Capítulo II

MARCO TEORICO

A continuación se presenta información técnica de los diferentes puntos a tratar en esta tesis, ofreciendo un apoyo en la investigación para obtener los objetivos planteados.

2.1 Agentes de Software

2.1.1Concepto.

Actualmente existen numerosas definiciones acerca del concepto de Agentes. En una de ellas, se establece que los agentes son, desde el punto de vista del usuario final, programas que los asisten y que actúan a su favor. La funcionalidad de los agentes consiste, entonces, en permitirles a los usuarios delegarles tareas que interactúen con información [Large,Oshima98].

Un agente es una entidad computacional (Software, Hardware) que percibe y actúa en un ambiente de manera autónomo, y cuyo desempeño depende de su diseño y en algunas veces de su representación del ambiente y su experiencia [Ayala01].

2.1.2 Características.

Los agentes como entidad computacional tiene numerosas actividades, las cuales son:

- Está situado en un ambiente y es capaz de acciones para las cuales fue diseñado, llevándolas acabo de manera autónoma.

- Los agentes operan en ambientes abiertos, cambiantes de manera impredecibles [Lange, Oshima98]
- Un agente trabaja en beneficios del usuario.
- A los agentes se les delega tareas. La delegación se relaciona con el hecho de que el usuario pueda desentenderse de una tarea en específico y delegarla para que otro ente la realice. En este caso el usuario delega ciertos trabajos al agente indicándole por dónde empezar a trabajar y después sólo solicita los resultados [Lieberman96]
- En términos de programación un agente es un objeto con características particulares.
- Pueden administrar tareas de forma paralela por la red.

2.1.3 Clasificación de Agentes.

En la tecnología de agentes existen diferentes clasificaciones en la que se contemplan los diferentes puntos de vista sobre las propiedades y definiciones de los mismos. En esta taxonomía se consideran en general tres tipos de agentes[Sánchez97]:

- Agentes de programación**.** Son abstracciones empleadas en beneficio del desarrollador de software para conceptuar, diseñar e implementar sistemas complejos, aquí se realza el modelado de los procesos realizados por la computadora.
- Agentes de Usuario. Son abstracciones que permiten la interacción de los usuarios finales con los sistemas. Estos últimos se dividen en: *agentes de información* que ayudan a los usuarios con la manipulación de información en complejos y vastos espacios de datos; *agentes sintéticos* que son aquellos que se basan en introducir caracteres animados o textuales en la interfaz; y *agentes de tareas* que observan las

actividades del usuario, automatizan algunas de sus tareas y otros las ejecutan de manera concurrente con las aplicaciones de su dueño.

- Agentes de Red o Agentes Móviles. Son entidades autónomas que migran a través de la red en ambientes distribuidos, se instalan en un nodo determinado para realizar una tarea específica y emplean sus recursos en beneficio del nodo inicial.

2.2 Agentes Móviles

2.2.1 Concepto.

Un agente móvil es un objeto de software que tiene la capacidad de migrar entre los nodos o dominios de una red en un sistema distribuido. Creado en un ambiente de ejecución, el agente puede transportar su estado (valores de sus atributos que le ayudan a determinar qué hacer cuando éste continúa la ejecución en el destino) y código (instrucciones que el agente debe ejecutar) a otro contexto en donde reanudará la ejecución. Estos agentes aprovechan los recursos del nodo destino en beneficio del nodo que los envió [Lange, Oshima98].

2.2.2 Características esenciales de Agentes Móviles.

- Es un proceso autónomo ya que el agente tiene la capacidad de decidir cuándo, cómo y en qué condiciones migrar a otro nodo de la red [Lange,Oshima98].

- Es persistente ya que se envía como objeto conservando su estado de ejecución, código y datos [Lange,Oshima98].

- Tiene la capacidad de poder suspender su acción (actividad) por diferentes circunstancias, y poder viajar a otro nodo en donde podrá reanudar su proceso o iniciarlo nuevamente [Lange,Oshima98].

- Es comunicativo con su medio como respuesta a los cambios de su ambiente; con otros agentes con el fin de intercambiar información; o bien con su dueño para informarle sobre su estado o resultados.

- Puede enfocarse a realizar tareas ya sea delegadas por su dueño o en respuesta a los cambios que se produzcan en su ambiente.

- Puede ejecutarse sin conexión, pues el agente es capaz de migrar, desactivarse o esperar mientras la conexión a la red se reanuda.

- Puede procesarse en forma asíncrona gracias a que tiene su propio hilo de ejecución. Esta característica le permite trabajar independientemente de los otros procesos que se estén ejecutando en el nodo.

- Puede duplicarse con el simple hecho de generar un clon [Pérez 98].

2.2.3 Diferencia de Agentes Móviles y Agentes Estáticos.

La diferencia es que los Agentes Móviles pueden transportarse a un nodo remoto y obtener beneficios de él en el momento de realizar su tarea, lo cual resulta en forma contraria en los Agentes estáticos, ya que estos se llevan a cabo únicamente en el sistema en donde inició su ejecución. Si éstos necesitan información o requieren interactuar con otros agentes ubicados en otro sistema, entonces hacen uso de mecanismos de procedimientos remotos [Lange, Oshima98].

2.2.4 Lenguajes de programación.

A continuación se mencionarán y se dará una breve explicación de algunos lenguajes de programación adecuados para la implementación de Agentes Móviles [Pérez 98]:

- Aglets.
- Voyager.
- Mole.
- Concordia.
- Odyssey.

Aglets.

Aglets fue desarrollado en los laboratorios de investigación IBM de Tokio por los investigadores Lange y Oshima. Es una librería de clases Java únicamente para Agentes Móviles. En donde cada Aglet es un objeto persistente y transportable que corre de manera asíncrona en un nodo con contexto de ejecución que provee de un ambiente seguro y protegido, tanto al nodo como al aglet [Lange, Oshima98].

Voyager.

Voyager fue creado por ObjectSpace, y es una plataforma que se escribió en Java para el desarrollo de sistemas distribuidos basados en agentes, la cual provee un extenso conjunto de objetos con capacidades de mensajeria, y objetos que se mueven como agentes en una red [ObjectSpace99].

Mole.

Fue creado por el grupo de sistemas distribuidos de la Universidad de Sttutgart, Alemania. Este lenguaje contiene un esquema de nombrado global para agentes, una unidad de manejo de hilos, un director de servicios y esquema de comunicación entre agentes. También considera los conceptos de control, coordinación y terminación de agentes y grupos de agentes, así como la seguridad de estas entidades con transacciones, y su ejecución en nodos dudosos [Baumann99].

Concordia.

Es un marco de trabajo para el desarrollo y manipulación de aplicaciones basadas en agentes móviles desarrollado por Mitsubishi. Este consiste de múltiples componentes, todos basados en Java, que se combinan para proveer un ambiente completo para la generación de aplicaciones distribuidas. Considera a fondo el concepto de movilidad, seguridad, persistencia y colaboración [Mitsubishi00].

Odyssey.

Es una librería de clases Java que permiten al usuario crear sus propias aplicaciones de agentes móviles. Este surgió por General Magic Inc, cuando creó el primer sistema de agentes móviles denominado *Telescript*, cuyo periodo de vida fue muy corto pese a que estaba pensado para trabajar en una arquitectura de red. En respuesta a la popularidad de Internet y al gran auge del lenguaje de programación Java, General Magic decidió reimplementar todos los conceptos considerados en el desarrollo de *Telescript* pero ahora en este lenguaje. Fue así como surgió Odysey [General Magic97].

2.3 Jni

Jini Technology es una infraestructura simple para proveer servicios en una red, y crear interacciones espontáneas entre programas que usen estos servicios. Cuando un usuario interactúa con un servicio, lo hace a través de un objeto Java provisto por el servicio. Este objeto es cargado dentro de un programa del usuario para comunicarse con el servicio aún si nunca ha visto algo así antes (el objeto cargado sabe cómo hacer la comunicación). [Keith,Bill99]

2.3.1 Java

Concepto.

Java es un lenguaje de programación basado en el concepto de contenido ejecutable vía web. El poder de Java descansa en la habilidad de programar aplicaciones en un lenguaje de programación real que puede ser dinámicamente distribuido y usado por cualquier usuario en la Internet [Deitel98].

2.3.2 Jini

Concepto.

Jini es un sistema distribuido que forma una federación de JVMs o Máquinas Virtuales Java. Se pretende que los objetos que forman parte de un sistema o federación Jini ofrezcan servicios que puedan ser utilizados por cualquier usuario/objeto que se conecte a él. Los servicios pueden ser tanto acciones realizadas por dispositivos (hardware), programas de software o distintas combinaciones de ambas [jiniTech99].

Los principales objetivos de la arquitectura Jini son los siguientes:

- Posibilitar que los usuarios compartan servicios y recursos en la red.
- Proveer a los usuarios de un fácil acceso a esos recursos desde cualquier lugar de la red, incluso aunque estos cambien de lugar (flexibilidad)
- Simplificar lo más posible las tareas de implementación, mantenimiento y gestión del sistema tanto de los dispositivos como de los usuarios.

El sistema está orientado siempre a un grupo o grupo de trabajo. Se presupone que todos los miembros del grupo tienen un nivel de confianza al menos básico y un acuerdo en cuanto a administración e identificación.

Jini utiliza el lenguaje Java y aprovecha toda su potencia para la computación distribuida, permitiendo que el código Java y los datos se muevan de una máquina a otra de forma segura, identificando en todo momento las clases y los objetos. Jini a su vez añade mecanismos que hacen más fácil aún el funcionamiento del sistema, para que los usuarios puedan unirse y salir del sistema de forma transparente; también facilita el control de los dispositivos y sus servicios, consiguiendo un sistema totalmente dinámico [Keith,Bill99].

2.3.3 Funcionalidad.

Jini construye una red abstracta llamada federación, de todos los servicios disponibles y los organiza con un sistema dinámico conocido como *lookup servic*e, mismo que se almacena en la red. Al conectarse un usuario a la red, tiene acceso a un ambiente de trabajo que él mismo personalizó, además de una serie de iconos que representan los servicios disponibles en esta red. Para conectarse, Jini asume que todos los equipos tienen cierta memoria y poder de procesamiento, en caso contrario, serán controlados por un hardware

y/o software llamados *proxy* que posee estas dos características, memoria y poder de procesamiento [JiniTech99].

Para integrar los servicios y clientes, Jini utiliza dos protocolos y un servicio, llamados: [Keith,Bill99]

- ***Discovery*** - El dispositivo lanza una señal *multicast* para localizar algún servicio Lookup para así poder identificarse o registrarse como servicio en la red.
- ***Join*** - Una vez encontrado el servicio *Lookup*, un objeto representativo del nuevo dispositivo más sus atributos se cargan en el servicio *Lookup*. Ese objeto es, como no podía ser de otra manera, código en lenguaje Java (y es que Java, ante todo es un lenguaje de programación orientado a objetos): contiene el interfaz con los métodos a los que los usuarios podrán llamar, además de los atributos que también podrán ser manipulados, siempre dependiendo de como se haya implementado el objeto.
- ***Lookup*** - Un cliente quiere ahora buscar un servicio concreto que necesita. Gracias al servicio *Lookup*, localiza un servicio que se ajusta al perfil de sus necesidades, de tal forma que una copia del objeto antes mencionado se carga en el cliente. A partir de este momento, el cliente ya es capaz de invocar a los métodos del servicio y dialogar directamente con el dispositivo.

Estos tres realizan la acción de descubrir el servicio y/o cliente, conectarlo a la red y ponerlo a disposición de todos los usuarios.

2.3.4 Metas de Jini

Jini tiene como metas el crear una red de conectar y trabajar de manera que el usuario podrá conectar un servicio (hardware o software) a la red y tenerlo visible y disponible para

aquellos que quieran usarlo. Conectar algo a la red es todo o casi todo lo que se necesita para explotar el servicio. Borrar la distinción de hardware y software por que el usuario quiere un servicio y no le interesa saber que parte es software y que es hardware tanto como lo que necesita. Un servicio en la red debería estar disponible de la misma manera y bajo las mismas reglas sin importar que esté implementado en hardware, software o combinación de los dos. Habilitar el trabajo en red espontáneamente para que cuando se conecta un servicio a la red y está disponible, puede ser descubierto y usado por clientes y otros servicios. Por último el promover arquitecturas basadas en servicios y traer consigo simplicidad [Keith,Bill99].

2.3.5 Características esenciales de Jini.

- Es un conjunto de interfaces y protocolos para afrontar los "problemas comunes" de los sistemas distribuidos.
- Posibilita las "redes espontáneas" de servicios software integrándolos en grupos de trabajo, o federaciones.
- Posibilita el "auto-saneamiento" cuando uno o más servicios, y/o uno o más clientes dejan de estar disponibles.
- Está construido sobre la plataforma J2EE.
- Ofrece servicios que pueden ser parte de un sistema federado y que ofrecen funcionalidad a cualquiera de sus miembros, facilitando la construcción e implementación de sistemas distribuidos
- JINI es utilizable tanto por servicios de negocio como por dispositivos representados por objetos java [Keith,Bill99].

2.3.6 Java provee a Jini los siguientes módulos:

- Método de Invocación Remota (RMI).

  Las comunicaciones entre servicios pueden llevarse acabo usando *Java Remote Method Invocation* **(RMI**). Es parte de la infraestructura de Jini. RMI provee mecanismos para encontrar, activar y recolectar la basura en grupos de objetos. RMI no sólo permite el paso de datos de objeto a objeto en la red, sino objetos completos incluyendo código. La simplicidad de los sistemas Jini es gracias a la habilidad de mover código en la red, encapsulándolo en objetos[Keith,Bill99].

- Máquina Virtual de Java (JVM).

  La arquitectura Jini depende de muchas propiedades de la JVM como son [Keith,Bill99]:

  - *Homogeneidad*: El código será el mismo donde sea que se necesite correr.
  - *Single Type System*: En la misma plataforma siempre será el mismo gracias a la propiedad anterior. El mismo sistema de escritura puede ser usado para objetos locales o remotos y estos pasados entre ellos mismos.
  - *Serialization*: Los objetos java pueden ser serializados en una forma transportable que puede ser deserializada después.
  - *Code Downloading*: La serialización puede marcar un objeto con un código base: el lugar o lugares de donde el código del objeto será cargado. La deserialización puede entonces cargar el código para un objeto cuando sea necesario.
  - *Safety and Security*: La JVM protege la máquina cliente de virus que pudieran llegar con un código cargado. La descarga de código es restringido a operaciones que la seguridad de la JVM permite.

2.3.7 Seguridad en Jini

El diseño del modelo de seguridad para la tecnología Jini está construido sobre las nociones de un *director* y una *lista de control de acces*o. Los servicios Jini son accesados en nombre de algunas entidades – el director – el cual generalmente rastrea a un usuario del sistema. Los servicios en si pueden requerir acceso a otros servicios basados en la identificación del objeto que implementa el servicio. El que el acceso a un servicio sea permitido o no depende del contenido de una lista de control de acceso que es asociada con un objeto [JinSpec 99].

Cualquier programa en Java que usará código descargable debería inicializar un agente de seguridad, invocando a un paquete llamado System.setSecurityManager() que se encargará de que cualquier clase que es descargada remotamente (a través de un código base que es llevado vía RMI) no realice operaciones que no son permitidas. Si no se inicializa el agente entonces ninguna clase será cargada de no ser que se encuentre en el classpath. Si sólo se prueba localmente entonces posiblemente si funcione ya que las clases si serán encontradas pero definitivamente fallará si se prueba en la red. [JinSpec99]

Cuando se corre un programa con un agente de seguridad, los métodos de seguridad de Java 2 usará (por *default*) una política de seguridad estándar, que es, desafortunadamente, muy restrictiva en cuestión de correr algunas aplicaciones de Jini. Así en la mayoría de los casos será necesario especificar un nuevo archivo de política que permita correr al programa. [JinSpec99].

2.4 Métricas de Software.

En la mayoría de los desafíos técnicos, la medición y las métricas nos ayudan a entender tanto el proceso técnico que se utiliza para desarrollar un producto, como el propio

producto. El proceso se mide para intentar mejorarlo. El producto se mide para intentar aumentar su calidad. Es por esto que se aplicara en este proyecto la métrica que se describe mas adelante para medir el software de cada una de las tecnologías estudiadas en este proyecto, analizando su complejidad y puntos funcionales, los cuales se utilizarán como uno de los diferentes objetivos que abarca el estudio comparativo de las dos tecnologías estudiadas en este proyecto.

En principio, podría parecer que la necesidad de la medición es algo evidente. Después de todo es lo que nos permite cuantificar, y por consiguiente, gestionar de forma más efectiva. Pero la realidad puede ser muy diferente. Frecuentemente, la medición conlleva a una gran controversia y discusión, ¿Cuáles son las métricas apropiadas para el proceso y para el producto?, ¿Cómo se deben utilizar los datos que se recopilan?, ¿Es bueno usar medidas para comparar gente, procesos o productos?. Estas preguntas y otras tantas docenas de ellas siempre surgen cuando se intenta medir algo que no se ha medido en el pasado. Tal es el caso de la Ingeniería de Software (el proceso) y del software (el producto) [Pressman98].

Dedicados a la investigación de las métricas del software han tratado de desarrollar métricas que ofrezcan cantidades completas sobre la complejidad del software. Actualmente existen docenas de ellas, pero si bien es cierto cada una de ellas tiene un punto de definición diferente [Pressman98].

A continuación se presenta el concepto de lo que es en sí una Métrica de Software

2.4.1 Concepto

Una Métrica de Software es la aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo del software y sus productos, para suministrar información relevante a tiempo, así el administrador junto con el empleo de estas técnicas mejorara el proceso y sus productos [Pressman98].

Es una medida cuantitativa del grado en que un sistema o proceso posee un atributo dado. Por lo general relaciona una o más medidas, por ejemplo, el número de errores encontrados por cada mil líneas de código.

Existen varias razones por las que medir el software:

- Para indicar la calidad del producto.
- Para evaluar la productividad de la gente que desarrolla el producto.
- Para evaluar los beneficios (en términos de productividad y de calidad) derivados del uso de nuevos métodos y herramientas de ingeniería de software.
- Para establecer una línea de base para la estimación.
- Para ayudar a justificar el uso de nuevas herramientas o de formación adicional.

Existe un proceso de medición, el cual se puede caracterizar por cinco actividades:

- **Formulación:** La obtención de medidas y métricas del software apropiadas para la representación del software en cuestión.
- **Colección:** El mecanismo empleado para acumular datos necesarios para obtener las métricas formuladas.
- **Análisis:** El cálculo de las métricas y la aplicación de herramientas matemáticas.
- **Interpretación:** La evaluación de los resultados de las métricas en un esfuerzo por conseguir una visión interna de la calidad de la representación.

- **Realimentación:** Recomendaciones obtenidas de la interpretación de métricas técnicas transmitidas al equipo de software [Pressman98].

Las Métricas de Software, pueden englobarse en dos categorías, *medidas directas* y *medidas indirectas*.

Entre las *medidas directas* del proceso de ingeniería del software se encuentran el coste y el esfuerzo aplicado. Entre las medidas directas del producto se encuentran las líneas de código (LDC) producidas, la velocidad de ejecución, el tamaño de memoria y los defectos observados en un determinado periodo de tiempo.

Entre las *medidas indirectas* del producto se encuentran la funcionalidad, la complejidad, eficiencia, fiabilidad, facilidad de mantenimiento y muchas otras más.

2.4.2 Métricas Orientadas a la Función

Las métricas de software orientadas a la función son medidas indirectas del software y del proceso por la cual se desarrolla. En lugar de calcular las LDC, las métricas orientadas a la función se centran en la funcionalidad o utilidad del programa o software para medir su complejidad.

Los puntos de función se calculan rellenando la tabla que se muestra en la figura 2.1. Se determinan cinco características del ámbito de la información y los cálculos aparecen indicados en la posición apropiada de la tabla. Los valores del ámbito de la información están definidos de la siguiente manera:

- **Número de entradas del usuario:** Se cuenta cada entrada del usuario que proporciona al software diferentes dados orientados a la aplicación.

- **Número de salidas de usuario:** Se cuenta cada salida que proporciona al usuario información orientada a la aplicación. En este contexto, la salida se refiere a informes, pantallas, mensajes de error, etc.
- **Número de peticiones al usuario:** Una petición esta definida como una entrada interactiva que resulta de la generación de algún tipo de respuesta en forma de salida interactiva.
- **Número de archivos:** Se cuenta cada archivo maestro lógico (o sea, una agrupación lógica de datos que puedan ser una parte de una gran base de datos o un archivo independiente).
- **Número de interfaces externas:** Se cuentan todas las interfaces legibles por la maquina (por ejemplo, archivos de datos en cinta o disco) que son utilizados para transmitir información a otro sistema.

A continuación se muestra la tabla 2.1, en las cuales son obtenidos los ajustes en la complejidad y la tabla 2.2 en donde se obtiene la suma de todas las entradas funcionales, para obtener el punto de función:

Se evalúa cada factor en una escala de 0 a 5

0	1	2	3	4	5
Sin influencia	Incidental	Moderado	Medio	Significativo	Esencial

1.¿Requiere el sistema copias de seguridad y de recuperación fiables?	
2.¿Se requieren comunicaciones de datos?	
3.¿Existen funciones de procesamiento distribuido?	
4.¿Es critico el rendimiento?	
5.¿Será ejecutado el sistema en un entorno operativo existente y fuertemente utilizado?	
6.¿Requiere el sistema entrada de datos interactiva?	
7.¿Requiere la entrada de datos interactiva que las transacciones de entrada se lleven a cabo sobre múltiples pantallas o variadas operaciones?	
8.¿Se actualizan los archivos maestros de forma interactiva?	
9.¿son complejas las entradas, las salidas, los archivos o las peticiones?	
10.¿Es complejo el procesamiento interno?	
11.¿Se ha diseñado el código para ser reutilizable?	
12.¿Están incluidas en el diseño la conversión y la instalación?	
13.¿Se ha soportado el sistema para soportar múltiples instalaciones en diferentes organizaciones?	
14.¿Se ha diseño la aplicación para facilitar los cambios y para ser fácilmente utilizadas por el usuario?	

Tabla 2.1 Cálculo de ajuste en la complejidad

Parámetro de medida	**Cuenta**	**Factor de Peso**			**Resultado**
		Simple	**Medio**	**Complejo**	
Numero de entradas de usuario					
Numero de salidas de usuario					
Numero de peticiones al usuario					
Numero de archivos					
Numero de interfaces externos					
Cuenta – total ----------→					

Tabla 2.2 Cálculo para Punto de Función

Una vez obtenidos estos datos se aplica la formula del punto funcional:

PF=cuenta-total * [0.65 + 0.01 * SUM (Fi)] [Pressman98]

En donde cuenta-total es la cantidad obtenida en tabla 8.2, (Fi), es la sumatoria de los resultados obtenidos en la tabla 8.1 y los valores 0.65 y 0.01 son valores definidos por el autor de la métrica y ocupados como constantes en esta formula.

Cabe mencionar que los datos constantes manejados fueron obtenidos empíricamente y basados en las aplicaciones de autor de esta métrica Albretch [Pressman98].

Se definen a continuación una serie de atributos que deben acompañar a las métricas de software para resultados más efectivos, así las métricas y las medidas que conducen a ello deben cumplir con las siguientes características fundamentales [Heidi01]:

- **Simple y fácil de calcular:** debería ser relativamente fácil de aprender a obtener la métrica y su cálculo no obligara a un esfuerzo o a una cantidad de tiempo inusuales.
- **Empírica e intuitivamente persuasiva:** la métrica debería satisfacer las nociones intuitivas del ingeniero de software sobre el atributo del producto en cuestión (por ejemplo: una métrica que mide la cohesión de un módulo debería aumentar su valor a medida que crece el nivel de cohesión).
- **Consistente en el empleo de unidades y tamaños:** el cálculo matemático de la métrica debería utilizar medidas que no lleven a extrañas combinaciones de unidades. Por ejemplo, multiplicando el número de personas de un equipo por las variables del lenguaje de programación en el programa resulta una sospechosa mezcla de unidades que no son intuitivamente concluyentes.
- **Independiente del lenguaje de programación:** las métricas deberían apoyarse en el modelo de análisis, modelo de diseño o en la propia estructura del programa. No

deberían depender de los caprichos de la sintaxis o semántica del lenguaje de programación.

- **Un mecanismo eficaz para la realimentación de calidad:** la métrica debería suministrar al desarrollador de software información que le lleve a un producto final de superior calidad.

2.5 Tablas de Decisión.

Las tablas de decisión son usadas para determinar cuales acciones requieren ser tomadas, durante tomas de decisión sobre determinados productos a utilizar.

Una tabla de decisión se divide en dos partes (condiciones y acciones), y formadas por 4 secciones. La parte de condiciones establece todas las condiciones que se aplican a los datos. Las acciones son las acciones distintas que se pueden tomar dependiendo de las condiciones. Una tabla de decisión se construye usando columnas, de forma que cada columna corresponda a una combinación de condiciones [DeMarco 79].

2.5.1 Desarrollo de tablas de decisión.

Para construir tablas de decisión el analista necesita determinar el tamaño máximo de la tabla, eliminando cualquier situación imposible, inconsistencias o redundancias y simplificando la tabla lo más posible. Los siguientes pasos proporcionan al analista un método sistemático para el desarrollo de tablas de decisión [Kendall97]:

- Determinar la cantidad de decisiones que puedan afectar la decisión. Combine renglones que se traslapan, por ejemplo condiciones que son mutuamente excluyentes.

- Determine la cantidad de acciones posibles que puedan ser tomadas. Esta llega a ser la cantidad de renglones en la mitad inferior de la tabla de decisión.
- Determine la cantidad de alternativas de condición para cada condición. En la forma más simple de tabla de decisión habrá dos alternativas (s o n), para cada condición. En una tabla de entradas extendidas puede haber muchas alternativas para cada condición.
- Calcule la cantidad máxima de columnas en la tabla de decisión multiplicando la cantidad de alternativas para cada condición.
- Llene las alternativas de condición. Comience con la primera condición y divida la cantidad de columnas entre el numero de alternativas.

2.6 Sistemas Distribuidos.

Un sistema distribuido es aquel, en que se manejan los sistemas de aplicaciones estructurados para satisfacer los requerimientos de información a todos los niveles y funciones de organización, o de una forma más sencilla, es el que tiene alguno o todos sus componentes (Hardware,Software, Aplicaciones y datos) distribuidos.

El procesamiento de datos distribuidos también está constituido por un grupo de subsistemas (procesamiento de datos, sistemas de control, bases de datos, redes de tele-procesamiento) cada uno de ellos posee sus propias funciones, que operando en conjunto proporcionan al usuario más flexibilidad y respuestas adecuadas a sus necesidades de información [Sape93].

2.6.1 Operaciones de un sistema de información distribuido.

Para explicar las operaciones de un sistema de información distribuida, vamos a definir los diferentes estratos u operaciones que lo componen [Sape93]:

- Estrato de hardware: Está constituido por los procesadores, memorias unidades de entrada/salida, terminales, etc.
- El estrato del núcleo del sistema operativo distribuido: Es el que provee las bases necesarias para apoyar a los componentes conectados en los multiplexores, los mecanismos de protección y seguridad del sistema, los componentes para relacionar la estructura de entrada/salida al modelo de mensajes y los mecanismos básicos de comunicación entre procesos y de sincronización.
- El estrato del sistema operativo distribuido de servicio: Es aquel que provee los servicios de comunicación para las diferentes aplicaciones que residen en el sistema.
- Estrato de aplicaciones: este contiene las diferentes aplicaciones o sistemas de usuario.
- Interfaz: Es la comunicación existente entre los diferentes estratos de un sistema distribuido.

Cada uno de los estratos se construye sobre el estrato inferior de un modo concéntrico, surgiendo una comunicación entre ellos. Estos estratos pueden descomponerse en estratos más sencillos, lo que facilita el diseño e implantación de los sistemas distribuidos.

2.6.2 Tipos de sistemas distribuidos

Según la topología de la red [Sape93]:

Procesos distributivos verticales: Esta configuración esta asociada a un esquema " maestro-esclavo", donde los niveles superiores tienen los computadores más potentes y de mayor capacidad. Cada nodo actúa como maestro para aquellos situados debajo de el y como esclavo para los superiores.

Procesos distributivos horizontales: Aquí se emplean estaciones que tienen todas la misma categoría, y las estaciones o nodos se encuentran conectadas en serie y todas tiene el mismo nivel jerárquico.

Procesos distribuidos combinados: Este proceso dependen de las necesidades especificas de la organización, la cual combina la configuración topología vertical con horizontal.

Según el tipo de distribución.

- Distribución funcional: Aquí el sistema se organiza para efectuar un conjunto de tareas especificas vinculadas entre si de algún manera.
- Distribución geográfica: Por medio de este se organizan los procesos para efectuar algunas tareas especificas en ámbitos geográficos diferentes.

Según las características del hardware y software.

- Homogéneos: Un sistema distribuido es homogéneo cuando el equipamiento elegido, tanto en lo que respecta al hardware como al software, presenta problemas de compatibilidad mínimos, por provenir de una misma "familia" de equipamiento o ser producidos en su totalidad por el mismo fabricante. Esto determina que cuando se desea mantener la homogeneidad del sistema y es necesario una ampliación del mismo, es necesario recurrir a los mismos fabricantes que hicieron la primera provisión del equipamiento.

- Heterogéneos: Nos referimos a cuando el equipo elegido tanto en hardware como en software presentan problemas de compatibilidad y requiere de equipos especiales o software de colectividad para asegurar el funcionamiento del mismo.

2.6.3 Razones para usar un sistema de información distribuida

Debemos utilizar un sistema de información distribuida porque:

- Se logran instalaciones más económicas y modulares. Estas se expanden en forma paulatina, a medida que van incrementándose sus necesidades debido al crecimiento de las empresas u organizaciones.
- Diversifica los puntos de acceso al computador y además permite hacerlos crecer en base a la capacidad creativa de los usuarios y al criterio de la dirección de la organización.
- Evita que se subutilicen los bancos de datos que se forman en las organizaciones, colocando a la disposición de proveedores y clientes, toda la capacidad computacional que de otra manera quedaría desaprovechada.
- Optimiza el acceso a los computadores de gran porte.

2.7 Conclusiones

La información presentada en este capitulo es para llevar a cabo el estudio y comprensión de las dos tecnologías a investigar en este proyecto. De esta manera ayudara a definir las herramientas que se utilizaran en este proyecto y la forma en como se aplicaran para lograr los objetivos específicos. En el caso de Agentes Móviles, se revisaron sus lenguajes de programación existentes y se obtendrá el que mejor características presente

para el estudio del comportamiento del Agente Móvil, en este caso se observo que Aglets y Voyager ofrecen mejores características que los demás lenguajes, así también se lograron definir las métricas que ayudaran a comprar algunos puntos importantes de comparación de estas tecnologías.