica de negocios, y se describen en la tabla 9.

3.1.2 Capa de lógica de negocio

La capa lógica de negocio tiene como fin ser una capa intermedia entre la interfaz y el acceso a datos, contiene operaciones de administración y seguridad, operaciones como: verificación de datos de usuario, sesión y conexión de trabajo, de esta manera

se logra un control ante usuarios no identificados, o usuarios no habilitados para el uso del sistema MRaaS. La capa Interfaz de Servicio realiza llamadas a los métodos en esta capa, que a su vez realiza llamadas a varios métodos en la capa Acceso de Datos, se implementa la reutilización de código con el fin de disminuir las líneas de código repetidas que se puedan llegar a tener, se muestra en la figura 37 el diagrama de clases UML de la capa Lógica de Negocios.

Figura 37. Diagrama de clases UML de la Capa Lógica de Negocio del servicio MRaaS.

Elaborado por Edison Pilatasig

Se describen en la tabla 9 los métodos existentes por clase de la capa Lógica de Negocio.

Tabla 9.

Descripción de capa lógica de negocios

CLASE	MÉTODO	DESCRIPCIÓN
NegocioComponente	+ComponentesListar(int tokenCifrado) : FeedBackResponse<List<Componente>>	Método que lista los componentes habilitados para un manipulador conectado en el sistema, recibe el token de operador, devuelve un tipo FeedBackResponse con un listado de datos de componentes encapsulados en un objeto tipo Componente.
NegocioRegistro	+RegistroConsultar(int tokenCifrado) : FeedBackResponse<RegistroTransporte>	Método para consultar el movimiento ingresado en la base de datos por el operador, recibe el token de estación de trabajo, devuelve un tipo FeedBackResponse con datos del movimiento encapsulados en un objeto tipo RegistroTransporte.
	+RegistroAgregar(int tokenCifrado, int	Método para agregar un nuevo

	componente, int valor) : FeedBackResponse	movimiento en la base de datos, recibe el token de operador, el ID de extremidad a manipular y valor de movimiento, devuelve un tipo FeedBackResponse de confirmación.
NegocioConexión	+ConexionCrear(int tokenCifrado, int idSesionModulo) : FeedBackResponse<int>	Método para crear una nueva conexión en el sistema, recibe el token de operador e ID de módulo de trabajo, devuelve un tipo FeedBackResponse con un atributo entero, que es el ID de conexión.
	+ConexionFinalizar(int tokenCifrado) : FeedBackResponse	Método para finalizar la conexión en el sistema, recibe el token de usuario y devuelve un tipo FeedBackResponse de confirmación.
NegocioSesión	+SesionCrear(string alias, string pase, string dirMac) : FeedBackResponse<int>	Método para crear una nueva sesión en el sistema, recibe: alias, contraseña, y dirección Mac, devuelve un tipo FeedBackResponse con un atributo entero, que es el token de usuario.
	+SesionFinalizar(int token) : FeedBackResponse	Método para finalizar la sesión en el sistema, recibe el token de usuario y devuelve un tipo FeedBackResponse de confirmación.
NegocioMódulo	+ModuloListar(int tokenCifrado) : FeedBackResponse<List<ModuloTransporte>>	Método para consultar los módulos o estaciones de trabajo operativas para su manipulación, recibe el token de operador, devuelve un tipo FeedBackResponse con un listado de estaciones encapsulados en un objeto tipo ModuloTransporte.
NegocioUsuario	+AbonoConsultar(int tokenCifrado) : FeedBackResponse<int>	Método para consultar el abono o tiempo de conexión disponible, recibe el token de operador, devuelve un tipo FeedBackResponse con un tipo entero, que es el tiempo disponible de uso.

Nota: Descripción de los métodos implementados en la capa lógica de negocios.

Elaborado por: Edison Pilatasig

3.1.3 Capa de acceso de datos

La capa de acceso de datos es la encargada de todas las operaciones de acceso hacia la base de datos, no incluye operaciones de seguridad porque se suponen hechas en la capa lógica de negocios como es el presente caso, o se hayan implementado seguridades para el servicio web - WSE (Web Services Enhancements), de igual forma en la figura 38 se muestra el diagrama de clases UML que muestra los métodos creados y clasificados en clases para poder lograr una mejor administración, comprensión de código y abrir la posibilidad de agregar más métodos de forma

organizada para necesidades futuras de ampliación de servicios del sistema MRaaS. Mientras que en la tabla 10 se da una descripción del objetivo de cada método en su clase. La capa acceso de datos actúa directamente sobre la capa entidades, de donde se obtiene un acceso a la base de datos por medio de Entity Framework, para realizar las operaciones de consulta, escritura o borrado de registros.

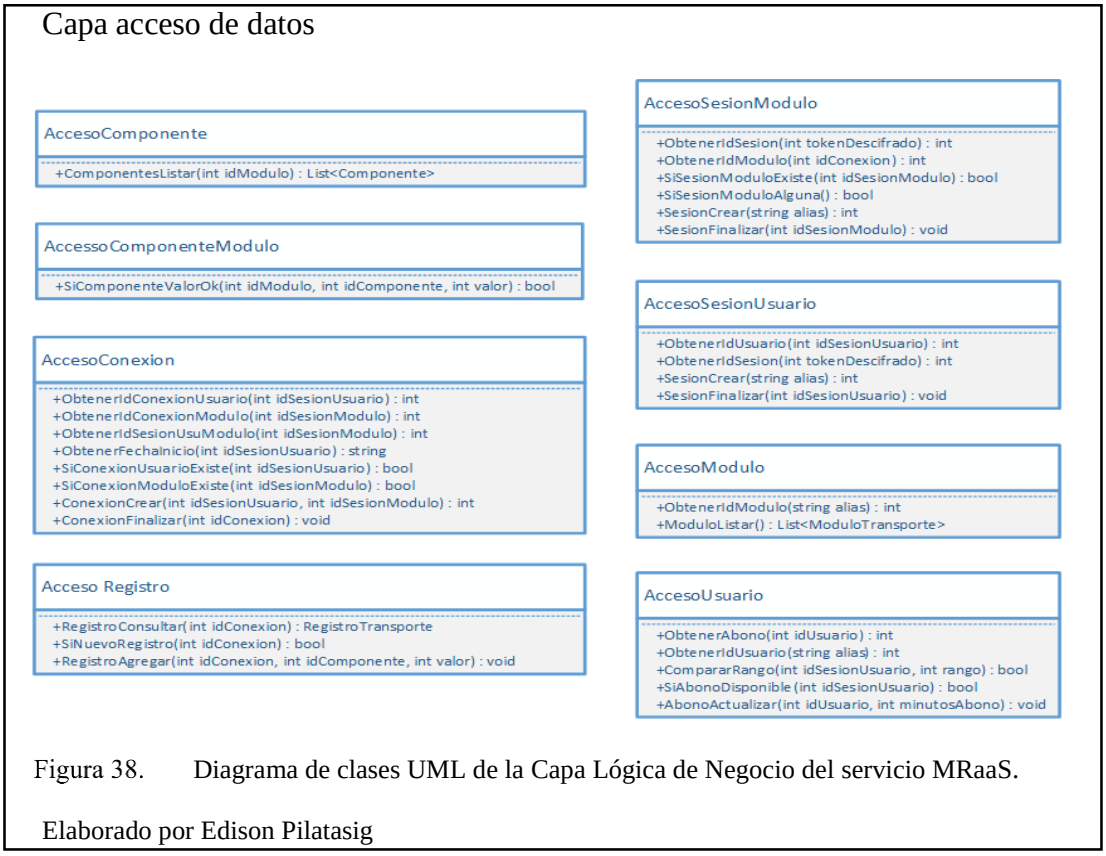


Figura 38. Diagrama de clases UML de la Capa Lógica de Negocio del servicio MRaaS.

Elaborado por Edison Pilatasig

Tabla 10.

Descripción de capa acceso de datos

CLASE	MÉTODO	DESCRIPCIÓN
AccesoComponente	+ComponentesListar(int idModulo) : List<Componente>	Método para listar los componentes habilitados dependiendo del módulo escogido, recibe el ID de módulo y devuelve un listado tipo Componente de los componentes habilitados.
AccessoComponenteModulo	+SiComponenteValorOk(int idModulo, int idComponente, int valor) : bool	Método para comprobar si el valor de movimiento ingresado es válido, recibe el ID de módulo, el ID de componente y el valor, devuelve un tipo Bool dependiendo si es verdadero o falso.

AccesoConexion	+ObtenerIdConexionUsuario(int idSesionUsuario) : int	Método para obtener el ID de conexión de usuario operador, recibe el ID de sesión del mismo, devuelve un entero que es el ID de conexión.
	+ObtenerIdConexionModulo(int idSesionModulo) : int	Método para obtener el ID de conexión de módulo de trabajo, recibe el ID de sesión del mismo, devuelve un entero que es el ID de conexión.
	+ObtenerIdSesionUsuModulo(int idSesionModulo) : int	Método para obtener el ID de sesión de usuario operador mediante el ID de sesión de módulo, recibe el ID de sesión de módulo, devuelve un entero que es el ID de conexión de usuario.
	+ObtenerFechaInicio(int idSesionUsuario) : string	Método para obtener la fecha de inicio de sesión de usuario operador, recibe el ID de sesión del mismo, devuelve la fecha en una cadena de caracteres.
	+SiConexionUsuarioExiste(int idSesionUsuario) : bool	Método para comprobar si un usuario operador tiene una conexión activa, recibe el ID de sesión de usuario operador, devuelve un tipo Bool dependiendo si es verdadero o falso.
	+SiConexionModuloExiste(int idSesionModulo) : bool	Método para comprobar si un módulo de trabajo tiene una conexión activa, recibe el ID de sesión de módulo, devuelve un tipo Bool dependiendo si es verdadero o falso.
	+ConexionCrear(int idSesionUsuario, int idSesionModulo) : int	Método para crear una nueva conexión, recibe los ID de sesión de Módulo y Usuario operador, devuelve un entero que es el ID de conexión.
	+ConexionFinalizar(int idConexion) : void	Método para finalizar una conexión activa, recibe el ID de conexión.
AccesoRegistro	+RegistroConsultar(int idConexion) : RegistroTransporte	Método para consultar un movimiento en los registros de la base de datos, recibe el ID de conexión y devuelve un objeto tipo RegistroTransporte con los datos del movimiento.
	+SiNuevoRegistro(int idConexion) : bool	Método para consultar si existen movimientos en los registros de la base de datos que no se han leído, recibe el ID de conexión y devuelve un tipo Bool dependiendo si es verdadero o falso.
	+RegistroAgregar(int idConexion, int idComponente, int valor) : void	Método para agregar un nuevo movimiento en los registros de la base de datos, recibe el ID de conexión, el ID de componente y el valor.
AccesoSesionModulo	+ObtenerIdSesion(int tokenDescifrado) : int	Método para obtener el ID de sesión del módulo por token, recibe el token y devuelve un entero con el ID de sesión.
	+ObtenerIdModulo(int idConexion) : int	Método para obtener el ID de módulo, recibe el ID de conexión y devuelve un entero con el número de ID.
	+SiSesionModuloExiste(int idSesionModulo) : bool	Método para comprobar si un

		módulo tiene sesión activa, recibe un ID de sesión y devuelve un tipo Bool dependiendo si es verdadero o falso.
	+SiSesionModuloAlguna() : bool	Método para comprobar si existe algún módulo con sesión activa, devuelve un tipo Bool dependiendo si es verdadero o falso.
	+SesionCrear(string alias) : int	Método para crear una nueva sesión de módulo, recibe el alias y devuelve un entero que es el ID de sesión creada.
	+SesionFinalizar(int idSesionModulo) : void	Método para finalizar la sesión de módulo, recibe el ID de sesión.
AccesoSesionUsuario	+ObtenerIdUsuario(int idSesionUsuario) : int	Método para obtener el ID de usuario operador, recibe el ID de sesión y devuelve un entero que es el ID de usuario.
	+ObtenerIdSesion(int tokenDescifrado) : int	Método para obtener el ID de sesión de usuario, recibe el token y devuelve un entero que es el ID de sesión.
	+SesionCrear(string alias) : int	Método para crear una nueva sesión de usuario operador, recibe el alias y devuelve un entero que es el ID de sesión creada.
	+SesionFinalizar(int idSesionUsuario) : void	Método para finalizar la sesión de usuario, recibe el ID de sesión.
AccesoModulo	+ObtenerIdModulo(string alias) : int	Método para obtener el ID de módulo, recibe el alias y devuelve un entero que es el ID de módulo.
	+ModuloListar() : List<ModuloTransporte>	Método para listar los módulos registrados en la base de datos en un objeto tipo ModuloTransporte.
AccesoUsuario	+ObtenerAbono(int idUsuario) : int	Método para obtener el tiempo de uso restante para el usuario operador, recibe el ID de usuario operador y devuelve un entero que es el valor de tiempo de uso.
	+ObtenerIdUsuario(string alias) : int	Método para obtener el ID de usuario, recibe el alias y devuelve un entero que es el ID de usuario.
	+CompararRango(int idSesionUsuario, int rango) : bool	Método para comparar el rango de jerarquía de un usuario operador, recibe el ID se sesión de usuario operador, el rango solicitado a comparar y devuelve un tipo Bool dependiendo si es verdadero o falso.
	+SiAbonoDisponible(int idSesionUsuario) : bool	Método para consultar si un usuario operador aún dispone de tiempo de uso, recibe el ID de sesión de usuario, y devuelve un tipo Bool dependiendo si es verdadero o falso.
	+AbonoActualizar(int idUsuario, int minutosAbono) : void	Método para actualizar el abono de usuario operador, recibe el ID de usuario, y el abono a asignar.

Descripción de los métodos implementados en la capa acceso de datos.

Elaborado por: Edison Pilatasig

3.1.4 Capa entidades

En este trabajo se ha decidido por llamar capa entidades a la representación en código del modelo de la base de los datos relacional utilizada, aunque es igualmente una capa de acceso de datos. Al utilizar el ADO .NET Entity Framework se busca que el servicio web esté libre de dependencias de codificación rígida del motor de datos para trabajar con datos en forma de objetos y propiedades sin tomar en cuenta la estructura de tablas y columnas en donde se almacenan estos datos.

Se puede deducir que las clases no poseen métodos pero si atributos, y cada clase corresponde con cada tabla mostrada en la figura 30 y descrita en la tabla 8.

3.1.5 Base de datos

Entity Framework provee de flexibilidad en la programación del servicio al brindar la capacidad de crear clases del modelo sin tener que asociarlas con alguna base de datos, el esquema de la base de datos es creado automáticamente, se crea una clase MiContexto.cs que hereda la clase DbContext de Entity Framework, el método DbContext() crea una nueva instancia de contexto por medio de convenciones para crear el nombre de la base de datos con la que se realizará una conexión, consultar una base de datos o agrupar los cambios para escribirlos como una unidad, o lo que es lo mismo, el contexto es una representación en memoria a partir del código de programa del esquema de la base de datos.

Contexto en el sistema MRaaS

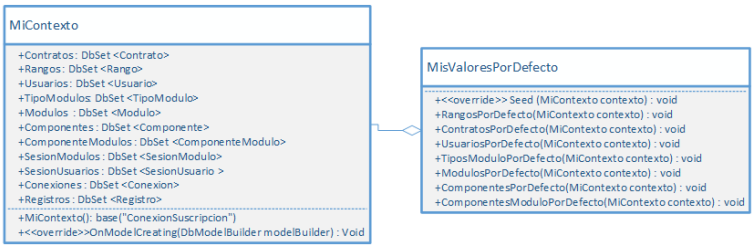


Figura 39. Diagrama de clases UML del contexto implementado para la base de datos.

Elaborado por: Edison Pilatasig

Se definen clases planas de objetos CLR "antiguos" - POCO (Plain Old CLR Objects), clases que no tienen persistencia lógica como almacenar o devolver datos de una base de datos, POCO son clases públicas que solo contienen métodos, propiedades y representan el esquema de la base de datos, los datos de inicialización entran a IDatabaseInitializer para ser implementados por DropCreateDatabaseIfModelChanges en la clase MisValoresPordefecto.cs, muy útil para la etapa de desarrollo ya que permite eliminar, recrear, y opcionalmente repoblar la base de datos automáticamente solo si el modelo ha cambiado desde que la base de datos fue creada, de lo cual se concluye que la primera vez que se hace una consulta a la base de datos y esta no está creada, Entity Framework la crea basándose en el contexto MiContexto.cs, en las clases POCO de la capa Entidades y almacenando datos si así se solicita a través del método Seed() en la clase MisValoresPordefecto.cs, todo esto a través de una conexión realizada a la base de datos especificada en ConexionSuscripcion en el archivo web.config del servicio MRaaS, en la figura 39 se muestra el diagrama de clases UML del contexto implementado.

3.1.6 Hosting

El proyecto descrito se compone de cuatro capas con un total de 37 clases, para alojar el servicio MRaaS en la nube se utiliza el hosting ASP .NET de la empresa SmarterASP creando un nuevo sitio mostrado en la figura 40 configurado para utilizar la tecnología ASP .NET 4, como para pruebas locales se utiliza la versión 2012 de MSSQL, mostrado en la figura 41, se procede de igual manera a utilizar un servidor de bases de datos en MSSQL 2012, mostrado en la figura 42, y cuya configuración se realiza de acuerdo a la figura 43, tomando en cuenta las pruebas realizadas en el siguiente capítulo se creará automáticamente el esquema de la base de datos relacional como se indica en la figura 44. Se generan los binarios del servicio web que son alojados mediante FTP usando el software FileZilla, véase figura 45, en el sitio web creado, para finalmente poder alcanzar el servicio MRaaS visto en la figura 46 desde cualquier lugar y cualquier momento a través de la dirección URL: http://edymaur1-001-site1.myasp.net/ServicioMRaaS.svc.

Hosting MRaaS

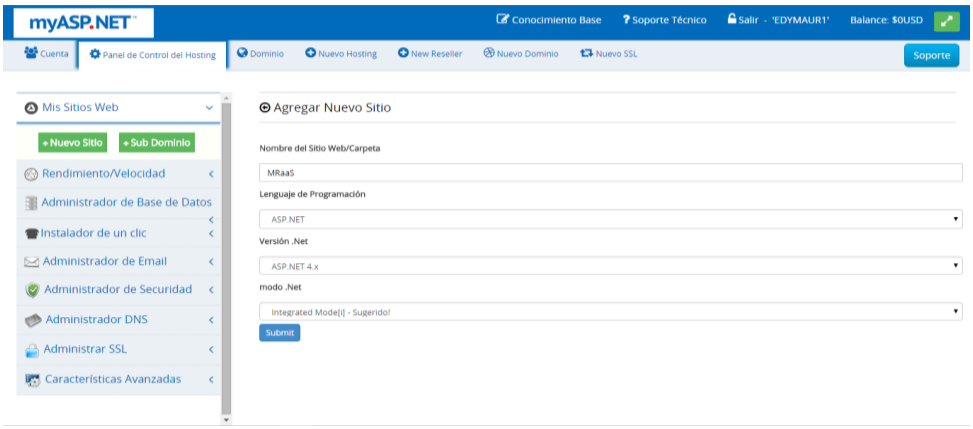


Figura 40. Hosting del servicio web para el sistema MRaaS.

Elaborado por: Edison Pilatasig

Versión MS SQL en pruebas locales

Microsoft SQL Server 2012 Service Pack 1 Setup Discovery Report

Product	Instance	Instance ID	Feature	Language	Edition	Version	Clustered
Microsoft SQL Server 2012	MSSQLSERVER	MSSQL11.MSSQLSERVER	Servicios de Motor de base de datos	1033	Enterprise Edition	11.0.2218.0	No
Microsoft SQL Server 2012	MSSQLSERVER	MSSQL11.MSSQLSERVER	Servicios de Motor de base de datos	3082	Enterprise Edition	11.0.2218.0	No
Microsoft SQL Server 2012	MSSQLSERVER	MSSQL11.MSSQLSERVER	Replicación de SQL Server	1033	Enterprise Edition	11.0.2218.0	No
Microsoft SQL Server 2012	MSSQLSERVER	MSSQL11.MSSQLSERVER	Replicación de SQL Server	3082	Enterprise Edition	11.0.2218.0	No
Microsoft SQL Server 2012	MSSQLSERVER	MSSQL11.MSSQLSERVER	Extracciones de texto completo y semánticas de búsqueda	1033	Enterprise Edition	11.0.2100.60	No
Microsoft SQL Server 2012	MSSQLSERVER	MSSQL11.MSSQLSERVER	Data Quality Services	1033	Enterprise Edition	11.0.2100.60	No
Microsoft SQL Server 2012	MSSQLSERVER	MSSQL11.MSSQLSERVER	Data Quality Services	3082	Enterprise Edition	11.0.2100.60	No
Microsoft SQL Server 2012	MSSQLSERVER	MSAS11.MSSQLSERVER	Analysis Services	1033	Enterprise Edition	11.0.2218.0	No
Microsoft SQL Server 2012	MSSQLSERVER	MSAS11.MSSQLSERVER	Analysis Services	3082	Enterprise Edition	11.0.2218.0	No
Microsoft SQL Server 2012	MSSQLSERVER	MSRS11.MSSQLSERVER	Reporting Services - Nativo	1033	Enterprise Edition	11.0.2218.0	No
Microsoft SQL Server 2012	MSSQLSERVER	MSRS11.MSSQLSERVER	Reporting Services - Nativo	3082	Enterprise Edition	11.0.2218.0	No
Microsoft SQL Server 2012			Herramientas de administración - Básica	3082	Enterprise Edition	11.0.2218.0	No
Microsoft SQL Server 2012			Herramientas de administración - Completa	3082	Enterprise Edition	11.0.2218.0	No
Microsoft SQL Server 2012			Conectividad con las herramientas de cliente	3082	Enterprise Edition	11.0.2218.0	No
Microsoft SQL Server 2012			Compatibilidad con versiones anteriores de las herramientas de cliente	3082	Enterprise Edition	11.0.2218.0	No
Microsoft SQL Server 2012			SDK de las herramientas de cliente	3082	Enterprise Edition	11.0.2218.0	No
Microsoft SQL Server 2012			Integration Services	3082	Enterprise Edition	11.0.2218.0	No
Microsoft SQL Server 2012			Reporting Services - SharePoint			11.0.2218.0	No

Figura 41. Descripción del software Microsoft SQL Server 2012 en pruebas locales.

Elaborado por: Edison Pilatasig

Versión MS SQL en la nube

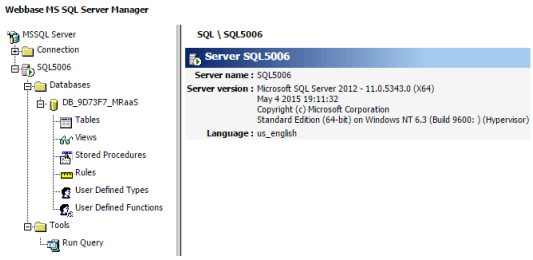


Figura 42. Descripción del software Microsoft SQL Server 2012 en pruebas en la nube.

Elaborado por: Edison Pilatasig

Creación de la base de datos

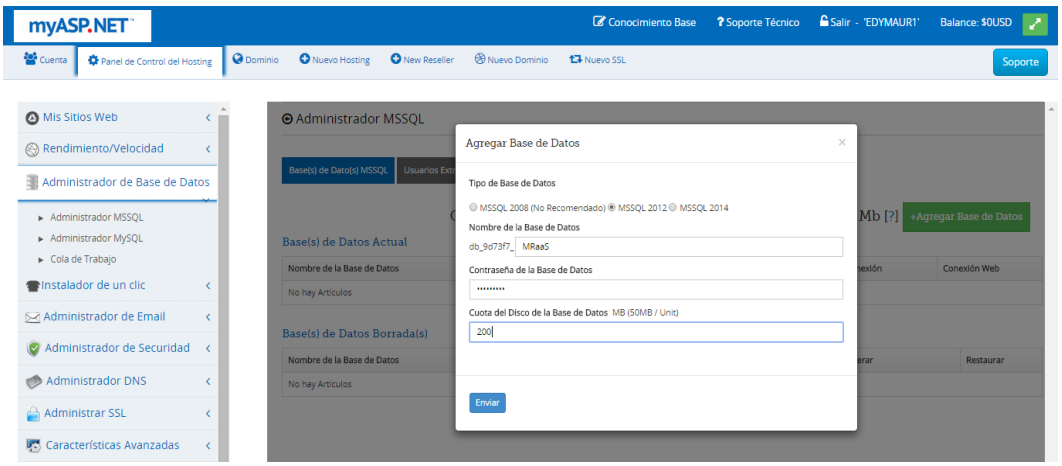


Figura 43. Creación de la base de datos en la nube.

Elaborado por: Edison Pilatasig

Base de datos en la nube

Figura 44. Tablas de la base de datos en la nube.

Elaborado por: Edison Pilatasig

Servicio MRaaS hacia la nube

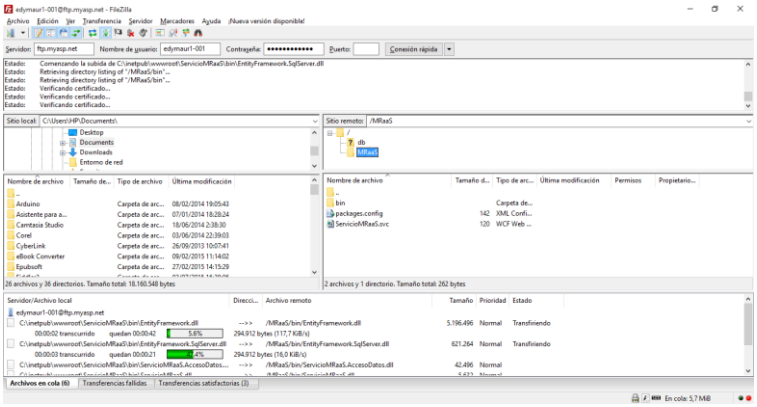


Figura 45. Descripción de archivos binarios alojándose en la nube mediante FTP.

Elaborado por: Edison Pilatasig

Servicio MRaaS en la nube

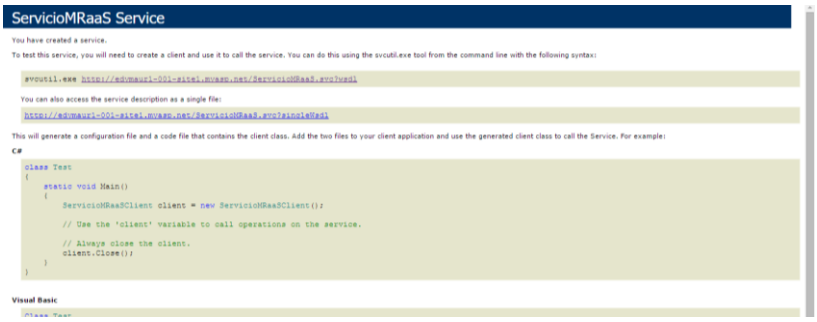


Figura 46. Descripción de archivos binarios alojados en la nube.

Elaborado por: Edison Pilatasig

A modo de resumen: el Servicio Web utiliza la tecnología WCF (Windows Communication Foundation) provista por Microsoft para logar una intercomunicación entre la estación de control y la estación de trabajo, este servicio web estará conectado a una base de datos, a través de Entity Framework para el acceso a los datos relacionales, en la que se almacenará información como: datos de estaciones de trabajo o control registradas, datos y características de movimientos por cada robot, sesiones y conexiones realizadas, y restricciones de manipulación. El modelo usado consiste en MVC (modelo – vista – controlador), consiste en la interfaz del servicio (vista), la capa que maneja los eventos o acciones de los usuarios al invocar peticiones al servicio (controlador), y las capas de acceso a datos y lógica de negocio del servicio (modelo), se mencionó que el servicio trabaja con cuatro capas y esto es debido a que se toma en cuenta una capa llamada entidades, que es en sí una representación en código de la base de datos relacional utilizada.

3.2 Cliente módulo de operador

Se define como cliente módulo de operador al sistema en donde se realizan las peticiones al servicio MRaaS para manipular al brazo robótico SCARA ubicado remotamente en el lado del cliente módulo de trabajo.

El cliente módulo de operador es una estación de control, una aplicación basada en el modelo de abstracción MVVM (Modelo – Vista –Vista Modelo), esta se desarrolló en el lenguaje de alto nivel C# bajo la plataforma WPF (Windows Presentation Foundation), la aplicación interactuará con el periférico Kinect, mediante el Framework de este dispositivo, procesará la información de movimientos, dichos

movimientos son controlados de acuerdo a las características que el servidor expone, posteriormente se envía la información al Servicio Web. El modelo MVVM es utilizado para desacoplar el código de la interfaz de usuario (vista) y el código de programa (modelo) usando intermediario (modelo-vista), es así que se define una interfaz de usuario de forma declarativa mediante XAML (Extensible Application Markup Language) usando un marcado de enlace de datos para unirse a las capas de datos o comandos del usuario.

La Estación de Control presenta al usuario un sistema compuesto por un conjunto de vistas, configuraciones y parámetros del dispositivo Kinect, se implementa la vista principal encargada de la creación, verificación y conexión de la estación con el dispositivo Kinect.

Módulo del operador

Figura 47. Bloques de programa necesarios del cliente módulo del operador.

Elaborado por: Edison Pilatasig

La figura 47 muestra el diagrama de bloques de programa implementado para la creación de la aplicación del operador, el operador es la persona física que manipula remotamente al brazo robótico SCARA, mientras que el conjunto “módulo del operador” es un sistema compuesto por bloques de programa.

3.2.1 Interfaz de autenticación

Permite al usuario iniciar sesión, descubrir módulos de trabajo conectados, iniciar conexiones, verificar objetos o extremidades manipulables, y advertencia de tiempo de uso restante, como se muestra en la figura 48 un usuario llamado “operador” autenticado por medio de contraseña (con posibilidad de usar dirección MAC de equipo) única sesión en el sistema, descubre un módulo de trabajo llamado Escara, inicia conexión con el módulo, y es ahí cuando recibe un listado de componentes disponibles para su uso, en este caso son cinco extremidades posibles, adicionalmente el sistema muestra la cantidad de tiempo disponible en el cual el operador puede manipular el sistema, esta opción se la hace con propósitos académicos con el fin de extender el funcionamiento del sistema MRaaS hacia la educación virtual. El usuario deberá dar clic en el botón “CONTINUAR” para iniciar el proceso de manipulación remota.

Interfaz de autenticación

Figura 48. Captura de pantalla de la interfaz de autenticación en el módulo del operador con sesión iniciada.

Elaborado por: Edison Pilatasig

3.2.2 Interfaz de control

La interfaz de control representa aquel medio por el cual el operador al interactuar con la aplicación logra un movimiento del brazo robótico SCARA en la estación remota, como se muestra en la figura 49 esta interfaz posee tres modos de control de movimientos: Manual, para movimientos en valores discretos establecidos al pulsar el botón “OK”. Incremento, para movimientos en valores discretos al pulsar los botones “▲, ▼”, incremento, decremento respectivamente. Automático, para realizar un movimiento conjunto de todas las extremidades en un mismo tiempo, los valores los establece el operador, el sistema toma cada valor, los valida y los envía al servicio Web en la nube. Los pasos en los que se puede variar los movimientos se configuran en el cuadro “Factor”. La interfaz muestra además el listado de extremidades manipulables con sus respectivos umbrales de movimientos, se da a la interfaz la posibilidad de incorporar un control por medio de movimientos corporales del operador usando el dispositivo Kinect, esta opción permite expandir las aplicaciones del sistema para usar posteriormente algoritmos de movimientos cada vez más complejos, y de forma obligatoria se incluye un control de retorno de video que se conecta a la cámara IP instalada en el laboratorio remoto, con el fin de lograr la retroalimentación visual propuesta para este trabajo.

Interfaz de control

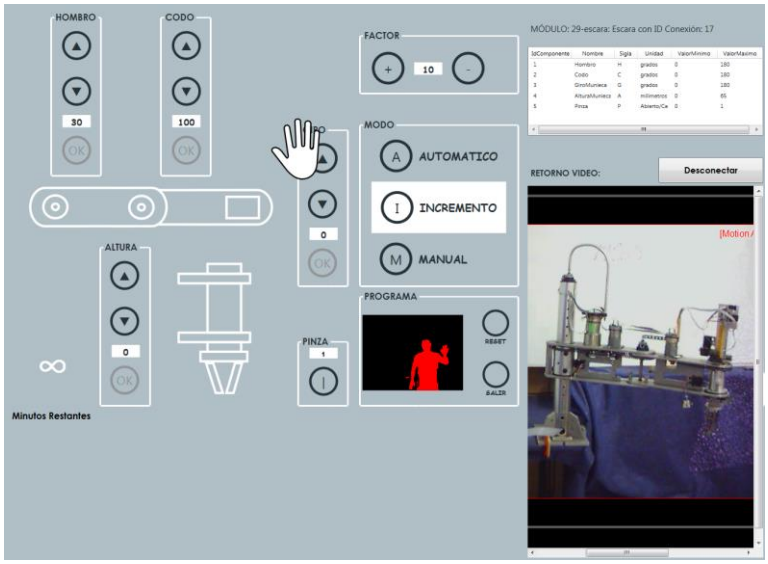


Figura 49. Captura de pantalla de la interfaz de control en el módulo del operador.

Elaborado por: Edison Pilatasig

3.2.3 Librería Kinect

La librería Kinect en la aplicación de control se basa fundamentalmente en archivos agregados en referencias como: “KinectInteraction180_32.dll”, “Microsoft.Kinect.dll”, “Kinect.Toolkit.dll”, “Kinect.Toolkit.Controls.dll” y “Kinect.Toolkit.Interaction.dll”, que contienen definiciones de código que permiten una programación sencilla en la etapa de desarrollo de la aplicación, y que permite establecer una interfaz amigable al usuario operador.

3.2.4 Librería de Conexión Servicio WEB

Se agrega una referencia al servicio web MRaaS en la aplicación de control, una forma viable y segura de consumir el servicio web, y acceder a las operaciones permitidas, como se muestra en la figura 50, posteriormente se creará un objeto que

se comporte como cliente de consumo del servicio web, dicho objetos nos permitirá consumir las operaciones del servicio web en forma de propiedades en las líneas de código de programa, de esta manera para el programador de la aplicación no se tiene que preocupar por armar manualmente el mensaje SOAP con XML, a menos que sea necesario.

Referencia de servicio

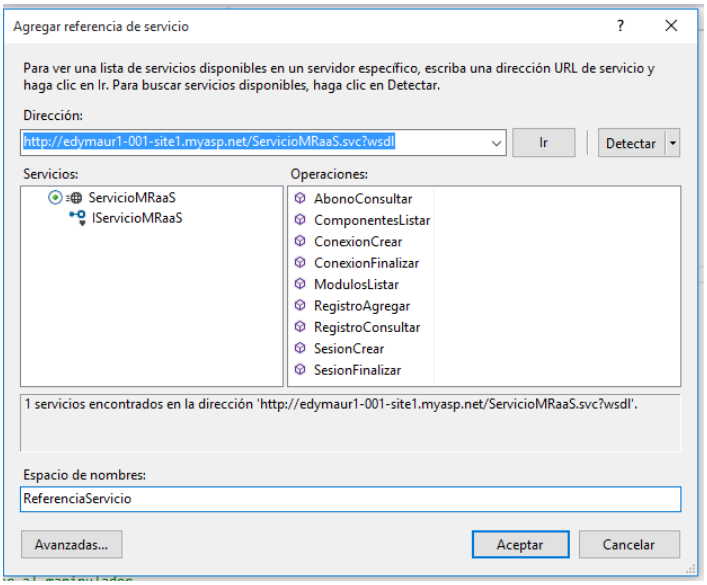


Figura 50. Creación de la referencia de servicio del servicio MRaaS en la aplicación del módulo operador.

Elaborado por: Edison Pilatasig

3.2.5 Control de movimientos

El control de movimientos en si un proceso encargado de validar la información ingresada por el operador en base a los parámetros recibidos por el servicio web MRaaS, con el objetivo de evitar enviar datos erróneos de manipulación, un ejemplo seria intentar mover el brazo fuera de los umbrales admitidos o mover una extremidad no habilitada.

3.2.6 Administrador de conexión

El administrador de conexión del sistema monitorea el tiempo de uso disponible para el operador, y también podría integrar en su futuro más capacidades para un control más inteligente que interactúe con el servicio web MRaaS o se adapte a los estados de la estación de trabajo, un ejemplo sería el fallo de alguna extremidad en el lado remoto, que haga que se bloquee inmediatamente el control gráfico en el módulo del operador.

Se muestra en la figura 51 los componentes utilizados en el módulo del operador, dichos componentes se hacen necesarios: sensor Kinect para la detección del movimiento natural humano, y el ordenador en el cual funciona la aplicación de control sobre un sistema operativo Windows 7 en adelante.

Componentes en el módulo del operador

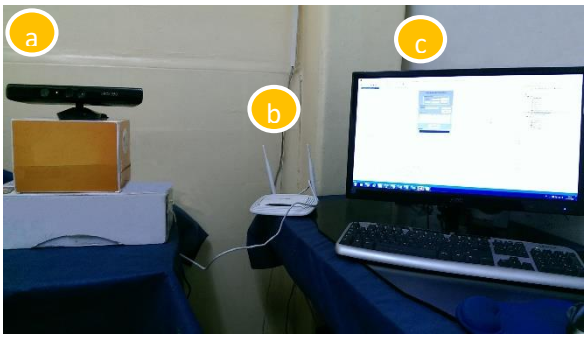


Figura 51. Fotografías de los componentes: a) Kinect, b) Punto de acceso y c) ordenador.

Elaborado por: Edison Pilatasig

3.3 Cliente módulo de trabajo

Se define como cliente módulo de trabajo al sistema en donde se realizan peticiones de consulta de nuevos movimientos para el brazo robótico SCARA con el fin de realizar la trayectoria solicitada remotamente por el cliente módulo de operador.

Módulo de trabajo

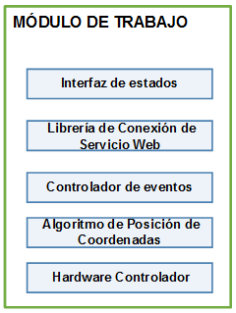


Figura 52. Bloques de programa necesarios del cliente módulo de trabajo.

Elaborado por: Edison Pilatasig

El cliente módulo de trabajo es una estación de trabajo se encarga de la comunicación serial hacia el brazo robot, y considera la posibilidad de utilizar una derivación del modelo MVVM, a fin de tener un sistema modular, pero dentro de un sistema embebido que utiliza el sistema operativo Linux en una placa Dragino Yún Shield que se comunica por el puerto serial a una placa Arduino Mega, que permita controlar el brazo robótico SCARA a través de un driver controlador, ahorre recursos de procesamiento y reduzca costos de la implementación física en el laboratorio remoto.

La figura 52 muestra el diagrama de bloques de programa implementado para la creación de la aplicación de trabajo, la aplicación actúa ante la petición de

movimiento sobre el brazo robótico SCARA, mientras que el conjunto “módulo de trabajo” es un sistema compuesto por bloques de programa.

3.3.1 Interfaz de estados

Con el fin de lograr un sistema modular en el laboratorio remoto, la interfaz de estados se implementa simplemente sobre un LCD, como se muestra en la figura 53 la interfaz muestra el estado conectado/desconectado del módulo de trabajo en el sistema MRaaS, muestra además los datos que va recibiendo de la nube a medida que el hardware controlador los va ejecutando.

Interfaz de estados

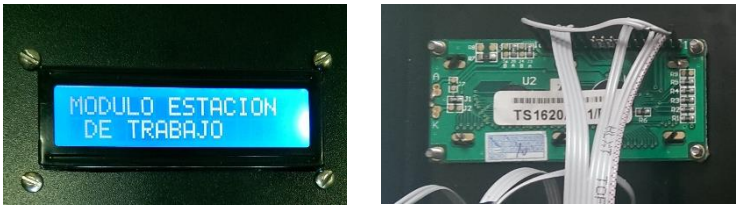


Figura 53. Interfaz de estados implementada en un LCD en el módulo de trabado.

Elaborado por: Edison Pilatasig

3.3.2 Librería de Conexión de Servicio Web

La librería de conexión de servicio web se implementa dentro de Python a través a de la librería SUDS, librería que facilita manejar las operaciones expuestas por el servicio web en forma de objetos, que simplifica el código de programa y lo vuelve más óptimo. Es necesario instalar la librería en Python a través de la consola de OpenWrt, posteriormente en la aplicación se usa la directiva “from suds.client import Client” para importar la librería al código de programa, de donde se crea un objeto

cliente usando el método “client = Client(url)”, donde url es la dirección web donde reposa la definición WSDL del servicio MRaaS, un ejemplo de implementación de este objeto cliente es: “respuesta = client.service.SesionCrear(alias,pase,dirMac)” línea de programa en la cual se consume el servicio web en la operación “SesionCrear”, enviándole atributos (alias,pase,dirMac).

3.3.3 Controlador de eventos

El controlador de eventos es el proceso descrito en la figura 34, que discrimina por software, las entradas de datos que el hardware, mostrado en la figura 54, en este caso un arreglo de pulsadores conectados a la placa Arduino Mega, que un usuario encargado del laboratorio remoto configura para conectar el módulo de trabajo con el servicio en la nube. Este controlador de eventos en el encargado de iniciar/finalizar una sesión/conexión bajo demanda, y de recibir las peticiones de movimientos del usuario operador remoto.

Dispositivos de entrada

Figura 54. Conjunto de dispositivos de entrada del módulo estación de trabajo.

Elaborado por: Edison Pilatasig

3.3.4 Algoritmo de Posición de Coordenadas

Bloque de programa en donde se procesan las peticiones de movimientos enviadas por el operador, este algoritmo de posición es aplicado en la placa Arduino, que mediante lecturas de los sensores de posición del brazo robótico SCARA puede calcular la compensación de movimiento para alcanzar una nueva posición en x,y,z del gripper o pinza del mismo. En este proceso se pueden implementar algoritmos de cinemática directa para mejorar la precisión, además de incorporar un espacio para el procesamiento avanzado de movimientos para compensación de extremidades fuera de línea (dañadas) en un futuro.

3.3.5 Hardware Controlador

El hardware controlador es el sistema conformado por dispositivos de electrónica de control, mostrado en la figura 55, en este caso son tres placas integrada con un puente H L298n, que servirán para el control de velocidad y giro de los motores para posicionamiento de las extremidades a ser manipuladas, posicionamiento que es realizado además por la lectura de sensores de posición potenciómetros y un sensor de distancia, por la razón de tener un brazo robótico SCARA re ensamblado, pero cabe notar que se pueden agregar circuitos más complejos que brinden un mejor posicionamiento (encoder o motores de paso), que no es posible ya que se tiene un módulo robótico des continuado y cuyas piezas ya no se comercializan.

Hardware controlador

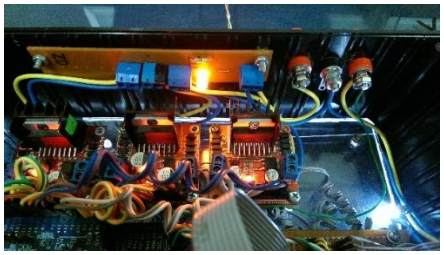


Figura 55. Hardware controlador utilizado para la manipulación del brazo robótico SCARA.

Elaborado por: Edison Pilatasig

3.3.6 Firmware Controlador

Es el software y hardware, o sistema operativo embebido dentro de la placa Dragino Yún Shield, que opera como controlador principal para realizar peticiones, interpretar las respuestas recibidas y que a través de un hardware mueve los actuadores de acuerdo al algoritmo en software programado, de tal forma el firmware sería la unión del programa en Python que funciona dentro del sistema operativo OpenWrt, que a su vez funciona y hace uso del hardware de la placa electrónica Dragino.

3.3.7 Brazo robótico Scara

Para el brazo robótico Scara luego de realizar modificaciones, como se observa en la figura 56, se tiene un motor en el eslabón A para el movimiento sobre el eje xy, llamado Hombro, un motor en el eslabón B para el movimiento sobre el xy, llamado Codo, un motor en el eslabón C para el movimiento sobre el eje z llamado AlturaPinza, un motor para rotación en el punto D, llamado GiroMunieca (ojo, no se

usa la ñ, para evitar errores de codificación de caracteres), y un motor para apertura y cierre de pinza en el punto E.

Así mismo se dispone de un potenciómetro en el punto F, para control de posición del eslabón A, un potenciómetro en el punto G, para control de posición del eslabón B, un sensor analógico tipo Sharp en el punto H, para control de posición del eslabón C, así como también sensores finales de carrera I, J para los extremos del mismo.

Se conectan los elementos del brazo robótico SCARA a través de un cable ribbon, que termina en un conector DB-20 hembra, se observa en la figura 57 la disposición de pines del mismo.

Brazo robótico Scara modificado

Figura 56. Brazo robótico Scara utilizado en el presente trabajo.

Elaborado por: Edison Pilatasig

Pines de conexión del brazo robótico Scara

Motor C +	**Motor E +**	**Motor D +**	**Motor B +**	**Motor A +**	**Sensor I**	**Sensor J**	**GND**	**Potenciómetro G**	**+5VDC**
Motor C -	**Motor E -**	**Motor D -**	**Motor B -**	**Motor A -**	**+5VDC**	**+5VDC**	**Sharp H**	**GND**	**Potenciómetro F**

Figura 57. Descripción de pines de conexión en el conector DB-20 del brazo robótico Scara.

Elaborado por: Edison Pilatasig

3.3.8 Circuito propuesto

En la figura 58 se muestra el diagrama de bloques del circuito propuesto para el módulo de trabajo, mostrando la interacción entre sub módulos. Los “Dispositivos de entrada” permiten que un administrador envié comandos hacia el “Firmware controlador”, el cual se encarga de procesar comandos, realizar peticiones al servicio web MRaaS, interpretar la respuesta obtenida, todo esto dentro de la placa Dragino Yún Shield para luego enviar datos de movimientos hacia la placa Arduino Mega, la cual despliega el proceso actual del sistema a través de la “Interfaz de estados”, se realiza el proceso de posicionamiento del brazo robótico a través del módulo “Hardware controlador” haciendo uso de las lecturas de los sensores instalados en el módulo “Brazo Robótico Scara”.

Diagrama de bloques del circuito propuesto

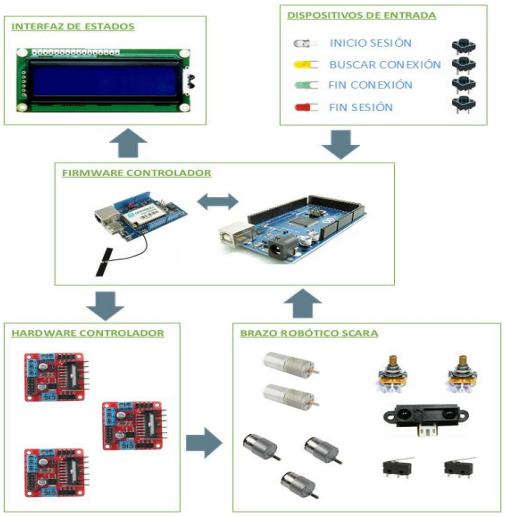


Figura 58. Diagrama de bloques del circuito para los sub módulos en el módulo de trabajo.

Elaborado por: Edison Pilatasig

En la tabla 11 se muestra la conexión de pines realizada en la estación módulo de trabajo, para cada elemento mostrado en la figura 58.

Tabla 11.

Conexión de pines

ELEMENTO		ARDUINO MEGA	BRAZO SCARA	DESCRIPCIÓN
Botón Inicio Sesión		22		Botón para iniciar sesión en el servicio MRaaS.
Botón Buscar Conexión		24		Botón para petición de movimientos ante una conexión realizada.
Botón Fin Conexión		26		Botón para finalizar una conexión en el servicio MRaaS
Botón Fin Sesión		28		Botón para finalizar una sesión en el servicio MRaaS
Led de Sesión		30		Led indicador se prende si la sesión es iniciada.
Led de Petición		32		Led indicador intermitente al recibir respuesta de movimiento.
Led de Error		34		Led indicador de errores en el sistema
LCD 2*16	RS	23		Pines del LCD a ser conectados a la placa Arduino Mega
	E	25		
	D4	27		
	D5	29		
	D6	31		
	D7	33		
L298N - 1	IN1	2		Pines de control de inversión de giro y velocidad del motor A.
	IN2	3		
	IN3	4		Pines de control de inversión de giro y velocidad del motor C.
	IN4	5		
	OUT3/4		Motor C	Conexión al motor C del brazo SCARA.
	OUT1/2		Motor A	Conexión al motor A del brazo SCARA.
L298N - 2	IN1	6		Pines de control de inversión de giro y velocidad del motor D.
	IN2	7		
	IN3	8		Pines de control de inversión de giro y velocidad del motor E.
	IN4	9		
	OUT1/2		Motor D	Conexión al motor D del brazo SCARA.
	OUT3/4		Motor E	Conexión al motor E del brazo SCARA.
L298N - 3	IN3	10		Pines de control de inversión de giro y velocidad del motor B.
	IN4	11		
	OUT3/4		Motor B	Conexión al motor B del brazo SCARA.
Sensor I	Resistencia Pull Down	37		Sensor final de carrera en la parte superior del eslabón C.
Sensor J	Resistencia Pull Down	39		Sensor final de carrera en la parte inferior del eslabón C.
Sharp H	Out	A0		Sensor análogo Sharp para el control de altura del eslabón C.
Potenciómetro F	Out	A2		Sensor análogo para el control de posición del eslabón A.
Potenciómetro G	Out	A1		Sensor análogo para el control de posición del eslabón B.
Dragino Yún Shield	SPI	SPI		Pines de control SPI para subir el sketch de programa a la placa Arduino.
	Uart	Uart (0,1)		Pines de comunicación serial entre la placa Dragino y Arduino.

Nota: Descripción de la conexión de pines realizados para el módulo de trabajo.

Elaborado por: Edison Pilatasig

El módulo de trabajo finalmente, se presenta en la figura 59, se observa que es un sistema modular e inalámbrico, que puede funcionar como parte de un sistema más grande, como lo sería un Laboratorio Remoto.

Módulo de trabajo

Figura 59. Módulo de trabajo creado: a) Interfaz de conexión con el brazo robótico Scara, b) Alimentación 5VDC (rojo), tierra (negro), 12 VDC (amarillo), c) Interfaces de conexión USB, Ethernet y Usb, d) Vista del circuito interno, e) Cámara IP y Punto de acceso.

Elaborado por: Edison Pilatasig

CAPITULO 4

PRUEBAS Y RESULTADOS

4.1 Consumo del servicio web usando SOAPUI

El servicio web puede ser accesible a cualquier hora desde cualquier lugar a través de la dirección web: http://edymaur1-001-site1.myasp.net/ServicioMRaaS.svc?wsdl, aquí el servicio se encuentra definido en formato XML a través del estándar WSDL, se procede a crear un protocolo de pruebas usando la aplicación SoapUI, se muestra en la figura 60 el proyecto SOAP creado, y en la figura 61 la configuración detectada automáticamente por SoapUI a través de la lectura del WSDL del servicio, se utiliza la versión 1.1 del protocolo SOAP, y se tienen expuestas nueve operaciones como se definió en la tabla 9.

Nuevo proyecto SOAP

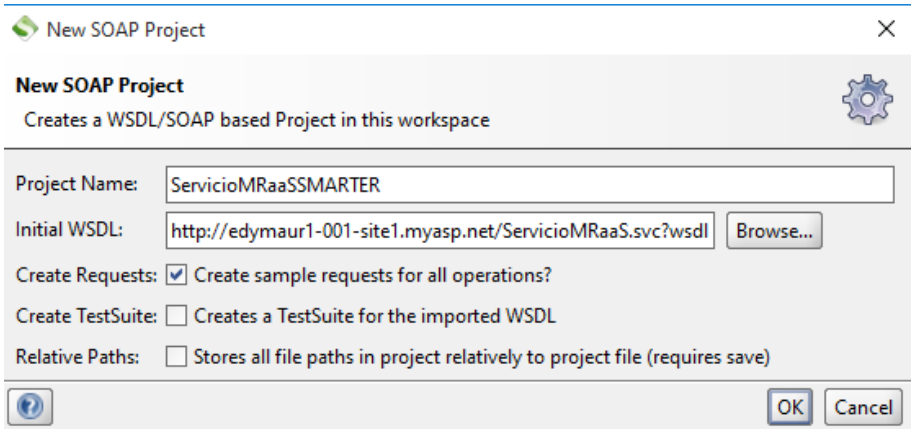


Figura 60. Creación del proyecto SOAP para consumir el servicio web MRaaS.

Elaborado por: Edison Pilatasig

El intercambio de mensajes entre clientes (módulo operador y trabajo) se hace usando el protocolo HTTP, se estableció un intercambio de mensajes en modo

petición / respuesta, es decir los clientes envían peticiones para ejecutar operaciones y el servidor procesará la petición, la ejecutará, y responderá un mensaje, estos son mensajes SOAP descritos en formato XML, que son interpretados fácilmente dada su estructura. Al lado izquierdo en la figura 61 se observan las operaciones detectadas por SoapUI,

Panorama general del servicio MRaaS en SoapUI

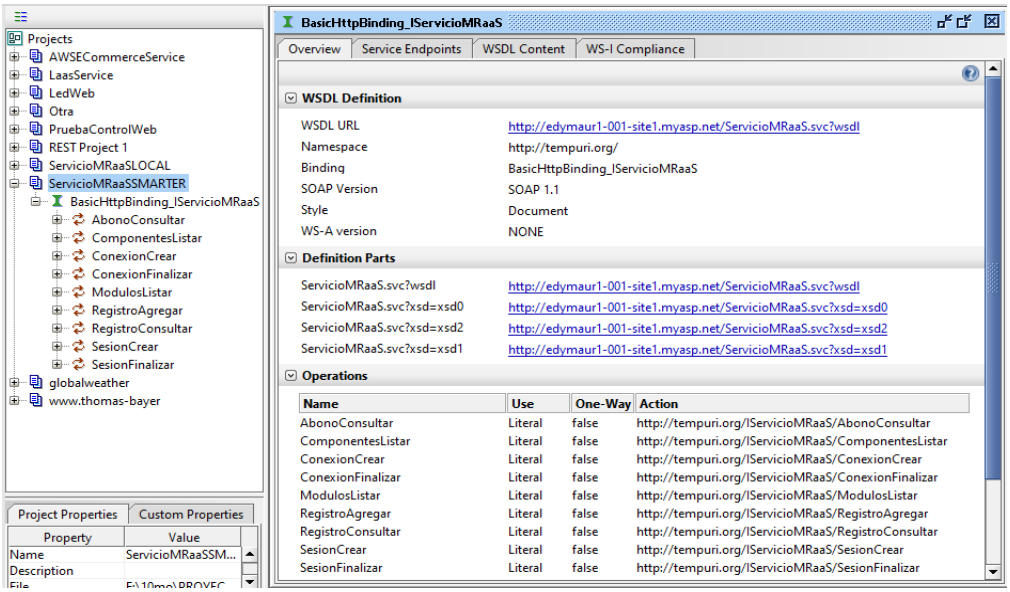


Figura 61. Descripción obtenida por SoapUI de la definición WSDL del servicio MRaaS.

Elaborado por: Edison Pilatasig

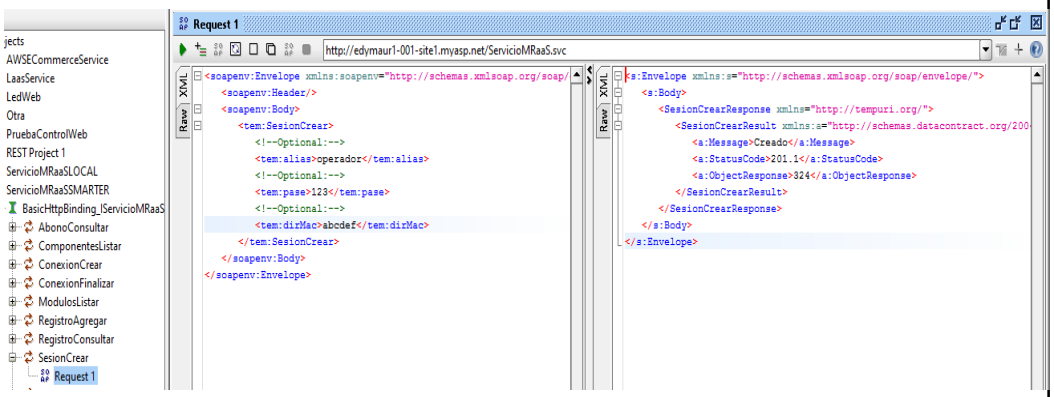
4.1.1 Sesión de clientes

El cliente en el módulo del operador, realiza una petición de inicio de sesión, usando un mensaje SOAP como se muestra en la figura 62.a, el servidor devuelve un mensaje de estado basado en la tabla 9 dependiendo del evento ocurrido, si las credenciales son correctas el sistema devuelve un código 201.1 acompañado de un token que identificará al usuario de manera única en el sistema MRaaS, de igual

manera el usuario operador finaliza la sesión tal como se muestra en la figura 62.b haciendo uso solamente del token asignado.

Sesión del operador

a.



b.

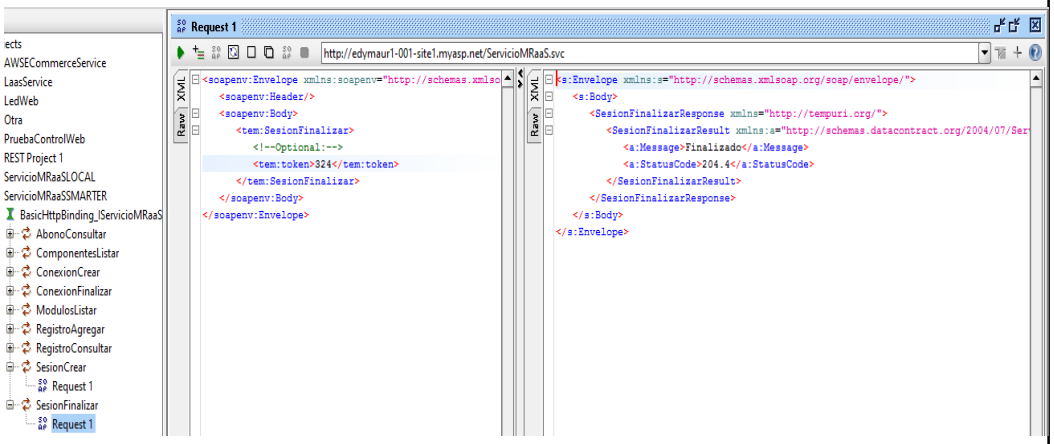


Figura 62. a. Inicio de sesión y b. Fin de sesión del módulo del operador en el servicio MRaaS.

Elaborado por: Edison Pilatasig

Al iniciar una sesión de operador se observa en la figura 63.a que se crea un registro en la base de datos en la tabla SesionUsuario, con la fecha de inicio de sesión, así mismo en la figura 63.b se observa que cuando se finaliza la sesión se llena el campo de fecha de fin de sesión.

Registros de sesión del operador

a. b.

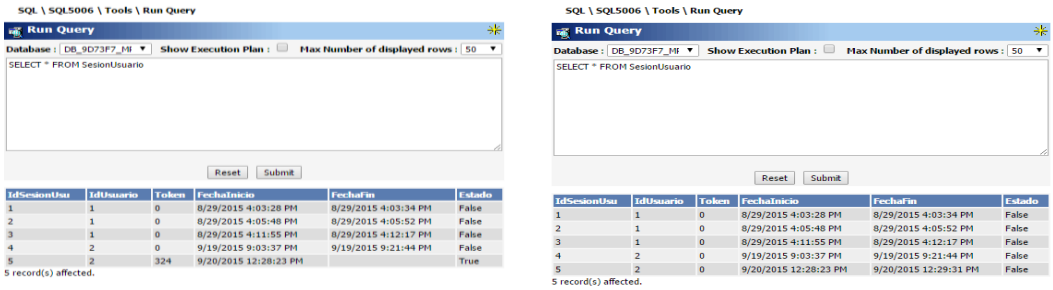


Figura 63. Consulta en lenguaje SQL en la base de datos MRaaS sobre la tabla SesionUsuario: a. Sesión iniciada y b. Sesión Finalizada.

Elaborado por: Edison Pilatasig

De igual manera para el módulo de trabajo se inicia una sesión como se observa en la figura 64.a consumiendo la misma operación en el servicio, se devuelve el token de sistema y se finaliza sesión como se muestra en la figura 64.b.

Sesión del brazo robótico SCARA

a.

b.

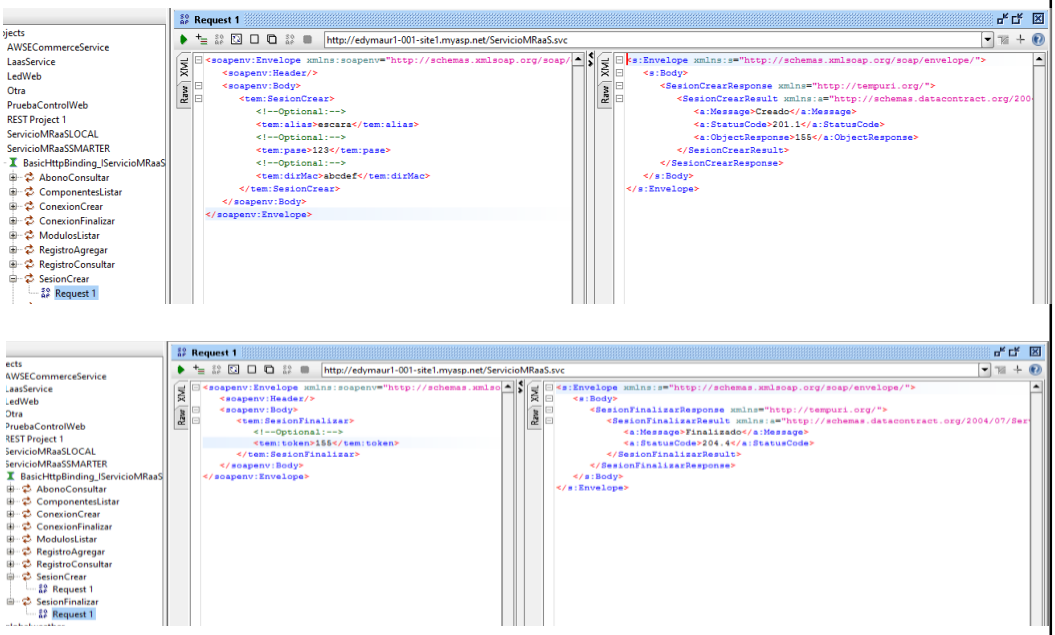


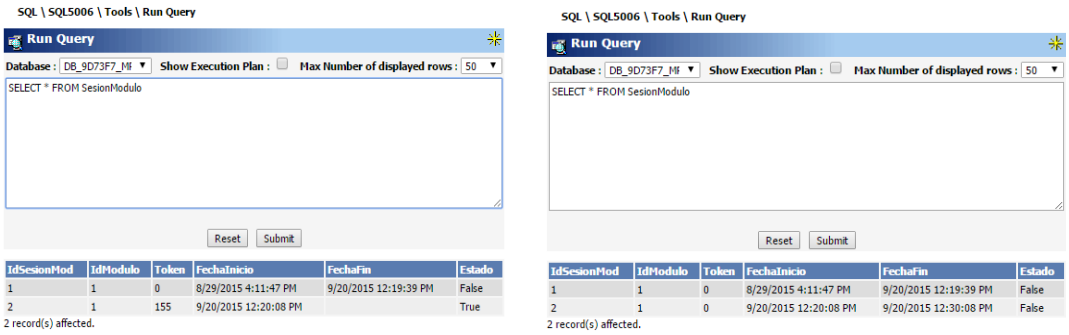
Figura 64. a. Inicio de sesión y b. Fin de sesión del módulo de trabajo en el servicio MRaaS.

Elaborado por: Edison Pilatasig

Al finalizar una sesión el sistema MRaaS verificará si se tiene una conexión activa para proceder a eliminarla primero y luego cerrar la sesión.

Así mismo se crea un registro en la base de datos con fecha de inicio de sesión, pero ahora en la tabla SesionModulo, como se observa en la figura 65.a, mientras que al finalizar una sesión se completa el campo de fecha fin de sesión como se muestra en la figura 65.b.

Registros de sesión del brazo SCARA



a. b.

Figura 65. Consulta en lenguaje SQL en la base de datos MRaaS sobre la tabla SesionModulo: a. Sesión iniciada y b. Sesión Finalizada.

Elaborado por: Edison Pilatasig

Si un cliente (modulo operador o módulo de trabajo), intenta iniciar sesión por segunda vez, sin antes haber finalizado la primera, el sistema simplemente rechaza esta petición y devuelve un código de estado 403.2.

Acción no permitida de inicio de sesión

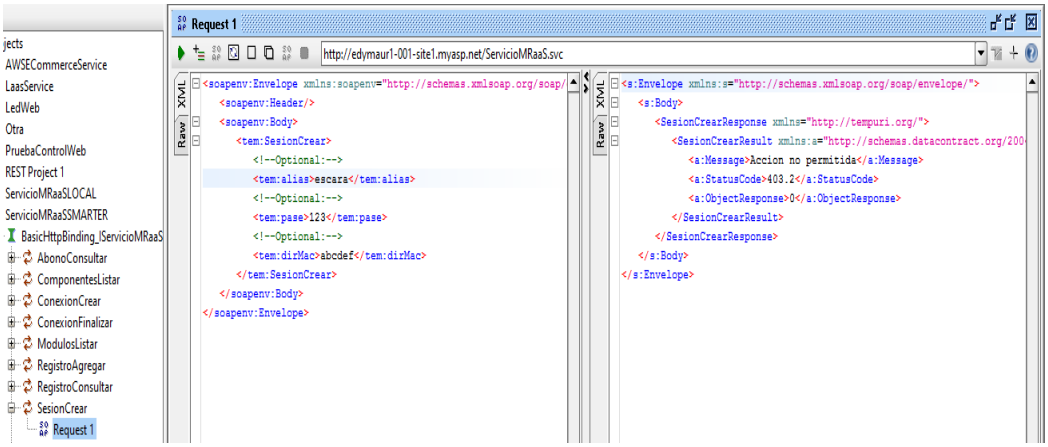


Figura 66. Código de estado devuelto ante el intento de inicio de sesión doble por parte de los clientes.

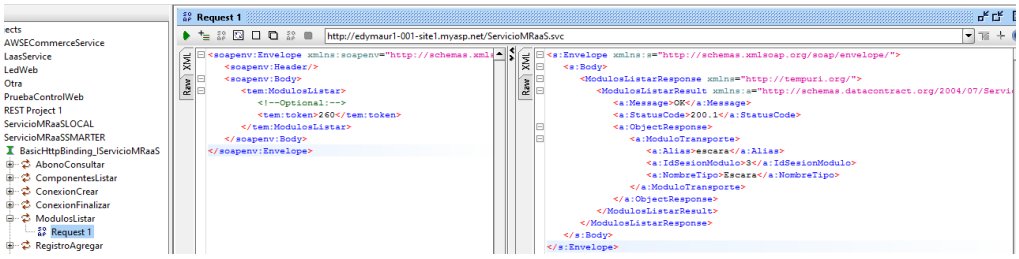
Elaborado por: Edison Pilatasig

4.1.2 Descubrimiento de módulos de trabajo

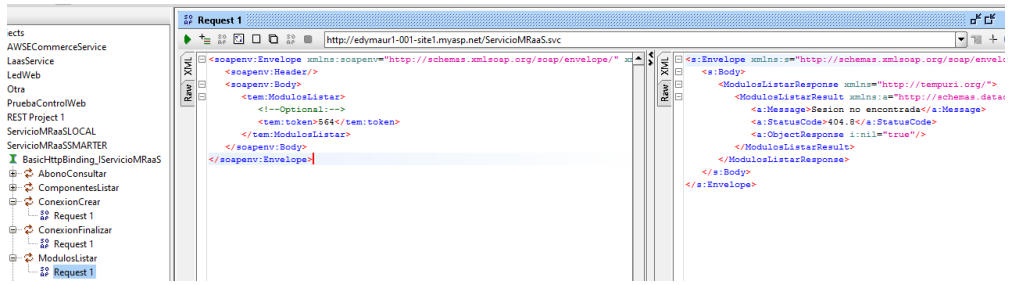
El cliente en el módulo del operador, realiza una petición de descubrimiento para obtener los módulos de trabajo actualmente conectados al sistema MRaaS, usando un mensaje SOAP sobre la operación “ModulosListar” como se muestra en la figura 67.a, el servidor devuelve un mensaje de estado basado en la tabla 9 dependiendo del evento ocurrido y un listado de módulos conectados (Alias, Id de sesión y nombre del tipo de robot), de no tener una sesión iniciada en el sistema MRaaS se devolverá un mensaje de error tal como se muestra en la figura 67.b, es decir no se puede consultar los módulos sin antes iniciar una sesión, un módulo de trabajo con sesión iniciada si puede consultar los módulo de trabajo vecinos conectados como se muestra en la figura 67c, se hace necesario solo hacer uso del token de usuario operador asignado, las estaciones de trabajo no podrán acceder a esta operación.

Descubrir módulos de trabajo

a.



b.



c.

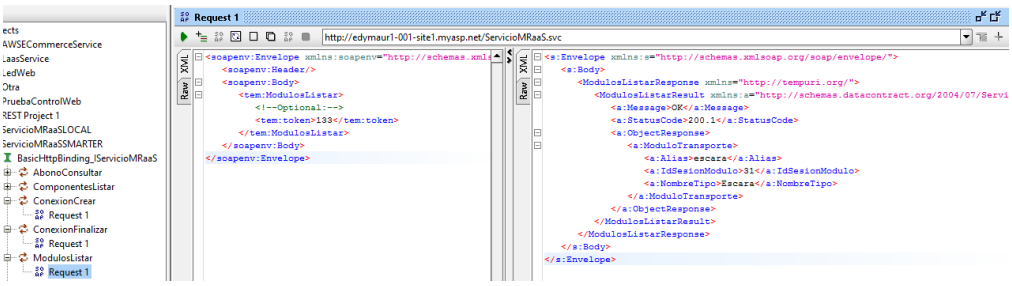


Figura 67. Petición de consulta de descubrimiento de módulos de trabajo actualmente conectados, a) y c) con sesión iniciada, b) con sesión no iniciada.

Elaborado por: Edison Pilatasig

4.1.3 Conexión con el módulo de trabajo

Habiéndose elegido el módulo a conectar, el cliente en el módulo del operador, realiza una petición de conexión, usando un mensaje SOAP sobre la operación "ConexionCrear" como se muestra en la figura 68.a, el servidor devuelve un mensaje de estado basado en la tabla 9 dependiendo del evento ocurrido y de ser válida la operación se devolverá una ID de conexión creada, en caso de intentos dobles de sesión se devolverá un mensaje de error como se muestra en la figura 68.b, el usuario no puede iniciar una conexión si no ha iniciado antes una sesión tal como se muestra

en la figura 68.c, no se puede intentar iniciar sesión con un módulo de trabajo no conectado tal como se muestra en la figura 68.d, un módulo de trabajo no puede iniciar una conexión como se muestra en la figura 68.e, se hace necesario hacer uso del token de usuario operador asignado junto con el ID de sesión del módulo de trabajo, las estaciones de trabajo no podrán acceder a esta operación.

Conexión con el módulo de trabajo

a.

b.

c.

d.

e.

Figura 68. Petición de nueva conexión con el módulo de trabajo deseado, a) Petición correcta, b) Petición negada.

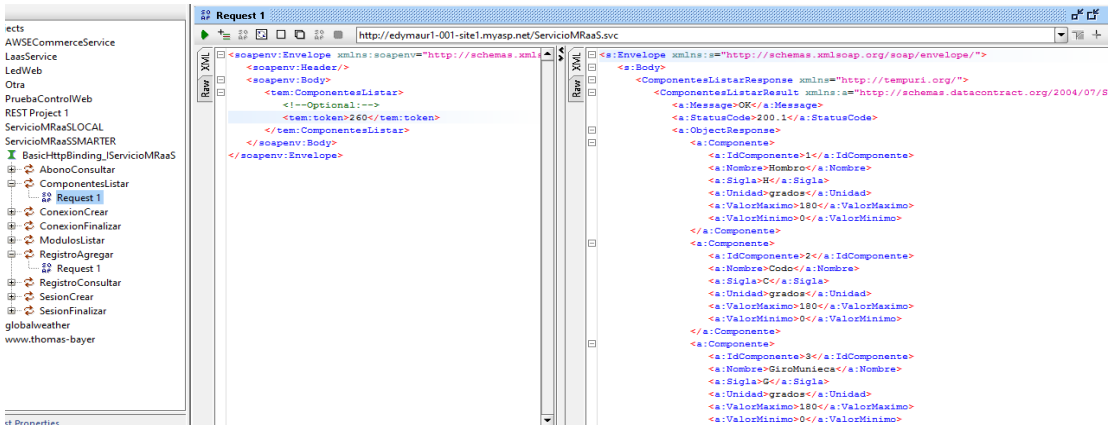
Elaborado por: Edison Pilatasig

4.1.4 Consulta de componentes manipulables

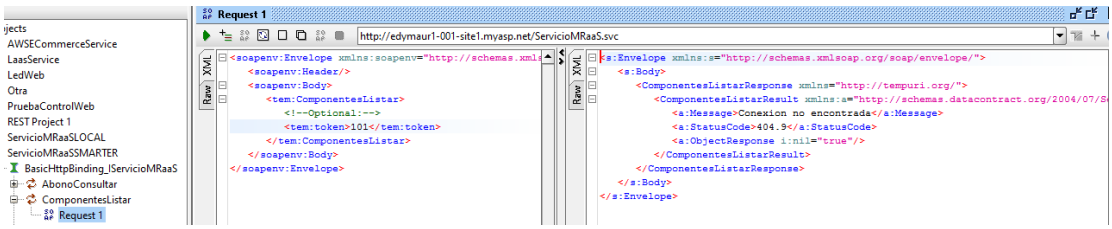
Para poder discriminar entre diferentes brazos robóticos de varias características que coexisten sobre la misma plataforma MRaaS, el cliente en el módulo del operador, realiza una petición de consulta de componentes aceptables para su manipulación, usando un mensaje SOAP sobre la operación “ComponentesListar” como se muestra en la figura 69.a, el servidor devuelve un mensaje de estado basado en la tabla 9 tras un evento satisfactorio ante una conexión previamente iniciada y un listado de componentes admitidos junto con sus características (Id, nombre, sigla, unidad de medida, rangos de movimientos), no se puede listar los componentes si no se tiene una conexión activa (figura 69.b) y mucho menos sin una sesión activa (figura 69.c), se hace necesario hacer uso solamente del token de usuario operador asignado , las estaciones de trabajo no podrán acceder a esta operación.

Consulta de componentes manipulables

a.



b.



c.

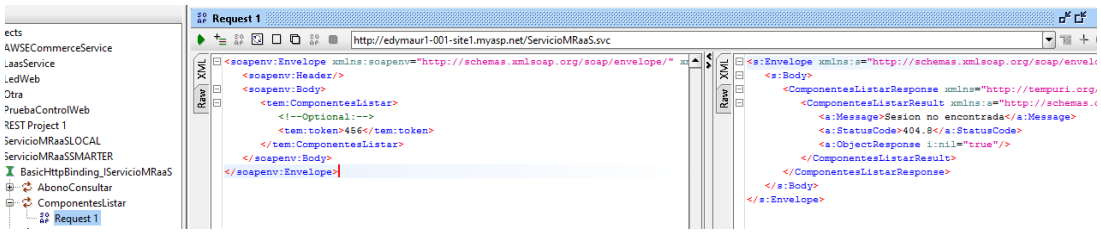


Figura 69. Petición de consulta de componentes manipulables y sus características, a) Con conexión iniciada, b) Con conexión no iniciada, c) Con sesión no iniciada.

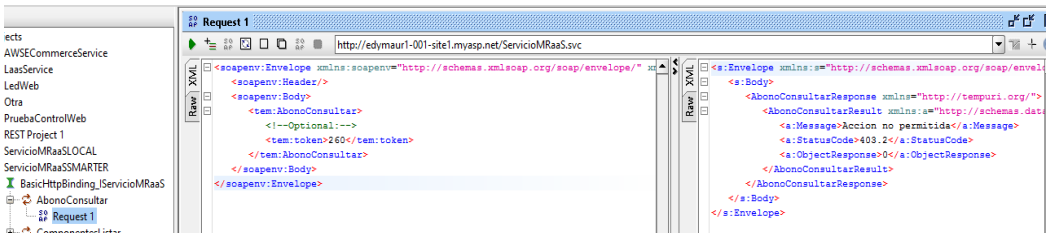
Elaborado por: Edison Pilatasig

4.1.5 Consulta de tiempo de uso

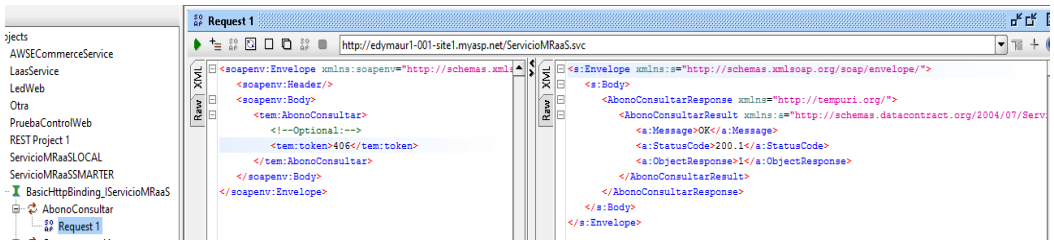
La política establecida de asignar un tiempo de uso máximo del sistema, es válida solo para usuarios comunes (véase Figura 70a) pero sin tiempo de uso para usuarios administradores (véase Figura 70b), en ambos casos el cliente en el módulo del operador, realiza una petición de consulta de tiempo de uso, usando un mensaje SOAP sobre la operación “AbonoCosultar”, el servidor devuelve un mensaje de estado basado en la tabla 9 dependiendo del evento ocurrido y una respuesta indicando el tiempo de uso restante si así fuera el caso, no se puede consultar el tiempo de uso sin una sesión iniciada (figura 70c), se hace necesario hacer uso solamente del token de usuario operador asignado, las estaciones de trabajo no podrán acceder a esta operación.

Consulta de tiempo de uso

a.



b.



c.

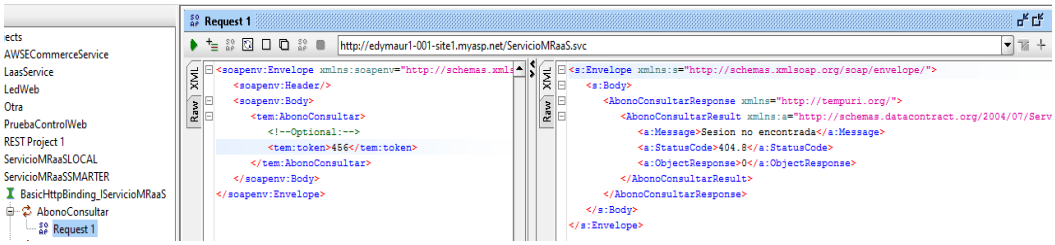


Figura 70. Petición de consulta del tiempo de uso restante, a) para usuarios administradores no aplica, b) para usuarios comunes se detalla el valor en minutos y c) sesión no iniciada.

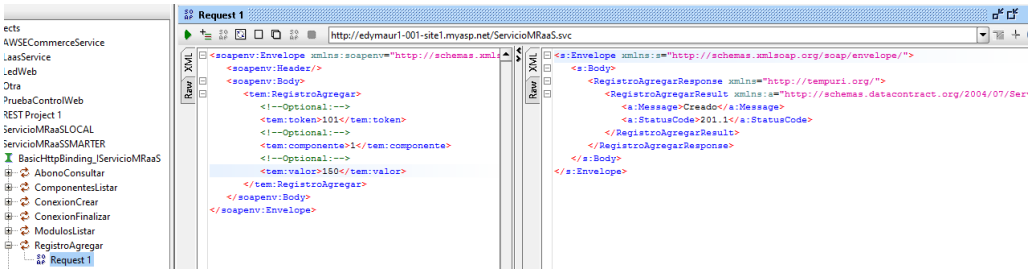
Elaborado por: Edison Pilatasig

4.1.6 Inserción de registros de movimientos

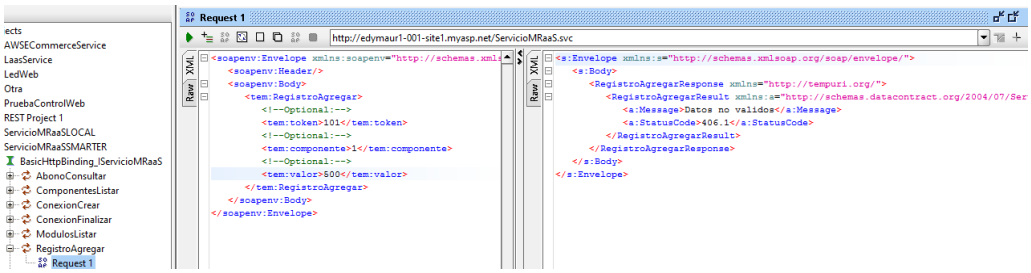
El cliente en el módulo del operador, realiza una petición de inserción de nuevos movimientos, usando un mensaje SOAP sobre la operación “RegistroAgregar”, el servidor devuelve un mensaje de estado basado en la tabla 9 dependiendo si el evento ocurrido fue exitoso y se agregó el registro como se muestra en la figura 71.a, no se pueden agregar datos con valores fuera del rango admitido como se muestra en la figura 71.b, no se puede insertar registros sin una conexión activa (71.d) y mucho menos sin una sesión activa (figura 71.c), un cliente módulo de trabajo no puede insertar registros de movimientos (figura 71.e), se hace necesario hacer uso del token de usuario operador asignado, Id de componente a manipular, y el valor de posición deseado.

Inserción de nuevo registro de movimiento

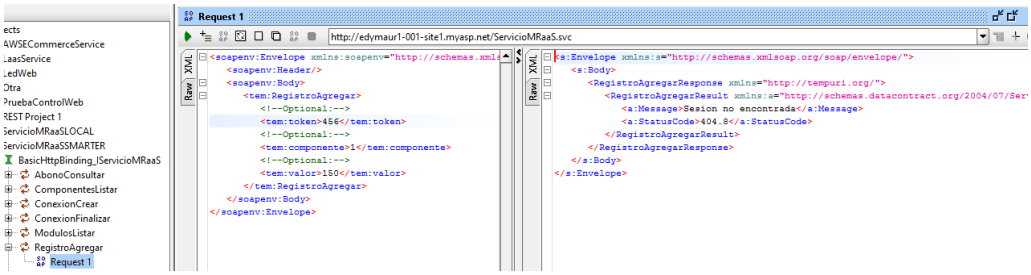
a.



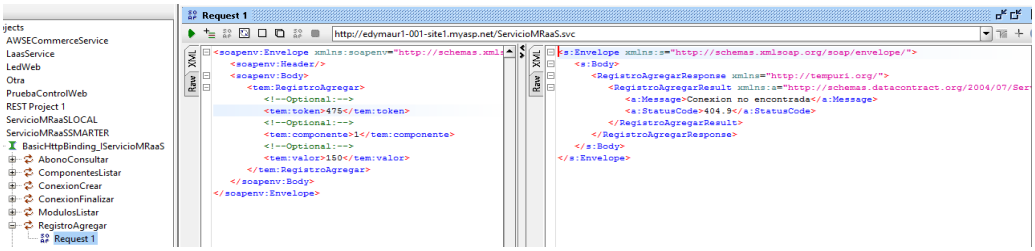
b.



c.



d.



e.

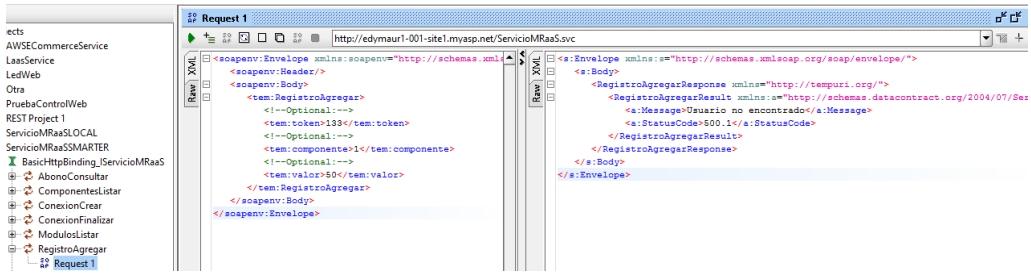


Figura 71. Inserción de nuevo movimiento para su ejecución en el módulo de trabajo, a) Con conexión activa y valores dentro del rango, b) Con conexión activa y valores fuera del rango, c) Con sesión no iniciada, d) Con conexión no iniciada y e) No válido para módulos de trabajo.

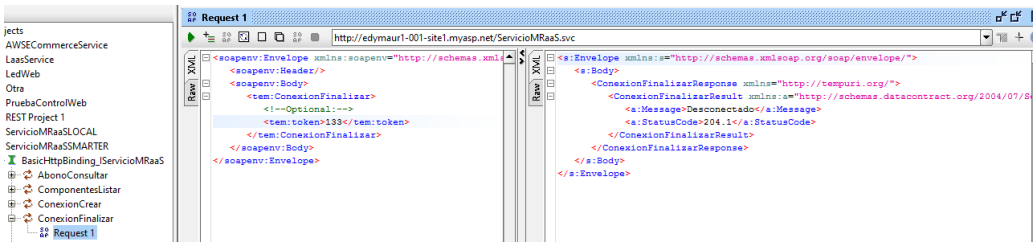
Elaborado por: Edison Pilatasig

4.1.7 Fin de conexión

El cliente en el módulo del operador o módulo de trabajo, realiza una petición de fin de conexión, usando un mensaje SOAP sobre la operación “ConexionFinalizar”, el servidor devuelve un mensaje de estado basado en la tabla 9 dependiendo si el evento ocurrido fue exitoso o no, ambos clientes pueden finalizar su conexión establecida como se muestra en la figura 72.a, no se puede finalizar la conexión sin una conexión activa (72.b) y mucho menos sin una sesión activa (figura 72.c), se hace necesario hacer uso solamente del token de usuario operador o trabajo asignado.

Fin de conexión

a.



b.

c.

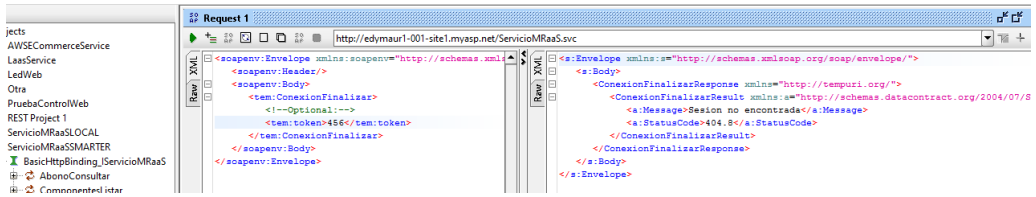


Figura 72. Fin de conexión, a) Con conexión activa, b) Con Conexión no iniciada, y c) Con sesión no iniciada.

Elaborado por: Edison Pilatasig

4.2 Manipulación remota

Una vez comprobado manualmente el funcionamiento el sistema usando SoapUI, se procede a probar el sistema MRaaS a través de la aplicación de consumo de escritorio en WPF en el módulo del operador y usando una aplicación de consumo en Python en el módulo de trabajo.

El módulo de trabajo necesita estar conectado a la red local (que a su vez tiene salida a la red pública de internet), como se detalló en la figura 59 y como se muestra en la figura 73 si bien la IP se asignó dinámicamente a través de DHCP se debe tener en cuenta de reservar esta dirección en modo permanente en el Access Point de la red local, con el fin de tener la misma IP siempre para un módulo específico (o asignarla manualmente en el Dragino modificando el archivo ifcfg-eth0).

Asignación de IP dinámica en la red local

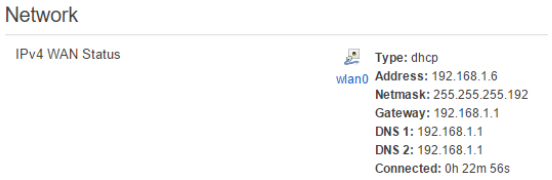


Figura 73. Asignación de IP dinámica en la red local visto desde el panel web gráfico LUCI integrado en el módulo Dragino.

Elaborado por: Edison Pilatasig

Con el fin de orientar el sistema hacia aplicaciones educativas se asume que el modulo estará siempre conectado a la red eléctrica, pero que la aplicación de consumo en Python en OpenWrt se ejecutará todos los días a las nueve de la mañana y se cerrará todos los días a las cuatro de la tarde (dependiendo del criterio de

funcionamiento), esto se logra programando actividades en crontab de OpenWrt como se muestra en la figura 74, se puede realizar este procedimiento utilizando el panel LUCI o directamente en la consola de OpenWrt (conexión SSH y ejecutando crontab -e).

Configuración de crontab

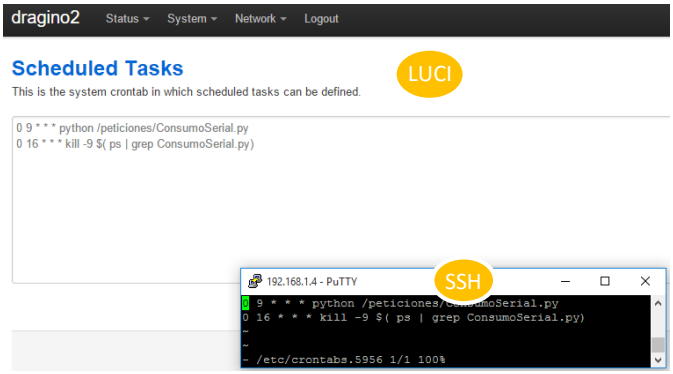


Figura 74. Fichero crontab configurado para ejecutar actividades programadas.

Elaborado por: Edison Pilatasig

Se puede iniciar el proceso manualmente como se indica en la figura 75, el archivo con el programa en Python se encuentra en el directorio /peticiones y tiene por nombre ConsumoSerial.py.

Inicio manual del proceso

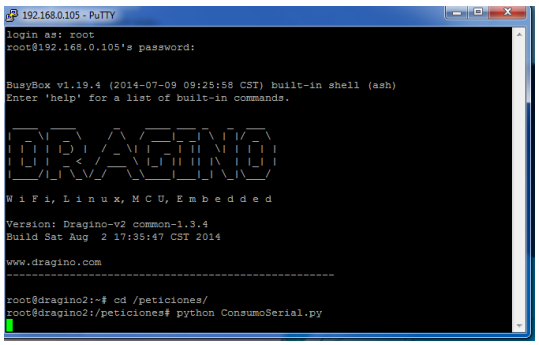


Figura 75. Inicio manual del proceso Python en el módulo de trabajo.

Elaborado por: Edison Pilatasig

En los dispositivos de entrada (figura 76) para comandos en el módulo de trabajo se encuentra: botón de inicio de sesión (76a), botón de fin de sesión (76b), botón de búsqueda de movimientos (76c), botón de fin de conexión (76d), led de estado de sesión (76e), led de estado de petición de movimientos (76f), led de estado de error (76g) y el LCD para mostrar mensajes y estados del sistema (76h).

Presentación del módulo de trabajo

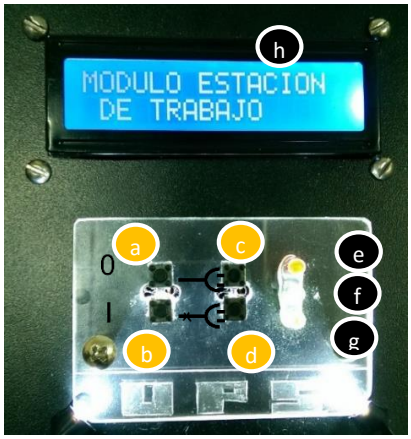


Figura 76. Fotografía de la presentación del módulo de trabajo.

Elaborado por: Edison Pilatasig

Al iniciar sesión el servicio MRaaS devolverá al módulo de trabajo el token correspondiente mostrado en la figura 77a, se enciende el led correspondiente, con el cual al pulsar el botón de búsqueda de movimientos y ante una conexión existente se puede recibir datos como se muestra la figura 77b, se puede hacer que el led de estado de petición de movimientos parpadee ante la recepción de nuevos datos, el led de error se encenderá solo si ocurren eventos como: no se recibe un token, no se puede obtener movimientos de una conexión, o ante el intento de fin de conexión sin tener una sesión iniciada. Se muestra en el LCD en la figura 77b en la parte inferior la recepción RX desde el MRaaS del código de movimiento y su interpretación en la parte superior por parte del programa en Python.

Estados en el sistema módulo de trabajo

a. b.

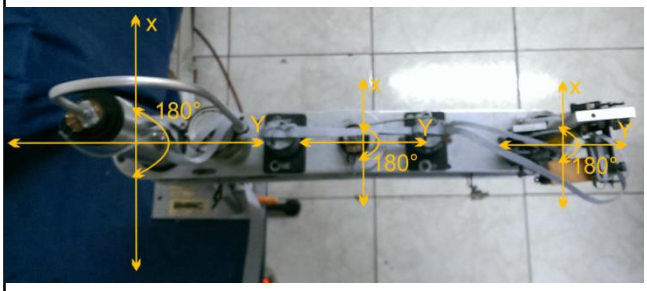
Figura 77. Estados del sistema módulo de trabajo, a) al obtener el token, y b) al recibir un movimiento.

Elaborado por: Edison Pilatasig

Los rangos de movimiento de cada extremidad se definen en la tabla *componente* de la base de datos, dicha tabla es relacionada con cada módulo existente, para el presente trabajo el brazo robótico SCARA presenta un rango de movimientos mostrado en la figura 78.a que va de 0 a 180 grados para las extremidades hombro, codo y giro de muñeca, los tres sobre el eje XY, mientras que como se muestra en la figura 78b el rango de movimiento de la altura de muñeca va de 0 a 65 milímetros en el eje Z.

Configuración de movimiento del brazo robótico SCARA

a. b.



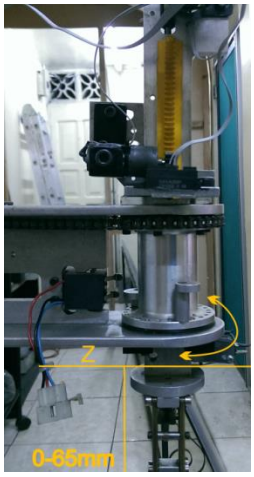


Figura 78. Ejes y rangos máximos y mínimos, del brazo robótico SCARA utilizado.

Elaborado por: Edison Pilatasig

Con el fin de realizar un control por autonomía supervisada se hace necesario agregar la cámara IP, en el mismo lugar donde se encuentra el módulo de trabajo, con el fin de tener un retorno de video en tiempo real del funcionamiento del sistema mientras se manipula remotamente.

Se procede a conectar los componentes en el módulo del operador como se mostró en la figura 58, se ejecuta la aplicación de control en donde, la primera pantalla en ejecutarse es la pantalla de autenticación (figura 79a), aquí el usuario podrá autenticarse en el sistema MRaaS para iniciar sesión ya sea como usuario administrador u operador (puede autenticarse usando contraseña o usando la dirección MAC leída automáticamente de su equipo), posteriormente se actualiza el listado de componentes conectados sabiendo que pueden haber varios, se procede a iniciar una conexión que siendo realizada desplegara el listado de características de componentes disponibles así como también el rango de movimiento de cada uno, se dispone de una pequeña consola en la parte inferior para visualizar los mensajes en tiempo real del sistema de las acciones que se vayan solicitando, lo siguiente es continuar hacia la interfaz de control.

En la interfaz de control (figura 79b) se puede observar los controles de movimiento de los componentes admitidos, tres modos de uso del sistema: en el modo manual el usuario deberá presionar el botón OK de cada componente para enviar datos al servicio MRaaS, mientras que en el modo de incrementos el usuario podrá enviar datos de movimientos instantáneamente al presionar los botones de comando de arriba y/o abajo o cierre de pinza, finalmente el modo automático permitirá al usuario

configurar todos los valores de movimiento deseados para enviar simultáneamente estos datos al servicio MRaaS.

Se dispone además de un control del factor de incremento o decremento que permitirá variar a diferentes saltos las cantidades solicitadas de movimientos, se puede además ingresar manualmente el valor de movimiento. Cabe destacar que todos estos controles permiten ser manejados a través del sensor Kinect mediante la detección del movimiento corporal humano detectado y presentado en la parte inferior media de la pantalla de control (en color rojo la silueta de la persona).

Se muestra además en la parte superior derecha una pequeña consola para mostrar mensajes de estados del sistema y el listado de componentes permitidos para su manipulación, mientras que en la parte inferior derecha se muestra el retorno de video de la cámara remota que se encuentra en el mismo laboratorio del brazo robótico SCARA, con el fin de completar el lazo de trabajo mediante autonomía supervisada que permite al usuario visualizar los movimientos a medida que los vaya realizando.

Aplicación de control y autenticación

a.

b.

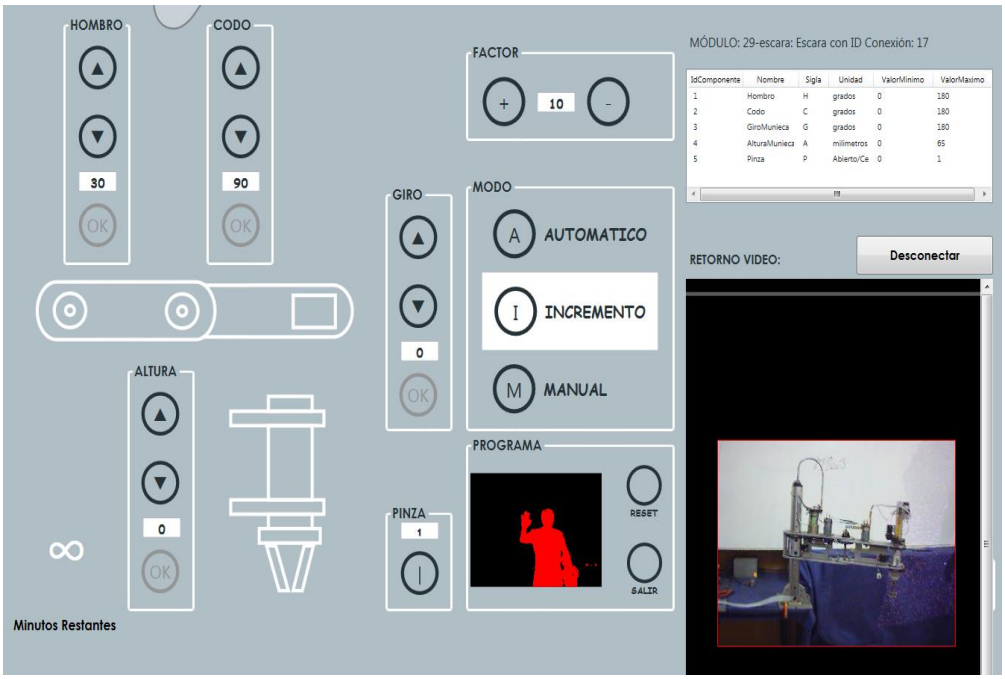


Figura 79. Funcionamiento de la aplicación de control y autenticación del módulo del operador.

Elaborado por: Edison Pilatasig

Los registros de todas las operaciones invocadas al servicio MRaaS, se van almacenando en la base de datos en su correspondiente tabla y con la lógica que

maneja el servicio MRaaS, como se observa en la figura 80, se puede acceder a estos datos a través de comandos en lenguaje SQL sobre la base datos mencionada.

Registros en la base de datos

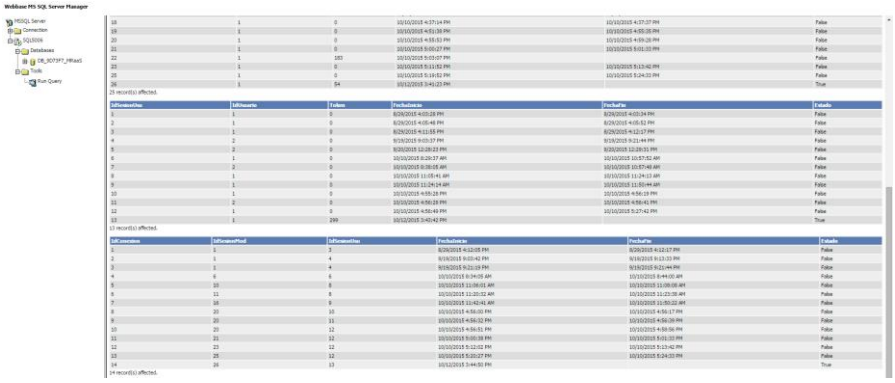


Figura 80. Captura de pantalla de algunos de los registros agregados en la base de datos al momento de realizar pruebas del sistema.

Elaborado por: Edison Pilatasig

El servicio MRaaS se encuentra publicado hasta la presente fecha en la dirección web:

http://edymaur1-001-site1.myasp.net/ServicioMRaaS.svc

Mientras que a la plataforma donde se aloja la base de datos se encuentra en:

https://mssql.myasp.net/default.asp

El código de programa de la aplicación del módulo del operador se encuentra detallado en el Anexo 1 del presente trabajo, mientras que el código fuente se encuentra almacenado digitalmente en el CD entregado en la carpeta /Binarios/AplicacionOperador. El código de programa del servicio MRaaS se encuentra detallado en el Anexo 2 del presente trabajo, mientras que el código fuente se encuentra almacenado digitalmente en el CD entregado en la carpeta /Binarios/ ProyectoMRaaS. El código de programa de la aplicación en Python y Arduino del módulo de trabajo se encuentra detallado en el Anexo 3 del presente trabajo,

mientras que el código fuente se encuentra almacenado digitalmente en el CD entregado en la carpeta /Binarios/ AplicacionTrabajo.

CONCLUSIONES

Al implementar el sistema de teleoperación se generó una unidad tipo MRaaS, o manipulador robótico como servicio en la nube, se logró interpretar los comandos de movimientos ingresados mediante el sensor Kinect en el módulo del operador, estos comandos de movimientos fueron enviados al servicio MRaaS en la nube, servicio del cual el módulo de trabajo pudo obtener los comandos de movimientos, interpretarlos, realizar lecturas de sensores y enviar señales a los actuadores para lograr el movimiento del brazo robótico SCARA, este manipulador robótico puede ser accesible desde cualquier lugar y en cualquier momento por medio de un servicio web instalado en la nube.

Se creó interfaces de control amigables con el usuario, se espera que una vez entendido el concepto y funcionamiento del sistema, se pueda ampliar las capacidades de manera notoria tanto a interfaces gráficas como a algoritmos de funciones agregadas que puedan fusionarse sobre la plataforma del servicio MRaaS almacenado en la nube.

Una de las necesidades relacionadas al campo de la educación es la demanda, por tal motivo se dotó al sistema MRaaS la capacidad de soportar múltiples módulos robóticos funcionando simultáneamente con diferentes operadores separados geográficamente, a través de un control de usuarios basado en autenticación, jerarquía, y descripción de características de trabajo de los módulos individuales. Esto se logró a través de la programación en código C# en WPF Y WCF, junto con un correcto diseño de una base de datos relacional en MSSQL y la implementación

de sistemas embebidos de bajo coste que se conectan al MRaaS, con el fin de reducir costos en equipos de computación grandes y sobredimensionados.

El sistema MRaaS implementado, junto con la evolución de la educación y el aumento de las necesidades de conocimiento de nuestra sociedad, se orientó al campo de la educación al abrir las puertas a conjuntos de sistemas modulares que conformen un Laboratorio como Servicio o LaaS, se fomentó la investigación en los campos de las telecomunicaciones y robótica de control ya que al implementar esta aplicación en la nube se pueden generar modelos, algoritmos y protocolos desarrollándose de esta manera un banco de pruebas diverso para un vehículo virtual, considerado como un manipulador robótico.

El internet es un sistema dinámico con retardos entrantes y salientes y flujos de realimentación, lo que significa que la estabilidad del sistema en cuanto a retardos no pudo ser garantizada. Se espera que se proporcione una base para entender el enfoque analítico de los sistemas de control en red.

Los sistemas modulares dotan de flexibilidad, y reducción de costos al sistema MRaaS, se optó por usar un sistema operativo embebido basado en Linux que se conecta a la red interna del bucle local de control del módulo o estación de trabajo, con esto se da paso a la posibilidad de implementación de un laboratorio que puede ser alcanzado remotamente.

RECOMENDACIONES

En base al sistema implementado se hace viable la necesidad de plantear el desarrollo de una aplicación exclusiva dedicada a la administración del sistema MRaaS, aplicación que tenga como objetivo tareas como: gestión de clientes (operadores y de trabajo), gestión de sesiones, gestión de conexiones, módulos de estadísticas de conexiones y porcentajes de movimientos ejecutados con éxito.

El sistema demanda además de la implementación de seguridades del mismo, seguridades basadas en Web Services Enhancements – WSE, así como también la construcción de un sistema centralizado de control de estados del sistema, capaz de detectar e intentar corregir ciertos comportamientos inesperados que pudieran presentarse.

Al tomar como punto la orientación del sistema hacia la educación, demandaría además de una aplicación dedicada a recolectar peticiones de usuarios, realizar encuestas y brindar ayudas específicas de los módulos, pudiendo ser alojada en la nube.

LISTA DE REFERENCIAS

Guzmán Luna, J., Torres Pardo, I., & Galeano-Hincapié, P. (2014). Propuesta de un Modelo de Gestión de Servicios en la Nube para la Manipulación de Sistemas Robóticos con el uso de Dispositivos Móviles. *Investigación, Desarrollo e Innovación Tecnológica*, 9.

¿Qué es XAML? (1 de Enero de 2015). Recuperado el 2 de Febrero de 2015, de https://msdn.microsoft.com/es-es/library/cc295302.aspx

¿Qué es XAML? (05 de 02 de 2015). Obtenido de https://msdn.microsoft.com/es-es/library/cc295302.aspx

Abhijit, J. (2012). *Kinect for Windows SDK Programming Guide.* Birmingham: Packt Publishing Ltd.

ADO.NET. (23 de 12 de 2014). Obtenido de https://msdn.microsoft.com/es-es/library/e80y5yhx(v=vs.110).aspx

ADO.NET. (1 de Enero de 2015). Recuperado el 29 de Enero de 2015, de https://msdn.microsoft.com/es-es/library/e80y5yhx(v=vs.110).aspx

Arduino. (1 de Enero de 2015). Recuperado el 20 de Noviembre de 2014, de https://www.arduino.cc/

Belqasmi, F., Glitho, R., & Chunyan, F. (5 de Diciembre de 2011). RESTful web services for service provisioning in next-generation networks: a survey. *Communications Magazine, IEEE*, 8. doi:10.1109/MCOM.2011.6094008

Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Frystyk, H., . . . Winer, D. (8 de Mayo de 2000). *Simple Object Access Protocol (SOAP) 1.1*. Obtenido de The World Wide Web Consortium (W3C): http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

C30 SD PTZ IP Camera . (1 de Enero de 2015). Recuperado el 24 de Diciembre de 2014, de Tenda Technology Co.,Ltd. : http://www.tenda.cn/uk/product/C30.html

Caminero, A., Robles-Gomez, A., Ros, S., & Tobarra, L. (3 de Abril de 2014). Deconstructing remote laboratories to create Laboratories as a Service (LaaS). *Global Engineering Education Conference (EDUCON)*, 7. doi:10.1109/EDUCON.2014.6826158

CASTRO MANTILLA, S. G., & JIMÉNEZ ALMAGRO, F. E. (2014). *Diseño, desarrollo e implementación de un sistema que permita controlar un brazo robótico mitsubishi rv-2aj a través del periférico kinect de microsoft como interfaz de usuario.* Quito, Pichincha, Ecuador: Universidad Politécnica Salesiana. Recuperado el 2015

Ceballos, J. (2013). *Enciclopedia de Microsoft®Visual C#* (4 ed.). Madrid: RA-MA.

Chen, Y., Du, Z., & García Acosta, M. (4 de Junio de 2012). Robot as a Service in Cloud Computing. *Service Oriented System Engineering (SOSE)*, 8. doi:10.1109/SOSE.2010.44

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (15 de Marzo de 2001). *Web Services Description Language (WSDL) 1.1*. Obtenido de The World Wide Web Consortium (W3C): http://www.w3.org/TR/wsdl

COM+ (Component Services). (1 de Enero de 2015). Recuperado el 3 de Febrero de 2015, de https://msdn.microsoft.com/en-us/library/windows/desktop/ms685978(v=vs.85).aspx

COM+ (Component Services). (20 de 01 de 2015). Obtenido de https://msdn.microsoft.com/en-us/library/windows/desktop/ms685978(v=vs.85).aspx

Common Language Runtime (CLR). (23 de 12 de 2014). Obtenido de https://msdn.microsoft.com/es-es/library/8bs2ecf4(v=vs.110).aspx

Common Language Runtime (CLR). (1 de Enero de 2015). Recuperado el 28 de Enero de 2015, de https://msdn.microsoft.com/es-es/library/8bs2ecf4(v=vs.110).aspx

Conesa, J., Gavidia, À., Ceballos, J., & Gañán, D. (2010). *Introducción a .NET.* (S. Poch, Ed.) Barcelona: UOC.

Cordella, P. (2011). *USO DE INTERFACES TANGIBLES PARA MEJORAR LA INTERACCIÓN EN.* Chile: UNIVERSIDAD DE CHILE.

Cross platform, open source .NET framework. (1 de Enero de 2015). Recuperado el 25 de Enero de 2015, de Mono: http://www.mono-project.com/

Dinita, R., Wilson, G., Winckles, A., Cirstea, M., & Jones, A. (14 de Mayo de 2012). A cloud-based virtual computing laboratory for teaching computer networks. *Optimization of Electrical and Electronic Equipment (OPTIM)*, 5. doi:10.1109/OPTIM.2012.6231992

DirectX 11. (1 de Enero de 2015). Recuperado el 3 de Febrero de 2015, de http://windows.microsoft.com/es-419/windows7/products/features/directx-11

Gómez de Gabriel, J., Ollero, A., & García , A. (2006). *Teleoperación y Telerrobótica.* Madrid: Prentice Hall.

Hawke, S. (16 de Diciembre de 2011). *REST*. Obtenido de The World Wide Web Consortium (W3C): http://www.w3.org/2001/sw/wiki/REST

Hunziker, D., Gajamohan, M., Waibel, M., & D' Andrea, R. (6 de Mayo de 2013). Rapyuta: the RoboEarth cloud engine. *Robotics and Automation (ICRA)*, 7. doi:10.1109/ICRA.2013.6630612

Introducción a .NET Framework. (1 de Enero de 2015). Recuperado el 26 de Enero de 2015, de https://msdn.microsoft.com/es-es/library/hh425099(v=vs.110).aspx

Kimmel, P. (2008). *Manual de UML*. (C. Fernando, Ed.) Mexico: McGraw-Hill.

Linux, WiFi Shield for Arduino. (1 de Enero de 2015). Recuperado el 5 de Diciembre de 2014, de Dragino Technology Co., LTD: http://www.dragino.com/products/yunshield.html

Liu, M. (2012). *WCF 4.5 Multi-Layer Services Development with Entity Framework.* BIRMINGHAM: Packt Publishing Ltd.

MARK III Controller. (1 de Enero de 2015). Recuperado el 10 de Noviembre de 2014, de RHINO ROBOTICS LTD.: http://www.rhinorobotics.com/markiii.html

MARK IV Controller. (1 de Enero de 2015). Recuperado el 10 de Octubre de 2014, de RHINO ROBOTICS LTD. : http://www.rhinorobotics.com/markiv.html

Marshall, D. (1 de Mayo de 1999). *What Is RPC*. Recuperado el 27 de Enero de 2015, de http://www.cs.cf.ac.uk/Dave/C/node33.html

Micropik. (01 de 01 de 2015). Recuperado el 02 de 12 de 2015, de http://www.micropik.com/pag_potenciometro.htm

Microsoft Expression. (1 de Enero de 2015). Obtenido de https://www.microsoft.com/expression/esn/

Microsoft Silverlight. (1 de Enero de 2015). Recuperado el 2 de Febrero de 2015, de https://www.microsoft.com/silverlight/

Microsoft SQL Server. (1 de Enero de 2015). Recuperado el 4 de Febrero de 2015, de https://msdn.microsoft.com/es-es/library/bb545450.aspx

Microsoft Visual Studio. (01 de Enero de 2015). Recuperado el 05 de Enero de 2015, de https://www.visualstudio.com/

Modulo L298n Driver. (1 de Enero de 2015). Recuperado el 10 de Noviembre de 2014, de TecMikro Ecuador: http://tecmikro.com/modulo-l298n-driver-para-motores-arduino

OpenWrt Wireless Freedom. (1 de Enero de 2015). Recuperado el 25 de Enero de 2015, de https://openwrt.org/

Pololu Robotics. (01 de 01 de 2015). Recuperado el 2 de 12 de 2014, de https://www.pololu.com/product/2464

Quin, L. (19 de Mayo de 2015). *eXtensible Markup Language (XML).* Recuperado el 25 de Mayo de 2015, de The World Wide Web Consortium (W3C): http://www.w3.org/XML/

SCARA. (1 de Enero de 2015). Recuperado el 10 de Noviembre de 2014, de RHINO ROBOTICS LTD.: http://www.rhinorobotics.com/scara.html

Serialización (C# y Visual Basic). (1 de Enero de 2015). Recuperado el 2 de Enero de 2015, de https://msdn.microsoft.com/es-es/library/ms233843.aspx

Silverlight. (14 de 02 de 2015). Obtenido de https://www.microsoft.com/silverlight/

Tawfik, M., Salzmann, C., Gillet, D., Lowe, D., Saliah-Hassane, H., Sancristobal, E., & Castro, M. (26 de Febrero de 2014). Laboratory as a Service (LaaS): a Model for Developing and Implementing Remote Laboratories as Modular Components. *Remote Engineering and Virtual Instrumentation (REV)*, 10. doi:10.1109/REV.2014.6784238

Tawfik, M., Sancristobal, E., Martin, S., Diaz, G., & Castro, M. (13 de Junio de 2012). State-of-the-Art Remote Laboratories for Industrial Electronics Applications. *The Technologies Applied to Electronics Teaching (TAEE)*, 6. doi:10.1109/TAEE.2012.6235465

The Python Tutorial. (1 de Enero de 2015). Recuperado el 26 de Enero de 2015, de The Python Software Foundation: https://docs.python.org/2/tutorial/index.html

Web Services Specifications Index Page. (1 de Enero de 2015). Recuperado el 3 de Febrero de 2015, de https://msdn.microsoft.com/en-us/library/ms951274.aspx

Web Services Specifications Index Page. (15 de 01 de 2015). Obtenido de https://msdn.microsoft.com/en-us/library/ms951274.aspx

Windows Presentation Foundation. (05 de 02 de 2015). Obtenido de https://msdn.microsoft.com/es-es/library/ms754130(v=vs.110).aspx

XBOX. (1 de Enero de 2015). Recuperado el 5 de Febrero de 2015, de http://www.xbox.com/es-es/

Yun Shield Quick Start Guide. (08 de Julio de 2014). Recuperado el 25 de Diciembre de 2015, de Dragino Technology Co., LTD.: http://www.dragino.com/downloads/downloads/YunShield/YUN_SHIELD_QUICK_START_GUIDE_v1.0.pdf