UNAP

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

EXAMEN DE SUFICIENCIA PROFESIONAL

SISTEMA COMPUTACIONAL DE PROGRAMACIÓN FLEXIBLE

PARA OPTAR EL TÍTULO PROFESIONAL DE

INGENIERO DE SISTEMAS E INFORMÁTICA

PRESENTADO POR:

ERWIN ANTONIO ASPAJO SORIA

IQUITOS, PERÚ

2016

UNIVERSIDAD NACIONAL DE LA AMAZONIA PERUANA
FACULTAD DE INGENIERIA DE SISTEMAS E INFORMATICA

ACTA DE EXAMEN ORAL DE SUFICIENCIA PROFESIONAL

Siendo las 19.50 horas del día 20 de Octubre del 2016, en las instalaciones del Auditorio de la Facultad de Agronomía de la Universidad Nacional de la Amazonia Peruana, el Jurado Examinador, compuesto por los siguientes miembros:

Presidente : Ing. José Edgar García Díaz
Primer Miembro : Ing. Alejandro Reátegui Pezo
Segundo Miembro : Lic. Adm. Ángel Ildefonso Catashunga Torres

Posteriormente se procedió al acto académico del examen oral de suficiencia profesional del bachiller: **Erwin Antonio Aspajo Soria**, quien sustento el tema, **"Sistema Computacional de Programación Flexible"**.

ERWIN ANTONIO ASPAJO SORIA

Se procedió al cálculo de la Calificación y Condición Final, obteniéndose el siguiente resultado:

	Calificaciones	
	En número	**En letras**
Promedio de la Calificación Final de las Asignaturas.	15.00	Quince y 00/100
Calificación de la Sustentación del Informe Final.	14.80	Catorce y 80/100
Calificación Final	14.90	Catorce y 90/100

Se desprende que la Condición Final del Bachiller es (marcar el que corresponde):

( ) Aprobado con excelencia (18 a 20 puntos).
( ) Aprobado por unanimidad (15 a 17.9 puntos).
(X) Aprobado por mayoría (12 a 14.9 puntos).
( ) Desaprobado (Menos de 12 puntos).

Siendo las 20:40 horas del mismo día, se da por concluido el acto, firmando en conformidad los miembros del Jurado Examinador.

Presidente

Primer Miembro

Segundo Miembro

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

RESUMEN

La forma en que se programa una computadora depende en gran medida de lo que los futuros usuarios quieran de esa computadora. En el entorno informático de la oficina de ingeniería, existe la necesidad de utilizar aplicaciones relacionadas con el diseño asistido por ordenador. Sin embargo, el proceso más común en las estructuras bancarias es el procesamiento de grandes cantidades de información.

La elección del lenguaje de programación utilizado para implementar la aplicación tiene un amplio abanico de posibilidades dependiendo del tipo de aplicación que se esté desarrollando.

El estudio de un lenguaje de programación y su propósito limita un poco la facilidad con la que puede usarse para resolver ciertos tipos de problemas. Los lenguajes de propósito especial se enfocan en desarrollar una clase específica de aplicaciones de la manera más simple posible, y su semántica se limita al propósito para el cual fueron creados. Por otro lado, el objetivo de los llamados lenguajes generales es poder representar soluciones a cualquier tipo de dificultad computacional. Sin embargo, existen diferencias significativas en esta clasificación.

Lenguajes como Java [Gosling96] y C++ [Stroustrup98], se pueden seleccionar genéricos, orientados a objetos y similares a la sintaxis, p. para desarrollar aplicaciones portables y distribuidas -en el primer caso, bien asentadas, en las que domina la eficiencia de ejecución- caso del lenguaje C. La semántica computacional de un lenguaje de programación es fija y se aplica a los programas codificados en él. Una aplicación no puede cambiar la semántica del lenguaje en el que fue creada para cumplir con los nuevos requisitos que puedan surgir durante su existencia sin cambiar su código actual, su función central.

JUSTIFICACIÓN

Además de limitar las capacidades computacionales de las aplicaciones al elegir el lenguaje de programación a desarrollar, la interacción entre aplicaciones desarrolladas con referencia a diferentes lenguajes se suele realizar a través de mecanismos adicionales.
El uso de modelos binarios de componentes (como por ejemplo COM [Microsoft95]) o de arquitecturas de objetos distribuidos y middlewares (CORBA), son necesarios para permitir la interacción entre aplicaciones desarrolladas en distintos lenguajes de programación y, en ocasiones, sobre distintas plataformas.
Por tanto, justificamos la necesidad de explorar alternativas para crear entornos informáticos de programación que sean independientes del lenguaje y plataforma elegidos, permitan desarrollar aplicaciones en cualquier lenguaje de programación e interactuar entre sí sin utilizar mecanismos intermedios. Asimismo, la semántica de un lenguaje de programación no debe ser fija durante el tiempo de ejecución de la aplicación escrita en él; su significado (semántica) puede adaptarse a futuros contextos impredecibles a medida que evoluciona.

OBJETIVOS

OBJETIVO GENERAL

El objetivo de este trabajo es conocer la importancia de los sistemas computacionales de programación flexible, sus conceptos, su importancia en el desarrollo de aplicaciones multilenguaje y multiplataforma.

OBJETIVOS ESPECÍFICOS

- ✓ Dar a conocer los fundamentos de los sistemas computacionales de programación flexible
- ✓ Revisar varias alternativas y mencionaremos otras para desarrollar un entorno informático de programación flexible.
- ✓ Analizar el uso de los sistemas computacionales de programación flexible
- ✓ Conocer las plataformas, herramientas, metodologías basadas en los sistemas computacionales de programación flexible.

1. FUNDAMENTOS BASICOS DE SISTEMAS COMPUTACIONALES DE PROGRAMACION FLEXIBLE

Indicaremos los diversos objetivos que se incluyeron en el desarrollo del entorno de programación descrito anteriormente

1.1. LENGUAJE DE PROGRAMACIÓN E INDEPENDENCIA DE LA PLATAFORMA

Los sistemas informáticos se pueden programar en cualquier lenguaje y sus aplicaciones se pueden ejecutar en cualquier plataforma. Existe la necesidad de limitar la dependencia actual de sistemas informáticos específicos donde los lenguajes de programación de aplicaciones y las plataformas de desarrollo imponen plataformas específicas en su implementación. La elección del lenguaje de programación depende de cómo su expresión modela el problema y las preferencias del programador.
En nuestro sistema se pueden desarrollar aplicaciones en múltiples lenguajes de programación sin utilizar una capa de comunicación mutua entre diferentes módulos. La programación del sistema no necesita estar en un lenguaje conocido; el lenguaje de codificación utilizado se puede especificar en su código fuente, lo que permite que el sistema ejecute aplicaciones sin conocimiento previo del lenguaje que se utilizará.

1.2. INTEROPERABILIDAD DIRECTA

Actualmente, el uso de intercapas de software es la solución más común para lograr la interoperabilidad entre aplicaciones desarrolladas en diferentes lenguajes de programación o sobre diferentes plataformas físicas. La conversión entre modelos de componentes, middleware distribuido o arquitectura de objetos distribuidos permite que las aplicaciones desarrolladas en diferentes lenguajes de programación, sistemas operativos y plataformas se comuniquen entre sí. Sin embargo, el uso de estas capas adicionales tiene algunas desventajas: desarrollo de aplicaciones más complejo, más conversión entre diferentes modelos de computadora, dependencia de los mecanismos utilizados (conectividad) y mantenimiento reducido debido a cambios futuros.

1.3. EXTENSIBILIDAD

Los sistemas informáticos comunes amplían su nivel de abstracción de varias formas; los ejemplos incluyen extensiones de lenguaje de programación, bibliotecas o implementaciones de API, o desarrollo de componentes de software.
Cambiar el lenguaje de programación requiere una nueva versión del procesador de lenguaje, lo que resulta en la pérdida de portabilidad del código desarrollado para versiones anteriores. El uso de componentes, API o bibliotecas no proporciona un conocimiento dinámico de los servicios que brinda y carece de un marco general para extender su funcionalidad al resto del sistema, a veces con dependencias adicionales en plataformas o

lenguajes de programación. Nuestro sistema debe soportar un mecanismo para ser extensible: aprender dinámicamente sobre los servicios existentes para que puedan ser ampliados si es necesario, lograr un mayor nivel de abstracción para todo el entorno de programación, independientemente del lenguaje y la plataforma.

1.4. ADAPTABILIDAD NO RESTRICTIVA

La adaptabilidad de un sistema informático es la capacidad de adaptarse a los requisitos generados dinámicamente que son desconocidos en la etapa de diseño. La mayoría de los marcos existentes proporcionan mecanismos limitados para desarrollar aplicaciones adaptables dinámicamente. Las aplicaciones desarrolladas en nuestro entorno de programación deben ser capaces de adaptarse dinámicamente a nuevos contextos sin anticiparlos durante la fase de desarrollo. Durante la ejecución, la aplicación puede encontrar nuevos requisitos y adaptarse a ellos sin detener su ejecución, cambiar el código fuente y reiniciarlo.

La estructura de la aplicación en ejecución, así como la semántica del lenguaje de programación especificado, deben poder ajustarse sin restricciones. Como veremos más adelante, algunos marcos le permiten ajustar dinámicamente solo una parte de la estructura de una aplicación o lenguaje de programación. En nuestro marco informático, una aplicación, independientemente de su lenguaje de programación, se puede desarrollar separando su código funcional de un aspecto independiente de la funcionalidad y, por lo tanto, se puede reutilizar (depuración, persistencia, sincronización, distribución o monitoreo). Varios aspectos de un programa se pueden comprobar y cambiar dinámicamente sin interrumpir su ejecución. Cabe señalar que, como exploraremos en profundidad, el objetivo principal del sistema buscado es su flexibilidad, que muchas veces está reñida con su eficiencia; cuanto más flexible sea el entorno informático, menos eficiente será. Por tanto, encontrar sistemas informáticos más eficientes, pero más flexibles que los estudiados aún no está entre nuestros principales objetivos. Se pueden usar varias técnicas para los sistemas propuestos para lograr mejoras en el rendimiento.

2. REQUISITOS DEL SISTEMA

En este capítulo se describen los requisitos del sistema para este informe. Se identificarán y describirán brevemente diferentes requisitos, incluidos los requisitos en diferentes categorías funcionales. El conjunto de requisitos serían los requisitos para la plataforma informática subyacente buscada, los requisitos necesarios para desarrollar un entorno de programación con las características deseadas y los diversos niveles de flexibilidad necesarios para lograr los objetivos establecidos. Finalmente, definiremos un conjunto común de requisitos para el sistema desde una perspectiva global. El propósito de la especificación de requisitos del sistema realizada en este capítulo es verificar el sistema que se está creando, así como explorar varios sistemas existentes similares al sistema que se busca. Los requisitos del sistema nos permiten reconocer las fortalezas de los sistemas reales, para una mayor adopción o investigación, así como cuantificar sus debilidades y sugerir algunas modificaciones y/o extensiones.

2.1. REQUISITOS DE LA PLATAFORMA DE COMPUTACIÓN

Para desarrollar un sistema informático flexible y adaptable que pueda ejecutarse en cualquier plataforma independientemente del lenguaje y cuyas aplicaciones sean distribuibles e interoperables, la capa de ejecución subyacente, la plataforma virtual, debe cumplir varios requisitos. Su selección e implementación será un trabajo muy importante, ya que todo el sistema se desarrollará sobre esta plataforma; si no proporciona suficiente flexibilidad, es posible que el entorno de programación no se pueda personalizar por completo. Definamos brevemente los requisitos que debe cumplir la plataforma virtual de nuestro sistema para lograr nuestros objetivos.

a) Sistema computacional multiplataforma.

La plataforma informática subyacente a nuestro sistema debería poder implementarse en cualquier sistema actual. Debe ser completamente independiente del microprocesador, las características del sistema operativo o cualquier detalle del sistema en cuestión. La implementación de nuestra plataforma informática central debería poder integrarse en cualquier sistema informático con requisitos mínimos de procesamiento.

b) Independencia del lenguaje de programación.

El uso de plataformas virtuales, abstrayendo la descripción de máquinas, es conocido en entornos distribuidos y portátiles. Sin embargo, estas definiciones de plataforma suelen estar asociadas a un lenguaje de programación específico. La base informática de nuestro sistema debe ser independiente del lenguaje de programación utilizado para acceder a él. a través de la escena compilación deberá ser factible una traducción a partir diversos lenguajes de programación al código nativo de la plataforma.

c) Independencia del problema.

Las descripciones de la plataforma no deben centrarse en resolver una clase de problemas. Algunas implementaciones describen una plataforma para resolver problemas como la persistencia en sistemas orientados a objetos o la comunicación entre aplicaciones distribuidas. Dichas plataformas se definen de manera especial para solucionar los problemas que se presentan. Dados los desafíos a la mano, nuestra plataforma informática debe describirse de manera flexible. No se trata de implementar soluciones a la mayoría de los problemas existentes, sino de poder adaptar sus modelos semánticos y computacionales de manera general para resolver todos los problemas que se presenten.

d) Tamaño y semántica computacional reducida.

El primer requisito descrito, "sistema informático multiplataforma", asegura que la plataforma sea independiente del sistema en el que se implementa. Al mismo tiempo, decidimos que debería poder funcionar en sistemas con contenido computacional reducido. Para cumplir con este requisito, la plataforma subyacente debe implementarse para reducir el tamaño.

La semántica computacional subyacente de la plataforma, las primitivas computacionales, debe ser lo más simple posible. Esta semántica es una descripción del comportamiento normal de una máquina abstracta. Sin embargo, las primitivas o funciones de operación en lenguaje máquina deben ser externas, accesibles y extensibles, específicas para cada implementación. Al separar la semántica operativa y computacional de la plataforma, nuestro objetivo es minimizar la plataforma raíz para que pueda implementarse en cualquier entorno. Si el sistema tiene suficiente poder de cómputo, podrá aumentar su número de operaciones de cómputo. Este capítulo establece los siguientes dos requisitos.

e) Flexibilidad computacional

Tal y como definimos en nuestro requerimiento de “Issue Independence”, la plataforma debe ser lo suficientemente flexible para adaptarse a la solución de cualquier problema. No se trata de anticipar y satisfacer todas las necesidades del sistema, sino de crear una plataforma manejable que te permita adaptarte a cualquier desafío. Para ello, hemos dividido este requisito de personalización en tres partes:

- ✓ **Semántica operacional abierta.**

  Las operaciones primitivas del lenguaje de la plataforma deben ser modificables para cualquier implementación. Dentro de cada máquina de abstracción, podemos aumentar la cantidad de primitivas existentes para agregar niveles más altos de abstracción al entorno. Lo que se consigue es la flexibilidad de la semántica operativa del lenguaje.

- ✓ **Introspección y acceso al entorno.**

  Una aplicación portátil debe poder comprender la funcionalidad del entorno existente en el que se ejecuta. A través del autodiagnóstico, la aplicación podrá saber si se ha realizado la acción solicitada y actuar en consecuencia. De esta forma, utilizando una única plataforma proporcionamos diferentes niveles de abstracción y la capacidad de saber en qué plataforma estamos.

- ✓ **Semántica computacional abierta.**

  El mayor grado de flexibilidad de la plataforma se logrará gracias a su mecanismo de cambio de semántica computacional, abstrayendo el mismo lenguaje en la máquina. Al permitir que se cambie el comportamiento de la máquina, sería más fácil para los programadores cambiar la interpretación del programa sin cambiar una sola línea de código. Además, la implementación de una máquina nunca cambia: sus operaciones se modifican en su lenguaje.

2.2. REQUISITOS DEL ENTORNO DE PROGRAMACION

El diseño de la plataforma informática se centra en desarrollar una máquina de abstracción flexible con menor tamaño y semántica operativa.
Para ampliar su semántica computacional y proporcionar a los programadores de aplicaciones un mayor nivel de abstracción, desarrollaremos un entorno de programación con servicios comunes específicos del sistema operativo (sin propiedades del sistema operativo). Para probar su modo de flexibilidad, el entorno de programación implementará varias funciones de manera adaptativa para que puedan seleccionarse y modificarse dinámicamente.

a) Desarrollo de Características de Computación (Extensibilidad)

Las plataformas informáticas deben ser lo suficientemente escalables para permitir el desarrollo de funciones informáticas adicionales. Dadas las primitivas de la plataforma (como el paso de mensajes, la creación de objetos o la evaluación de métodos), el entorno de programación debe poder complementar estas otras capacidades con un mayor nivel de abstracción o reemplazar las capacidades existentes.

b) Selección de Características de Computación (Adaptabilidad)

Las nuevas funciones informáticas desarrolladas en el entorno de programación deben ser reemplazables dinámicamente, con la capacidad de cambiar o seleccionar estas funciones según las necesidades del programador. Por ejemplo, si los mensajes se transforman para su distribución, el protocolo de comunicación utilizado debe poder seleccionarse dinámicamente.

c) Identificación del Entorno de Programación

La única condición para formar parte de la plataforma distribuida es estar ejecutando la máquina virtual. Como mencionamos, su tamaño y su conjunto de primitivas son pequeñas, por lo que las ampliamos con un entorno de programación. Si la memoria o el espacio de procesamiento de un sistema físico es pequeño, entonces la porción del entorno de programación en él será pequeña. Para ello, un sistema de cómputo heterogéneo debe contar con un mecanismo para conocer o identificar el conjunto de servicios instalados en el entorno de programación.

d) Autodocumentación Real

En los requisitos anteriores, indicábamos la posibilidad de infinitas versiones del entorno de programación, cada una dependiendo de los intereses y limitaciones de la unidad de procesamiento. Se necesita un sistema de autodocumentación para obtener una imagen precisa del entorno existente. La plataforma debe proporcionar información sobre todos los objetos, métodos y propiedades que existen real y directamente. Si alguno de estos cambios, el sistema debería poder cambiar automáticamente el documento, nunca desactualizado con lo que realmente existe en el sistema.

2.3. REQUISITOS CONCERNIENTE AL GRADO DE FLEXIBILIDAD

El objetivo principal del sistema buscado en este informe es crear un sistema con un alto grado de flexibilidad, convirtiéndolo en una plataforma de ejecución general sin cambiar la implementación de la máquina abstracta.
Desde la perspectiva de la seguridad de los sistemas informáticos, acceder y modificar diversas aplicaciones es una tecnología restringida por mecanismos de seguridad. A medida que desarrollemos este informe, nos centraremos en desarrollar un sistema que sea flexible en este departamento, con consideraciones de seguridad como una capa adicional para restringir el acceso a estas funciones.

a) Desarrollo de Características de Computación (Extensibilidad).

Cualquier programa debe poder conocer el estado del sistema u otro programa mientras se está ejecutando. Gracias a esta característica, podremos desarrollar aplicaciones que se adapten dinámicamente al contexto, ya que se pueden analizar en tiempo de ejecución.

b) Modificación Estructural Dinámica.

Cualquier aplicación que se ejecute en la plataforma puede cambiar dinámicamente los objetos, las aplicaciones o la estructura de todo el entorno de programación. La capacidad de cambiar la estructura de cualquier dispositivo en tiempo de ejecución permite sistemas dinámicamente escalables.

2.4. REQUISITOS GENERALES DEL SISTEMA

a) Independencia del hardware.

La interfaz de acceso a la plataforma no deberá ser dependiente del hardware en el que haya sino implantado. La plataforma, y en grupo todo el sistema, deberá poder instalarse sobre distintos sistemas hardware.

b) Independencia del sistema operativo.

El sistema deberá lograr implantarse en cualquier sistema operativo, y por lo tanto no deberá ser diseñado con características particulares de algún sistema concreto.

c) Interacción con sistemas externos.

Las aplicaciones desarrolladas en el sistema deben poder interactuar con las aplicaciones nativas existentes en el sistema operativo utilizado. El acceso a la plataforma debe proporcionar una interfaz similar para cada sistema operativo utilizado, siguiendo en cada caso métodos estándar documentados de interacción entre aplicaciones. Usando mecanismos estándar seleccionados, cualquier aplicación en el sistema podrá interactuar con la plataforma desarrollada, y el código portátil de nuestra plataforma podrá acceder a los recursos del sistema actual en ejecución.

3. MÁQUINAS ABSTRACTAS

Un programa es un conjunto ordenado de instrucciones dadas a una computadora que le indica una acción o tarea a realizar [Cueva94]. Estos programas serán ejecutados, animados o interpretados por el procesador de la computadora. El procesador de una computadora ejecuta instrucciones de un programa que accederá, examinará o modificará los datos propiedad del programa. La implementación del procesador de una computadora puede ser física (hardware) o lógica (software) al desarrollar otro programa.

3.1. PROCESADORES COMPUTACIONALES

a) Procesadores Implementados Físicamente

Un procesador físico es un intérprete de programa diseñado en forma física, generalmente como un circuito integrado. Los procesadores más utilizados son los procesadores digitales síncronos. Se componen de una unidad de control, una memoria y una unidad aritmética lógica, todas interconectadas [Mandado73]. Una computadora es un procesador digital síncrono cuya unidad de control es un sistema secuencial síncrono que recibe del exterior (del programador) una serie de instrucciones que le indican las microoperaciones a realizar [Mandado73]. El orden en que se ejecutan estas operaciones se define en la memoria que describe el programa, pero la semántica de cada instrucción y el conjunto global existente del procesador son constantes. La ventaja de este tipo de procesadores sobre los procesadores lógicos es la velocidad a la que pueden desarrollarse físicamente. Como decíamos en el apartado anterior, su principal inconveniente es la falta de flexibilidad.

b) Procesadores Implementados lógicamente

Un procesador de software es un programa que a su vez interpreta otro programa procesador [Cueva98]. Es un programa capaz de explicar o simular acciones procesador dado. Es mucho más fácil cambiar de programa a procesadores analógicos en lugar de cambios físicos en el procesador de hardware. Esto significa que, entre otras cosas, los emuladores de software se utilizan como herramientas de simulación diseñadas para implementar físicamente los procesadores que emulan. La principal desventaja de este tipo de procesador es la velocidad de ejecución frente al hardware o procesadores físicos. A medida que se construye el procesador lógico otro nivel de computación (realizado o interpretado por otro procesador), inevitablemente requieren más computación para interpretar el programa Que su versión de hardware. Esta sobrecarga computacional es visible en el gráfico.

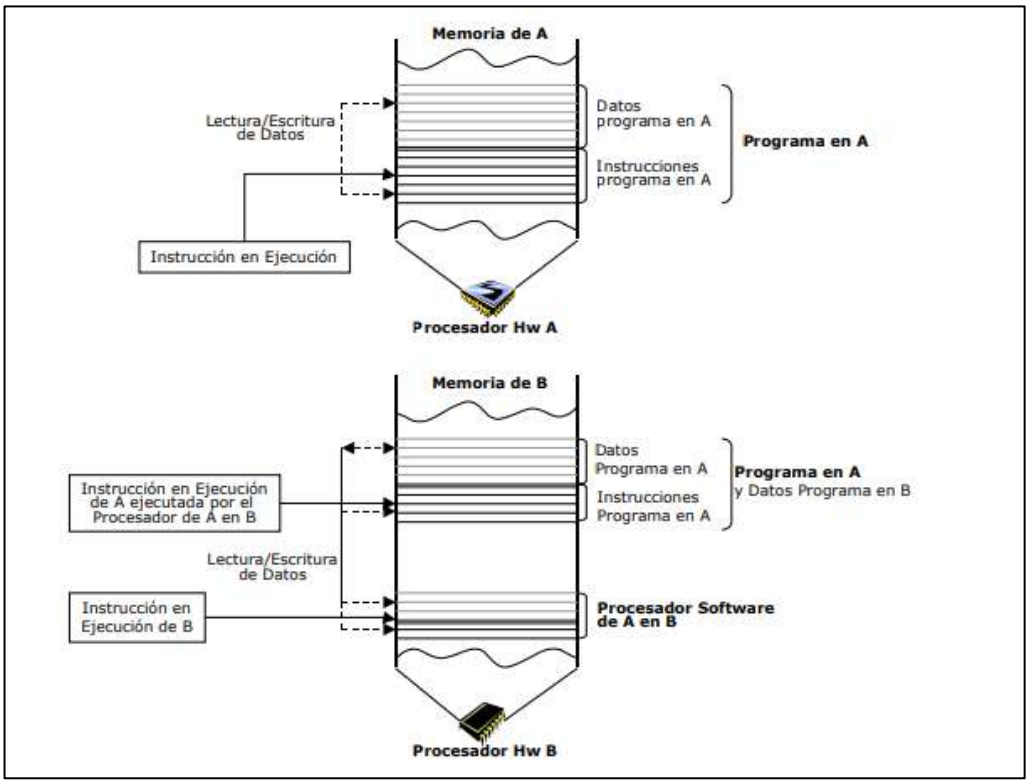

Fig. 1. Ejecución de un procesador lógico frente a un procesador físico.

3.2. APORTACIÓN DE LA UTILIZACIÓN DE LAS MÁQUINAS ABSTRACTAS

La definición de máquina abstracta y las ventajas que aporta la utilización de estas. De todas ellas podemos decir que las aportaciones a nuestro sistema buscado, descrito en el capítulo 1, son:

1. Independencia del lenguaje de programación.
2. Portabilidad de su código.
3. Independencia de la plataforma.
4. Interacción única entre aplicaciones, utilizando un único modelo de computación.
5. Programación interactiva y continua.
6. Distribución de aplicaciones
7. Interoperabilidad nativa de aplicaciones distribuidas.

4. UTILIZACIÓN DE LAS MÁQUINAS ABSTRACTAS

Presentamos los conceptos de máquinas abstractas y máquinas virtuales y sus diversas posibilidades prácticas para los sistemas informáticos. Del abanico general de ventajas que ofrece su uso, destacamos el conjunto de ventajas presentadas en el Capítulo 1 de las que dispone el sistema. Con base en una taxonomía práctica de los sistemas existentes, examinaremos las diversas implementaciones realizadas y resaltaremos los requisitos alcanzados y las brechas existentes, siguiendo los requisitos establecidos en el Capítulo 2. Una vez descritos y analizados los diferentes sistemas, determinaremos qué requisitos con los ejemplos prácticos existentes, cómo adaptarlos a nuestros objetivos, así como sus carencias y la necesidad de superarlas.

4.1. PORTABILIDAD DEL CÓDIGO

a) Estudios de sistemas existentes.

Máquina-p

p-code es un lenguaje intermedio de la máquina de abstracción p-machine [Nori76], que se usó originalmente para desarrollar el compilador del lenguaje Pascal. La Universidad de California, San Diego (UCSD) ha desarrollado un procesador que puede ejecutar código binario desde una máquina p (p-code). Adoptó la especificación de máquina abstracta para desarrollar el proyecto Pascal portátil. El soporte para multitarea estuvo disponible y se desarrolló p-System: un sistema operativo portátil codificado en Pascal y traducido a p-code. La única parte del sistema que debe implementarse en una plataforma específica es el emulador de máquina p. El sistema se basa en varios microprocesadores como DEC LSI-11, Zilog Z80, Motorola 68000 e Intel 8088.

Forth

Otro ejemplo de especificación de una máquina abstracta para la portabilidad del código es la máquina virtual Forth. Este lenguaje fue desarrollado en la década de 1970 por Charles Moore para controlar telescopios. Es un lenguaje simple, rápido y extensible que se interpreta en una máquina virtual, lo que lo hace portátil entre plataformas y útil para integrarse en los sistemas. El emulador de máquina virtual Forth tiene dos pilas. El primero es la pila de datos: los parámetros de operación se toman de la parte superior de la pila y el resultado se coloca en la pila. La segunda pila es la pila de valores devueltos: el valor del contador del programa se coloca en la pila antes de llamar a la subrutina, de modo que vuelve al punto de ejecución original cuando finaliza la llamada.

4.2. INTEROPERATIVIDAD DE APLICACIONES

Para permitir que distintas aplicaciones puedan interoperar entre sí. Si bien el concepto de máquina abstracta no es utilizado como una plataforma completa de computación, veremos cómo puede describirse como un pequeño middleware para interconectar distintas aplicaciones.

a) Estudios de sistemas existentes.

PerDis

PerDis es un middleware que proporciona un entorno de programación para aplicaciones orientadas a objetos, distribuidas, persistentes y transaccionales que pueden desarrollarse de manera limpia, escalable y eficiente [Kloosterman98]. La creación de la plataforma PerDiS surgió de la necesidad de desarrollar aplicaciones de ingeniería colaborativa en una "empresa virtual": un grupo de empresas o departamentos que trabajan juntos en la implementación del proyecto. Las diferentes partes de una empresa virtual colaboran y coordinan su trabajo compartiendo datos en su red informática. PerDiS proporciona un entorno persistente distribuido que combina las características de un modelo de memoria compartida, un sistema de archivos distribuido y una base de datos orientada a objetos.

Al igual que en el entorno de programación de memoria compartida, PerDiS le permite desarrollar aplicaciones que acceden a objetos en la memoria independientemente de su ubicación. Si es necesario, los objetos se envían a la memoria local sin ambigüedades, lo que hace que la programación distribuida sea menos explícita sobre la ubicación de los objetos. Al igual que los sistemas de archivos distribuidos, PerDiS almacena bloques de datos denominados clústeres en el disco y los almacena en el lado del cliente.

Un clúster es un conjunto de objetos que define una unidad de almacenamiento, denominación, protección y uso compartido, similar a un sistema de archivos [Sun89]. PerDiS también proporciona un conjunto básico de funciones de bases de datos orientadas a objetos, como almacenamiento local y operaciones de objetos de lenguajes orientados a objetos (como C), varios modelos de transacciones y tolerancia a fallas.

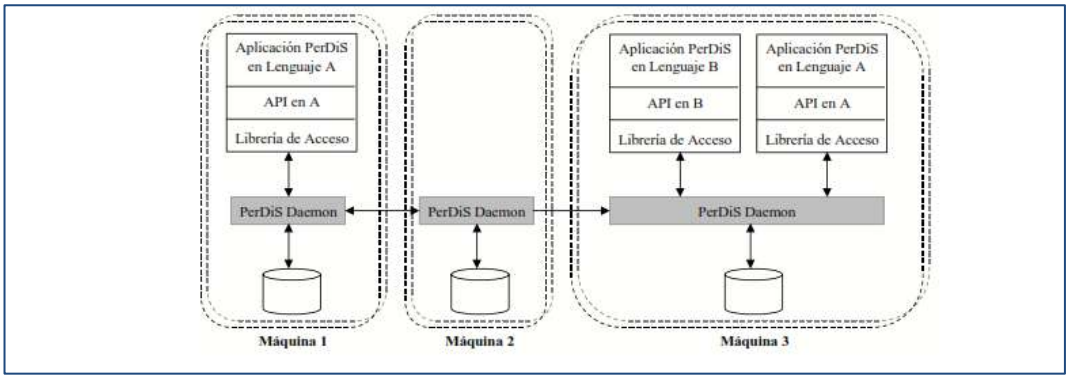

Fig. 2. Arquitectura de la plataforma PerDiS

4.3. PLATAFORMAS INDEPENDIENTES

En esta taxonomía, analizaremos casos prácticos en los que se utilizan máquinas abstractas como herramientas para construir plataformas independientes del lenguaje, del sistema operativo y del microprocesador. El sistema estudiado reconoce una máquina abstracta como base de la computación y desarrolla toda la plataforma independientemente del entorno informático real sobre esta base.

a) Estudios de sistemas existentes.

Java

En 1991, un grupo de ingenieros trabajó en Project Green: un sistema que podía conectar cualquier dispositivo electrónico. El objetivo es poder programar cualquier dispositivo utilizando un lenguaje simple, un intérprete conciso y el código es totalmente portátil. Especificaron una máquina abstracta con código binario de bytes y probaron C como su lenguaje de programación [Stroustrup98]. Al darse cuenta de su complejidad y dependencia de la plataforma de ejecución, lo simplificaron al lenguaje Oak, que más tarde pasó a llamarse Java [Gosling96]. Ninguna empresa de equipos estaba interesada en el producto y en 1994 el proyecto fracasó. En 1995, con el uso generalizado de Internet, desarrollaron un navegador HTTP en Java capaz de ejecutar programas Java previamente descargados del servidor en el lado del cliente, este último llamado HotJava. Como resultado de esta implementación, Netscape introdujo un emulador de máquina abstracta en su Navegador, que

permite que los subprogramas agreguen computación a las páginas HTML estáticas que se usan en Internet; Java es mundialmente famoso. Se compila un programa en lenguaje Java para ejecutarse en una plataforma independiente. Esta plataforma de ejecución independiente consta básicamente de las siguientes partes:

- ✓ La máquina virtual de Java (Java Virtual Machine) [Sun95].
- ✓ La interfaz de programación de aplicaciones en Java o Core API (Application Programming Interface).

Esta dualidad de lograr la independencia de la plataforma y las aplicaciones de abstracción de mayor nivel es similar a lo que reconocimos en Smalltalk-80: imágenes y máquinas virtuales. Una API es un conjunto de clases compiladas en un formato de código binario de máquina abstracto [Sun95]. Los programadores usan estas clases para construir aplicaciones más fácilmente. La Máquina Virtual Java es el procesador de código binario de la plataforma. El soporte orientado a objetos está definido en su código binario, aunque su arquitectura no está definida12. Por lo tanto, la implementación del intérprete y la representación de objetos en la memoria aún están a cargo de los diseñadores de simuladores que se beneficiarán de la plataforma adecuada. La plataforma Java se creó a partir de la necesidad existente de abrir la aplicación al dominio informático. Las redes informáticas conectan diferentes tipos de computadoras y dispositivos entre sí. Se han desarrollado diversas aplicaciones, protocolos, middleware y arquitecturas cliente-servidor para resolver el llamado problema de la programación distribuida.

Las redes de computadoras están conectadas a diferentes tipos de dispositivos y computadoras con diferentes arquitecturas de hardware y sistemas operativos. Java define una máquina abstracta para implementar aplicaciones distribuidas que se ejecutan en todas las plataformas en red (independientes de la plataforma). De esta forma, cualquier elemento conectado a la red, siempre que cuente con un procesador de máquina virtual y la API adecuada, puede procesar una aplicación o parte de una aplicación implementada en Java. Para realizar este tipo de programación distribuida, la especificación abstracta de la máquina Java básicamente sigue tres estándares:

- ✓ Búsqueda de una plataforma independiente.
- ✓ Movilidad del código a lo largo de la red.
- ✓ Especificación de un mecanismo de seguridad robusto.

La definición de funciones independientes de la plataforma está cubierta por una especificación de máquina abstracta. Sin embargo, para permitir el movimiento de código a través de una red informática, hay otra consideración: la especificación de la máquina debe permitir que el código de la aplicación se recupere desde una máquina remota. La función "Classloader" [Gosling96] admite directamente la distribución del código de la aplicación Java; puede especificar cómo obtener software (en este caso, clases) que se encuentra en máquinas remotas. De esta

forma, la distribución del software es automática, ya que todo el código puede centralizarse en una única máquina y cargarse desde el propio cliente al inicio de cada ejecución.

Adicionalmente, cabe indicar la importancia que se le ha dado a la seguridad y robustez de la plataforma. Las redes representan un instrumento para aquellos programadores que deseen dañar información, escamotear recursos o simplemente fastidiar. Un ejemplo de este tipo de aplicaciones puede ser un virus distribuido, que se ejecute en nuestra máquina una vez demandado por la red.
El primer módulo en la definición de una plataforma orientada a la robustez es el "verificador de código" [Sun95]. Una vez que se carga el código, entra en la fase de verificación: la máquina se asegura de que la clase esté codificada correctamente y no viole la integridad de la máquina abstracta. La seguridad de ejecución de código que proporciona la máquina virtual la proporciona el "gestor de seguridad"; las aplicaciones cliente pueden especificar una política de seguridad que restrinja el acceso a cualquier recurso que la propia máquina virtual considere apropiado. Así, se pueden definir los límites del software adquirido; por ejemplo: ejecutar un subprograma descargado de un servidor web evitará que los archivos se eliminen del disco duro.

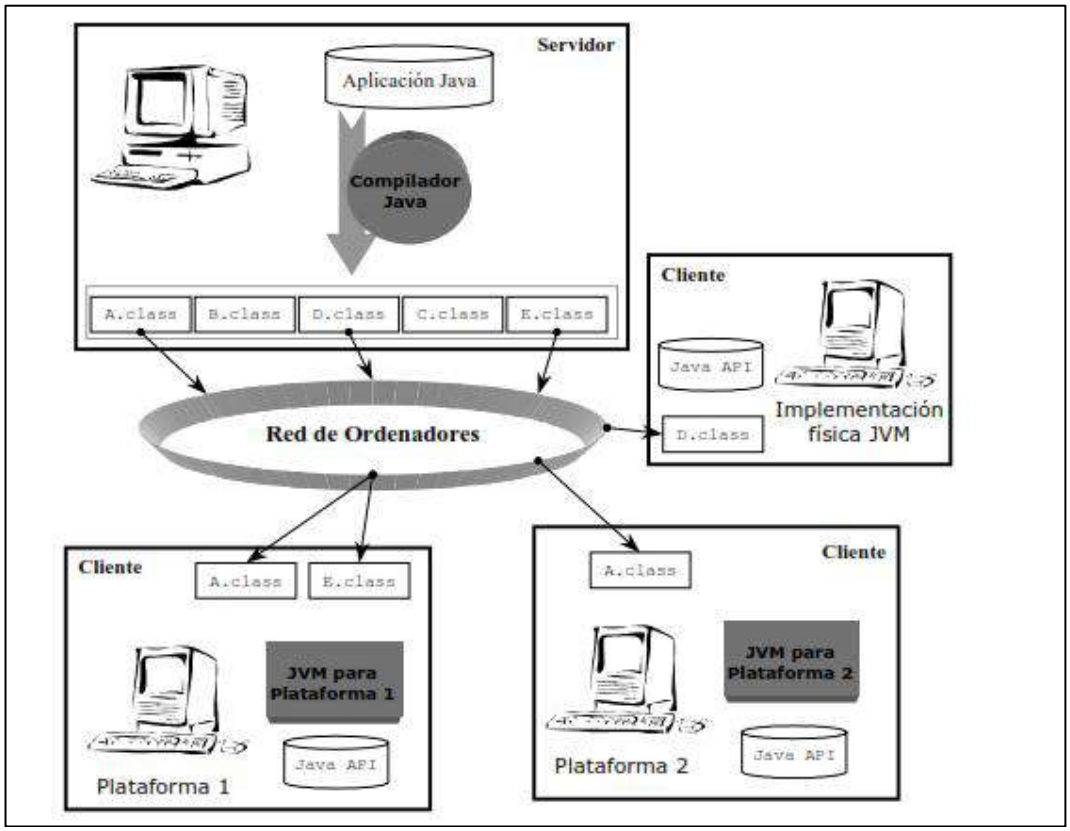

Fig. 3. Ejemplo de entorno de programación distribuida en Java.

.NET

En.NET es una combinación de la estrategia de negocios y el diseño de la plataforma de Microsoft enfocado en introducir la computación de cualquier ordenador en Internet [Microsoft2000]. El objetivo principal es proporcionar una plataforma en la que los usuarios individuales y las empresas puedan disfrutar de un entorno de interoperabilidad sin fisuras en Internet. Las aplicaciones se pueden desarrollar y distribuir fácilmente a través de la red, independientemente de la plataforma física utilizada.

La versión final de .NET se lanzó en 2002. Desde julio de 2000, la primera versión de la herramienta de desarrollo "Visual Studio .NET" está disponible en el portal de desarrolladores de Microsoft. Toda la información relacionada con el desarrollo de la plataforma está publicada en el sitio .NET [Microsoft2000]

Con la plataforma .NET, Microsoft pasará de ser un proveedor de software a un proveedor de servicios, desarrollando la funcionalidad que obtienen los usuarios a través de sus conexiones a Internet. Los consumidores de los Servicios no siguen el ciclo de compra, instalación y mantenimiento de cada aplicación; con .NET, compran una licencia para un servicio que se instala y actualiza automáticamente en las computadoras de sus clientes.

Los tres principales servicios identificados por la empresa son: almacenamiento, autenticación y notificación. Con estos, .NET proporcionará [Microsoft2000]:

- La simplicidad de interconectar y hacer que se comuniquen entre sí distintos sistemas de computación, haciendo que la información de usuario sea actualizada y sincronizada de forma automática.

- Un mayor nivel de interacción para las aplicaciones Web, habilitando el uso de información en XML (Extensible Markup Language).

- Un servicio de suscripción en línea que permita ingresar y obtener, de modo personalizado, productos y servicios de la computadora servidor.

- Un almacenamiento de datos centralizado, que aumentará la eficiencia y facilidad de acceso a la información, al mismo tiempo que los datos estarán sincronizados entre los distintos usuarios y dispositivos.

- La posibilidad de interconexión de equipos mediante distintos medios, como correo electrónico, faxes y teléfonos.

Para los desarrolladores de software, la posibilidad de crear módulos reutilizables e independientes de la plataforma, que aumenten la productividad.

.NET (Figura 4) se encuentra en una máquina virtual llamada Common Language Runtime, que proporciona un motor de ejecución para el código, independientemente del sistema en el que se haya desarrollado y compilado. Lenguaje de programación En la raíz de la plataforma (un lenguaje de tiempo de ejecución genérico), se proporcionará un marco básico: código básico que es aplicable a cualquier plataforma y se puede utilizar en cualquier lenguaje de programación. El entorno de programación se establece en XML [W3C98]. Se puede acceder desde una aplicación web (utilizando un navegador de Internet) o una interfaz gráfica. Cualquier comunicación entre aplicaciones se realizará mediante mensajes codificados en XML, por lo que es independiente de la aplicación de envío y del sistema operativo. 1991

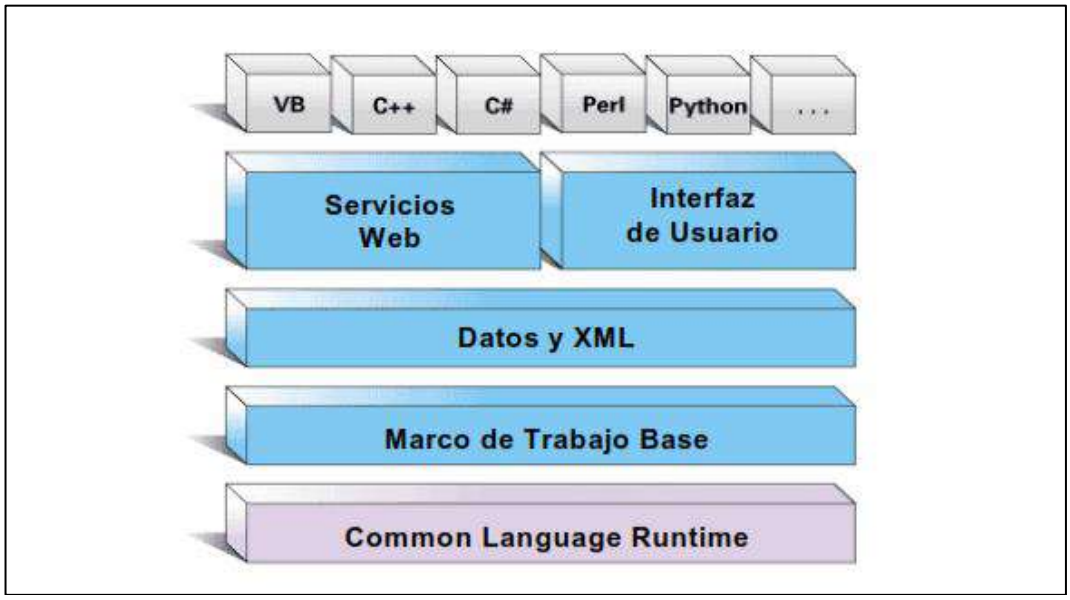

Fig. 4. Arquitectura de la plataforma .NET.

Las ventajas propias de desarrollar aplicaciones en .NET, usando la arquitectura mostrada en la Figura 4., son:

- Utilización del mismo código base en .NET, puesto que su codificación es independiente de la computadora hardware.

- La depuración de aplicaciones se realizará de forma indiferente al lenguaje de implementación utilizado. Se podrán depurar aplicaciones, incluso si están codificadas en diferentes lenguajes de programación.

- Los desarrolladores podrán reutilizar cualquier clase, mediante herencia o composición, indistintamente del lenguaje empleado.

- Unión de tipos de datos y manejo de errores (excepciones).

- La máquina virtual ejecutará código con un sistema propio de seguridad.

- Se puede examinar cualquier código con una clase, obtener sus métodos, mirar si el código está firmado, e incluso conocer su árbol de jerarquía.

b) Aportaciones y Carencias de los Sistemas Estudiados

Se creó un sistema informático multiplataforma en todos los sistemas probados.
Java Abstract Machine fue diseñado para proporcionar soporte portátil para lenguajes de programación. Su arquitectura está básicamente diseñada para trabajar con el lenguaje. Sin embargo, mientras .NET está en desarrollo, su arquitectura parece ser independiente del lenguaje de programación, pero C# parece haber sido desarrollado específicamente para esta plataforma [Microsoft2000]. Por lo tanto, .NET es la única plataforma diseñada con el requisito de independencia del lenguaje de programación en mente. Aunque las plataformas estudiadas intentaron lograr una variedad de resultados cuando se diseñaron, ninguna fue diseñada para resolver un tipo específico de problema. Si los juzgamos por el ejemplo del requisito de tamaño y la semántica computacional limitada, los tamaños de todas las plataformas de investigación son muy grandes. El costo de implementar una máquina virtual en una nueva plataforma o sistema integrado es alto; esto lo confirma la distribución de la máquina virtual java de javasoft mediante complementos: las diversas modificaciones y la complejidad de la JVM ralentizan su implementación en los navegadores de Internet, por lo que comparte un complemento con el sistema.
Para el grado de flexibilidad requerido en los requisitos de flexibilidad computacional, todas las máquinas disfrutan del autoexamen: cualquier objeto puede explorarse en tiempo de ejecución. Finalmente, la investigación de Microsoft en la creación de la plataforma .NET muestra que es necesaria hasta cierto punto en computación, una solución global para desarrollar aplicaciones portátiles distribuidas. Compatible. Hoy en día, existen varias alternativas para crear este tipo de aplicaciones, pero no existe una solución única para todos. Lo que Microsoft está tratando de lograr con .NET, y lo que está tratando de lograr en este informe, es la necesidad de desarrollar una nueva plataforma informática virtual.

5. CONCLUSIONES

Hemos visto cómo el uso original de plataformas de máquinas virtuales abstractas buscaba asegurar la portabilidad de su código binario. Se desarrollaron sistemas que lo tradujeron al lenguaje; otras plataformas más modernas y comercialmente exitosas se diseñaron para admitir principalmente un solo lenguaje de programación (como Java Virtual Machine). Uno de los objetivos del diseño de plataformas informáticas flexibles es que su construcción no esté ligada a un determinado lenguaje de programación de alto nivel (independencia del lenguaje de programación).

La portabilidad de código proporcionada por las máquinas virtuales se utiliza en entornos informáticos distribuidos. Si el código de la plataforma se puede ejecutar en cualquier sistema, se puede distribuir fácilmente a través de una red de computadoras. Además, las aplicaciones distribuidas en entornos informáticos heterogéneos con una única representación de datos y modelo de cálculo pueden intercambiar datos localmente sin utilizar una interfaz de representación común. Como apuntábamos, la creación de diferentes máquinas de abstracción se ha utilizado para resolver problemas concretos. Pero el objetivo de este informe es encontrar una plataforma informática que sea independiente del problema en particular. Estos sistemas permiten la programación portátil, la distribución de código, la interacción de aplicaciones distribuidas y un modelo informático único basado en el paradigma de programación orientada a objetos. En un entorno heterogéneo distribuido, la implementación de la aplicación es más uniforme y sencilla. Vale la pena señalar que el proyecto ".NET" de Microsoft (4.3.1), que se encuentra actualmente en desarrollo, combina sus sistemas operativos en una sola plataforma basada en una máquina abstracta.

En los sistemas estudiados en el punto anterior se desarrollaron diversas máquinas de abstracción para lograr el entorno de programación. Sin embargo, la funcionalidad proporcionada también se ha utilizado para el desarrollo de sistemas operativos. La única diferencia entre los dos es la semántica de los servicios implementados. El principal inconveniente de todos los sistemas mencionados es su arquitectura común: proporcionan modelos computacionales estáticos que carecen de flexibilidad computacional; las características de cálculo de la plataforma no se pueden cambiar.

6. BIBLIOGRAFIA

1. **[Gosling96]** James Gosling, Bill Joy y Guy Seele. The Java Language Specification. Addison-Wesley. 1996.
2. **[Stroustrup98]** Bjarne Stroustrup. “The C++ Programming Language”. Third Edi- tion. Addison-Wesley. October 1998.
3. **[Microsoft95]** “The Component Object Model Specification”. Version 0.9. Micro- soft Corporation. Octubre de 1995.
4. **[Cueva94]** J. M. Cueva Lovelle, P.A. García Fuente, B. López Pérez, C. Luengo Díez y M. Alonso Requejo. “Introducción a la Programación Estruc- turada y Orientada a Objetos con Pascal”. ISBN: 84-600-8646-1.
5. **[Mandado73]** Enrique Mandado. “Sistemas Electrónicos Digitales”. Marcombo Boixareu Editores. 1973.
6. **[Cueva98]** Juan Manuel Cueva Lovelle. “Conceptos Básicos de Procesadores de Lenguaje”. Cuaderno Didáctico número 10. Editorial Servitec. Di- ciembre de 1998.
7. **[Nori76]** K. V. Nori, U. Ammann, K. Jensen, H. H. Nageli y C. Jacobi. “The Pascal-P Compiler: Implementation Notes”. Bericht 10, Eidgengös- sische Technische Hochschule. Zurich, Suiza. Julio 1976.
8. **[Kloosterman98]** Sytse Kloosterman y Marc Shapiro. “Large Scale Distributed Object