Universidad de Los Andes
Facultad de Ingeniería
Postgrado de Computación

Diseño de un Sistema basado en tecnologías multiagente y del conocimiento para el modelado y simulación de Organizaciones

Tesis de Grado presentada por:
Ing. María D. Ferrer G.
para optar al título de:
Magíster Scientiae en Computación

Mérida, 2003

Agradecimientos

A Dios por darme la vida e iluminarme el camino que hace brillar mi vida en éxitos y triunfos.

A mis padres, quienes han llenado mi vida de amor y han derrochado sobre mí, ternura y abnegación.

A mis hermanos por brindarme su apoyo y confianza en los momentos más difíciles.

Al profesor Jacinto Dávila quién con sus conocimientos y constante preocupación, supo orientarme y aclarar los puntos necesarios para la culminación exitosa de este proyecto.

Al profesor Rodrigo Martínez, por brindarme su ayuda para superarme intelectualmente y estimularme para seguir adelante.

A Luis por brindarme su apoyo y colaborar conmigo en la disposición de computadoras, para la búsqueda de información necesaria para la realización de este proyecto.

A Alexander por colaborar conmigo durante la realización de este proyecto, colocando a mi alcance materiales y equipos necesarios para el desarrollo del mismo.

A mis tías, Teresa y María, quienes me han tendido su mano y me han brindado su ayuda incondicional para lograr alcanzar mis triunfos.

Resumen

En este trabajo se presenta el comportamiento de organizaciones, haciendo uso de las tecnologías de agentes y del conocimiento, específicamente se trabaja con un modelo general de una empresa para desarrollo de software y con un modelo de un centro de entrenamiento. El estudio de sistemas sociales artificiales augura muchas ventajas para analizar y obtener datos significativos sobre sociedades y su comportamiento de masas emergentes. Por otra parte, la tecnología de agentes está recibiendo una gran atención en los últimos años y, como consecuencia, la industria está comenzando a interesarse en adoptar esta tecnología para desarrollar sus propios productos. Es por ello, que los agentes y sistemas multiagente son los componentes básicos para el desarrollo de simulaciones de organizaciones.

Este trabajo está enmarcado dentro de la disciplina de la Inteligencia Artificial, específicamente en Sistemas Multiagente, los cuales usan agentes de software que intentan imitar el comportamiento de entes que viven en sociedad, con el objetivo final de alcanzar simulaciones de sistemas tan complejos como las organizaciones humanas. A los modelos propuestos para cada una de las organizaciones señaladas anteriormente, se ha incorporado un componente novedoso, como lo es la Gestión de Calidad. El aspecto metodológico para lograr la incorporación de este componente consistió en la revisión de los diferentes esquemas que proporcionan los elementos básicos para iniciar programas de mejora de los diferentes procesos involucrados en las organizaciones.

Contenido

Lista de Figuras

Lista de Tablas

Introducción

En la última década se han consolidado una serie de áreas de investigación en donde confluyen trabajos en ciencia de la computación y teoría de la organización, dando lugar a especialidades nuevas como teoría computacional de la organización, modelado de empresas y administración de conocimiento. Estas diferentes aproximaciones pueden agruparse en dos grandes corrientes. Por un lado, aquellas cuyo enfoque se centra en el modelado, y por el otro, aquellas que buscan ser herramientas de apoyo a la operación y control de la organización.

El enfoque centrado en el modelado busca desarrollar técnicas que permitan la creación de modelos de organizaciones detallados y completos con el fin de estudiar su desempeño y poder planificar posibles modificaciones en la estructura, responsabilidades y tareas asignadas a cada persona. Asociado a este tipo de enfoque se ha desarrollado un trabajo importante en temas como: ontologías de organizaciones y modelado basado en multiagentes.

Este trabajo versa sobre dos tipos específicos de organizaciones como son una Empresa para Desarrollo de Software y un Centro de Entrenamiento; el primer tipo de organización genera un producto, mientras que el segundo tipo presta o proporciona un servicio. El desarrollo de la simulación de estas organizaciones se realizará utilizando tecnologías de agentes, ya que los agentes y sistemas multiagentes son los componentes básicos para el desarrollo de simulaciones de organizaciones. Además, se incorpora dentro de estas organizaciones la Gestión de Calidad, para asegurar la calidad del producto generado o el servicio prestado.

Este trabajo está enmarcado dentro de la disciplina de la Inteligencia Artificial, específicamente en Sistemas Multiagente (SMA), los cuales usan agentes de software que intentan imitar el comportamiento de entes que viven en sociedad, con el objetivo final de alcanzar simulaciones de sistemas tan complejos como lo son las organizaciones humanas. Este tipo de simulaciones es una gran fuente de conocimiento sobre la dinámica de sistemas

complejos y permite realizar estudios e investigaciones que aportan resultados concluyentes de una manera más eficiente, ya que éstas simulaciones están sometidas al control del investigador.

El objetivo del sistema multiagente, consiste en proporcionar a las organizaciones humanas un sistema de información distribuido que apoye la toma de decisiones a través de toda la organización, y que facilite la coordinación y cooperación a lo largo de los procesos de grupos de trabajo y tareas individuales que se desarrollan en la organización. Los modelos desarrollados en este trabajo, tanto el de la Empresa para Desarrollo de Software, como el del Centro de Entrenamiento, ponen en primer plano la estructura de administración y control de los agentes de cada organización. El principal formalismo de representación del conocimiento está basado en reglas lógicas.

El capítulo 1 trata de las organizaciones y su importancia, de la evolución de la teoría organizacional, de la perspectiva de sistemas, del modelado y simulación de las organizaciones y de la Gestión de Calidad; además, se habla del problema y objetivo específicos. El capítulo 2, trata acerca de los sistemas multiagente en organizaciones, haciendo referencia a las metodologías de diseño orientadas a agentes y de las plataformas existentes de sistemas multiagente para representar organizaciones. El capítulo 3, trata del modelado de una empresa de desarrollo de software. El capítulo 4 hace una descripción y simulación del flujo secuencial de dicha empresa garantizando la calidad del proyecto realizado y también hace referencia al modelado de la empresa basado en sistemas multiagente. El capítulo 5, trata del modelado y simulación de un centro de entrenamiento; nos proporciona una descripción e implementación del flujo secuencial de un centro de entrenamiento y el modelado de dicho centro basado en sistemas multiagente e incorporando en ese flujo la gestión de calidad para garantizar la calidad del servicio ofrecido. Finalmente se hace referencia a los resultados obtenidos, las lecciones aprendidas, los aportes y las perspectivas en esta área.

Capítulo 1

Marco Teórico

La complejidad y diversidad de los problemas asociados al diseño de un sistema basado en tecnologías multiagente y del conocimiento, para el modelado y simulación de organizaciones, hace necesario un análisis y estudio de dichos problemas. Los agentes y sistemas multiagente son los componentes básicos para el desarrollo de simulaciones de organizaciones; los sistemas multiagente usan agentes de software que intentan imitar el comportamiento de entes que viven en sociedad, con el objetivo final de alcanzar simulaciones de sistemas tan complejos como las organizaciones humanas. Este tipo de simulaciones es una gran fuente de conocimiento sobre la dinámica de sistemas complejos y permite realizar estudios e investigaciones que aportan resultados concluyentes de una manera más eficiente, ya que éstas simulaciones estarían sometidas al control del investigador.

A pesar del rápido desarrollo de teorías, arquitecturas y lenguajes de agentes, se ha realizado muy poco trabajo en la especificación y aplicación de técnicas para desarrollar y simular aplicaciones empleando tecnología de agentes. La introducción de la tecnología de agentes en la industria requiere de metodologías que asistan en todas las fases del ciclo de vida del sistema de agentes y la necesidad de estas técnicas, será especialmente requerida a medida que el número de agentes de los sistemas aumente.

En la teoría de sistemas multiagente se discute una diversidad de conceptos considerados desde diferentes puntos de vista como son: reactividad, comunicación, negociación, cognición, entre otros; por lo que han sido usados como técnica integradora de otras áreas como son los sistemas expertos y las redes neuronales artificiales para la resolución de diferentes problemas.

1.1 Las Organizaciones y su Importancia

Las organizaciones constituyen un punto importante a tratar, debido a que en la mayor parte de nuestra existencia hemos pertenecido a una organización u otra: una universidad, un equipo deportivo, un grupo musical o teatral, una organización religiosa o civil, un cuerpo de las fuerzas armadas o una empresa específica. Algunas organizaciones, como el ejército y las grandes corporaciones, tienen una estructura muy formal; otras, como un equipo de baloncesto de escuela, tienen una estructura más informal. Sin embargo, todas las organizaciones, sean formales o informales, están compuestas y reunidas por un grupo de personas que busca los beneficios de trabajar juntas con el propósito de alcanzar una meta común. Por consiguiente, un elemento básico de toda organización es su meta o propósito; la meta puede cambiar – ganar el campeonato de liga, entretener al público, vender un producto, ofrecer un servicio – pero sin una meta, ninguna organización tendría razón de ser.

Las organizaciones son sistemas sociales, complejos y dinámicos y, el estudio de sistemas complejos ha llegado a ser reconocido en los años recientes como una nueva disciplina científica; el último de los campos interdisciplinarios. Este campo está afianzado en los avances hechos en diversas áreas que van desde la física hasta la antropología.

Muchos de los sistemas que nos rodean son complejos; una definición de diccionario de "complejo" es: "consistente de partes interconectadas o interpuestas". Sin embargo, los sistemas simples también están formados por partes. Para explicar la diferencia entre los sistemas simples y complejos, los términos "interconectadas" e "interpuestas" son de algún modo esenciales. Cualitativamente, para entender el comportamiento de un sistema complejo se debe entender no sólo el comportamiento de las partes, sino también cómo ellas actúan juntas para formar el comportamiento del todo, es por eso y porque cada parte puede ser descrita en relación a las otras partes, que no se puede describir el todo sin describir cada parte y que los sistemas complejos resultan difíciles de entender. Esto es relevante a otra definición de "complejo": "no fácil de entender o analizar" .

1.2 Evolución de la Teoría Organizacional

Existen diversas teorías organizacionales o perspectivas, las cuales representan diferentes intentos para explicar el fenómeno organizacional, caracterizándose cada una de estas perspectivas por el uso de ciertas herramientas y por ciertas suposiciones que las fundamentan [18, 13].

La teoría organizacional se inicia en los días de la revolución industrial, sin embargo, los primeros trabajos formales en el área aparecen a finales del siglo XIX (1890 aproximadamente) y principios del XX. Las organizaciones son producto de su momento y su contexto histórico y social; por tanto, la evolución de la teoría organizacional se entiende en términos de cómo han resuelto las personas las cuestiones de sus relaciones en momentos concretos de la historia [6].

Los primeros trabajos en teoría organizacional están considerados como parte de la teoría clásica que perdura hasta los años 1940; ésta teoría de la organización, surgió de la necesidad de encontrar lineamientos para administrar organizaciones complejas, por ejemplo las fábricas. Entre los investigadores más conocidos de esta teoría están Fayol, Taylor y Weber.

La perspectiva clásica surgió, en parte, por la necesidad de elevar la productividad [6], pues a principios del siglo XX, había poca oferta de mano de obra y la única manera de aumentar la productividad era elevando la eficiencia de los trabajadores, es decir, aumentando la capacidad de reducir al mínimo los recursos usados para alcanzar los objetivos de la organización, o sea que, nació de la necesidad de administrar organizaciones complejas y giró en torno a la administración, tratando de identificar los principios y las capacidades básicas de una administración eficaz.

La noción fundamental que guía a los investigadores de esta teoría, es la búsqueda de eficiencia en la empresa [27]. Esta búsqueda de eficiencia ha tenido dos tendencias, algunos autores afirman que su logro depende del grado de cooperación entre las personas

involucradas en la organización, mientras que otros autores la relacionan con la noción de coordinación.

En forma más específica, se tiene que la cooperación dentro de la organización (entre y dentro de los grupos de trabajadores y de gerentes) es un factor clave para elevar el desempeño organizacional. Esta tendencia está relacionada con las ciencias del comportamiento; se propone como parte importante del trabajo gerencial la motivación de los empleados para incrementar su colaboración, siendo ésta conveniente tanto para los gerentes como para los trabajadores. Por otra parte, la coordinación asume que la eficiencia puede ser incrementada mejorando la coordinación dentro de la organización; se habla de buscar los métodos, procedimientos y estructura organizacional convenientes para realizar las funciones organizacionales de forma que exista buena coordinación y así lograr un alto rendimiento de la organización y un balance apropiado entre la diferenciación vertical (jerarquía) y horizontal (división del trabajo) .

Dentro de la teoría clásica surgió un conjunto de principios que se conocen como la teoría de la administración científica; enfoque formulado por Frederick W. Taylor y otros entre los años de 1890 y 1930 que pretendía determinar, en forma científica, los mejores métodos para realizar cualquier tarea, así como para seleccionar, capacitar y motivar a los trabajadores [6]; posteriormente, después de la II guerra mundial (1941 - 1945) y sobre todo en los años 1950, aparece la perspectiva neoclásica que surgió, debido a que el enfoque clásico no lograba suficiente eficiencia productiva ni armonía en el centro de trabajo. Entre los investigadores más conocidos de esta teoría están Simon, Cyert y March. En particular Herbert Simon realiza un trabajo muy importante donde califica de poco realistas las suposiciones más importantes de la teoría clásica y propone nuevos métodos para el estudio organizacional, sugiriendo ideas de la inteligencia artificial. Simon propone, entre otras cosas, elaborar modelos más realistas de los agentes organizacionales proponiendo recurrir a la Psicología Experimental (el individuo es lo más importante de las organizaciones). Las ideas de Simon han sido referencia clave en los nuevos intentos metodológicos para el estudio de las organizaciones y de la economía. Sus ideas y su trabajo lo hacen ganador del premio nobel en economía [24].

Herbert Simon en ‘The Proverbs of Administration’ (1946) [27] ofrece la primera crítica a los principios generales de organización, calificándolos de inconsistentes, inaplicables y poco realistas y, da mucha importancia al proceso de toma de decisiones. Según Shafritz [27], Simon considera que la teoría organizacional debe ser la ‘ciencia de la racionalidad limitada’ de los humanos participantes, quienes persiguen resultados satisfactorios en vez de maximizar la productividad. Simon ha sido el pionero en el modelado de organizaciones, dando énfasis al proceso de toma de decisiones y a la aplicación de métodos cuantitativos, tales como la investigación de operaciones y los métodos computacionales.

Otra escuela organizacional importante es la escuela conductista, la cual se desarrolla entre los años 1950 y 1970. Esta perspectiva se basó en tratar de descubrir de manera sistemática, los factores sociales y psicológicos que crearían relaciones humanas eficaces. Esta escuela fue liderizada por un grupo de investigadores, con estudios de sociología, psicología y campos relacionados, que usan sus conocimientos interdisciplinarios para proponer formas más eficaces para dirigir a las personas en las organizaciones [6]. De aquí que esta perspectiva hizo mayor énfasis en el estudio del comportamiento humano, en la cooperación y en la coordinación, es decir, que la mayoría de los trabajos realizados durante esta escuela están dirigidos a estudiar las interrelaciones entre los individuos, los grupos y el ambiente organizacional, así como también el comportamiento de los individuos en términos del rendimiento organizacional; siendo los temas de estudio de esta perspectiva los siguientes: motivación, comportamiento de grupos y dentro de grupos, liderazgo, dinámica de grupos y cambio organizacional y, entre los investigadores más resaltantes de esta perspectiva se tienen: Elton Mayo, Fritz J. Roethlisberger y William J. Dickson.

Otra perspectiva que comenzó a dominar a la teoría organizacional a partir de 1960 fue la perspectiva de sistemas. Este enfoque en lugar de abordar los diversos segmentos de una organización por separado, piensa que la organización es un sistema único, que tiene un propósito y está compuesto por partes que se interrelacionan. El enfoque de sistemas dice

que la actividad de un segmento de la organización afecta, en diferentes grados, la actividad de todos sus otros segmentos [6].

A partir del año 1990 se ha ingresado en una era de compromiso dinámico donde se ha desarrollado el estudio de una gran variedad de temas organizacionales y la aparición de muchos puntos de vista. Aparecen estudios de la organización relacionados con la cultura, la tecnología de la información, la ecología, entre otras y también, aparecen estudios sobre el modelado computacional de organizaciones, los cuales son utilizados para representar ya sea organizaciones empíricas o modelos más abstractos de organizaciones como aquellos obtenidos de teorías organizacionales. El modelado de las organizaciones empíricas, puede estar orientado a conocer mejor una organización específica, a evaluar cambios, como la introducción de un sistema de información, a entrenar a los gerentes haciendo un análisis de escenarios acerca de las posibilidades de la organización, o a determinar las variables importantes en la toma de decisiones y, en el modelado abstracto de organizaciones, los modelos están orientados a representar aspectos de teorías organizacionales con el fin de verificar la validez y consistencia de las suposiciones de la teoría [18].

1.3 Perspectiva de Sistemas

Los elementos básicos de las organizaciones han permanecido relativamente constantes a través de la historia; las organizaciones tienen propósitos (los cuales pueden ser explícitos o implícitos), atraen participantes, adquieren y asignan recursos para acompañar sus metas, usan alguna forma de estructura para dividir y coordinar sus actividades, y cuentan con ciertos miembros para dirigirlas. Aunque los elementos de las organizaciones han permanecido relativamente constantes, sus propósitos, estructuras, formas de hacer las cosas, y los métodos para coordinar las actividades, han variado ampliamente, estas variaciones reflejan en parte la adaptación de las organizaciones a su ambiente, ya que las organizaciones son “sistemas abiertos” que son influenciados e impactados por el mundo que los rodea, siendo las organizaciones partes inseparables de la sociedad y de la cultura en las cuales ellas existen y funcionan.

La perspectiva de sistemas considera a la organización como un todo, constituida por partes interrelacionadas que persiguen un propósito organizacional común. La teoría de sistemas presenta dos temas o componentes conceptuales principales como son : a) impulsar el desarrollo de una terminología general que permita describir las características, funciones y comportamientos sistémicos y, b) el uso de herramientas y técnicas cuantitativas para entender las relaciones complejas entre las variables organizacionales y ambientales y por lo tanto, optimizar las decisiones [27].

Una gran diferencia entre la teoría organizacional clásica y la teoría de sistemas, es que la primera es unidimensional mientras que la segunda es multidimensional; así mismo, la teoría clásica considera que la organización tiene carácter estático, en tanto que la teoría de sistemas considera, que la organización está inmersa en un proceso donde tanto ella como su ambiente cambian continuamente [27].

En un sentido amplio, la teoría de sistemas se presenta como una forma sistemática y científica de aproximación y representación de la realidad y, en poco tiempo, ha generado un gran interés y se han desarrollado bajo esta perspectiva diversas tendencias, entre las que destacan la cibernética, la teoría de la información y la dinámica de sistemas.

Las herramientas básicas en la teoría de sistemas son el computador, los modelos y los grupos interdisciplinarios de análisis; el computador es usado como una herramienta de experimentación en técnicas de investigación de operaciones tales como la simulación. Precisamente, la investigación de operaciones utiliza métodos cuantitativos, incluyendo herramientas matemáticas y estadísticas, para el estudio de la toma de decisiones en organizaciones y, entre los pioneros en el estudio organizacional usando métodos cuantitativos para analizar los procesos de toma de decisiones, se mencionan a H. Simon, Newell, March y Cyert.

Por otra parte, un sistema es cualquier colección organizada de partes unidas por interacciones preestablecidas y diseñadas para acompañar sus metas específicas o sus

propósitos generales. De esta manera es fácil ver por qué la teoría de sistemas proporciona una perspectiva importante para entender las organizaciones modernas.

El énfasis de esta perspectiva está en entender y describir a la organización como un sistema, es decir, como un conjunto de elementos interrelacionados que persiguen algún fin. Los elementos más importantes usados por esta teoría son las nociones de proceso, de elemento o parte, de interacción, y de objetivo de un sistema. Las preguntas más importantes que se hacen los investigadores de sistemas acerca de las organizaciones están relacionadas con la comprensión no sólo de su estructura estática sino también de su cambio en el tiempo, de su dinámica.

1.3.1 Algunos Conceptos Claves de la Teoría de Sistemas

Muchos de los conceptos de la teoría general de los sistemas están formando parte de la gran mayoría de organizaciones, es por ello que los miembros de las mismas y en especial los gerentes, deben estar familiarizados con este vocabulario para estar al tanto de los avances actuales.

Individuo. Las características más importantes del individuo son sus motivos y actitudes, las cuales describen su comportamiento de acuerdo a lo que él espera de su participación en la organización.

Organización formal. Asignación formal de funciones o asignación de tareas. En este aspecto se le da importancia a la interacción de este concepto o elemento con respecto al anterior, referente a la asignación de roles al individuo, de lo que el individuo espera de la organización formal y de la interacción y adaptación en el tiempo entre ambos elementos.

Organización informal. Son las normas de interacción no expresadas formalmente pero que representan en buena medida la cultura organizacional. Se da importancia a la interacción entre la estructura informal y el individuo, particularmente en cuanto a qué

espera el individuo del comportamiento de los otros individuos en relación a la realización de sus tareas, es decir, respecto a cómo la estructura informal de la organización impone al individuo formas de comportarse y tareas a través de las normas organizacionales, y en cuanto a cómo un individuo contribuye a la formación y cambio de la organización informal.

Subsistemas. Las partes que constituyen un sistema entero. Además, cada sistema puede ser, a su vez, un subsistema de un todo.

Sinergia. Significa que el todo es mayor que la suma de sus partes. En términos organizacionales, sinergia, significa que conforme los departamentos independientes de una organización cooperan e interactúan, resultarán más productivos que si cada uno de ellos actuara en forma aislada. Por ejemplo, en una empresa pequeña, habrá más eficiencia si cada departamento se relaciona con un departamento de finanzas, que si cada departamento tiene su propio departamento de finanzas independiente.

Sistemas abiertos y cerrados. Un sistema es abierto si interactúa con su ambiente; un sistema es cerrado si no lo hace.

Ambiente. Se refiere al área de sucesos y condiciones que influyen sobre el comportamiento de un sistema.

Flujo. Un sistema tiene flujos de información, materiales y energía (incluso energía humana). Estos flujos del ambiente entran en el sistema en forma de insumos (por ejemplo, materias primas), pasan por procesos de transformación en el sistema (operaciones que los modifican) y salen del sistema en forma de productos (bienes y servicios).

Retroalimentación. Son los procesos mediante los cuales un sistema abierto recoge información sobre los efectos de sus decisiones internas en el medio, información que actúa sobre las decisiones (acciones) futuras. Mediante los mecanismos de retroalimentación, los

sistemas regulan sus comportamientos de acuerdo a sus efectos reales y no a programas de salidas fijas (Figura 1.1).

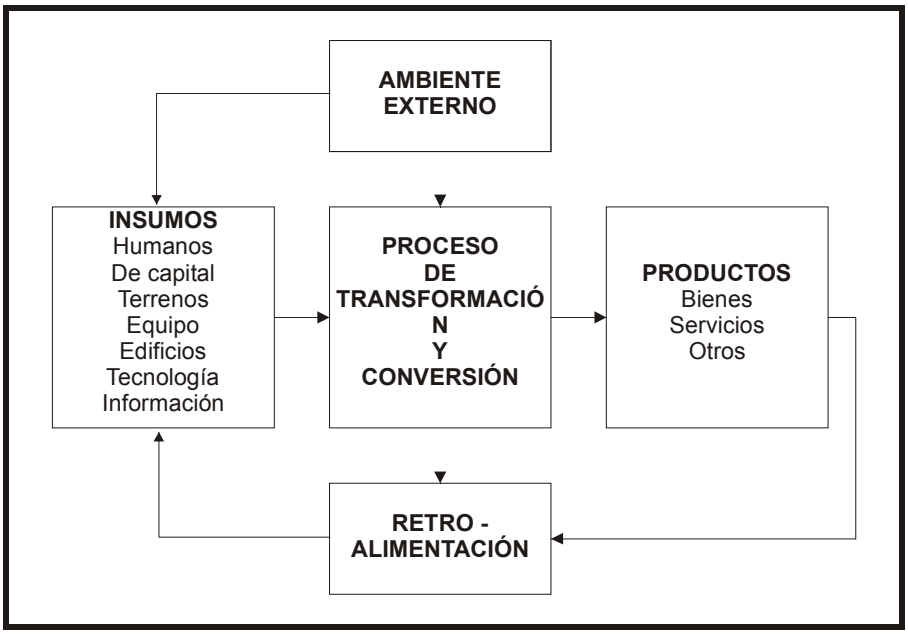


Figura 1.1. Los flujos y la retroalimentación en un sistema abierto

1.4 Modelado y Simulación de las Organizaciones

Los sistemas pueden ser agrupados en: reales, ideales y modelos. Mientras los primeros presumen una existencia independiente del observador (quien los puede descubrir), los segundos son construcciones simbólicas, como el caso de la lógica y las matemáticas, mientras que el tercer tipo, corresponde a abstracciones de la realidad, en donde se combina lo conceptual con las características de los objetos. También, con relación a su origen los sistemas pueden ser naturales o artificiales. Los sistemas sociales artificiales representan un intento por imitar el comportamiento de seres vivos que viven en sociedad, estos sistemas se basan en agentes de software para lograr la simulación de dichas sociedades y para ello, se hace uso de ciertas herramientas, tales como la dinámica de sistemas y los sistemas multiagente.

Un estudio organizacional con dinámica de sistemas implica el modelado de aquellos aspectos que inciden sobre su dinámica de comportamiento. Este proceso de estudio puede ir desde la creación de modelos cualitativos formales, hasta la posible elaboración de modelos matemáticos de simulación.

Actualmente, los sistemas multiagente (SMA por sus siglas) son considerados una vía para modelar y simular sistemas reales [10]. El rango de aplicaciones varía desde sistemas para la gestión del suministro eléctrico hasta sistemas de búsqueda de información en Internet; un nuevo campo en el que se intenta introducir este tipo de sistemas es el de la representación, modelado y sistemas de ayuda en las organizaciones, fundamentalmente organizaciones humanas.

El uso de los SMA para modelar la actuación en organizaciones humanas, es un tema de interés reciente motivado por la necesidad de obtener modelos de comportamiento de individuos inmersos en sociedades, modelos teóricos en los que la interdisciplinaridad (recursos humanos, toma de decisión, gestión del conocimiento, agentes, SMA, psicología del comportamiento, gestión estratégica) es una pieza clave. Para poder realizar un análisis de los SMA como herramienta de modelado de la actuación en organizaciones humanas, la fase inicial debe consistir en delimitar las funciones y características de los diferentes componentes de una organización humana, para establecer cuáles de éstos son susceptibles de su modelado.

1.4.1 Una Definición de Modelos Computacionales

En la actualidad los modelos computacionales constituyen una herramienta imprescindible en el estudio y la solución de una gran variedad de fenómenos y problemas en las ciencias e ingenierías modernas. Estos modelos son el resultado de la implantación en programas de cómputo de los modelos matemáticos que describen el comportamiento de los sistemas. Si bien la manera sistemática para la obtención de dichos modelos

matemáticos lo constituye la teoría de los sistemas continuos, su implementación computacional se hace posible gracias a los métodos numéricos.

1.4.2 Modelos Computacionales de Organizaciones

Se han desarrollado algunos modelos computacionales de organizaciones, entre los cuales se encuentran modelos clásicos; que están orientados a conocer mejor una organización específica, tal como sucede con el modelo presentado por Cohen *et al*. elaborado a principios de los años 70 y destinado a evaluar aspectos de la teoría del comportamiento y, modelos no clásicos; orientados a representar aspectos de teorías organizacionales para verificar la validez y consistencia de las suposiciones de la teoría, tal como sucede con el modelo presentado por Scott Moss (1999).

Dentro de los modelos computacionales clásicos, se tienen los trabajos realizados por Cohen (1972) [6], Masuch *et al.* (1989) [22] y dentro de los modelos computacionales no clásicos se tienen los trabajos propuestos por Carley (1995) [1], Carley *et al.* (1999) [2], Scott Moss (1998) [23]. En el año 1972 Cohen publicó un artículo denominado Anarquías Organizadas, en donde presenta, con ciertas modificaciones, los aspectos más resaltantes de la teoría del comportamiento. Según el autor, muchas de las ideas desarrolladas en el modelo presentado en este artículo, han sido propuestas por investigadores que han estado trabajando en los más recientes enfoques para la época, como lo son Simon y Schelling. Los aspectos del modelo relacionados con la toma de decisiones se han considerado novedosos, sobre todo en aquellos casos en que el objetivo es ambiguo y, también la manera como son activados los diferentes miembros de la organización durante la simulación (participación fluida) .

Posteriormente, en el año 1989 Masuch publicó un artículo en donde la idea principal era mejorar el modelo de anarquías organizadas propuesto por Cohen, agregando ideas de la inteligencia artificial. Entre las desventajas de los modelos previos citados por el autor, se tienen: (1) los individuos no son representados explícitamente; (2) las dificultades para representar de forma adecuada espacios organizacionales.

Las nuevas ideas propuestas por Masuch en su artículo, son: (1) Propone dar mayor importancia a la representación de los individuos como agentes o actores; (2) Uso del paradigma de programación por objetos para intentar modelar con más detalle la organización y su ambiente; (3) Uso de lógica simbólica; se propone el uso de lenguajes de programación declarativos de propósito general, como el Prolog y Lisp. También existen lenguajes de programación declarativos de propósitos más específicos como SDML (Stridly Declarative Modelling Language) y GALATEA, usados para construir modelos de sistemas multiagente en simulación social.

Luego, en el año 1995 Carley publica un artículo donde propone los elementos más importantes para entender el fenómeno organizacional, considerados por el autor, como los agentes y los procesos. Considera que los procesos son generados por la interacción de los agentes y que el comportamiento de los agentes afecta y es afectado por el ambiente externo.

Carley publica un nuevo artículo en 1999, donde hace una revisión específica de los aspectos metodológicos del modelado de organizaciones. Inicialmente, hace una revisión de las nociones de organización, luego examina la utilidad de los métodos computacionales para estudiar organizaciones. Posteriormente, analiza los diferentes elementos en los que se basa el modelado computacional para describir las organizaciones y, finalmente realiza un análisis de los principales lenguajes y paquetes computacionales para modelar organizaciones.

En 1998 Scott Moss publicó un artículo en el cual propone un modelo computacional de organizaciones para manejo de incidentes críticos. El propósito principal de este artículo es demostrar un enfoque empírico para simulación social; en él se modelan los procesos y el comportamiento de administradores de nivel medio de una compañía real. Los agentes elaborados en el modelo presentan características de comunicación, capacidades de aprendizaje y de conocimiento de acuerdo a teorías de conocimiento y de modelado de sistemas sociales. Moss da importancia a los resultados de la simulación que informen acerca del comportamiento de la organización como un todo y, a los modelos de decisión y

procedimientos que puedan surgir como resultado de la aplicación del conocimiento de los agentes.

1.5 Gestión de Calidad

Un sistema de gestión de calidad es la colección de procesos, documentos, recursos y sistemas de monitoreo (supervisión) que dirigen el trabajo de una organización respecto a la calidad de productos y servicios.

1.5.1 Gestión de la Calidad Total

La gestión de calidad total es una gestión sistemática a través de la cual la empresa satisface las necesidades y expectativas de sus clientes, de sus empleados, de los accionistas y de toda la sociedad en general, utilizando los recursos de que dispone: personas, materiales, tecnología, sistemas productivos, entre otras [9].

Como se ha dicho, la gestión de calidad trabaja en la organización en función de la calidad de sus productos y servicios. Existen varias definiciones de calidad, entre las que se tienen las siguientes:

- "Es el grado en que un producto cumple con las especificaciones técnicas que se han establecido cuando fue diseñado."

- "Es la adecuación al uso del producto o más detalladamente, el conjunto de propiedades y características de un producto o servicio que le confieren su aptitud para satisfacer unas necesidades expresadas o implícitas."

- "Todas las formas a través de las cuales la empresa satisface las necesidades y expectativas de sus clientes, sus empleados, las entidades implicadas financieramente y toda la sociedad en general."

1.5.2 Conceptos Fundamentales en Gestión de Calidad Total

Existen algunos conceptos fundamentales en gestión de calidad total, entre los cuales se tienen:

Liderazgo y consistencia en los objetivos. El sistema de gestión de calidad total debe incentivar y motivar a su equipo y a todos los miembros de la organización para mejorar sus procesos y garantizar que las actividades realizadas están acordes con las metas y objetivos de la organización.

Enfoque hacia el cliente. La alta gerencia debe asegurar que los requerimientos del cliente son atendidos y alcanzados, con la meta de mejorar la satisfacción del mismo.

Establecer la política y estrategia. Las políticas de calidad identifican las metas principales del sistema de gestión de calidad. Las políticas de calidad deben: ser apropiadas a los propósitos de la organización; incluir un compromiso para alcanzar los requerimientos del cliente y los requerimientos legales y regulatorios; crear una base para establecer los objetivos de calidad; difundir y comunicar los objetivos de calidad a través de la organización; revisar el desarrollo de las necesidades de la organización y sus clientes.

Gestión por procesos. Tradicionalmente, las organizaciones se han estructurado sobre la base de departamentos funcionales que dificultan la orientación hacia el cliente. La gestión por procesos percibe la organización como un sistema interrelacionado de procesos que contribuyen conjuntamente a incrementar la satisfacción del cliente. Supone una visión alternativa a la tradicional caracterizada por estructuras organizativas de corte jerárquico – funcional, que pervive desde mitad del siglo XIX, y que en buena medida dificulta la orientación de las empresas hacia el cliente.

Modelo de gestión de calidad total. Es una representación descriptiva de los elementos, y sus relaciones a alto nivel, a tener en cuenta para facilitar la explicación, razonamiento y simulación, entre otros, del camino hacia la calidad total.

Mejora continua e innovación. Los modelos de gestión de calidad total apoyan su implantación en un ciclo de mejora continuo, el cual permite introducir y refinar los conceptos de calidad total en la organización de una manera progresiva.

Desarrollo e involucramiento del personal. El sistema de gestión de calidad total proporciona entrenamiento a su personal con la finalidad de implantar efectivamente el ciclo de mejora continuo.

Relaciones de asociación con entidades externas. Relaciones con otras empresas para conciliar un proyecto. Es beneficioso mantener una relación a largo plazo con los proveedores. La relación debe ser de mutuo beneficio.

Orientación hacia los resultados. El ciclo de mejora continuo debe trabajar siempre en función de alcanzar los resultados deseados.

Modelado de procesos. Un modelo es una representación de una realidad compleja. Realizar el modelado de un proceso es sintetizar las relaciones dinámicas que en él existen, probar sus premisas y predecir sus efectos en el cliente. Constituye la base para que el equipo de proceso aborde el rediseño y mejora y establezca indicadores relevantes en los puntos intermedios del proceso y en sus resultados.

Rediseño y mejora de procesos. El análisis de un proceso puede dar lugar a acciones de rediseño para incrementar la eficacia, reducir costes, mejorar la calidad y acortar los tiempos, reduciendo los plazos de producción y entrega del producto o servicio.

Responsabilidad social. Significa ser capaz de satisfacer a la sociedad donde las personas se desenvuelven.

1.5.3 Ciclo de mejora

El ciclo de mejora continuo se describe mediante cuatro etapas básicas [15], como son (Figura 1.2):

- Responsabilidad de la dirección
- Gestión de recursos
- Realización del producto
- Medición, análisis y mejora.

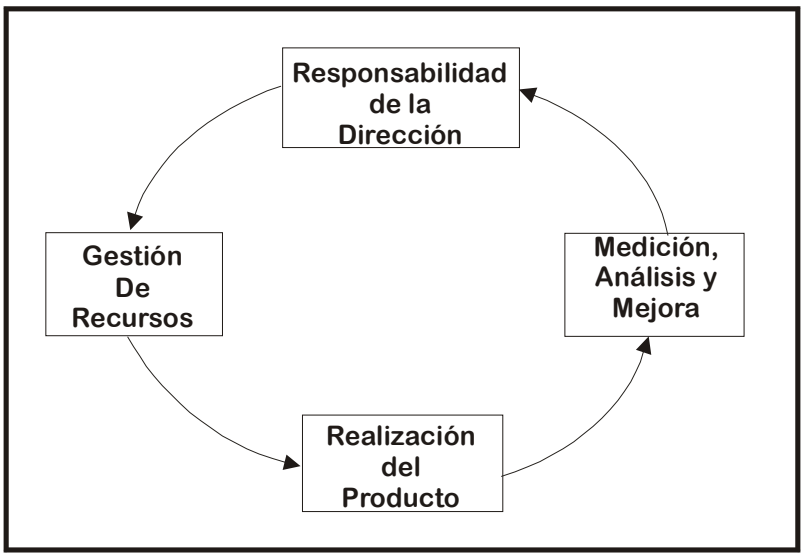


Figura 1.2. Ciclo de mejora continuo

Responsabilidad de la Dirección

Un programa de calidad efectivo requiere de la participación y el compromiso de la alta gerencia de la organización. La alta gerencia tiene las siguientes responsabilidades:

- Supervisar la creación del sistema de gestión de calidad
- Comunicar la importancia de los requerimientos a alcanzar, incluyendo requerimientos del cliente, requerimientos legales y regulatorios.
- Establecer las políticas de calidad y los objetivos de calidad.

- Comunicarse con las partes responsables para la calidad del producto y servicio.
- Proporcionar recursos adecuados para la operación del sistema de gestión de calidad.
- Revisar y supervisar la operación del sistema de gestión de calidad.

Gestión de Recursos

Proporciona las personas, equipo, herramientas y materiales necesarios para construir y mantener el sistema de gestión de calidad y, para mejorar continuamente la efectividad del mismo y alcanzar los requerimientos necesarios. Las personas que realizan el trabajo que afecta la calidad del producto, deben ser competentes para llevarlo a cabo. Esta competencia es lograda a través de una combinación de educación, entrenamiento, habilidades, destrezas y experiencia.

Realización del producto

Es el trabajo que la organización realiza para desarrollar, manufacturar y repartir o entregar los servicios o bienes finalizados. Un sistema de gestión de calidad efectivo incluye un enfoque comprensivo para ir desde el concepto de producto deseado al producto finalizado, es decir, debe conocer detalladamente los procesos necesarios para lograr el producto final.

Medición, análisis y mejora

Consiste en planificar y llevar a cabo la inspección, prueba, medición, análisis y mejora de las actividades necesarias para:

- Asegurar que el producto alcance los requerimientos del producto
- Asegurar que el trabajo del sistema de gestión de calidad sea como fue planeado
- Mejorar la operación y resultados del sistema de gestión de calidad.

También, realiza el monitoreo de la opinión final del cliente del producto y servicio para realizar las medidas correctivas que sean necesarias.

1.6 Problema Específico y Objetivo General

En este trabajo se especificarán e implementarán dos tipos específicos de modelos de organizaciones: una empresa para desarrollo de software y un centro de entrenamiento, utilizando la tecnología de agentes y haciendo las simplificaciones correspondientes, con el propósito de observar el comportamiento de estas organizaciones e incorporando como parte de las organizaciones un componente novedoso importante, como lo es la gestión de calidad en los productos y servicios que estas organizaciones proporcionan.

La metodología que se utiliza para el desarrollo del trabajo, se basa en la revisión de los diferentes esquemas que proporcionan los elementos básicos para iniciar con un programa de mejora o de calidad de los procesos involucrados en las organizaciones, cuyo resultado principal es un producto, como por ejemplo una empresa para desarrollo de software y para organizaciones cuyo resultado principal es un servicio, tal como sucede con el centro de entrenamiento.

Estos esquemas se usarán conjuntamente con las tecnologías y metodologías orientadas a agentes, para generar un modelo de cada una de las organizaciones referidas anteriormente, que permita describir su funcionamiento teniendo en cuenta el aspecto de calidad. Se cree que el modelo puede producir resultados importantes y concluyentes acerca de la dinámica de estas organizaciones considerando en ellas la calidad.

1.6.1 Objetivos Específicos

- Obtención del modelo general de una Empresa para Desarrollo de Software, incorporando en el mismo, la Gestión de la Calidad.

- Descripción del modelo general de la empresa basado en sistemas multiagente, para ello se hace uso parcial de una metodología orientada a agentes: Metodología orientada a agentes para el modelado de empresas.

- Implementación del conocimiento de cada uno de los agentes involucrados en la empresa, en el lenguaje de programación lógica Prolog, pero haciendo uso bajo este lenguaje, de un interpretador de la integración de los lenguajes de programación lógica ACTILOG [16] y OPENLOG [17].

- Simulación del modelo de la empresa basado en sistemas multiagente, bajo el lenguaje y plataforma de simulación multiagente GALATEA.

- Descripción del flujo secuencial seguido por un proyecto desarrollado por la empresa.

- Implementación del flujo secuencial en el lenguaje de simulación GLIDER.

- Obtención del modelo de funcionamiento de un Centro de Entrenamiento, incorporando dentro del mismo, el aseguramiento de la calidad del servicio prestado.

- Descripción del modelo del Centro de Entrenamiento basado en sistemas multiagente.

- Implementación del conocimiento de cada uno de los agentes involucrados en el centro, en el lenguaje de programación lógica Prolog, pero haciendo uso bajo este lenguaje, de un interpretador de la integración de los lenguajes de programación lógica ACTILOG y OPENLOG.

- Simulación del modelo del centro de entrenamiento basado en sistemas multiagente, bajo el lenguaje y plataforma de simulación multiagente GALATEA.

- Descripción del flujo secuencial de un curso dictado por el centro de entrenamiento.

- Implementación del flujo secuencial en el lenguaje de simulación GLIDER.

Capítulo 2

Sistemas Multiagente en Organizaciones

Tanto en los hospitales, como en las escuelas, salas de espera, y organizaciones en general, las personas encuentran que sus acciones afectan y son afectadas por diversas organizaciones; por las normas, procedimientos y miembros de las mismas, por ello, en el mundo organizacional, agentes (humanos y artificiales) necesitan inteligencia social y organizacional. Esta inteligencia organizacional comprende diversas dimensiones, incluyendo capacidades de comunicación, conocimiento acerca de otros y conocimiento acerca de las normas, procedimientos y cultura de la organización, entre otras.

La capacidad de una organización para actuar, es ciertamente dependiente de la inteligencia de los agentes que se encuentran dentro de ella; sin embargo, las organizaciones muestran con frecuencia una inteligencia y un conjunto de capacidades que son distintas de la inteligencia y capacidades de los agentes involucrados en un sistema multiagente que intenta representarlas. No es difícil encontrar sistemas multiagente que muestren patrones y procesos de acción, comunicación y conocimiento, no aleatorios y repetidos, indistintamente de que los agentes sean humanos o no. Dicho de otra manera, algunos sistemas multiagente exhiben características de organización y algunas veces de diseño intencional de organización. Diseños de organizaciones pueden emerger espontáneamente o ser impuestos y, pueden estructurar actividades e intenciones dentro de un sistema o controlar las acciones de un sistema como una entidad corporativa [11].

Existen diferencias y similitudes en los patrones y procesos que conectan a agentes individuales y, en las formas que toman las organizaciones. Al hablar de organizaciones y de alcanzar resultados no alcanzables por agentes individuales, o exhibir capacidades no mantenidas por agentes individuales, las mismas (y en efecto todos los sistemas multiagente) necesitan actuar como procesadores inteligentes de información, capaces de responder como una sola entidad corporativa, y coordinar agentes individuales usando

principios y diseños de organizaciones. Investigaciones en el área de organización computacional, emplean técnicas computacionales para teorizar y analizar acerca de organizaciones y los procesos de las mismas.

2.1 Las Organizaciones y los Sistemas Multiagente

Mientras no haya una sola definición de organizaciones que sea uniformemente aceptada, existen principios generales que son mas o menos compartidos con ellas y, en tal sentido, son caracterizadas como [11]:

- Sistemas resolviendo problemas de larga escala, ayudados de la tecnología existente.
- Comprendidas de múltiples agentes (humanos, artificiales o ambos).
- Ocupadas en una o más tareas; las organizaciones son sistemas de actividades.
- Orientadas a metas (sin embargo, las metas pueden cambiar, pueden no ser fácilmente manejables, y pueden no ser compartidas por todos los miembros organizacionales).
- Capaces de afectar y ser afectadas por su ambiente.
- Teniendo conocimientos, cultura, registros, historias y distintas capacidades por parte de cualquier agente simple.
- Teniendo un rango legal distinto al de los agentes individuales (como entidad corporativa).

Una razón para la existencia de organizaciones es que ellas existen para superar las limitaciones de la representación individual. Desde este punto de vista, hay cuatro limitaciones básicas a saber: cognitiva, física, temporal e institucional.

1. **Limitaciones cognitivas:** Los agentes son visualizados como actores con racionalidad limitada y por lo tanto, deben unirse para alcanzar altos niveles de realización.

2. **Limitaciones físicas:** Los agentes están físicamente limitados; tanto por sus limitaciones fisiológicas como por los recursos de los cuales disponen, por lo tanto,

deben coordinar sus acciones para alcanzar altos niveles de productividad. Toda acción toma lugar en una localización de espacio y tiempo específica y los agentes están limitados en sus accesos a otras localizaciones de espacio y tiempo; esta localidad fundamental de las acciones significa que acción distribuida es fundamentalmente un fenómeno multiagente.

3. **Limitaciones temporales:** Los agentes están temporalmente limitados y por lo tanto, deben unirse para alcanzar metas que pueden sobrepasar el tiempo de vida de cualquiera de los agentes.

4. **Limitaciones institucionales:** Los agentes están legalmente o políticamente limitados y por lo tanto, deben lograr un estatus organizacional para funcionar como un actor corporativo mas que como un actor individual.

Existen o hay ciertas formas para que las organizaciones puedan superar las limitaciones de la representación individual. Los investigadores en esta área se refieren al camino en el cual una organización es organizada como la forma, estructura, arquitectura o diseño de esa organización. Estas investigaciones han mostrado repetidamente, que no hay un solo diseño organizacional correcto o apropiado, que produzca la realización óptima bajo todas las condiciones. Para que un diseño organizacional sea óptimo, depende de una variedad de factores incluyendo las tareas específicas que están siendo ejecutadas, la inteligencia, las capacidades cognitivas, la formación de los agentes, la variabilidad del ambiente, las restricciones legales o políticas sobre el diseño organizacional, y el tipo de resultado deseado (por ejemplo; eficiencia, efectividad, precisión o costo mínimo).

2.2 Teoría Computacional de la Organización

Investigadores en el campo de la teoría computacional de la organización usan las matemáticas y métodos computacionales para estudiar las organizaciones humanas y automatizadas como entidades computacionales. Las organizaciones humanas pueden ser consideradas como inherentemente computacionales ya que cualquiera de sus actividades

transforma información de una forma a otra, y también porque la actividad organizacional frecuentemente está manejando información.

La teoría computacional de la organización intenta entender y modelar dos tipos distintos pero complementarios de organización. El primero, es la organización natural o humana, la cual continuamente adquiere, manipula y produce información a través de la unión de las actividades de personas y tecnologías de información automatizadas y el segundo, la teoría computacional de la organización estudia organizaciones computacionales artificiales, generalmente comprendidas de múltiples agentes distribuidos, los cuales exhiben propiedades organizacionales colectivas (tales como la necesidad de actuar colectivamente, la asignación de tareas, la distribución de conocimiento y capacidades a través de los agentes y las restricciones sobre las conexiones entre los agentes). Los investigadores usan análisis computacional para desarrollar un mejor entendimiento de los principios fundamentales de la organización con múltiple procesamiento de información por parte de los agentes y, de su naturaleza como entidades computacionales. Los objetivos generales de las investigaciones en esta área son construir nuevos conceptos, teorías y conocimiento acerca de organizar y de la organización en forma abstracta, para desarrollar herramientas y procedimientos que permitan la validación y análisis de modelos organizacionales computacionales y para reflejar estas abstracciones computacionales en la práctica organizacional actual, a través de estas herramientas y de estos conocimientos [11].

Estudios recientemente realizados han resultado en un largo número de modelos, cada uno con sus propias características especiales. Muchos de estos modelos se enfocan sobre aspectos específicos del comportamiento organizacional. Los mismos también han resultado en diversos modelos generales que pueden ser usados en un número de aplicaciones para su uso en teoría organizacional. Por ejemplo, un modelo general muy usado de búsqueda de información, toma de decisión y actividad de resolución de problemas en organizaciones, es la búsqueda distribuida.

Las teorías computacionales de la organización están frecuentemente basadas en teorías cognitivas existentes, teorías basadas en conocimiento y teorías de procesamiento de información del comportamiento individual. Las teorías computacionales de la organización extienden esto a un nivel organizacional y dan precisión a la noción de racionalidad limitada, especificando la naturaleza de las fronteras o límites. La perspectiva original de procesamiento de información básicamente nos señala simplemente, que los agentes están racionalmente limitados, que la información está siempre presente en la organización, y que la organización por sí misma llega a ser un sistema computacional. Hoy día hay una perspectiva de procesamiento de información sobre el comportamiento organizacional, cuyos principios básicos sobre organizaciones son:

- **Racionalidad limitada:** agentes organizacionales están racionalmente limitados. Hay dos tipos de límites; límites a capacidades y límites al conocimiento. Las capacidades dependen de la arquitectura cognoscitiva, computacional y física de los agentes. El conocimiento depende de la capacidad del agente para aprender y de la historia intelectual de los agentes. La posición de los agentes en una organización influye en la información a la cual un agente tiene acceso. De esta manera, el conocimiento del agente de cómo hacer tareas específicas, de cómo opera su organización y de cómo opera las organizaciones en general, es una función de las posiciones que el agente ha desempeñado.

- **Información siempre presente:** dentro de las organizaciones largas cantidades de información en diferentes formas están ampliamente distribuidas a través de múltiples agentes. La información puede no ser necesariamente correcta.

- **Orientación a tareas:** las organizaciones y los agentes dentro de ellas continuamente comprometidos en el desempeño de tareas. Las tareas en las cuales una organización y sus agentes están comprometidos, requiere éstos agentes para comunicar, generar, analizar, adaptar o procesar información organizacional usando varias tecnologías y buscar nueva información y nuevas soluciones.

- **Restricciones distribucionales:** desempeño organizacional es una función de qué información es compartida por quienes conforman la organización, y de los procesos de búsqueda para esa información. La cultura de una organización es la distribución del conocimiento y de los procesos a través de los agentes que están dentro de ella. Esta distribución afecta la extensión y el carácter del conocimiento socialmente compartido, del procesamiento de la información de grupo y el análisis de la información concurrente.

- **Incertidumbre:** incertidumbre acerca de los resultados de las tareas, condiciones ambientales y algunos otros aspectos de la vida organizacional que influye en la actividad organizacional.

- **Inteligencia organizacional:** reside en la distribución de conocimiento, procesos, procedimientos a través de los agentes y las conexiones entre los agentes

- **Cambio irrevocable (dependencia de camino):** tanto los agentes como las organizaciones aprenden, su inteligencia es irrevocablemente reestructurada. Esta evolución unidireccional significa que el tipo y el orden en el cual las cosas son aprendidas, pueden tener consecuencias dramáticas.

- **Necesidad de comunicación:** para que la organización funcione como una unidad corporativa, los agentes dentro de ella necesitan comunicarse. Esta comunicación puede tomar lugar explícitamente mediante el envío y recepción de mensajes o implícitamente, percibiendo las acciones de otros.

2.3 Agentes y Sistemas Multiagente

Para modelar una organización, los siguientes factores son modelados, en algún nivel de detalle: los agentes que comprenden la organización, el diseño o estructura de la organización, las tareas que la organización lleva a cabo, cualquier ambiente de la

organización, la transformación material de la organización y/o tecnología de procesamiento de información y las metas de la organización. La mayoría de los modelos en las teorías computacionales de la organización están compuestos de agentes. Estos agentes pueden ser humanos, artificiales o ambos. Los agentes realizan acciones, están obligados por su rol organizacional y pueden tomar decisiones. Las acciones que los agentes son capaces de realizar y las decisiones que pueden tomar, dependen de sus capacidades, conocimientos, de la situación en la cual ellos están sumergidos, y de las tareas que ellos están realizando. En las organizaciones, los agentes tienen algún nivel de capacidad cognitiva y ocupan una posición en su organización. Esta posición define qué tarea o tareas el agente realiza, con quienes el agente debe comunicarse, a quienes el agente debe reportar y quién le reporta al agente, entre otras. Los agentes tienen conocimiento, destrezas y capacidades específicas. Clases de agentes pueden ser definidos por sus diferencias con respecto a la posición, conocimiento, destrezas, capacidades o roles organizacionales. Por ejemplo, una clase de agentes son agentes administradores, otra sería la de agentes trabajadores. El conocimiento de un agente está potencialmente comprendido no sólo por el conocimiento basado en las tareas sino también del conocimiento social u organizacional. Clases de agentes difieren en su arquitectura cognitiva y / o conocimiento y en la capacidad de realizar acciones diferentes. Desde un punto de vista de agente artificial, qué acciones un agente puede realizar es función de las capacidades cognitivas del agente y de su conocimiento.

Finalmente, los sistemas multiagente no son mas que sistemas donde sus actividades son el fruto de la interacción entre entidades relativamente autónomas e independientes, llamadas agentes, que trabajan en la organización según modos complejos de cooperación, de conflicto y de concurrencia, lo que les permite sobrevivir y perpetuarse. Estos sistemas enriquecen el área de inteligencia artificial al usar metáforas sociológicas, tales como de cooperación, de negociación, de grupo y de equipo; y de la biología, tales como auto – organización, adaptación, etc.

2.3.1 Caracterización de los Agentes

Es difícil encontrar en la literatura una definición de agente que sea uniformemente aceptada. Aquí, se presentará la de Ferber [19], quién dice: *un agente es un hardware o software que puede actuar sobre sí mismo o sobre su ambiente. Además, tiene una representación parcial de su ambiente y puede comunicarse con otros agentes. Por otro lado, tiene objetivos individuales y su comportamiento es el resultado de las observaciones, conocimiento, habilidades e interrelaciones que él puede tener con otros agentes o con su ambiente.*

Existen aplicaciones que requieren sistemas capaces de decidir por sí mismos lo que necesitan hacer para satisfacer sus objetivos de diseño. Tales sistemas son conocidos como agentes. Los agentes que deben operar robustamente en un ambiente rápidamente cambiante, impredecible o ambientes abiertos, donde hay una posibilidad significativa que las acciones puedan fallar, son conocidos como agentes inteligentes o algunas veces como agentes autónomos. En general, un agente puede ser descrito por:

- **Sus tareas:** es decir, el conjunto de actividades de los agentes, las cuales le permiten cumplir sus objetivos y / o funciones.

- **Sus conocimientos:** es decir, las reglas que siguen los agentes para realizar sus tareas. Dicho conocimiento, puede ser adquirido según alguna de las siguientes técnicas: especificadas por el diseñador (a través de algún formalismo), incorporadas dinámicamente durante su evolución, proveniente de otras fuentes de conocimiento (agentes, etc.), o derivarse de sus propios mecanismos de aprendizaje.

- **Su comunicación:** definen su forma de interacción con el ambiente y con los otros agentes, por consiguiente, se deben definir los lenguajes, protocolos de comunicación, entre otros.

La figura 2.1 muestra un modelo de agente que abarca los aspectos mencionados anteriormente.

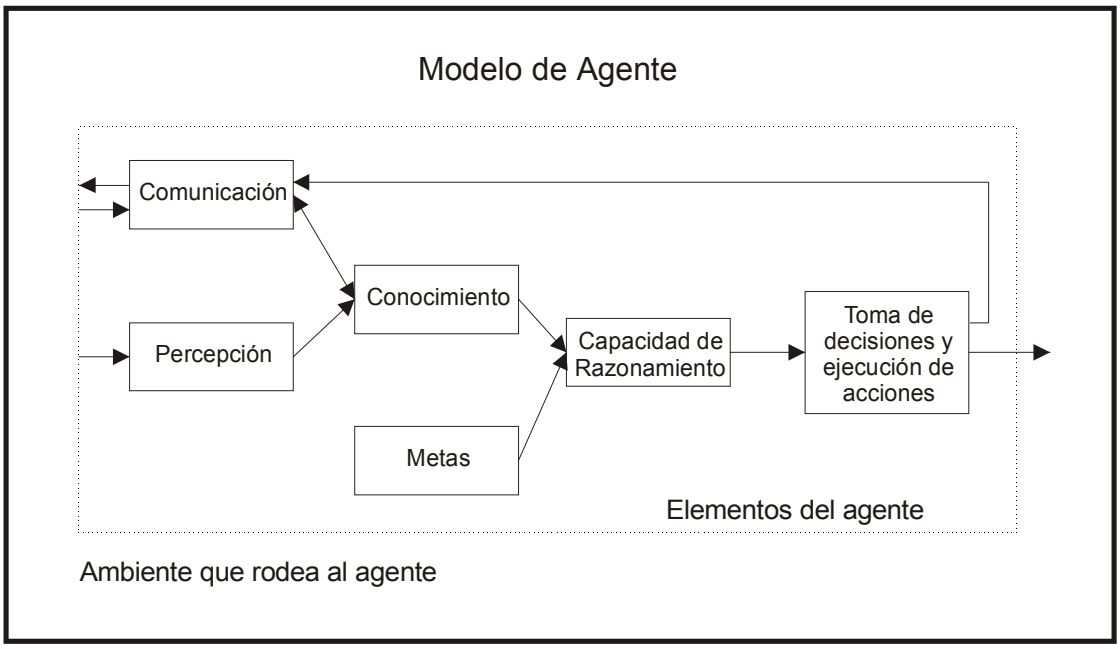


Figura 2.1. Modelo de Agente

Los agentes realizan un ciclo continuo de observación y acción. Perciben la información a través de sus sentidos y la transforman en conceptos que les permiten definir la situación en la cual se encuentran. De esta manera, comprueban si el objetivos perseguido ha sido alcanzado, y en caso de una respuesta negativa deciden, según ciertos criterios, cuáles acciones deben ejecutarse para lograr la meta, y a continuación, ejecutan dichas acciones.

2.3.2 Sobre Arquitecturas para Agentes Inteligentes

Existen diversas arquitecturas para representar el conocimiento de un agente inteligente. En la literatura encontramos arquitecturas abstractas, entre las cuales se señalan los agentes puramente reactivos, percepción y agentes con estado. También existen arquitecturas concretas para agentes inteligentes, entre las cuales se tienen arquitecturas basadas en lógica, arquitecturas reactivas, arquitecturas basadas en creencias, deseos e intención (BDI por sus siglas en inglés) y arquitectura por capas.

En este trabajo, la arquitectura cognoscitiva de los agentes para representar su conocimiento, será la arquitectura basada en lógica. El enfoque tradicional para construir sistemas artificialmente inteligentes (conocido como Inteligencia Artificial Simbólica) sugiere que el comportamiento inteligente puede ser generado en un sistema dándole al mismo una representación simbólica de su ambiente, su comportamiento deseado y cómo manipular esta representación. Estas representaciones simbólicas son fórmulas lógicas, y la manipulación sintáctica corresponde a la deducción lógica.

2.4 Metodologías de Diseño Orientadas a Agentes

La investigación en metodologías orientadas a agentes es un campo incipiente y debido a la relación del paradigma de la orientación a agentes con la orientación a objetos y con los sistemas basados en conocimiento; las metodologías orientadas a agentes no han surgido como metodologías totalmente nuevas, sino que se han planteado como extensiones tanto a metodologías orientadas a objetos como a metodologías de ingeniería del conocimiento [3].

Existen algunas razones para justificar la extensión de metodologías orientadas a objetos, hacia metodologías orientadas a agentes. En primer lugar, ambos paradigmas comparten el pase de mensajes para comunicarse y el empleo de herencia y agregación para definir su arquitectura. Las principales diferencias estriban en que los mensajes de los agentes tienen un tipo predeterminado (su acto comunicativo) y en que el estado mental del agente se basa en sus creencias, intenciones, deseos, acuerdos, etc. Otra posible ventaja proviene del empleo habitual de los lenguajes orientados a objetos para implementar sistemas basados en agentes porque se consideran un entorno natural de desarrollo. También se puede citar como ventaja la popularidad de las metodologías orientadas a objetos y que han sido empleadas en la industria con éxito como por ejemplo OMT (*Object Modeling Technique*), OOSE (*Object - Oriented Software Engineering*) y UML (*Unified Modeling Languaje*).

En cuanto a las tres vistas comúnmente empleadas en las metodologías orientadas a objetos para analizar un sistema, también son interesantes para describir a los agentes: una

vista estática para describir la estructura de los objetos/agentes y sus relaciones; una *vista dinámica* para describir las interacciones entre objetos/agentes; y una *vista funcional* para describir el flujo de datos entre los métodos/tareas de los objetos/agentes.

Por otra parte, a pesar de las similitudes entre los paradigmas de la orientación a objetos y a agentes, obviamente, los agentes no son simplemente objetos, los objetos tienen una estructura simple (atributos y métodos), mientras que los agentes tienen una estructura compleja. Aunque tanto los objetos como los agentes emplean paso de mensajes para comunicarse, mientras que en los objetos el pase de mensajes se traduce en invocación de métodos, en los agentes este pase de mensajes se suele modelar como el intercambio de un conjunto predeterminado de actos de habla, con protocolos asociados para negociar o responder a cada acto comunicativo. Además, los agentes realizan un procesado de los mensajes, y pueden decidir ejecutar o no, la acción correspondiente al mensaje recibido.

Dentro de las metodologías orientadas a agentes como extensión del paradigma de orientación a objetos, se tienen:

1. **Análisis y diseño orientado a agentes de Burmeister:** este método define tres modelos para analizar un sistema de agentes: el modelo de agente, que contiene los agentes y su estructura interna (creencias, planes, objetivos, etc); el modelo de organización, que describe las relaciones entre los agentes (herencia y papeles en la organización); y el modelo de cooperación, que describe las interacciones entre los agentes.

2. **Técnica de modelado de agentes para sistemas de agentes BDI:** este método define dos vistas (externa e interna) para modelar a los agentes BDI (Belief, Desire, Intention). La vista externa consiste en la descomposición del sistema en agentes y la definición de sus interacciones. Esto se lleva a cabo a través de dos modelos, el *modelo de agente*, que describe las relaciones jerárquicas entre clases de agentes y las relaciones entre agentes concretos; y el *modelo de interacción*, que

describe las responsabilidades, servicios e interacciones entre los agentes y los sistemas externos.

La vista interna realiza el modelado de cada clase de agente BDI a través de tres modelos: el *modelo de creencias*, que describe las creencias sobre el entorno, el *modelo de objetivos*, que describe los objetivos que un agente adopta o los eventos a los que responde; y el *modelo de planificación*, que describe los planes que un agente puede emplear para alcanzar sus objetivos.

3. **Método basado en escenarios multiagente (MASB, por sus siglas en inglés):** Este método propone el desarrollo de sistemas multiagente en el campo del trabajo cooperativo. La fase de análisis consta de las siguientes actividades: descripción de escenarios, descripción funcional de los papeles, modelado conceptual de los datos y del mundo, modelado de interacción sistema – usuario.

4. **Metodología orientada a agentes para modelado de empresas** [8]**:** esta metodología propone la combinación de metodologías orientadas a objetos OOSE (Object Oriented Software Engineering) y metodologías de modelado de empresas IDEF (Integration Definition for Function modelling) y CIMOSA (Computer Integrated Manufacturing Open System Architecture). Los modelos propuestos son:

   - Modelo de funciones: describe las funciones (entradas, salidas, mecanismos y control), empleando diagramas IDEF que incluyen la selección de los métodos posibles dependiendo de la entrada y el control.

   - Modelo de casos de uso: describe los actores involucrados en cada función, empleando notación de casos de uso OOSE.

   - Modelo dinámico: se utiliza para analizar las interacciones entre objetos.

- Sistema orientado a agentes: se compone de las siguientes fases: identificación de agentes, protocolos de coordinación, invocación de planes, creencias, sensores y actuadores.

Por otra parte, las metodologías de ingeniería del conocimiento pueden proporcionar una buena base para modelar sistemas multiagente ya que estas metodologías tratan del desarrollo de sistemas basados en conocimiento. Dado que los agentes tienen características cognitivas, estas metodologías pueden proporcionar las técnicas de modelado de la base de conocimiento de los agentes.

La definición del conocimiento de un agente puede considerarse un proceso de adquisición de conocimiento, y dicho proceso sólo es abordado por estas metodologías. La mayor parte de los problemas planteados en las metodologías de ingeniería del conocimiento también aparecen, obviamente, en el diseño de sistemas multiagente: adquisición del conocimiento, modelado del conocimiento, representación y reutilización. Sin embargo, estas metodologías conciben un sistema basado en conocimiento como un sistema centralizado. Por tanto, no abordan los aspectos distribuidos o sociales de los agentes, ni en general su conducta emprendedora, dirigida por objetivos.

Dentro de las metodologías orientadas a agentes como extensión de las metodologías de ingeniería del conocimiento, se tienen:

1. **Metodología CoMoMAS:** expresa Contribution to Knowledge Acquisition and Modelling in a Multi – Agent Framework. Esta metodología propone una extensión de CommonKADS para modelar sistemas multiagente, a través del desarrollo de los siguientes modelos: modelo de agente, modelo de la experiencia, modelo de tareas, modelo de cooperación, modelo del sistema y modelo de diseño.

2. **Metodología MAS – CommonKADS:** extiende los modelos definidos en CommonKADS, añadiendo técnicas de las metodologías orientadas a objetos (OOSE, OMT) y de ingeniería de protocolos (SDL y MSC96) para describir los protocolos entre

agentes. La fase de análisis se basa en los siguientes modelos: modelo de agentes, modelo de tareas, modelo de la experiencia, modelo de organización, modelo de coordinación, modelo de comunicación y modelo de diseño, el cual comprende el diseño de la red, el diseño de los agentes y el diseño de la plataforma.

2.5 Análisis de Sistemas Multiagente para Representar Organizaciones

En la actualidad, los sistemas multiagente son considerados una vía para modelar y simular sistemas reales. Un nuevo campo en el que se intenta introducir este tipo de sistemas es en la representación de organizaciones, fundamentalmente organizaciones humanas [10]. La cantidad de aspectos a tener en cuenta a la hora de representar una organización es muy elevada, por lo que para ello, se han hecho análisis a objeto de estudiar la capacidad de una serie de sistemas multiagente para representar organizaciones, bajo ciertos aspectos que se han considerado los más importantes y por medio de los cuales se puede distinguir y analizar mejor una organización [7].

- Enfoque composicional de la organización
- Proceso de definición del objetivo común de la organización
- Cooperación
- Planificación y modelado de la cooperación y toma de decisiones
- Dinámica de los agentes y del sistema.

El enfoque composicional de la organización abarca tanto la estructura global de la organización, como la estructura interna de un individuo perteneciente a la misma. En cuanto a la estructura interna de los individuos de las organizaciones, se habla de analizar cómo, en cierto modo, modelar personas mediante el uso de agentes, lo cual entraña una gran complejidad.

Con respecto al proceso de definición del objetivo común de la organización, en las organizaciones humanas, la definición del objetivo común se realiza generalmente mediante alguna forma de evaluación de varias posibilidades, bien sean consensuadas, elegidas por votación o establecidas por la autoridad responsable de una organización. Por tanto, suele haber una fase de evaluación de varias posibilidades y una deliberación para ver qué objetivo es el que se desea perseguir.

Los sistemas multiagente son diseñados para realizar una tarea en cuestión o para modelar un aspecto en concreto, por lo que no suele existir un proceso mediante el cual los agentes deciden el fin común de la comunidad de agentes, sino que se ayudan a conseguir sus metas particulares, es decir, cumplir las tareas para las que han sido diseñados. Esto tiene bastante relación con las organizaciones, pues sus miembros suelen colaborar para cumplir las tareas que los altos cargos han establecido y asignado y, sus miembros no deciden el objetivo común de la misma, sino que participan de su consecución, realizando una serie de tareas de acorde a sus capacidades y responsabilidades dentro de la organización. Los agentes del sistema (o miembros de una organización) pueden acordar algún fin intermedio entre varios de ellos, cosa que ocurre cuando colaboran, pero no pueden fijar el fin último del sistema u organización.

El aspecto de cooperación contempla la cooperación de los agentes del sistema para conseguir la realización exitosa de una tarea u objetivo en particular. Además de la cooperación, otro acto comunicativo, es el compartimiento de información entre distintos agentes del sistema, así un agente, además de pedir cooperación para resolver una tarea, puede pedir ayuda para localizar cierta información que es conocida por algún otro agente.

En el aspecto de planificación y modelado de la cooperación y toma de decisiones, analiza en mayor profundidad cómo se realiza el proceso de cooperación en los sistemas multiagente existentes, los pasos que se siguen desde que se solicita la ayuda, hasta que la cooperación ha concluido, bien teniendo resultados positivos o no. En una organización, la cooperación también suele conllevar un proceso de negociación mediante el cual cada

individuo que toma parte de la misma, expone su disponibilidad para cooperar, así como las condiciones bajo las cuales estaría dispuesto a prestar su cooperación o ayuda.

Finalmente, el aspecto de la evolución dinámica de los agentes y del sistema, se centra en analizar cómo varía el comportamiento y el conocimiento de los agentes a través del tiempo, con motivo de las relaciones con su entorno y otros agentes y ver además, si esto afecta al comportamiento global del sistema, si son capaces de aprender nuevas actividades y demás aspectos que se dan en organizaciones humanas, ya que un miembro de una organización humana seguramente mejorará su rendimiento, a medida que vaya adquiriendo experiencia en realizar una serie de tareas.

Los sistemas multiagente sobre los cuales se ha hecho el estudio bajo los aspectos señalados anteriormente, son: MAGSY, GRATE, DESIRE, RETSINA, MASK, INTERRAP [10].

No se especificará a detalle el resultado obtenido del análisis de cada uno de estos sistemas multiagente, pero en términos generales, se tiene que el principal aspecto negativo que se puede encontrar en el sistema MAGSY es que el conocimiento de los agentes se representa de forma totalmente plana y poco estructurada, ya que un agente MAGSY viene definido por los siguientes elementos: conjunto de hechos que conoce, reglas que definen sus estrategias de comportamiento y conjunto de servicios que ofrece el agente. De esta manera, un agente conoce unos hechos y sigue unas reglas atendiendo a unos servicios. Esto puede ser útil para agentes de tipo reactivo, que ante una variación, actúan según sus reglas de comportamiento, pero no parece en principio muy adecuado para modelar individuos que pertenezcan a organizaciones reales. Un individuo no se mueve únicamente por los hechos que conoce y por unas reglas de comportamiento, sino que se apoya en más elementos como pueden ser sus deseos o sus intenciones por citar un tipo de arquitectura de agente de amplia difusión como es la arquitectura BDI.

Se ha comprobado que MAGSY es un sistema eficiente y conveniente de implementar SMA, aunque su aplicabilidad a organizaciones humanas es mas bien escasa. Otro aspecto

no muy positivo es que un agente puede crear otros agentes según desee y sin que otros agentes conozcan su existencia, lo cual no es muy aconsejable dentro de una organización humana.

En cuanto al sistema GRATE, algunas consideraciones deben ser realizadas. En primer lugar, un aspecto importante de esta arquitectura es que la cooperación se establece para que un agente le proporcione a otro cierta información o le resuelva alguna tarea que no es capaz de realizar. Segundo, un factor a tener en cuenta en esta arquitectura es, que el agente que origina la petición, decide con qué agentes quiere cooperar, lo cual es también muy propio del mundo real. Tercero, existen dos protocolos de cooperación, el cliente – servidor (entre dos agentes) y el contrato de red (todos los agentes del sistema intervienen). Cada agente del sistema intenta cumplir con las tareas para las que ha sido diseñado. El módulo de evaluación de cada agente decide qué tarea realizar en cada momento y si es una tarea que no puede realizar localmente, entonces le indica al módulo de cooperación que busque a algún agente que pueda realizarlo.

En el sistema DESIRE el conocimiento se presenta estructurado, ya que se distingue entre habilidades, creencias, intenciones, motivaciones, mundo exterior, etc. El sistema DESIRE está organizado en forma de una descomposición jerárquica de tareas (que pueden ser delegadas) que van a ser realizadas por los agentes del sistema, lo que implica que es necesaria una alta coordinación de las tareas al intervenir generalmente más de un agente. Los agentes de este sistema están en contacto con el mundo exterior, recogen información del mismo, lo cual es un aspecto altamente positivo para el funcionamiento del sistema y modelado de organizaciones humanas, porque los miembros de estas organizaciones están en un contacto casi permanente con el mundo exterior, por ser sistemas abiertos.

El sistema RETSINA tiene una serie de carencias impuestas principalmente por la especificidad con la que fue diseñado. Este sistema fue concebido para realizar búsquedas inteligentes de información en diversas fuentes, lo cual determina fuertemente el tipo de sistema a construir y sus características. La organización de un sistema RETSINA se basa en la autoorganización y organización por demanda, de forma que es una arquitectura que

presenta cierta robustez ante ciertos fallos, porque los agentes podrán buscar nuevas vías para resolver sus tareas pendientes.

El sistema MASK, más que un sistema multiagente es una plataforma para el diseño y construcción de sistemas multiagente. Uno de los aspectos positivos es que permite definir múltiples tipos de organizaciones, aunque la estructura debe ser fijada en tiempo de diseño del sistema que queremos construir. Por lo tanto, cuando se diseña un sistema multiagente usando MASK, se tiene la posibilidad de definir el modelo de los agentes, el modelo organizativo del sistema, el entorno en el cual se va a ubicar el sistema y las interacciones que van a existir durante la evolución del mismo. Así pues, se pueden definir múltiples tipos de agentes según el tipo de organización que se quiera construir.

Otro aspecto interesante que presenta MASK con miras a representar organizaciones humanas a través de un sistema multiagente es que en cada momento cualquier agente puede conocer qué agentes están activos en el sistema y cuando éstos dejan de estarlo, de forma que los agentes pueden tomar las mejores decisiones posibles en cada instante, con el fin de conseguir tanto sus objetivos como los de la organización.

En el sistema INTERRAP no existe una estructura organizacional, sino que todos los entes del sistema están a un mismo nivel y se relacionan con el resto cuando se requiere de la resolución cooperativa de alguna tarea, lo cual no es del todo válido para modelar organizaciones humanas, ya que no se suele dar en ellas este tipo de estructuras. La división de las creencias de los agentes en tres grupos, también es un aspecto muy interesante y adecuado para modelar individuos de organizaciones humanas, ya que distingue entre creencias sobre sí mismo, creencias sobre el mundo exterior al sistema y sobre los demás miembros del sistema.

Otra plataforma de simulación reciente es GALATEA, la cual es presentada como un lenguaje para modelar sistemas multiagente a ser simulados en una plataforma multiagente DEVS. GALATEA es el producto de dos líneas de investigación: lenguajes de simulación basados sobre la teoría de simulación de Zeigler y agentes basados en lógica. Hay en

GALATEA un propósito para integrar, en la misma plataforma de simulación, herramientas conceptuales y concretas para simulación multiagente, distribuida, interactiva, continua y de eventos discretos. GALATEA es también un descendiente directo de GLIDER, un lenguaje de simulación basado en DEVS, el cual incorpora herramientas para modelado continuo. En GALATEA, GLIDER es combinado con una familia de lenguajes de programación lógica, específicamente diseñado para modelar agentes.

2.6 Ontologías

Un fenómeno moderno en el campo de la teoría representacional puede ser encontrado en el uso de ontologías para representar el conocimiento de los agentes [8]. Las ontologías son expresiones declarativas (manejo de datos) del mundo de un agente: los objetos, operaciones, hechos y reglas que constituyen el espacio lógico dentro del cual un agente se desempeña o se desenvuelve.

Las ontologías son una de las principales aproximaciones utilizadas en el ámbito de la gestión del conocimiento y la inteligencia artificial para resolver cuestiones relativas a la semántica. Con el incremento de información disponible dentro de las organizaciones y el constante flujo de información que hay entre personas y sistemas, la gestión del conocimiento, se ha convertido en una parte importante de toda organización. Las ontologías formales proporcionan una forma de compartir y reutilizar conocimiento entre personas y sistemas de aplicación heterogéneos.

2.6.1 Una definición de ontología

El término ontología tiene su origen en la filosofía y proviene del intento de Aristóteles por hacer una clasificación de las cosas en el mundo. En el campo de la inteligencia artificial, algunas definiciones han sido dadas al término de ontología, aunque la más aceptada es una que está dada en (Gruber, 1993) [27], donde es definida “una especificación explícita y formal de una conceptualización compartida”

- Es explícita porque define los conceptos, propiedades, relaciones, funciones, axiomas y restricciones que la componen.

- Es formal porque es legible e interpretable por máquinas.

- Es una conceptualización porque es un modelo abstracto y una vista simplificada de las entidades que representa.

- Finalmente, es compartida porque ha habido un consenso previo sobre la información, que ha sido acordado por un grupo de expertos,

En resumen se puede decir que una ontología es la definición de un conjunto de conceptos, su taxonomía, interrelación y las reglas que gobiernan dichos conceptos.

2.6.2 Características de las ontologías

Las ontologías proporcionan un entendimiento común y compartido de un dominio para facilitar la comunicación entre las personas y sistemas de las aplicaciones. Las ontologías son contenidos teóricos sobre objetos reales de interés (conceptos, entidades, eventos, acciones o procesos), las propiedades de los objetos y las relaciones entre ellos y un determinado dominio. Una ontología contiene definiciones que nos proveen del vocabulario para referirse a un dominio. Las definiciones dependen del lenguaje que se usa para describirlas.

Algunas de las características de las ontologías son:

- **Claridad:** una ontología debe ser capaz de comunicar de manera efectiva el significado de sus términos. Las definiciones deben ser objetivas y comentadas en lenguaje natural.

- **Coherencia:** una ontología debe permitir hacer inferencias que sean consistentes con las definiciones.

- **Multiplicidad de la representación:** un concepto puede ser representado de muchas formas, por lo que pueden coexistir múltiples representaciones de un mismo concepto.

- **Mapeo de ontologías:** establecer relaciones entre los elementos de una o más ontologías, para establecer conexiones, especializaciones, generalizaciones, entre otros.

La principal ventaja que aporta el uso de las ontologías en la representación del conocimiento, es que permite la reutilización y el compartimiento del conocimiento, en un formato tratable computacionalmente [7]. El conocimiento incluido en una ontología puede ser compartido, ya que este conocimiento suele ser consensuado, es decir, ha sido adoptado por un colectivo de personas y no por un individuo aislado [28].

Capítulo 3

Modelado de una Empresa para Desarrollo de Software

En este capítulo se pretende obtener un modelo general de una empresa para desarrollo de software, incluyendo en dicho modelo, la gestión de calidad, de tal manera que la empresa logre desarrollar software de calidad. La especificación del modelo de la empresa se realizará empleando una técnica de modelado, como lo es la notación IDEF0 [14].

El término software significa: "aquellos programas, procedimientos, reglas y documentación asociada con la computación, así como los datos pertenecientes a la operación de un sistema de computación" (IEEE, 1983).

El software es un objeto abstracto, producto del intelecto del hombre y, sus propiedades son:

- No es visible, no es tangible.
- No tiene propiedades físicas, tales como volumen, peso, masa, color u olor.
- No se deteriora, ni se desgasta con el tiempo.
- Tiene una estructura modificable; está sujeto a cambios continuos periódicos o no.
- Las modificaciones continuas proporcionan una pérdida de confiabilidad, es decir, existe mayor probabilidad de fallas.
- Se desarrolla, no se construye: sus componentes (programas, procedimientos, documentación y datos) crecen progresivamente mediante adiciones y correcciones sucesivas de sus componentes.
- Su mantenimiento es complejo: sus componentes no se reemplazan, se corrigen.

Una empresa para desarrollo de software está constituida, entre otros, por un conjunto de personas, recursos y actividades que se organizan para desarrollar y mantener software y productos asociados (documentos de diseño, casos de pruebas, manuales de usuario...) [9].

Para producir sistemas programados efectivos y eficientes, la empresa para desarrollo de software se fundamenta en la ingeniería del software. Existen diversas definiciones de ingeniería del software, entre las cuales se tienen las siguientes:

- "Es el establecimiento y uso de principios (métodos) sólidos de ingeniería con el fin de obtener sistemas programados en forma económica, confiable y que trabajen en máquinas reales (computadores)" [20].

- "Es la aplicación práctica del conocimiento científico en el diseño y construcción de programas para computadores y la documentación asociada requerida para desarrollarlos, operarlos y mantenerlos" [20].

- "Es el enfoque sistemático para el desarrollo, operación, mantenimiento y eliminación de software" [20].

- Es la aplicación sistemática de herramientas y técnicas en el desarrollo de aplicaciones o sistemas programados basados en computador (software). Un sistema programado se diseña como un conjunto de unidades o componentes – subsistemas – que se relacionan entre sí, mediante conexiones o interfaces claramente especificadas [20].

Los objetivos generales de la ingeniería del software son:

- Aumentar la calidad de los productos de software (gestión de calidad).

- Disminuir o controlar el costo, tiempo y demás recursos empleados en el desarrollo y mantenimiento de software (eficiencia).

- Aumentar la productividad de los grupos de desarrollo y mantenimiento y, de esta manera garantizar la productividad de los mismos.

- Garantizar un alto grado de satisfacción en los usuarios del software.

- Facilitar el trabajo del desarrollador, ya que hay mayor precisión de los planes del proyecto.

Entre los objetivos específicos de la ingeniería del software – con respecto a las características del sistema programado que se genera - se tienen:

- Alta confiabilidad: se refiere a la producción de software con una baja probabilidad de falla (libre de errores). Ocurre un error cuando el software no hace lo que se supone debería hacer. La medida de confiabilidad es el tiempo medio entre fallas.

- Alta flexibilidad o mantenibilidad: habilidad del sistema programado para adaptarse o ajustarse a los cambios, mediante la modificación de su estructura y componentes. Los cambios son ocasionados por nuevos requerimientos, detección de fallas o bajo rendimiento.

- Alta eficiencia y bajo costo: uso óptimo de recursos de computación (ej. espacio de memoria y tiempo de ejecución) y disminución de los costos de desarrollo, operación y mantenimiento.

- Alta comprensibilidad: lograr que el sistema sea entendible interna y externamente. Comprensibilidad interna se refiere a la claridad de la estructura del software y del código fuente y la comprensibilidad externa a la claridad y facilidad de comprensión y uso del sistema a través de su interfaz.

- Alta utilidad, como beneficio cuantitativo.

A continuación se muestra un modelo general del funcionamiento de una empresa para desarrollo de software, el cual, como se ha mencionado anteriormente, se fundamenta en la ingeniería del software para producir software efectivo y eficiente

3.1 Un modelo general del funcionamiento de una empresa para desarrollo de software

Un proceso es una forma sistemática de hacer las cosas. Se habla de la empresa para desarrollo de software como un proceso para subrayar el hecho de que todas las etapas o funciones involucradas en la empresa, están caracterizadas por el desempeño de ciertas actividades interrelacionadas, con el propósito de alcanzar las metas deseadas.

En el modelo que se presenta, se observan las diferentes funciones involucradas en la empresa, las entradas y salidas de cada una de las funciones y, las relaciones entre las mismas. El modelado tanto del proceso de ejecución como del sistema de gestión de calidad, se ha realizado usando la notación IDEF (Integration DEfinition modeling) [14]. IDEF significa la definición de la integración para el modelado de funciones y consiste en una serie de normas que definen la metodología para la representación de funciones correspondientes a un sistema. Este estándar surge durante los años 1970, cuando el Programa de la Fuerza Aérea de los Estados Unidos para Manufactura Integrada ayudada por Computadores (ICAM por sus siglas en inglés) buscaba incrementar la productividad de la manufactura a través de la aplicación sistemática de la tecnología del computador. El programa ICAM identificó la necesidad de técnicas para un mejor análisis y comunicación de las personas involucradas en mejorar la productividad de la manufactura.

Como resultado, el programa ICAM desarrolló una serie de técnicas conocidas como las técnicas IDEF, las cuales incluyeron lo siguiente:

1. IDEF0, usada para producir un “modelo de función”. Un modelo de función es una representación estructurada de las funciones, actividades o procesos dentro del sistema modelado.

2. IDEF1, usada para producir un “modelo de información”. Un modelo de información representa la estructura y semántica de información dentro del sistema modelado.

3. IDEF2, usada para producir un “modelo de dinámicas”. Un modelo de dinámicas representa las características de comportamiento a través del tiempo, del sistema modelado.

En este trabajo se utiliza la notación IDEF0 para modelar las funciones involucradas tanto en el proceso de ejecución como en el sistema de gestión de calidad. Estos modelos consisten en una serie de diagramas jerárquicos, los cuales se representan por medio de rectángulos o cajas, textos y una serie de flechas. Uno de los aspectos más importante de IDEF es que como concepto de modelado va introduciendo cada vez más niveles de detalle a través de la estructura del modelo. Un modelo IDEF está compuesto de una serie jerárquica de diagramas que gradualmente despliegan niveles crecientes de detalle, describiendo funciones y sus interrelaciones dentro del contexto de un sistema. Se utiliza solamente notación IDEF0 - “modelo de función” - ya que el modelo de la empresa para desarrollo de software se representa mediante las funciones involucradas en la empresa, es decir, mediante el ejercicio de un conjunto de actividades en forma jerárquica con la finalidad de lograr un producto de software eficiente y efectivo y además, porque las demás técnicas IDEF especificadas anteriormente han sido cubiertas, para efectos de este trabajo, por medio de la especificación lógica y el uso de ontologías.

Los objetivos principales del estándar IDEF, son:

1. Establecer una técnica unificada del modelado de las funciones involucradas dentro del sistema a modelar.

2. Proporcionar un medio de modelado completo y consistente de las funciones requeridas por un sistema y los datos y objetos que interrelacionan a estas funciones.

3. Proporcionar un lenguaje de modelado que tenga las siguientes características:

- Genérico (para análisis de sistemas de diversos propósitos, alcance y complejidad).
- Riguroso y preciso (para producción de modelos correctos y usables)
- Conciso (para facilitar el entendimiento, comunicación y validación de un sistema).
- Conceptual (para representación de requerimientos funcionales independientes de implementaciones físicas u organizacionales).
- Flexible (para soportar diversas fases del ciclo de vida de un proyecto).

Los componentes de la sintaxis de IDEF son cajas y flechas, reglas y diagramas. Las cajas representan funciones, definidas como actividades, procesos o transformaciones. Las flechas representan datos u objetos relacionados a las funciones. Las reglas definen cómo son usados los componentes, y los diagramas proporcionan un formato para describir modelos tanto verbalmente como gráficamente.

Una caja típica se muestra en la Figura 3.1. Cada caja tiene un nombre y un número dentro de los límites de la misma. El nombre es un verbo activo o una frase verbal que describa la función. Cada caja sobre el diagrama contiene un número en su esquina inferior derecha; este número es usado para identificar el tema de la caja en el texto asociado a la misma.

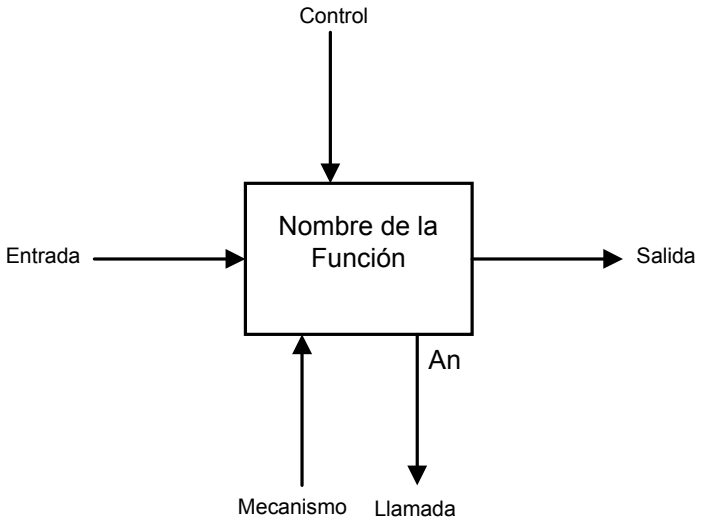


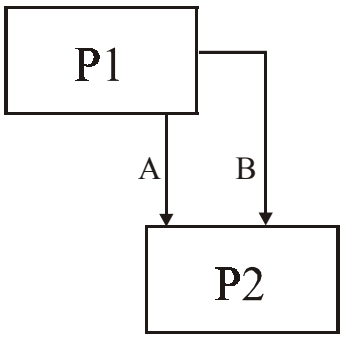
Figura 3.1. Posiciones de las flechas y roles de la Notación IDEF

Cada lado de la caja tiene un significado diferente en términos de las relaciones caja – flecha. Flechas entrando en el lado izquierdo de la caja son entradas; las entradas son transformadas o consumidas por la función para producir salidas. Flechas entrando a la caja por la parte superior son controles, los cuales especifican las condiciones o recursos requeridos por la función para producir salidas correctas. Flechas saliendo de la caja por el lado derecho son salidas; las salidas son datos u objetos producidos por la función.

Flechas conectadas a la parte inferior de la caja representan mecanismos. Flechas apuntado hacia arriba identifican algunos de los medios usados para soportar la ejecución de la función. Así que esta orientación (abajo – arriba, izquierda - derecha), es muy importante en estos modelos.

Las flechas mecanismos que apuntan hacia abajo se denominan flechas de llamada, estas flechas permiten el compartimiento de detalles entre modelos o entre porciones de un mismo modelo. Cada caja tendrá un mínimo de una flecha de control y una flecha de salida, cero o más flechas de entrada, cero o más flechas de mecanismos y cero o una flecha de llamada.

Esta orientación de las flechas es para permitir representaciones como estas:



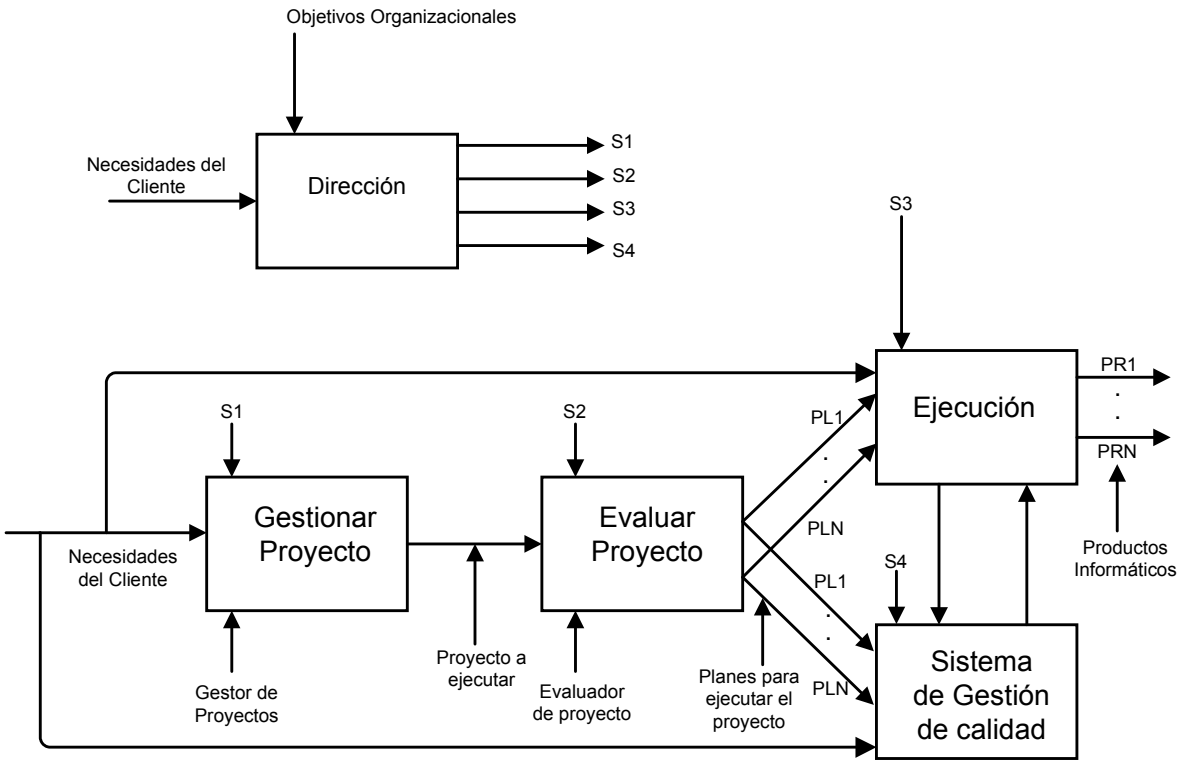
En el diagrama, la línea “A” es un componente que está al alcance de P1 y P2, es decir, ambos tienen acceso a ese componente, que en el caso de P3 es visualizado como un recurso o condición. Por otra parte, la línea “B” que representa una salida de P1, pasa a ser un recurso o requerimiento para P2.

En el modelo que se propone de una empresa para desarrollo de software, la misma está constituida por los siguientes elementos: dirección, gestión de proyectos, evaluación de proyectos, proceso de ejecución y sistema de gestión de calidad (Figura 3.2).

El gerente es el encargado de la *Dirección* y, es un proceso cuyo objetivo es producir, mediante la estipulación de normativas y, dentro de las restricciones de costo y tiempo, software que satisfaga los requerimientos impuestos por sus usuarios. La *Dirección* es quién establece las políticas de gestión y evaluación de proyectos y, las políticas de ejecución y calidad, para el desarrollo de los mismos. La función *Gestionar proyecto* tiene contacto directo con los clientes ya que plantea y realiza proyectos preliminares de acuerdo a las necesidades de los mismos y, la *Evaluación de proyectos*, evalúa y selecciona los proyectos que cumplan con el visto bueno de presupuesto, recursos, calendario y requisitos preliminares y, además, redefine el proyecto inicial o preliminar en caso de ser necesario.

En el modelo general de la Empresa para Desarrollo de Software, se encuentran involucrados los actores o agentes: Dirección, Gestor de Proyectos, Evaluador de Proyectos, el staff de ejecución y el grupo de desarrollo del Sistema de Gestión de Calidad (QMS por sus siglas en inglés).

Como ya se ha dicho, la *Dirección* es quien establece las políticas de gestión de proyectos, de evaluación de proyectos, de ejecución y las políticas de calidad; en función de las necesidades del cliente y de los requerimientos de calidad del producto generado. Las necesidades del cliente son conocidas por el *Gestor de Proyectos* en la etapa Gestionar Proyectos, ya que este actor tiene contacto directo con los clientes y, en función de estas necesidades plantea un proyecto preliminar donde define las especificaciones generales del sistema programado. Otra actividad que lleva a cabo este actor es que le hace ver al cliente cuándo alguno de sus requerimientos no puede ser alcanzado de la forma que él lo especifica, proponiendo cambios adecuados y convenientes, tanto para la empresa como para el cliente.



S1: Políticas de Gestión de Proyectos
S2: Políticas de Evaluación de Poyectos
S3: Políticas de Ejecución
S4: Políticas de Calidad
PLi: Planes para ejecutar el proyecto i; i = 1 .. N.

Figura 3.2. Modelo simple de una empresa para desarrollo de software

Los planes estratégicos preliminares del proyecto a ejecutar, son enviados al agente *Evaluador de Proyectos*, quién define o esquematiza con mayor nivel de detalle las necesidades del cliente que han sido puestas de manifiesto en los planes preliminares planteados por el *Gestor de Proyectos*. De esta manera, en la etapa Evaluar Proyectos, se pueden redefinir, de ser necesario, los planes preliminares; el *Evaluador de Proyectos* proporciona asesoramiento al cliente en caso de que se requiera que el cliente visualice otras necesidades u otros mecanismos para satisfacerlas. Una vez que el *Evaluador de Proyectos* ha establecido con mayor nivel de detalle los planes para el desarrollo del

proyecto (planes de tiempo, costo, recursos necesarios, disponibilidad de personal adecuado, etc), éstos son enviados tanto al personal de ejecución como al grupo de trabajo del Sistema de Gestión de Calidad (QMS).

En general, un gerente tiene como funciones básicas las siguientes:

1. **Planificación:** Para cada cliente, en cada nueva ocasión de un problema., las actividades involucradas con esta función son:

   - Definir el problema y establecer las necesidades del cliente.
   - Proponer y evaluar sistemas alternativos de solución.
   - Establecer los objetivos del sistema alternativo seleccionado.
   - Estimar los costos y tiempos de las actividades.
   - Estimar los recursos humanos, equipos y software requeridos.
   - Establecer los eventos críticos del proyecto.
   - Elaborar y mantener el Plan detallado inicial del proyecto.

2. **Organización:**

   - Identificar las necesidades de personal del proyecto.
   - Seleccionar una estructura organizativa para el grupo de desarrollo.
   - Definir las obligaciones y responsabilidades de cada miembro del grupo.
   - Establecer las líneas de comunicación del grupo con el cliente o usuario.

3. **Gestión de personal:**

   - Seleccionar el personal requerido por el proyecto.
   - Adiestrar al personal en el uso de las técnicas, métodos y herramientas a usar.
   - Evaluar el desempeño de cada miembro del grupo.

4. **Monitoreo y control:**

   - Medir el avance del proyecto en base al Plan del Proyecto.

   - Medir el cumplimiento de objetivos, metas y eventos críticos.

   - Definir las acciones necesarias para corregir las desviaciones del Plan.

5. **Innovación:** encontrar nuevas ideas, técnicas o herramientas que mejoren la creatividad y productividad del grupo. Además, aprovechar la experiencia, la intuición y la creatividad para anticipar los problemas que se puedan presentar en el proyecto de un cliente.

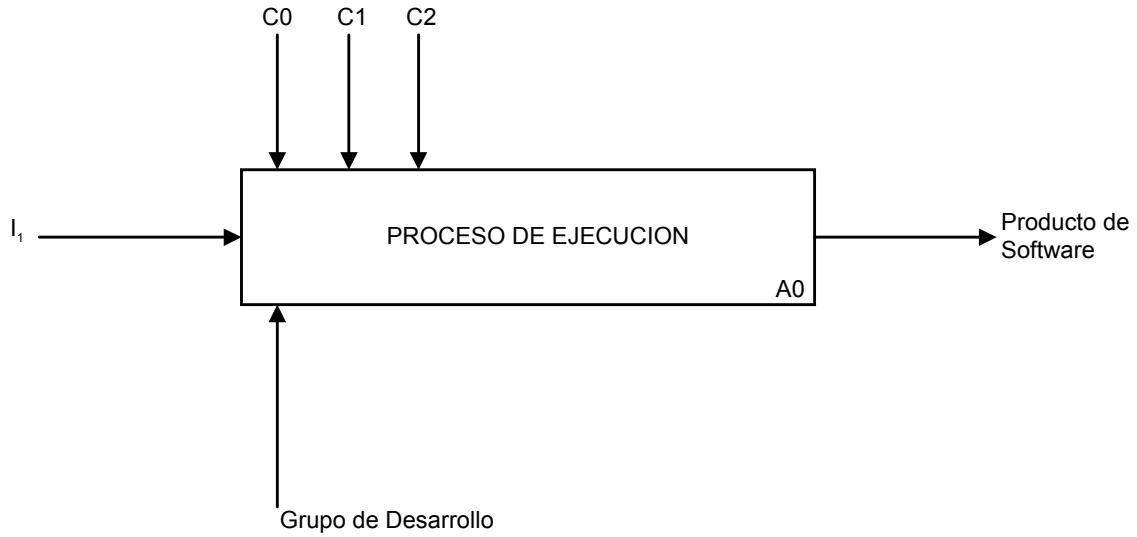
6. **Comunicación:** comunicar las decisiones tomadas en las otras funciones siguiendo las líneas de comunicación establecidas.

7. **Representación:** representar al grupo de desarrollo ante el cliente, los suplidores u organizaciones externas.

3.1.1 Modelado del Proceso de Ejecución

El modelo del proceso de ejecución se presenta en notación IDEF0. El nivel superior del diagrama en el modelo, proporciona la descripción más general o abstracta del tema representado por el modelo, es decir, el proceso de ejecución. Este diagrama es seguido por una serie de diagramas hijos que proporcionan más detalles acerca de funciones específicas del proceso de ejecución.

En la Figura 3.3 se presenta el nivel más abstracto o nivel superior, del proceso de ejecución.



C_0: Librerías de componentes
C_1: Solución de problemas similares
C_2: Conocimiento/información estratégico (a)
I_1: Necesidades del cliente

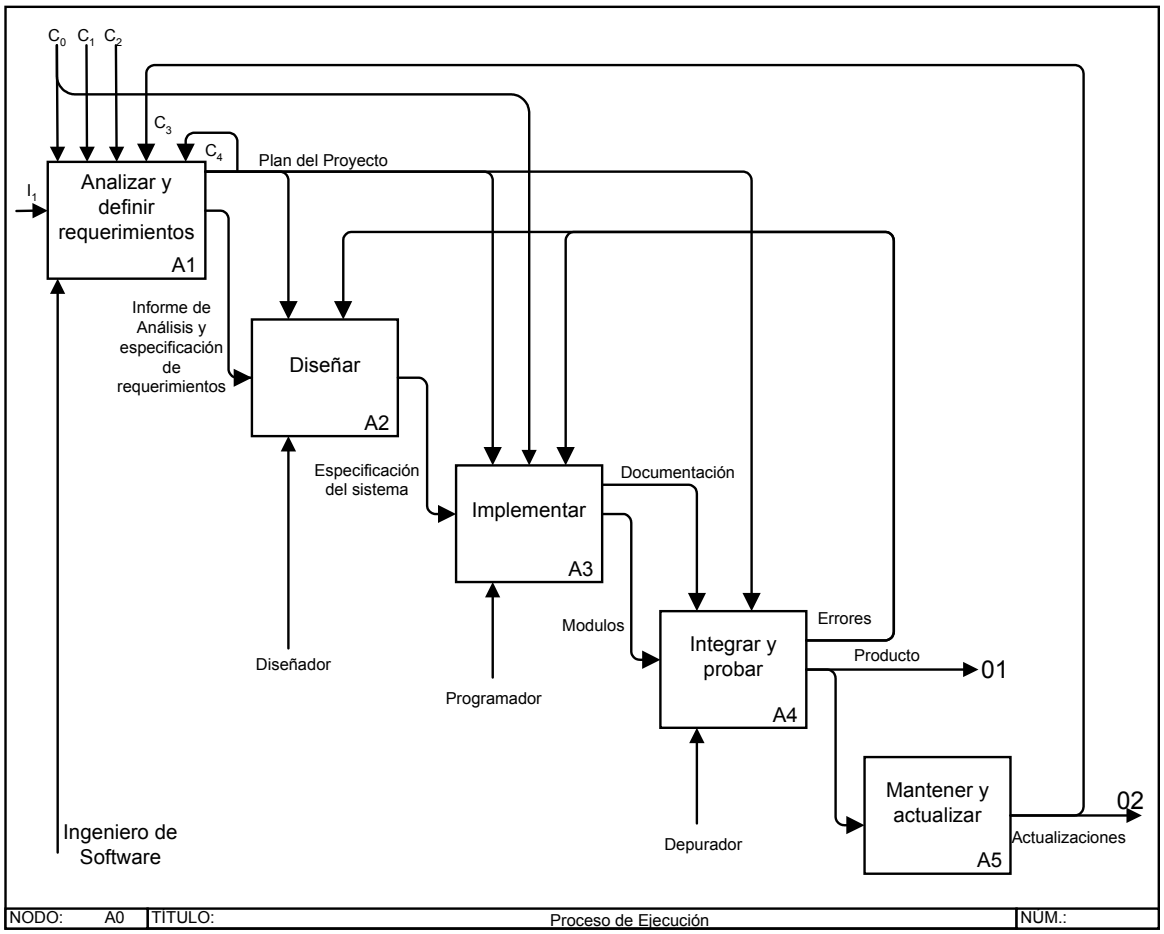
Figura 3.3. Proceso de Ejecución en Notación IDEF

El proceso de ejecución consiste en el desarrollo del proyecto o sistema programado y consta de cuatro etapas o funciones básicas:

1. **Análisis y especificación de requerimientos.** Comprende las siguientes etapas: Definir problema, Estudiar factibilidad, Analizar requerimientos,

2. **Diseño.** Es el proceso de definir y especificar los detalles de cómo el sistema cumplirá las especificaciones de requerimientos establecidas en el análisis. Su objetivo es establecer una correspondencia entre los requerimientos y el ambiente operativo del sistema. Comprende las siguientes etapas: Analizar el problema, Investigar soluciones, Tomar decisiones, Especificar o sintetizar

En la Figura 3.4 se muestran, en notación IDEF0, las etapas o funciones básicas del proceso de ejecución.



C0, C1, C2: Ver figura 3.3

Figura 3.4. Diagrama hijo en notación IDEF del proceso de ejecución.

3. **Implementación.** Comprende las siguientes etapas: Seleccionar el lenguaje, Codificar, Documentar

4. **Integración y pruebas.** Comprende las siguientes etapas: Probar módulos, Integrar módulos, Probar integración, Probar la aceptación del sistema (eficacia del sistema)

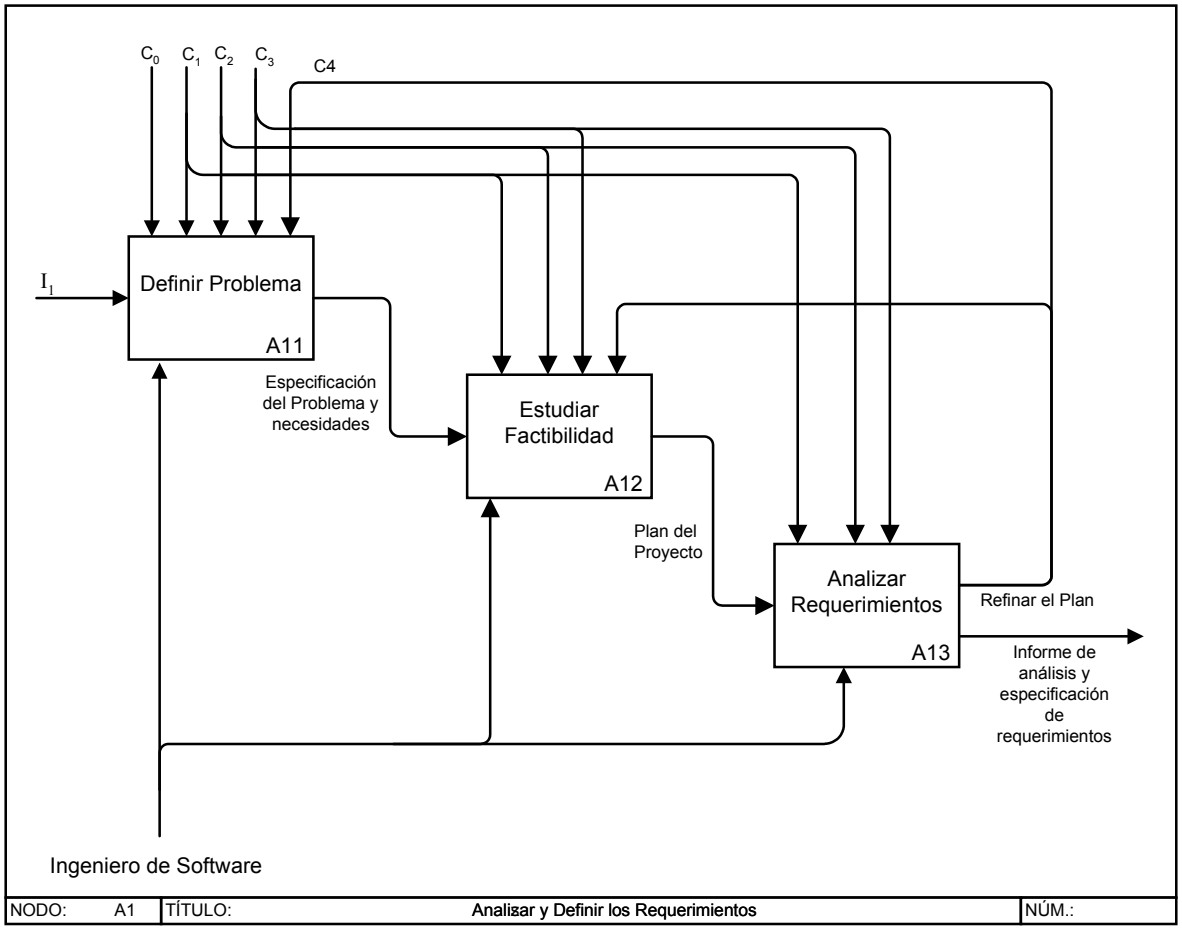
5. **Mantenimiento y actualización.**

Las funciones de la etapa funcional *Analizar y Definir requerimientos* (A1), se muestran en notación IDEF0, en la Figura 3.5 y son:

- **Definir el Problema:** Identifica las necesidades; Define el problema y Organiza el personal

- **Estudiar Factibilidad:** Analiza los riesgos, costos y beneficios del proyecto; Analiza factibilidad económica, técnica y organizacional.

- **Analizar requerimientos:** Define y especifica los requerimientos funcionales, de rendimiento, de interfaz, de diseño y de estándares.



C0, C1, C2: Ver figura 3.3

C3: Actualizaciones para nuevas versiones

Figura 3.5. Diagrama hijo de la etapa funcional Analizar y Definir los Requisitos

Esta etapa tiene como entrada las necesidades de los clientes y como salida el Informe de Análisis y Especificación de Requerimientos y el Plan de Proyecto. Este último tiene la siguiente estructura:

Introducción	▪ Bosquejo del plan del proyecto: objetivos, estructura y generalidades del Plan. ▪ Supuestos bajo los cuales se elabora el Plan. ▪ Calendario de eventos críticos.
Descripción del Proyecto	▪ Productos del proyecto que se entregarán al cliente. ▪ Criterios de aceptación para los productos. ▪ Participación del usuario en el proceso de desarrollo.
Plan de fases	▪ Fases, pasos y tareas del proyecto. ▪ Estimación de costos, tiempos y recursos humanos. ▪ Eventos críticos del proyecto y sus fechas.
Organización	▪ Estructura del grupo ▪ Responsabilidades de cada miembro del grupo.
Especificación de requisitos de soporte	▪ Requisitos de hardware requeridos. ▪ Requisitos de software requeridos.
Control de cambios	▪ Método de control de la configuración.
Plan de documentación	▪ Documentos que se producirán y sus formatos. ▪ Métodos para la producción de documentos.
Apéndices	▪ Plan de pruebas (tipos, procedimientos y herramientas de prueba que se usarán y responsabilidades) ▪ Plan de calidad (estándares, procedimientos y técnicas de control de calidad que se emplearán) ▪ Ejemplos de las notaciones textuales, gráficas o formales que se usarán.

Continuando con la especificación de las etapas o funciones necesarias para lograr un producto de software efectivo (Figura 3.4), la etapa *Diseñar* (A2), especifica un diseño preliminar de entradas/salidas, interfaz usuario/sistema y del esquema conceptual de la base

de datos; un diseño de la arquitectura del sistema programado y; el diseño de subsistemas y programas. La etapa *Diseñar* tiene como entrada el Informe de Análisis y Especificación de requerimientos y como salida la especificación del sistema para su posterior implementación.

En la Figura 3.6, se muestran, en notación IDEF0, las funciones involucradas en la etapa *Diseñar.*

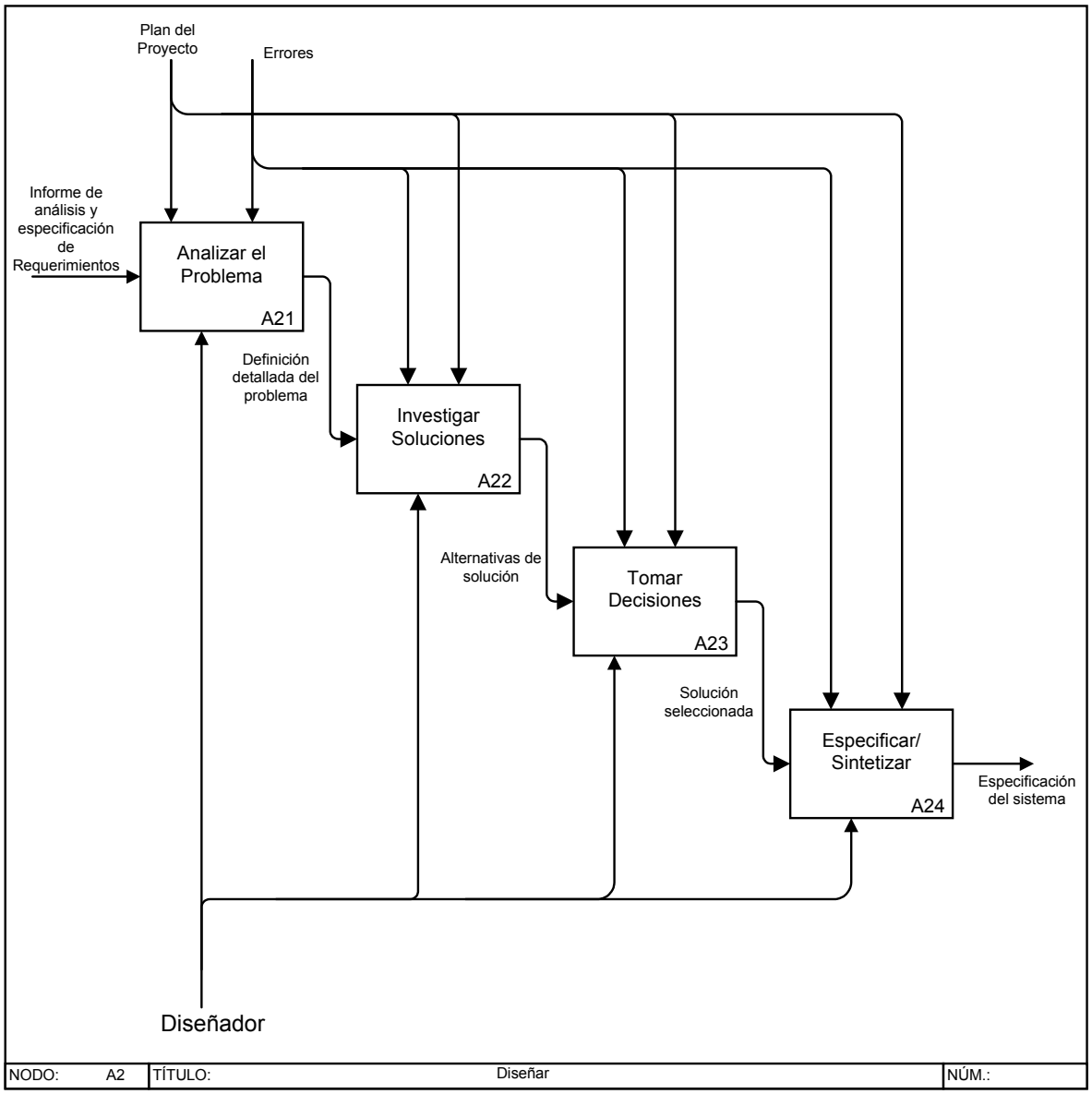


Figura 3.6. Diagrama hijo de la etapa funcional Diseñar

La etapa funcional *Diseñar*, está constituida por las siguientes funciones:

- **Analizar el problema:** Definición de la estructura del sistema en base a los requerimientos Entrada/Salida, interfaz Usuario/Sistema y; esquema conceptual de las bases de datos. Su salida principal es la definición detallada del problema.

- **Investigar soluciones:** busca diferentes alternativas de solución; expande cada requerimiento y establece su correspondencia con el ambiente operativo.; identifica los módulos y sus relaciones; establece la estructura y contenido de la base de datos.

- **Tomar Decisiones:** Selecciona la alternativa más adecuada de acuerdo al plan de proyecto.

- **Especificar o sintetizar:** Comprende diseño de la arquitectura del sistema, diseño de subsistemas y programas, diseño de la integración de los subsistemas; determinar la ubicación de los datos y procesos y; representación gráfica de los detalles del sistema.

La etapa *Implementar* (A3), tiene como actividades, la codificación de módulos; prueba y depuración de módulos; creación de la base de datos y; elaboración de la documentación de los módulos. Esta etapa tiene como entrada la Especificación del sistema y, como salida la Documentación de los Módulos.

En la figura 3.7, se muestran, en notación IDEF, las funciones involucradas en la etapa *Implementar.*

La etapa *Implementar*, consta de las siguientes funciones:

- **Seleccionar la plataforma:** Selección del lenguaje de programación de acuerdo a tipos de datos y verificación, constructos del lenguaje, modularización, manejo de memoria, manejo de excepciones, apoyo multiusuario, eficiencia del compilador y portabilidad.

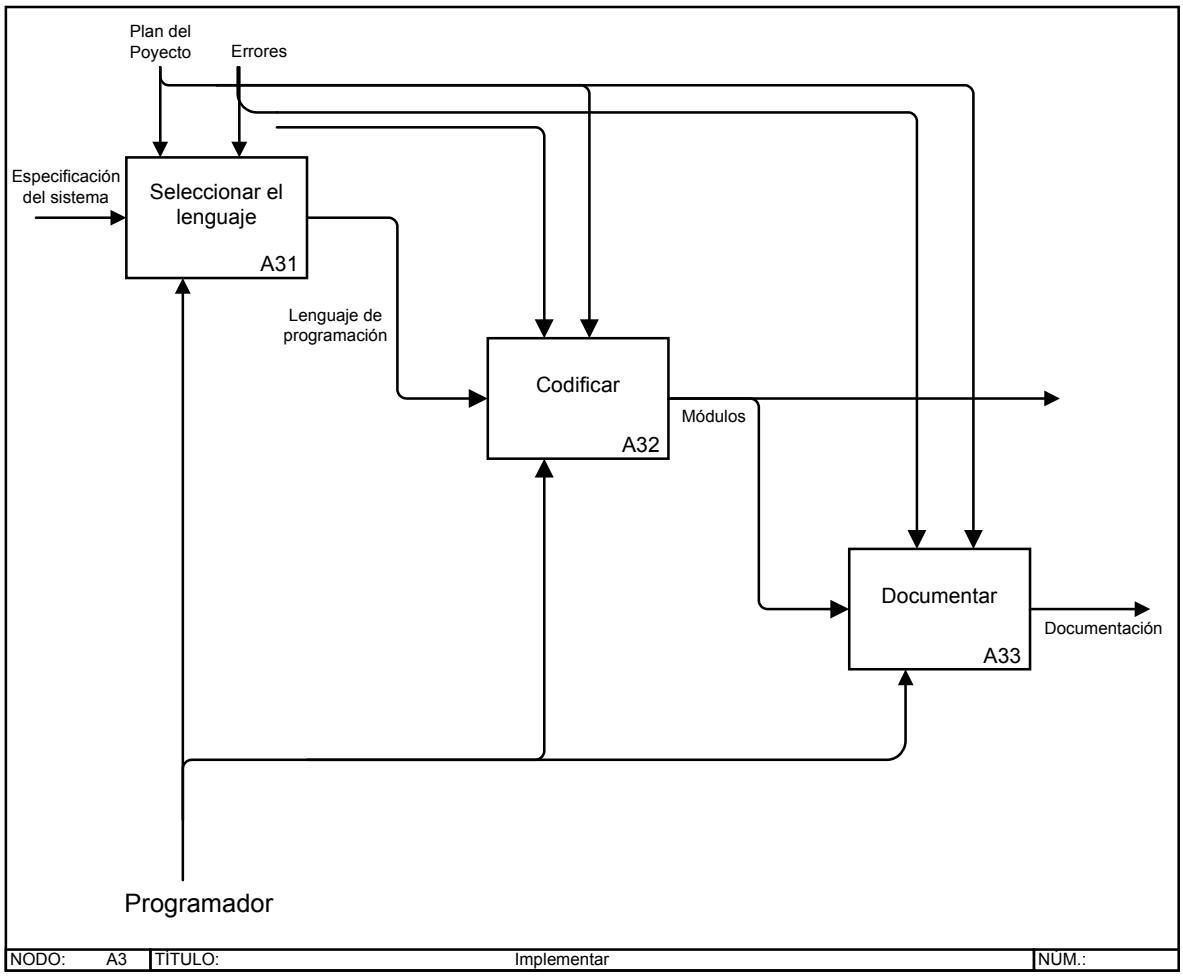


Figura 3.7. Diagrama hijo de la etapa funcional Implementar

- **Codificar:** Implementa el diseño y verifica los módulos del programa; crea las bases de datos.

- **Documentar:** Elaboración de la documentación de los módulos.

En la etapa funcional *Integrar y probar* (A4), el objetivo fundamental es asegurar la calidad del producto y validar el sistema programado. Las actividades realizadas en esta etapa son las pruebas de integración, pruebas del sistema y, pruebas de aceptación. Tiene como entrada los módulos ya programados y como salida la validación de la aceptación del sistema.

En la Figura 3.8, se muestran, en notación IDEF, las funciones involucradas en la etapa *Integrar y probar.*

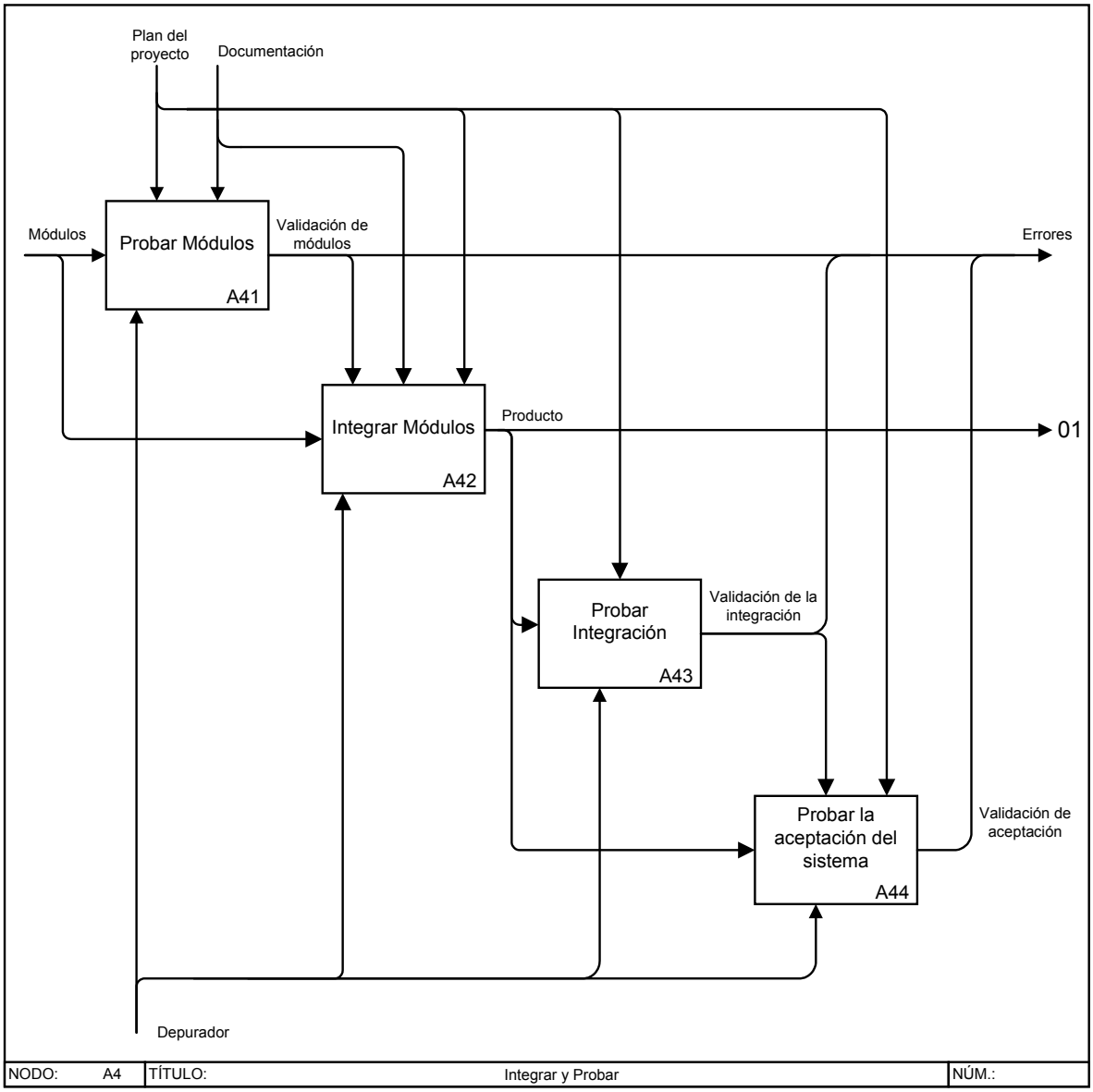


Figura 3.8. Diagrama hijo de la etapa funcional Integrar y Probar

Las funciones de la etapa *Integrar y probar*, son:

- **Probar Módulos:** Planificar las pruebas y; verificar y validar los módulos, a través de técnicas de verificación y validación.

- **Integrar Módulos:** integrar el sistema programado; conversión de datos; instalación; documentación de integración.

- **Probar Integración de los Módulos:** Validar las conexiones entre unidades o módulos, empleando estrategias de validación ascendentes o descendentes.

- **Probar la aceptación del sistema programado por parte del usuario.**

La etapa de *Mantener y Actualizar* (A5), tiene como objetivo integrar el sistema programado a su ambiente operativo para lograr que el sistema pueda operar de forma eficiente y eficaz. Entre las actividades realizadas en este etapa se tienen: monitoreo del sistema programado; mantenimiento correctivo, perfectivo y adaptativo y; auditoría del sistema programado. Los productos resultantes de esta etapa son: informe de cambios y documentación de nuevas versiones. Finalmente, esta etapa realiza actualizaciones para nuevas versiones del sistema programado generado y garantiza que el sistema esté operando eficientemente y eficazmente, realizando un monitoreo del mismo.

Ya se han especificado las funciones realizadas por la empresa para desarrollo de software y a su vez, el conjunto de actividades desempeñadas en cada una de las etapas. De esta manera, se ha presentado en notación IDEF0 el modelo del proceso de ejecución de la empresa. A continuación, se presenta el modelado del sistema de gestión de calidad.

3.1.2 Modelado del Sistema de Gestión de Calidad

La calidad consiste en el conjunto de características de una entidad (producto o servicio) que le confieren su aptitud para satisfacer las necesidades expresadas e implícitas (ISO 8402). Por otra parte, el aseguramiento de la calidad consiste en un conjunto de acciones planificadas y sistemáticas que son necesarias para proporcionar la confianza adecuada de que un producto o servicio satisfaga los requisitos dados sobre la calidad [20].

El control de calidad se refiere a las actividades y técnicas utilizadas para afectar (medir y corregir) la calidad del producto que se está desarrollando [20]. Para medir la calidad de un producto de software se emplean modelos que especifican la calidad mediante la evaluación de atributos. Entre estos modelos se encuentra el CMM (Capability Maturity Model – Modelo de Capacidad de Madurez) [9].

El modelo de capacidad de madurez (CMM) [9], es un modelo de referencia desarrollado por el Instituto de Ingeniería del Software, con la finalidad de evaluar la madurez de los procesos de desarrollo de software dentro de una organización y proponer un plan de mejoramiento de los mismos, en base a una serie de niveles que van desde un proceso caótico (inmaduro) hasta un proceso disciplinado y de mejoramiento continuo (maduro) [9]. El CMM presenta cinco niveles de madurez, que son : Inicial, Repetible, Definido, Gestionado y Optimizado. Cada nivel del CMM están conformado por Areas de Proceso Clave (KPA – por sus siglas en inglés); cada área de proceso clave corresponde a un área de la empresa, que ha sido considerada como clave para su control y mejora.

En este trabajo, el sistema de gestión de calidad propuesto se basa en el modelo CMM, específicamente en su segundo nivel de madurez y se incorpora a las KPA's del segundo nivel, una KPA del tercer nivel llamada Peer Review. Esta área de proceso clave del tercer nivel, tiene como propósito remover los defectos de los productos de software. Un nivel es una plataforma bien definida que se corresponde con un estado de madurez en los procesos de software de una organización. La Figura 3.9 muestra los niveles del modelo CMM [9].

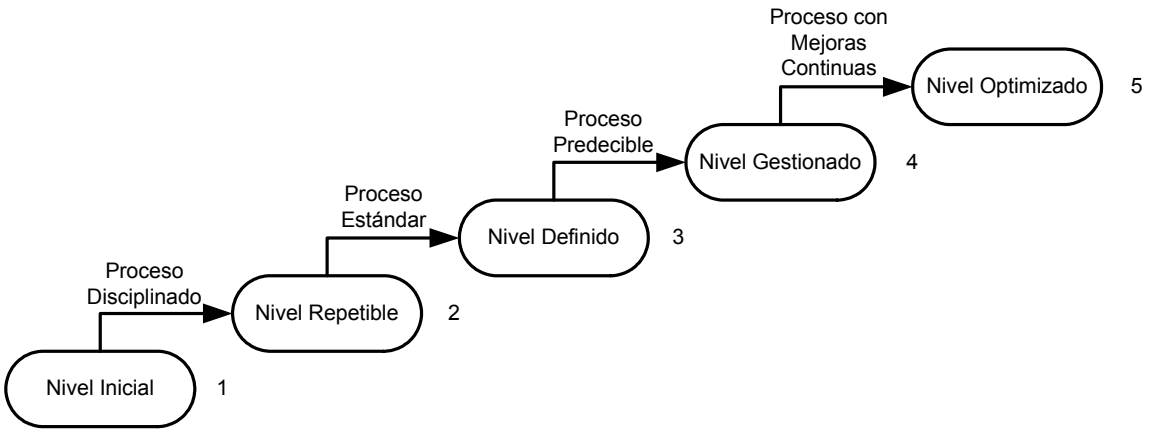


Figura 3.9. Niveles de Madurez del CMM

El sistema de calidad especifica la estructura organizacional, las responsabilidades, los procedimientos, los procesos y los recursos necesarios para implantar la gestión de calidad. La gestión de calidad son aquellos aspectos de la función gerencial que determinan e implementan la política de calidad. Las políticas de calidad son un conjunto de directrices y objetivos generales de una organización relativos a la calidad de sus productos o servicios y definidos por la gerencia.

Como ya se ha mencionado, cada nivel de madurez del CMM contiene Areas de Proceso Clave (KPA por sus siglas en inglés) y a su vez, cada KPA presenta características comunes, como son: Objetivos, Compromiso de realización, Capacidad de realización, Actividades realizadas, Medición y análisis y Verificación de implantación.

- **Objetivos:** se establecen los objetivos de mejora necesarios para esa KPA.

- **Compromisos de realización:** describe las acciones que la organización debe llevar a cabo para garantizar que los procesos, tanto de software como de calidad, queden establecidos y perduren.

- **Capacidad de realización:** describe los requisitos necesarios en los proyectos y en la organización para poder implementar el proceso de software competentemente.

- **Actividades realizadas:** describe las actividades, roles y procedimientos necesarios para implementar cada KPA.

- **Medición y análisis:** describe las prácticas de medición básicas necesarias para determinar el estado en relación con el proceso.

- **Verificación de implantación:** describe los pasos necesarios para garantizar que las actividades son llevadas a cabo de acuerdo al proceso definido.

3.1.3 Especificación del Segundo Nivel del Modelo de Capacidad de Madurez (CMM por sus siglas en inglés)

Se trabaja con el segundo nivel del CMM para tratar de adoptar un enfoque estándar repetible, de tal manera que cualquier empresa pueda repetir sus prácticas de calidad. Se incluye además el área de proceso clave "Peer Review" del tercer nivel del CMM, con la intención de contemplar, desde el primer momento, la delicada interacción entre los agentes que intervienen en el proceso de evaluación y depuración del software.

El segundo nivel del CMM, nivel repetible, presenta seis áreas de proceso clave, las cuales son [9]:

- Gestión de requisitos
- Planificación de proyectos de software
- Seguimiento y control de proyectos de software
- Gestión de subcontratación de software
- Aseguramiento de la calidad del software
- Gestión de configuración del software.

El propósito de *Gestión de requisitos* es establecer un entendimiento común entre el cliente y el equipo del proyecto sobre los requisitos del cliente que deben tenerse en cuenta. Sus objetivos son: documentar requisitos como base del proyecto y , gestionar y controlar los cambios que se hacen a los requisitos durante todo el ciclo de vida del proyecto.

El propósito de *Planificación de proyectos* de software es establecer unos planes razonables para realizar las actividades de ingeniería de software y para gestionar el proyecto. Sus objetivos son: hacer estimaciones del trabajo a realizar, establecer los compromisos para realizar el trabajo y definir los planes para ejecutar el trabajo.

El propósito de *Seguimiento y control de proyectos de software* es supervisar el progreso real del proyecto para tomar acciones a tiempo cuando el rendimiento del

proyecto se desvía significativamente de lo planificado. Sus objetivos son: seguir y revisar el progreso del proyecto en comparación con las estimaciones y los planes, tomar acciones correctivas cuando surjan discrepancias entre las estimaciones y los valores reales, para reconducir el proyecto y, reestablecer compromisos.

El propósito de *Gestión de subcontratación de software* es seleccionar subcontratistas calificados y dirigirlos eficazmente. Sus objetivos son: seleccionar un subcontratista adecuado, establecer compromisos y, seguir y revisar el rendimiento del subcontratista y sus resultados.

Aseguramiento de la calidad del software tiene como propósito proporcionar visibilidad sobre los procesos utilizados por el proyecto de software y sobre los productos que genera dicho proyecto. Sus objetivos son: planificar las actividades de aseguramiento de la calidad, revisar y auditar objetivamente los productos y las actividades para verificar que están conformes con los procedimientos y estándares aplicables y, proporcionar los resultados de estas revisiones o auditorías, informando a la dirección cuando sea necesaria su mediación.

Gestión de configuración del software tiene como propósito establecer y mantener la integridad de todos los productos del proyecto a lo largo de todo su ciclo de vida. Sus objetivos son: planificar las actividades de gestión de la configuración, identificar los elementos de configuración del software, controlar los cambios hechos a los elementos de configuración para mantener su integridad y, construir las versiones del producto final. La Gestión de la Configuración implica la identificación de las entidades del software y sus componentes, su monitoreo o seguimiento y el control de los cambios que involucren a esas entidades y sus componentes.

A continuación se bosquejan, en la Figura 3.10, las entradas, salidas y recursos del sistema de gestión de calidad. Más adelante se muestra en notación IDEF0, el ciclo correspondiente a este sistema.

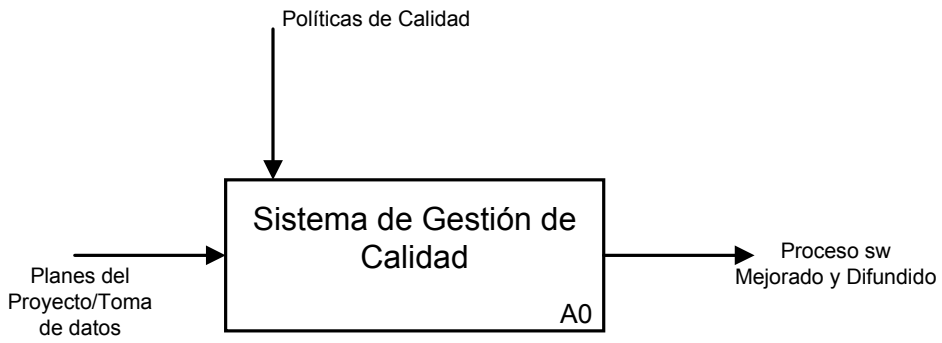


Figura 3.10. Sistema de Gestión de Calidad en notación IDEF

Como ya se ha dicho, los modelos de gestión de calidad total apoyan su implantación en un ciclo de mejora continuo, el cual permite introducir y refinar los conceptos de calidad total en la organización de una manera progresiva

Las actividades o procesos involucrados en el ciclo de mejora, se muestran en la Figura 3.11 y, son:

- Evaluación del Proceso de Software.
- Diseño del Plan de Mejora.
- Ejecución de la Mejora.
- Instutucionalización.

La *evaluación del proceso de software* consiste en un examen disciplinado de los procesos involucrados en el desarrollo del mismo, con respecto a un conjunto de criterios estipulados, con el objetivo de determinar la capacidad con la cual esos procesos se realizan en términos de calidad, tiempo, coste, etc.

El *plan de mejora* define alcance, objetivos, recursos y actividades necesarias para la implantación de una mejora. *Ejecución de la mejora* implica hacer operativo el plan de mejora y finalmente, *Institucionalización*, garantiza que la mejora permanezca en el largo plazo, mediante la creación de elementos físicos, metodológicos y administrativos suficientes.

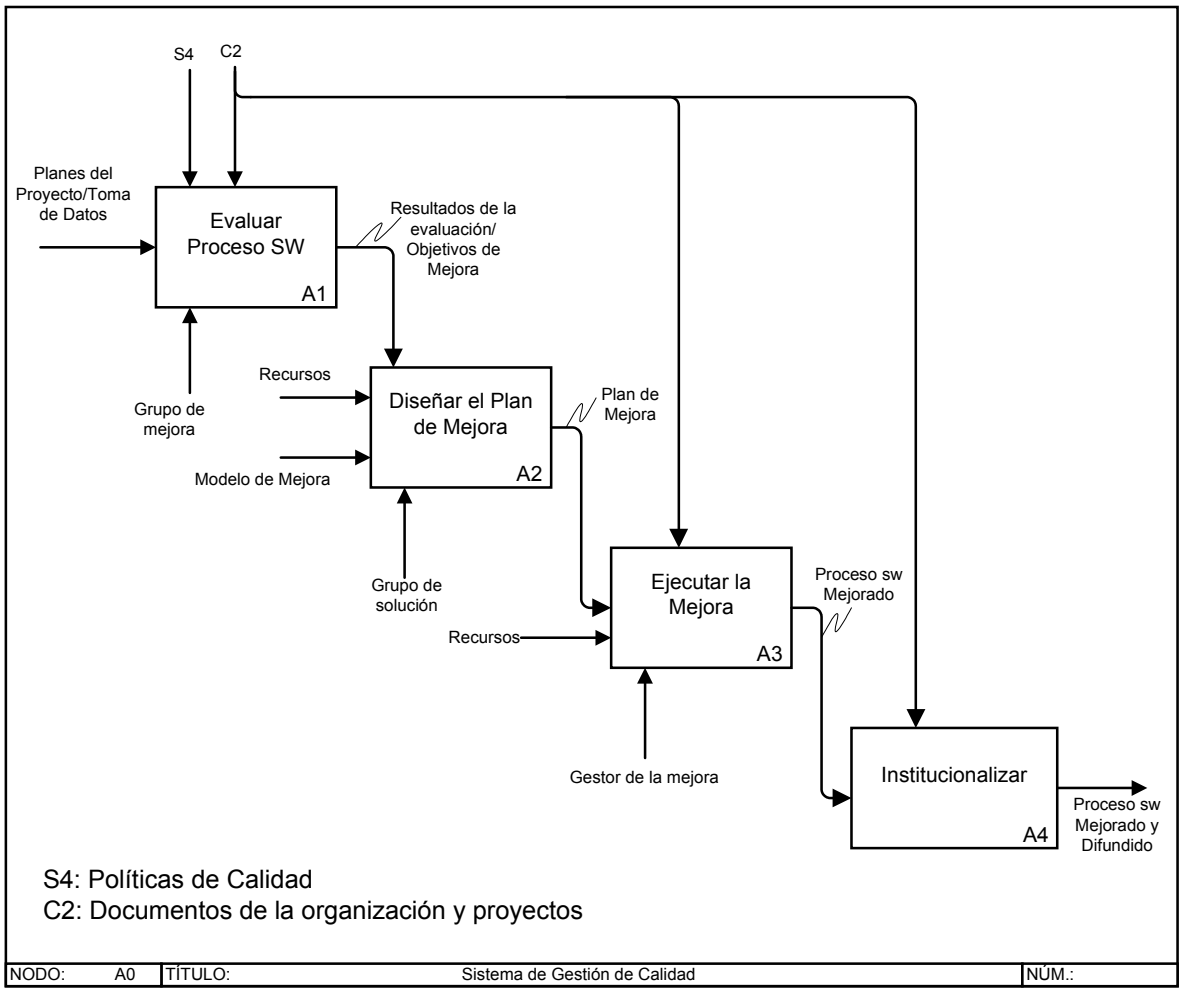


Figura 3.11. Diagrama Hijo del Sistema de Gestión de Calidad

Para finalizar, en este capítulo se ha obtenido el modelo de la empresa para desarrollo de software, el cual ha sido especificado en notación IDEF0. En el siguiente capítulo se hace referencia al modelado de dicha empresa basado en sistemas multiagente.

Capítulo 4

Simulación de una empresa para desarrollo de software basado en sistemas multiagente

Una vez obtenido el modelo de una empresa para desarrollo de software, se determinan a partir de allí, los actores (agentes para sistemas multiagente) involucrados tanto en el proceso de ejecución como en el sistema de gestión de calidad. De igual manera, se han considerado los actores de gestión y evaluación de proyectos.

4.1 Análisis Funcional de los Agentes Involucrados en el Sistema Multiagente

En el capítulo 3 se muestra un modelo general de una empresa para desarrollo de software (ver figura 3.1), en el cual se pueden observar los actores involucrados en la empresa, entre los cuales se tienen: Dirección, Gestor de Proyectos, Evaluador de Proyectos, actores del proceso de ejecución y actores del sistema de gestión de calidad.

Dentro del modelo general de la empresa, específicamente en el proceso de ejecución, se comienza a trabajar con la etapa correspondiente al *Análisis y Especificación de requisitos*, donde se hace una revisión detallada de cada sistema programado a desarrollar (especificación del problema), se hace un estudio completo de factibilidad y se generan como salidas el Informe de Análisis y Especificación de Requerimientos y el Plan del Proyecto. Las salidas de esta etapa son enviadas al QMS para su respectiva revisión, si el QMS da su visto bueno, entonces se pasa a la siguiente etapa, la cual corresponde al Diseño. Si el QMS no da su visto bueno entonces la etapa de Análisis y Especificación de Requerimientos sigue trabajando en el informe y en el plan del proyecto.

La etapa de *Diseño* cubre un análisis del problema, donde se hace una definición de la estructura del sistema en base a los requerimientos de entrada/salida, interfaz usuario/sistema y se realiza un esquema conceptual de la base de datos. La principal salida de esta etapa es una definición detallada del problema.

Una vez que el sistema programado sale de la etapa de *Diseño*, entra en la etapa de *Implementación*, donde se realiza la selección del lenguaje y luego, se hace la implementación del diseño en el lenguaje seleccionado y se verifican los módulos del programa. Además, se crea la base de datos y luego se hace la elaboración de la documentación de los módulos.

En la última etapa, correspondiente a la *Integración y pruebas*, se realizan las siguientes fases: Pruebas de Módulos, Integración de los módulos y pruebas de la integración. En la prueba de los módulos se realiza la planificación de las pruebas, verificación y validación de los módulos, a través de técnicas apropiadas. La integración de módulos, comprende la integración del sistema, la conversión de datos, y documentación de la integración.

El análisis funcional determina las responsabilidades de cada uno de los actores o agentes involucrados en el modelo general de la empresa.

Actores involucrados en el modelo general de la empresa:

1. **Dirección.** Se presenta un resumen de sus funciones, las cuales han sido expuestas en el capítulo 3:

    - Establecer las políticas de gestión de proyectos.
    - Establecer las políticas de evaluación de proyectos.
    - Establecer las políticas de ejecución.
    - Establecer las políticas de calidad.
    - Organizar el personal (División del trabajo).
    - Establecer políticas de motivación (Incentivos).

- Generar liderazgo en los grupos de trabajo (Dinámica de grupo).
- Establecer políticas de presupuesto.
- Diseñar la estrategia de mejora.
- Tomar decisiones en caso de presentarse problemas.
- Autorizar y proporcionar recursos humanos.
- Saber dirigir al personal para que las funciones que realizan con respecto al programa de mejora, estén en función de los objetivos de la organización.

2. **Gestor de proyectos.** Sus funciones son:

- Tener contacto directo con el cliente; plantear y realizar proyectos preliminares.
- Evaluar y seleccionar los proyectos que cumplan con el visto bueno de presupuesto, recursos, calendario y requisitos.
- Motivar al cliente; proporcionar información al cliente de otras opciones o modificaciones posibles.
- Discutir con el cliente el proyecto inicial de ser necesario.

3. **Evaluador de proyectos.** Sus funciones son:

- Definir las necesidades del cliente
- Proporcionar al cliente información que le permita visualizar otros requerimientos o cambiar los actuales de ser necesario.

4. **Cliente.** De este actor se espera que:

- Manifieste a la empresa sus necesidades.
- De el visto bueno para la ejecución de un proyecto en base a sus requerimientos.
- Proporcione a la empresa las características de diseño deseadas.
- Establezca los parámetros de aceptación requeridos.

5. **Usuario.** De este actor se espera que:

- Use el sistema y reporte los posibles errores.
- Sugiera alguna actualización apropiada del sistema programado.
- Solicite apoyo técnico en caso de ser necesario.

Actores involucrados en el proceso de ejecución:

1. **Ingeniero de Software.** Sus funciones son:

    - Desarrollar y mantener al sistema programado.
    - Realizar un diseño externo del sistema programado o software, el cual se inicia muy temprano en la fase de análisis y especificación de requerimientos y consiste en definir:
        - La estructura general del sistema programado.
        - Las funciones que deberá ejecutar el sistema bajo el ambiente operativo establecido.
        - La interfaz Usuario – Sistema, incluyendo la entrada de datos y salida de información y los atributos de calidad de diseño que deberá satisfacer el sistema.

2. **Diseñador.** Sus funciones son:

    - Refinar la estructura general del sistema en términos de módulos o componentes.
    - Diseñar los archivos y las bases de datos.
    - Diseñar los módulos, incluyendo el diseño de los algoritmos.
    - Diseñar las estructuras de datos.

3. **Programador.** Su función principal es la codificación de programas. El programador debe analizar las diferentes propiedades de los lenguajes de programación, con miras a seleccionar el lenguaje más apropiado para una aplicación. Es por ello, que un

programador debe tener conocimientos básicos en lenguajes de programación, algoritmos y sistemas operativos.

4. **Depurador.** Sus funciones son:

   - Analizar los diferentes métodos, técnicas y estrategias empleadas en la verificación y validación de software.
   - Planificar las pruebas.
   - Examinar el proceso de aseguramiento de la calidad del software y establecer su relación con el proceso de pruebas del software.
   - Realizar la integración de los módulos.
   - Realizar las pruebas de integración.

5. **Mantenimiento y Actualización.** Sus funciones son:

   - Recibir del usuario las sugerencias de actualización y los posibles errores.
   - Proporcionar apoyo técnico.
   - Estudiar el software para definir posibles mejoras.
   - Proporcionar nuevas versiones para nuevos recursos tanto de software como de hardware.

Actores involucrados en el sistema de gestión de calidad [9]:

1. **Grupo de mejora.** Sus responsabilidades son:

   - Diseñar y mantener la infraestructura física para las mejoras.
   - Diseñar la estrategia de gestión del cambio.
   - Diseñar, impulsar y supervisar las actividades de evaluación.
   - Mantener el proceso de la compañía.
   - Proporcionar soporte a los grupos en su implantación de las mejoras.

2. **Grupo de solución.** Diseña los productos que implantan la solución. Sus responsabilidades son:

    - Diseñar las soluciones.
    - Definir un plan estratégico para desarrollar el proyecto de mejora.
    - Implantar y refinar las soluciones.
    - Diseñar guías para la difusión de las soluciones.

3. **Gestor de la mejora.** Sus responsabilidades son:

    - Impulsar y difundir la iniciativa de mejora de procesos.
    - Diseñar un plan operativo, a partir de la evaluación, para llevar a cabo las mejoras.
    - Gestionar el proyecto de mejora, entre otras cosas: asegurar la ejecución del plan, monitorear el estado, revisar e informar, gestionar cambios.
    - Mantener las mejoras.

En el capítulo anterior se dijo, que cada área de proceso clave (KPA) presenta ciertas características comunes y por otra parte, hay tres actores o agentes involucrados en el sistema de gestión de calidad; es por ello que se va a especificar qué actor desarrolla o lleva a cabo cada característica para cada KPA.

- Objetivos: Grupo de mejora.
- Compromiso de realización: Grupo de mejora.
- Capacidad de realización: Grupo de mejora.
- Actividades realizadas: Grupo de solución.
- Medición y análisis: Grupo de solución.
- Verificación de implantación: Gestor de la mejora.

4.2 Análisis del Comportamiento

Los actores identificados anteriormente se derivan de los mecanismos señalados en los diagramas IDEF, tanto para el modelo general, como para el proceso de ejecución y sistema de gestión de calidad. Los modelos IDEF0 considerados, señalan de forma explícita las entradas y salidas para cada uno de los actores involucrados en cada proceso o etapa funcional, lo cual permite definir para cada uno de ellos, las acciones ejecutables y observables, a fin de facilitar la simulación de la empresa como un sistema multiagente bajo la plataforma de simulación GALATEA. Estas acciones ejecutables y observables permiten establecer las reglas lógicas que definen el conocimiento de cada agente.

En este trabajo, se hace uso de ontologías para representar el conocimiento de los agentes, específicamente se utiliza el tipo de ontología teleológica, la cual consiste en la descomposición funcional de tareas en planes para ejecutarlas. Es decir, la ontología teleológica es el mecanismo que se emplea en esta investigación para representar el conocimiento de los agentes y, para representar la ontología, se usa un conjunto de reglas lógicas.

A continuación, se va a ejemplificar (Figura 4.1) el uso de ontologías a través de la descomposición de las actividades de un actor en sus roles específicos, hasta llegar a las reglas lógicas correspondientes. Se va a trabajar con el actor Grupo de Mejora, específicamente con sus actividades involucradas en la KPA Aseguramiento de la Calidad del Software.

El actor Grupo de Mejora pertenece al Sistema de Gestión de Calidad y se ve involucrado en cada área de proceso clave (KPA), trabajando con las siguientes características de cada KPA: objetivos de mejora, compromiso de realización y capacidad de realización de la mejora. Es decir, que el grupo de mejora trabaja fundamentalmente con la definición de las metas que se desean obtener para el desarrollo eficiente del proceso de software. Además, describe las acciones que la organización debe llevar a cabo para garantizar que el proceso de software, incluyendo sus mejoras, quede establecido y perdure,

así como también, describe los requisitos necesarios en los proyectos y en la organización para poder implementar el proceso de software competentemente.

Las acciones que la organización debe llevar a cabo para que el proceso de software quede establecido y perdure, son fijadas con patrocinio de la dirección y, una vez que estas acciones han sido implantadas, se realiza un seguimiento de los proyectos en desarrollo, bajo supervisión de la dirección.

Las reglas lógicas del agente Grupo de Mejora en la KPA Aseguramiento de la Calidad del Software, son:

```
si kpa_aseguramiento_calidad, definidas_politicas_calidad entonces
solicitar_a_direccion_compromiso_realizacion.

si kpa_aseguramiento_calidad, establecido_compromiso_realizacion
entonces definir_plan_aseguramiento_mejora.
```

La explicación de estas reglas lógicas se expresan por medio de la figura 4.6.

En la Figura 4.1 se muestra la descomposición de estas reglas lógicas:

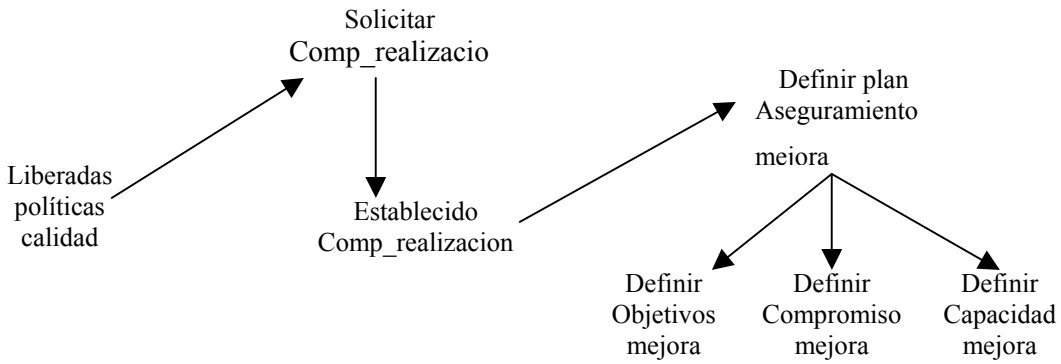


Figura 4.1. Arbol para ejemplificar el uso de ontología teleológica en la empresa

La implementación del conocimiento de cada uno de los agentes, se ha realizado bajo el lenguaje de programación lógica Prolog, pero haciendo uso bajo su contexto, de un

interpretador que permite la integración de dos lenguajes de programación lógica, como son: ACTILOG [16] y OPENLOG [17]. Este interpretador ha sido desarrollado por Jacinto Dávila. En ACTILOG se escriben las reglas que activan o disparan una meta que posteriormente producirá un efecto y, en OPENLOG se especifica el efecto producido.

El uso de este interpretador en el lenguaje Prolog consiste en colocar, en cada archivo donde se especifican las reglas lógicas de un agente, dicho interpretador como encabezado. A continuación, se muestra la implementación de las reglas lógicas del actor Grupo de Mejora bajo Prolog, haciendo uso de la notación permitida por el interpretador de la integración, es decir, que estas reglas lógicas se han convertido al formato que requiere el cerebro de los agentes GALATEA.

La base de conocimiento de un agente se especifica mediante las instrucciones "`observable`" y "`executable`". Por medio de la primera instrucción, se definen las percepciones del agente y a través de la segunda, se definen las acciones realizadas por el agente en base a sus percepciones.

```
observable(kpa_gestion_requisitos).
observable(liberado_informe_inicial_requisitos).
observable(establecido_compromiso_realizacion).
observable(kpa_planificacion_proyectos).
observable(liberado_plan_inicial_proyecto).
observable(kpa_seguimiento_proyecto).
observable(liberado_plan_proyecto_mejorado).
observable(kpa_gestion_subcontratacion_software).
observable(especificacion_recurso_a_contratar).
observable(kpa_aseguramiento_calidad).
observable(definidas_politicas_calidad).
observable(kpa_gestion_configuracion_software).
observable(especificacion_de_modulos).
observable(integracion).
observable(pruebas_iniciales).
observable(kpa_peer_review).
observable(modulos_hechos).
```

```
observable(partes_integradas).
observable(soy_peer).

executable(solicitar_a_direccion_compromiso_realizacion).
executable(definir_objetivos_mejora).
executable(definir_compromisos_mejora).
executable(definir_capacidad_mejora).
```

Una vez especificada la base de conocimiento, se tienen las reglas lógicas que definen el comportamiento del agente: "`Integrity constraints`". A continuación, debajo de algunas de las reglas lógicas se va a expresar cuál es su significado, sin embargo todas estas reglas lógicas son especificadas por medio de los diagramas de las figuras 4.2 – 4.8.

```
% Integrity constraints.

si kpa_gestion_requisitos, liberado_informe_inicial_requisitos entonces
solicitar_a_direccion_compromiso_realizacion.
```

Si estamos trabajando en el área de proceso clave Gestión de Requisitos y ya se tiene el informe inicial de requisitos entonces el grupo de mejora solicita a Dirección establecer, entre las partes involucradas, el compromiso de realización de la mejora.

```
si kpa_gestion_requisitos, establecido_compromiso_realizacion entonces
definir_plan_aseguramiento_mejora.

si kpa_planificacion_proyectos, liberado_plan_inicial_proyecto entonces
solicitar_a_direccion_compromiso_realizacion.
```

Si estamos trabajando en el área de proceso clave Planificación del Proyecto de Software y ya se tiene el plan inicial de proyecto entonces el grupo de mejora solicita a Dirección establecer, entre las partes involucradas, el compromiso de realización de la mejora.

```
si kpa_planificacion_proyectos, establecido_compromiso_realizacion
entonces definir_plan_aseguramiento_mejora.
```

```
si kpa_seguimiento_proyecto, liberado_plan_proyecto_mejorado entonces
solicitar_a_direccion_compromiso_realizacion.

si kpa_seguimiento_proyecto, establecido_compromiso_realizacion entonces
definir_plan_aseguramiento_mejora.
```

Si estamos trabajando en el área de proceso clave Seguimiento del Proyecto de Software y ya se ha establecido con Dirección el compromiso de realización de la mejora, entonces se define el plan de aseguramiento de la mejora. Este plan consiste en definir los objetivos de mejora y describir los requisitos necesarios para implementar el proceso de software competentemente.

```
Si kpa_gestion_subcontratacion_software,
especificacion_recurso_a_contratar entonces
solicitar_a_direccion_compromiso_realizacion.

si kpa_gestion_subcontratacion_software,
establecido_compromiso_realizacion entonces
definir_plan_aseguramiento_mejora.

si kpa_aseguramiento_calidad, definidas_politicas_calidad entonces
solicitar_a_direccion_compromiso_realizacion.

si kpa_aseguramiento_calidad, establecido_compromiso_realizacion entonces
definir_plan_aseguramiento_mejora.

si kpa_gestion_configuracion_software, especificacion_de_modulos entonces
solicitar_a_direccion_compromiso_realizacion.

si kpa_gestion_configuracion_software, integracion, pruebas_iniciales
entonces solicitar_a_direccion_compromiso_realizacion.

si kpa_gestion_configuracion_software, establecido_compromiso_realizacion
entonces definir_plan_aseguramiento_mejora.

si kpa_peer_review, modulos_hechos entonces
solicitar_a_direccion_compromiso_realizacion.
```

```
si kpa_peer_review, partes_integradas entonces
solicitar_a_direccion_compromiso_realizacion.

si kpa_peer_review, soy_peer, establecido_compromiso_realizacion entonces
definir_plan_aseguramiento_mejora.


% Definitions

para definir_plan_aseguramiento_mejora haga
           definir_objetivos_mejora,
           definir_compromisos_mejora,
           definir_capacidad_mejora.
```

Una vez que el archivo ha sido compilado, el Prolog genera una salida, la cual muestra, las influencias del agente, que corresponden a las acciones tratándose de ejecutar en ese instante y las metas que quedan pendientes por hacer. Las influencias y metas de salida, dependen del escenario planteado. A continuación se muestra la salida obtenida.

```
% c:/windows/pl.ini compiled 0.00 sec, 392 bytes
Welcome to SWI-Prolog (Multi-threaded, Version 5.2.7)
Copyright (c) 1990-2003 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?-
% c:/Tesis/Agentes_en_Prolog/Actores de Empresa de
Software/Actores_del_QMS/grupo_de_mejora.pl compiled 0.00 sec, 17,972
bytes
1 ?- c(I,A).

I = [definir_objetivos_mejora, definir_compromisos_mejora,
definir_capacidad_mejora]
```

La salida “I” representa el plan del agente cuando el escenario planteado es el siguiente:

```
observe solicitar_a_direccion_compromiso_realización.
observe establecido_compromiso_realización.
observe definir_plan_aseguramiento_calidad.
Observe kpa_gestion_requisitos.
```

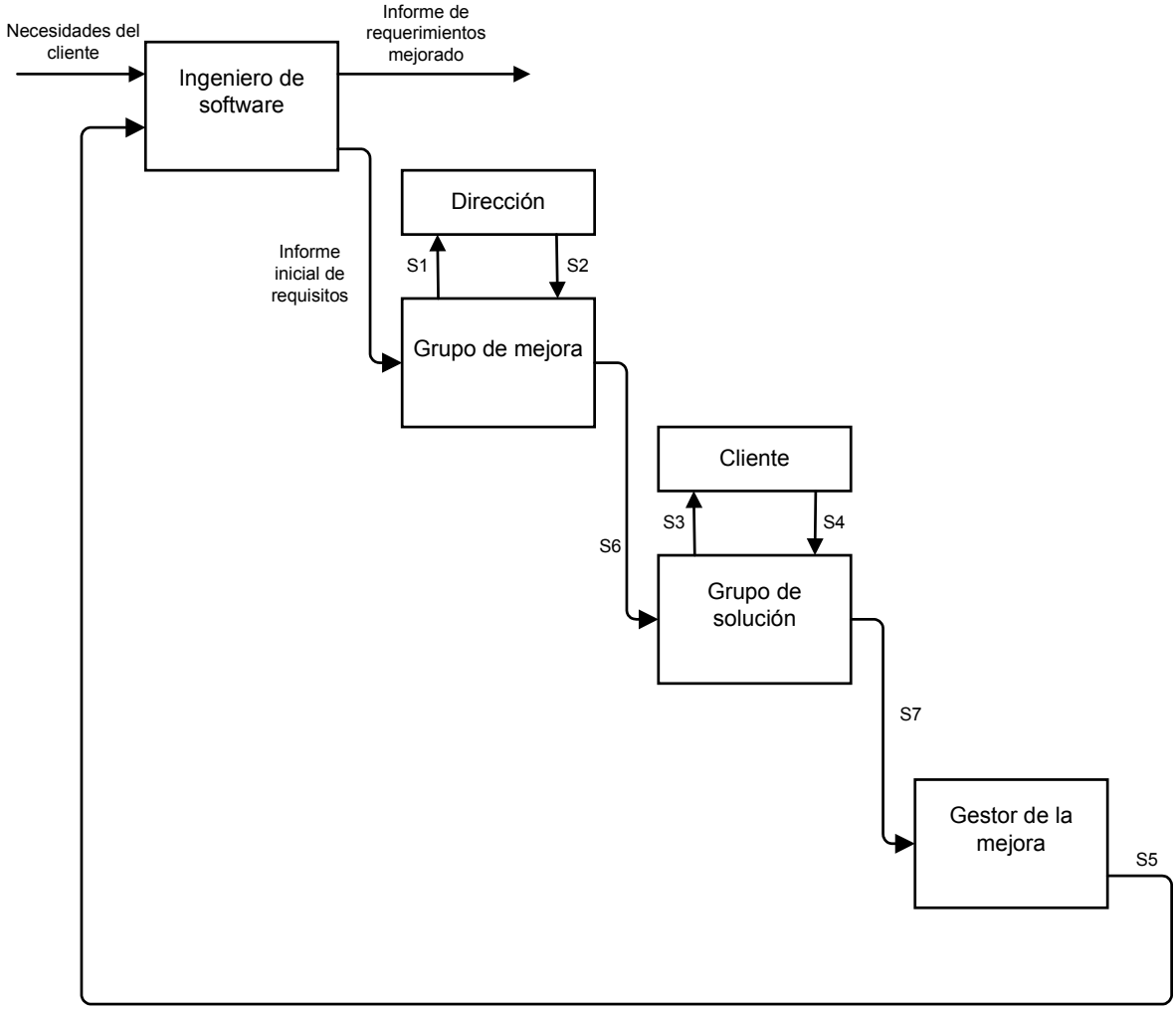
Por otra parte, para establecer las relaciones entre cada uno de los actores involucrados en una empresa para desarrollo de software, se trabaja por áreas de proceso clave, es decir, se identifica qué actor del proceso de ejecución y del sistema de gestión de calidad, tienen relación con cada una de las áreas de proceso clave. Eso se hace a continuación.

1. KPA: Gestión de requisitos

La Figura 4.2, muestra las relaciones entre los actores involucrados con la KPA gestión de Requisitos.

Cada caja en la figura anterior representa a cada uno de los actores involucrados con ésta área de proceso clave. El ingeniero de software, como actor principal de la etapa de *Análisis y Especificación de Requisitos*, conoce las necesidades del cliente para cada proyecto o sistema programado a desarrollar. En base a esas necesidades, el ingeniero de software genera como salida el “informe inicial de requisitos”, el cual es enviado al grupo de mejora. Este actor a su vez hace una evaluación de ese informe y solicita al actor Dirección establecer el compromiso de realización de la mejora entre las partes involucradas (S1). Cuando Dirección ya ha establecido el compromiso de realización de la mejora (S2), el grupo de mejora define los objetivos de mejora y, describe los requisitos necesarios para implementar el proceso de software competentemente (S6). Esta información es enviada al grupo de solución, quién realiza una encuesta al cliente (S3), obteniéndose ciertos resultados posteriores a la evaluación de la misma (S4). En función de los resultados obtenidos por medio de las encuestas, el grupo de solución describe las actividades, roles y procedimientos necesarios para implementar las mejoras

correspondientes a esta área de proceso clave y además, describe las prácticas de medición básicas necesarias para determinar el estado en relación con el proceso (S7). Este reporte de actividades es enviado al gestor de la mejora, quién genera un documento que establece si es necesario o no, realizar cambios al informe de requisitos. En caso de ser necesario, el ingeniero de software genera el "informe de requisitos mejorado", en caso contrario, se sigue trabajando con el "informe inicial de requisitos".



S1: Solicita compromiso de realización de la mejora
S2: Establece el compromiso de realización de la mejora
S3: Realiza encuesta al cliente
S4: Resultados de la evaluación de la encuesta
S5: Establece si es necesario o no realizar cambios al informe de requisitos
S6: Objetivos de mejora, compromiso y capacidad
S7: Actividades, medición y análisis

Figura 4.2. KPA: Gestión de Requisitos.

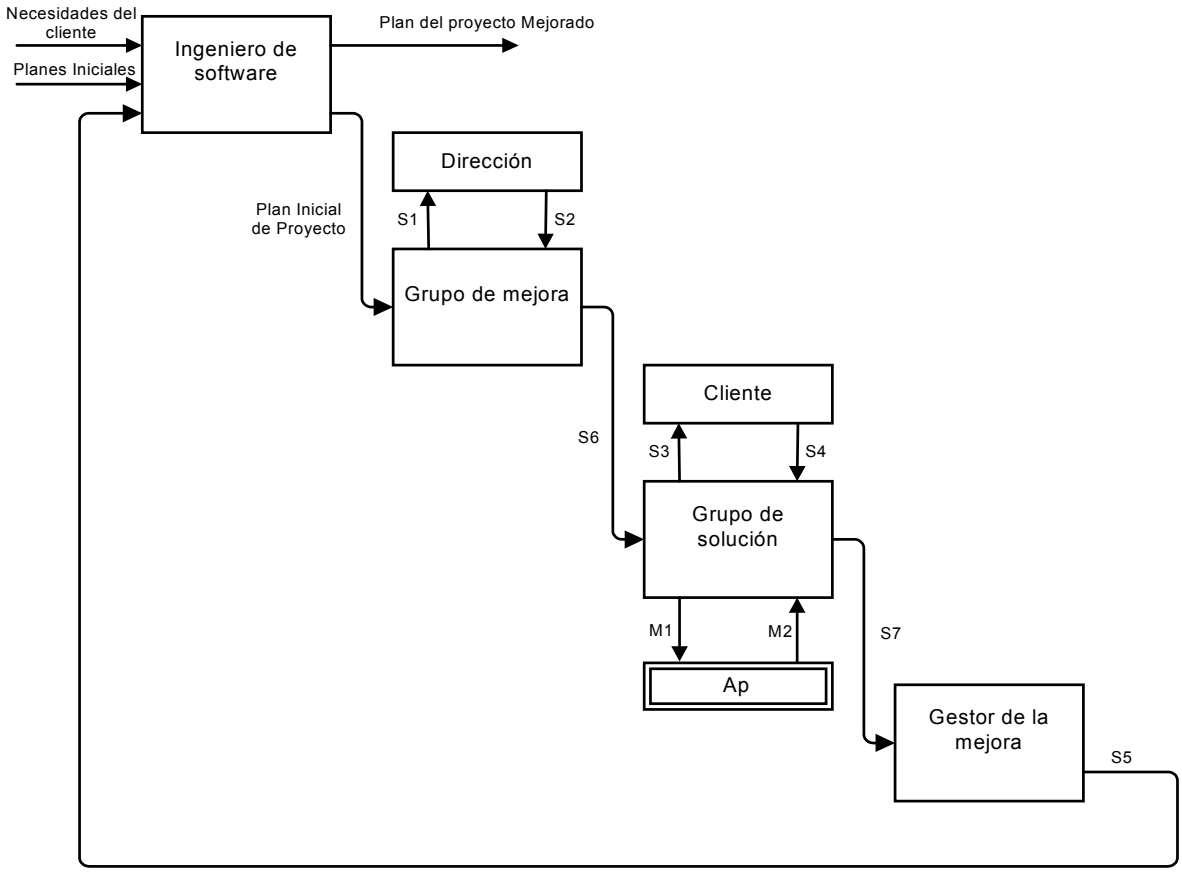
Las reglas lógicas correspondientes a las interacciones de estos actores o agentes, se presentan en los anexos 1, 6 - 10: KPA Gestión de Requisitos.

2. KPA: Planificación de los proyectos de software

La Figura 4.3, muestra las relaciones entre los actores involucrados con la KPA Planificación de Proyectos de Software.

El diagrama que se muestra en la figura 4.3 es genérico, ya que se observa que en el mismo existe un recuadro doblemente bordeado, el cual corresponde a diferentes actores Ap. Por lo tanto, es necesario especificar que existe un diagrama igual para cada uno de los actores Ap que se señalan.

El ingeniero de software conoce las necesidades del cliente y los planes iniciales para desarrollar un sistema programado efectivo, en base a esta información, genera el "plan inicial de proyecto", el cual es enviado al grupo de mejora. Este actor a su vez, realiza una evaluación del plan recibido y solicita a Dirección establecer el compromiso de realización de la mejora entre las partes involucradas (S1). Ya establecido este compromiso por parte del actor Dirección (S2), el grupo de mejora define los objetivos de mejora correspondientes a esta KPA, así como también, los requisitos necesarios para implementar el proceso de software competentemente (S6). Este reporte obtenido es enviado al grupo de solución, el cual realiza encuestas tanto al cliente como al personal de ejecución (Diseñador, Programador, Depurador y Mantenimiento). Posteriormente, a partir de los resultados obtenidos por medio de las encuestas, el grupo de solución describe las actividades y procedimientos necesarios para implementar las mejoras respectivas y además, describe las prácticas de medición necesarias para determinar el estado en relación con el proceso (S7). Este reporte de actividades es enviado al gestor de la mejora, quién genera un documento que establece si es necesario o no, realizar cambios al "plan inicial de proyecto". En caso de ser necesario, el ingeniero de software genera el "plan de proyecto mejorado" y en caso contrario, se sigue trabajando con el "plan inicial de proyecto".



S1, S2, S3, S4, S6, S7: Ver Figura 4.2
S5: Establece si es necesario o no realizar cambios al Plan Inicial de Proyecto
M1: Realiza encuesta al personal de ejecución
M2: Resultados de la evaluación de la encuesta

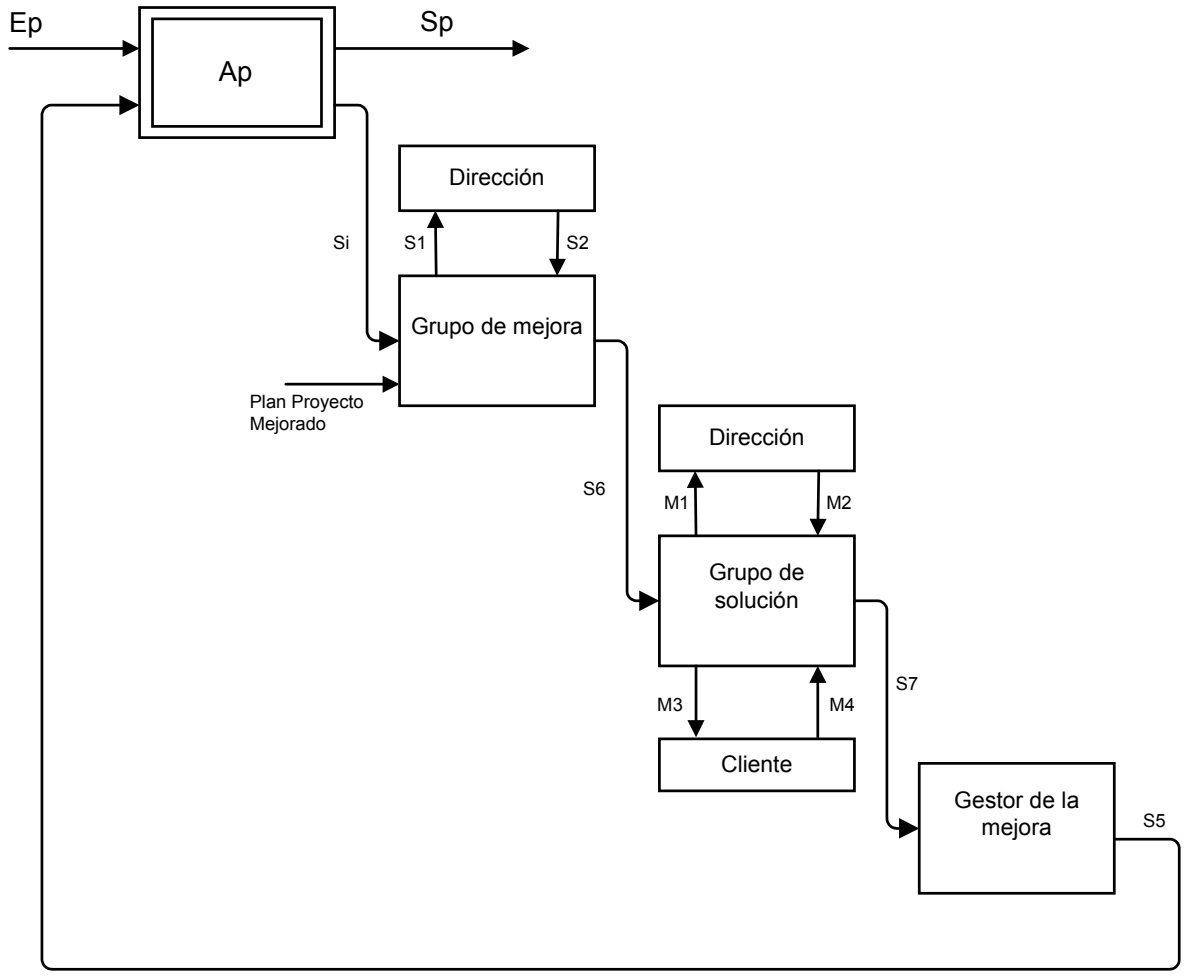
Ap: - Diseñador
- Programador
- Depurador
- Mantenimiento y Actualización.

Figura 4.3. KPA: Planificación del Proyecto de software

Las reglas lógicas correspondientes a las interacciones de estos actores o agentes, se presentan en los anexos 1 - 10: KPA Planificación del Proyecto de Software.

3. KPA: Seguimiento y control del proyecto

La Figura 4.4, muestra las relaciones entre los actores involucrados con la KPA Seguimiento del Proyecto de Software.



S5: Medidas Correctivas
M1: Solicita - Restablecer - Compromiso (Internos)
M2: Compromiso Interno Restablecido
M3: Solicita - Restablecer - Compromiso (Externos)
M4: Compromiso Externo Restablecido
S1, S2, S6, S7: Ver Figura 4.2

Ap: - Diseñador
- Programador
- Depurador
- Mantenimiento y actualización

Figura 4.4. KPA: Seguimiento del Proyecto de software

Al igual que en el gráfico anterior, se muestra para esta KPA, un diagrama genérico, es decir, que existe uno igual para cada actor Ap (Diseñador, Programador, Depurador y Mantenimiento) .

La entrada Ep que se muestra en el diagrama varía dependiendo del actor Ap con el cual se esté trabajando. A continuación se explicará el diagrama de la figura 4.4 para el caso en

que el actor Ap es el Diseñador y, debido a que dicho diagrama se repite para cada actor Ap, sólo se especificarán para cada uno de los restantes, las entradas Ep y las salidas Sp, respectivas.

El Diseñador tiene a su alcance el informe de análisis de requisitos y el plan de proyecto definitivos y, a partir de esta documentación describe las especificaciones de la arquitectura del sistema programado que se está gestionando (Si). Estas especificaciones del sistema son enviadas al grupo de mejora, quien conociendo el plan de proyecto definitivo, solicita al actor Dirección establecer el compromiso de realización de la mejora entre las partes involucradas (S1). Una vez que Dirección ha establecido el compromiso de realización (S2), el grupo de mejora define los objetivos de mejora y, describe los requerimientos necesarios para implementar el proceso de software competentemente (S6). Este reporte generado es enviado al grupo de solución, quién realiza una evaluación del mismo con el fin de supervisar el progreso real del sistema programado en comparación con las estimaciones y los planes y, tomar acciones correctivas cuando surjan discrepancias entre las estimaciones y los valores reales. Por otra parte, en caso de ser necesario acciones correctivas, el grupo de solución solicita a Dirección reestablecer el compromiso interno de realización de la mejora (M1) y a su vez, solicita al cliente reestablecer el compromiso externo para la realización de la misma (M3). Cuando el compromiso de realización, tanto interno como externo, hayan sido establecidos (M2, M4), el grupo de solución describe las prácticas de medición básicas necesarias para determinar el nuevo estado en relación con el proceso (S7). Por su parte, el actor gestor de la mejora realiza una evaluación para determinar el estado de desarrollo del sistema programado y genera las medidas correctivas necesarias, es decir, realiza una supervisión del proceso (S5). Si es necesario medidas correctivas, entonces se realizan los cambios correspondientes en las especificaciones de la arquitectura del sistema programado (Sp) y, en caso contrario se sigue trabajando con las especificaciones iniciales del sistema (Si).

Las reglas lógicas correspondientes a las interacciones de estos actores o agentes, se presentan en los anexos 2 - 10: KPA Seguimiento del Proyecto de Software.

Datos de los actores: Programador, Depurador, Mantenimiento y Actualización

Actor Ap	Entradas Ep	Salida Si	Salida Sp
Programador	▪ Especificaciones del sistema. ▪ Plan de proyecto definitivo	▪ Módulos ▪ Documentación de los módulos	▪ Módulos y documentación revisados
Depurador	▪ Módulos ▪ Documentación de los módulos	▪ Producto de software	▪ Producto de software mejorado
Mantenimiento y Actualización	▪ Producto de software mejorado	▪ Actualizaciones posibles	▪ Actualizaciones realizadas

4. KPA: Gestión de subcontratación del software

La Figura 4.5, muestra las relaciones entre los actores involucrados en la KPA Gestión de Subcontratación de Software.

Al igual que para el área de proceso clave descrita anteriormente, se presenta un diagrama genérico, donde el actor Ap puede ser el Programador o el Depurador. De nuevo, se hará la explicación del diagrama para el actor Programador y luego, se tabularán las entradas Ep y salidas Sp del actor Depurador.

El programador tiene como entradas las especificaciones de la arquitectura del sistema programado y el plan de proyecto definitivo. En función de la evaluación de estas entradas, el programador hace una especificación del recurso necesario a contratar (este recurso puede ser Software o Personal).

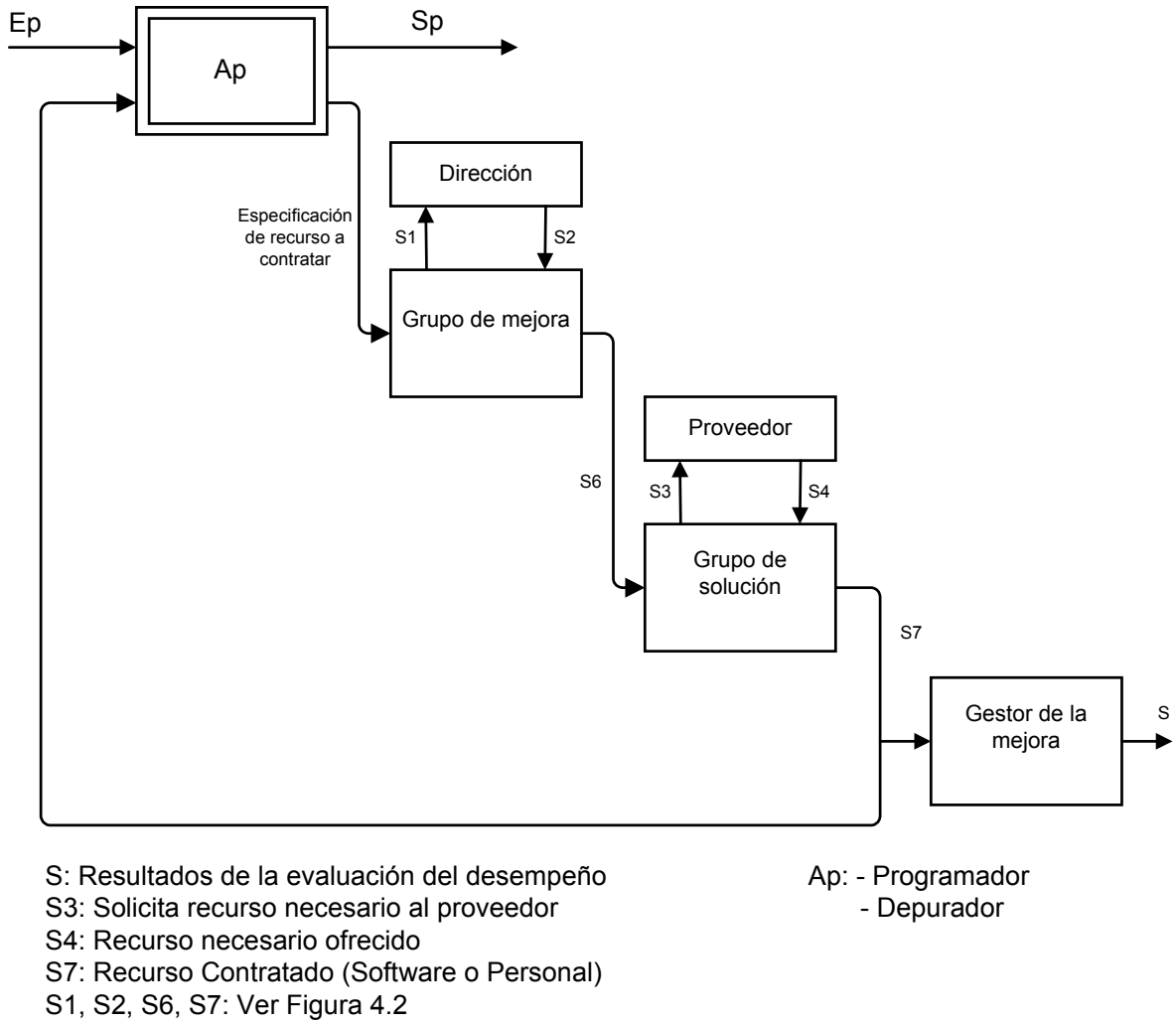


Figura 4.5. KPA: Gestión de Subcontratación de Software

Esta información es enviada al grupo de mejora para ser evaluada y posteriormente solicitar a dirección establecer el compromiso de realización para hacer efectiva la subcontratación necesaria (S1). Ya establecido este compromiso (S2), el grupo de mejora define los objetivos de la subcontratación y describe los requisitos necesarios para realizar la subcontratación (S6). Estos requisitos se hacen operativos a través del grupo de solución, quién solicita al proveedor correspondiente, el recurso a contratar (S3). Una vez contratado el recurso (S4), es utilizado apropiadamente por el grupo de solución (S7), para producir como salida fundamental en la fase de Implementación (ver figura 3.4), en el cual se ve involucrado como actor principal el programador, los módulos y su documentación (Sp). El

gestor de la mejora hace el seguimiento y revisión del desempeño del recurso contratado, generando como salida los resultados de esa evaluación (S).

Las reglas lógicas correspondientes a las interacciones de estos actores o agentes, se presentan en los anexos 3 – 8, 10: KPA Gestión de Subcontratación del Software.

Datos del actor Depurador

Actor Ap	Entradas Ep	Salida Sp
Depurador	▪ Módulos ▪ Documentación de los módulos	▪ Producto de software ▪ Errores del producto

5. KPA: Aseguramiento de la Calidad del Software

La Figura 4.6, muestra las relaciones entre los actores involucrados en la KPA Aseguramiento de la Calidad del Software.

El diagrama de la figura 4.6 es genérico, donde Ap puede ser cualquiera de los siguientes actores: Ingeniero de software, Diseñador, Programador, Depurador y Mantenimiento.

El actor Dirección genera las políticas para el aseguramiento de la calidad, es decir, planifica las actividades de aseguramiento de la calidad. Estas políticas establecidas por la Dirección son tomadas en cuenta por el actor grupo de mejora, quién solicita a Dirección establecer, entre las partes involucradas, el compromiso necesario para poder revisar y auditar objetivamente los productos y las actividades, de tal manera que se pueda verificar que están conformes con los procedimientos y estándares aplicables (S1). Por su parte, Dirección establece el compromiso y da su visto bueno al grupo de mejora (S2). Además, el grupo de mejora define los procedimientos o mecanismos que permitan lograr que las actividades realizadas estén acordes con las políticas de calidad establecidas por la

Dirección (S6). El grupo de solución hace operativos o efectivos estos mecanismos y describe las prácticas de medición básicas necesarias para determinar el estado en relación con el proceso (S7). Una vez hecha la evaluación del estado actual del proceso de desarrollo del sistema programado, el gestor de la mejora define las mejores prácticas, es decir, los procedimientos o métodos que permitan mejorar los resultados obtenidos por la realización de cierta actividad. Estas mejores prácticas son enviadas a cada actor Ap para ser puestas en funcionamiento.

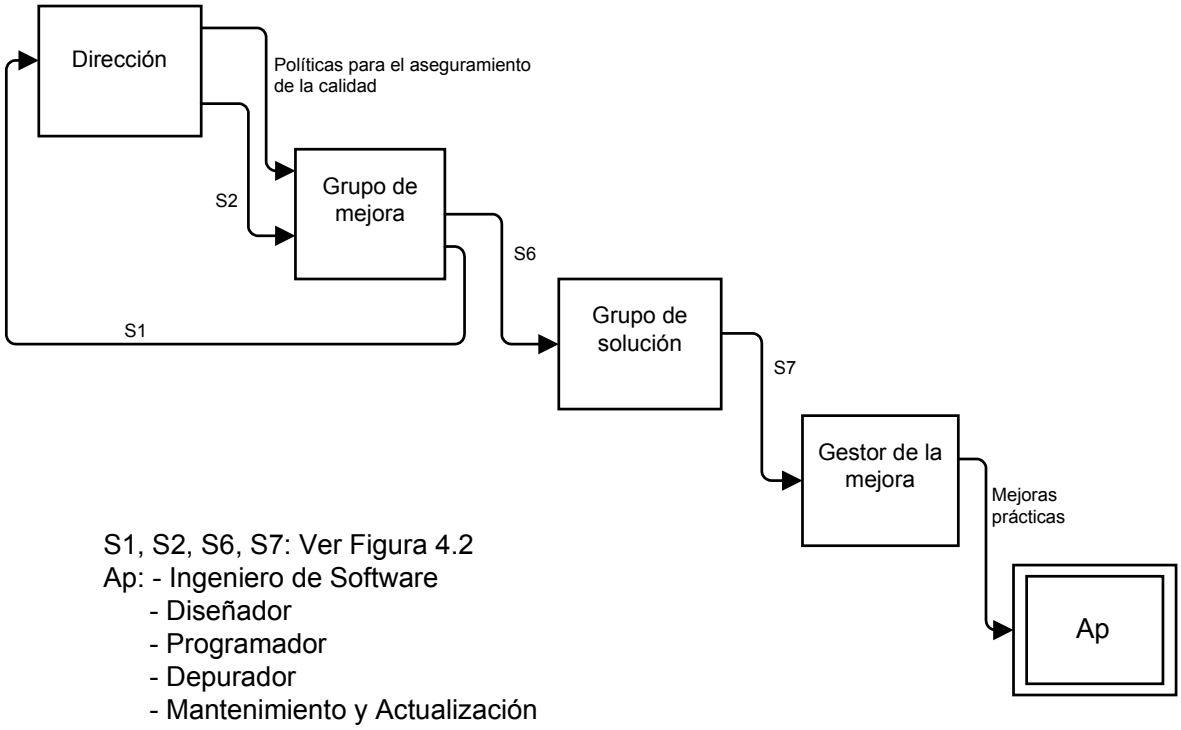


Figura 4.6. KPA: Aseguramiento de la Calidad del Software

Las reglas lógicas correspondientes a las interacciones de estos actores o agentes, se presentan en los anexos 1 – 8, 10: KPA Aseguramiento de la Calidad del Software.

6. KPA: Gestión de configuración del software

La Figura 4.7, muestra las relaciones entre los actores involucrados en la KPA Gestión de Configuración del Software, pero teniendo como actor principal, entre los actores del proceso de ejecución, al Programador.

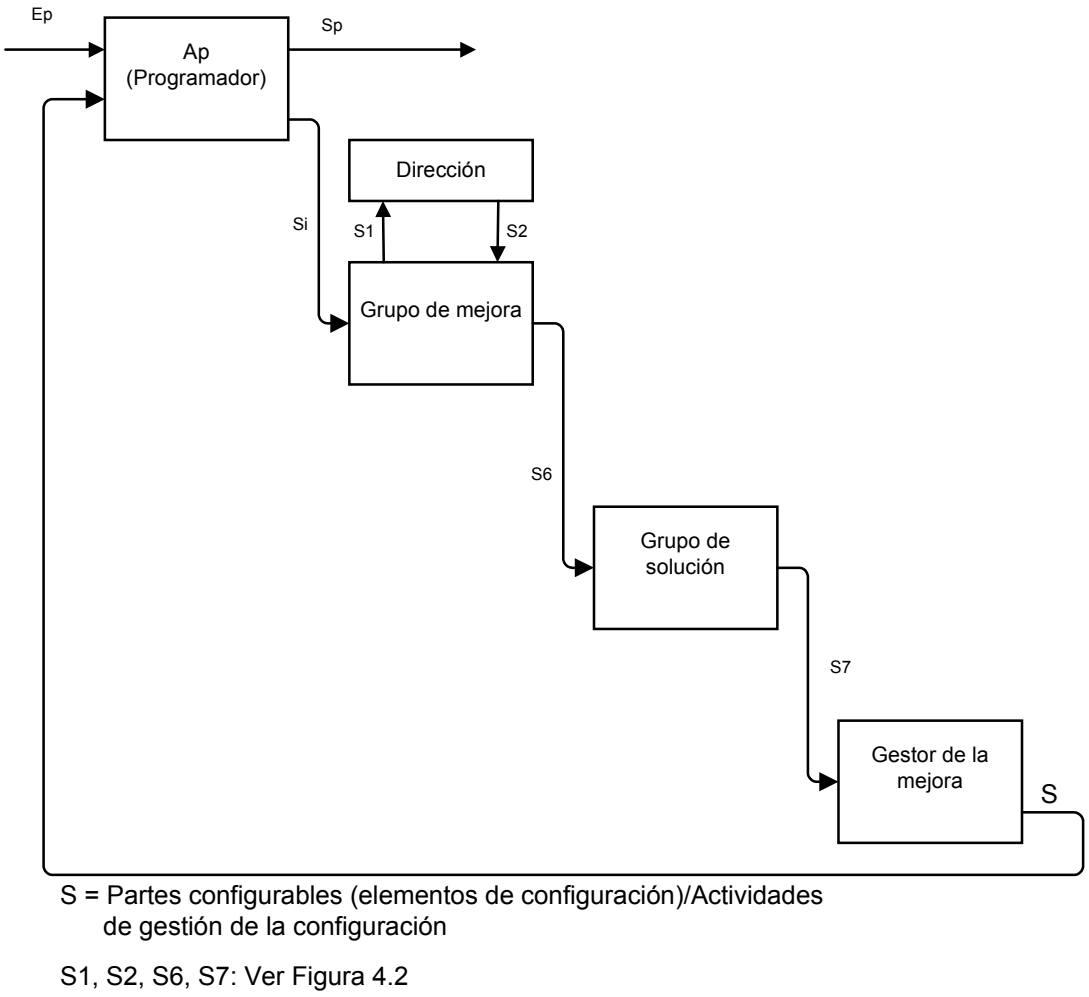


Figura 4.7. KPA: Gestión de Configuración del Software (Programador)

El diagrama que se presenta en esta KPA es genérico, donde Ap puede ser cualquiera de los dos actores siguientes: Programador o Depurador. Se hará la explicación del diagrama para el actor Programador y, para el actor Depurador se colocará en una tabla las especificaciones de Ep, Sp, Si y S.

El Programador tiene como entradas las especificaciones del sistema y el plan de proyecto definitivo. Posterior a la evaluación de estas entradas, el Programador genera un reporte donde hace una especificación de los módulos requeridos por el sistema programado (Si). Este reporte es enviado al grupo de mejora quién solicita a Dirección establecer el compromiso para realizar formalmente las gestiones necesarias (S1). Una vez establecido el compromiso (S2), el grupo de mejora define los requisitos de configuración y también los mecanismos que permitan implementarlos (S6). El grupo de solución realiza una evaluación del estado del sistema programado respecto a las actividades de gestión de la configuración (S7) y envía esa información al gestor de la mejora. El gestor de la mejora establece las partes configurables o los elementos de configuración y de acuerdo a esta información, el Programador tiene como salida principal los módulos codificados y su documentación (Sp).

Las reglas lógicas correspondientes a las interacciones de estos actores o agentes, se presentan en los anexos 3 - 4, 6 – 8, 10: KPA Gestión de Configuración del Software.

Datos del actor Depurador

Actor Ap	**Depurador**
Entradas Ep	▪ Módulos y su documentación ▪ Plan de proyecto definitivo
Salida Si	▪ Integración y pruebas iniciales
Salida Sp	▪ Producto de software definitivo
Salida S	▪ Propuestas para la versión final del producto

7. KPA: Peer Review

La Figura 4.8, muestra las relaciones entre los actores involucrados con la KPA Peer Review.

El diagrama que se presenta a continuación, es genérico, donde Ap puede ser cualquiera de los dos actores siguientes: Programador y Depurador. Esto quiere decir que existe un diagrama igual para cada actor Ap. Se hará la explicación del diagrama para el actor Programador y para el actor Depurador se resumirá en una tabla las entradas Ep y las salidas Sp y S.

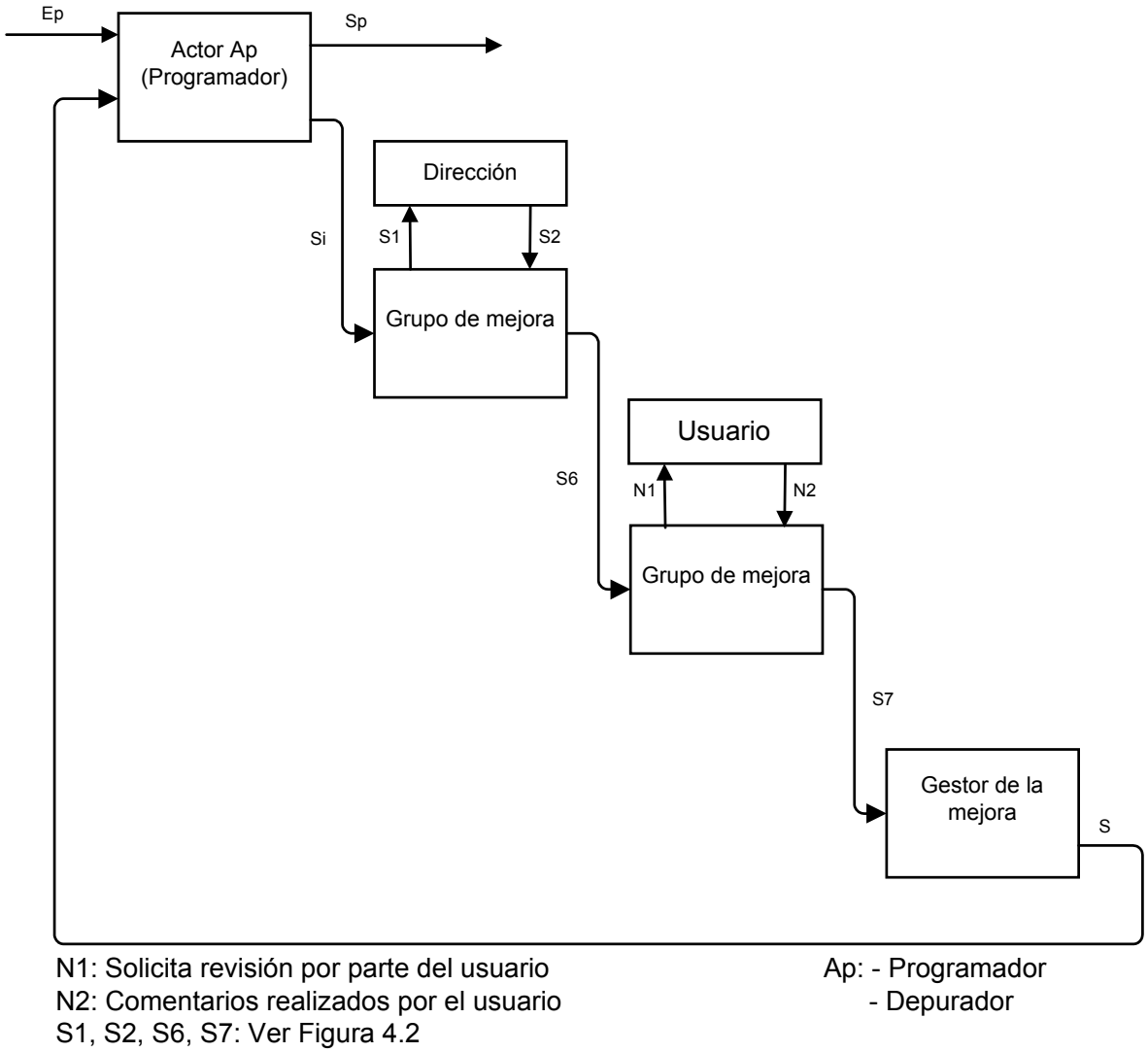


Figura 4.8. KPA: Peer Review

El Programador tiene como entradas las especificaciones de la arquitectura del sistema programado y el plan de proyecto definitivo (Ep) y, genera como una de sus salidas, los módulos ya codificados (Si). Estos módulos son enviados al grupo de mejora, el cual

cuenta con un experto en la codificación de módulos y que conjuntamente con el Programador hace una revisión o evaluación de los módulos generados, detectando los defectos, para ser removidos posteriormente. Esta revisión entre pares (Programador y experto), que corresponde a la KPA Peer Review, puede significar una fuente de ahorro considerable. Una vez hecha esta revisión, el grupo de mejora solicita a Dirección establecer el compromiso requerido entre las partes involucradas para hacer efectivos los cambios necesarios (S1). Cuando ya se ha establecido el compromiso por parte de Dirección (S2), el grupo de mejora describe los defectos encontrados y genera los resultados de la revisión (S6). Por su parte, el grupo de solución, en función de los defectos encontrados, solicita al usuario hacer nuevamente una revisión de los resultados obtenidos (N1) y el usuario hace saber sus comentarios al respecto (N2). Posteriormente, el grupo de mejora describe las prácticas de medición básicas para analizar el avance del desarrollo del sistema programado (S7). El gestor de la mejora le hace saber al Programador los errores detectados en los módulos y las propuestas de cambio a la documentación de los mismos (S) y finalmente, el Programador genera los módulos y su documentación definitivos (Sp).

Las reglas lógicas correspondientes a las interacciones de estos actores o agentes, se presentan en los anexos 3 - 4, 6 – 8, 10, 13: KPA Peer Review.

Datos del actor Depurador

Actor Ap	**Depurador**
Entradas Ep	▪ Módulos y su documentación ▪ Plan de proyecto definitivo
Salida Si	▪ Partes integradas
Salida Sp	▪ Producto de software definitivo
Salida S	▪ Errores detectados en la integración y en las pruebas

El actor *Mantenimiento y Actualización* del proceso de ejecución, no es afectado directamente por ninguna área de proceso clave y además, está relacionado con el actor Ingeniero de software y el actor Usuario. A continuación se presenta la Figura 4.9, la cual

muestra las relaciones entre los actores: Mantenimiento y actualización, Ingeniero de Software y Usuario.

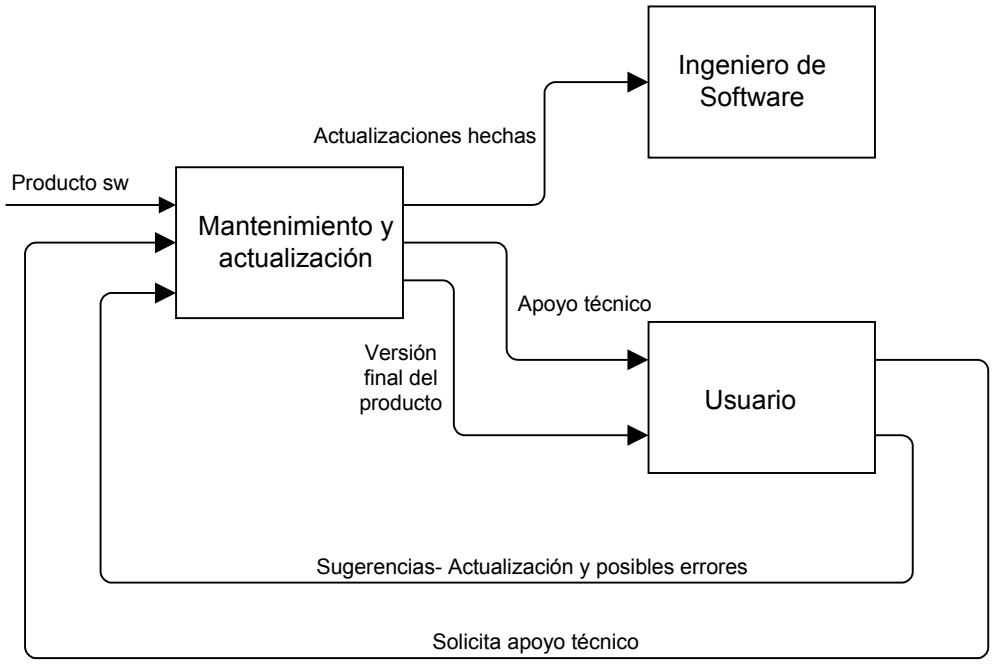


Figura 4.9. Actor Mantenimiento y actualización

Una vez que se tiene el sistema programado o producto de software, el actor Mantenimiento y Actualización genera la versión final del producto y le proporciona apoyo técnico al usuario en caso de que éste lo haya solicitado. El usuario detecta posibles errores en el software y le hace sugerencias al actor Mantenimiento y finalmente, éste último le proporciona las actualizaciones hechas al ingeniero de software.

Las reglas lógicas correspondientes a las interacciones de estos actores o agentes, se presentan en los anexos 1, 5 y 13.

Por otra parte, a través de la especificación de las relaciones de los actores por medio de las áreas de proceso clave, se ha podido establecer las relaciones entre los actores del proceso de ejecución y los actores del sistema de gestión de calidad. A continuación se establecen las relaciones entre los *actores involucrados en el proceso de ejecución.*

La Figura 4.10, muestra las relaciones entre algunos de los actores del proceso de ejecución (Diseñador, Programador y Depurador).

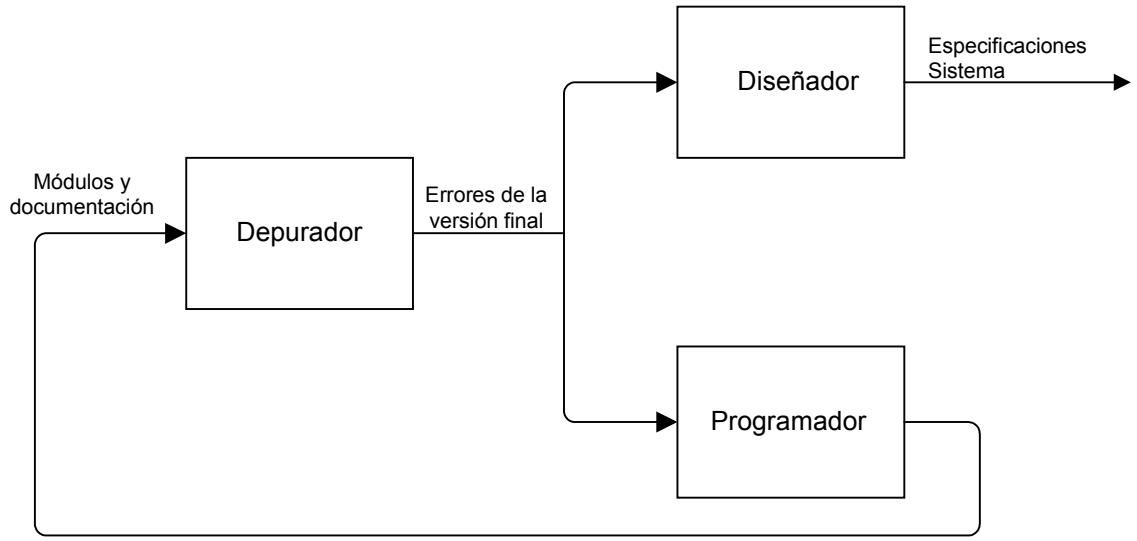


Figura 4.10. Relaciones entre los actores Diseñador, Programador y Depurador

En la figura anterior se observa que el actor Depurador envía a los actores Diseñador y Programador los errores de la versión final del producto de software. Por su parte, el actor Diseñador verifica las especificaciones del sistema y el actor Programador hace una revisión de los módulos y su documentación.

Las reglas lógicas correspondientes a las interacciones de estos actores o agentes, se presentan en los anexos 2, 3 y 4.

Una vez establecidas las relaciones entre los actores del proceso de ejecución, se procede a establecer las relaciones entre los *actores de gestión.*

La Figura 4.11, muestra las relaciones del actor, Gestor de Proyectos de los actores de Gestión, con los actores de Dirección y Cliente. El gestor de proyectos tiene contacto directo con el cliente, ya que éste último le hace saber sus necesidades o requerimientos (1). En función de estas necesidades y en base a las políticas de gestión de proyectos proporcionadas por el actor Dirección (2), el gestor de proyectos genera un proyecto inicial

(3), el cual es enviado a Dirección para su respectiva evaluación. Una vez hecha esta evaluación, Dirección da su visto bueno o proporciona al gestor de proyectos los posibles cambios a ese proyecto inicial (4). De acuerdo a esta información, el gestor de proyectos motiva al cliente a través de la innovación y la creatividad y se anticipa a los posibles problemas (5) y además, le envía al cliente un proyecto preliminar con toda la documentación necesaria (6). Por su parte, el cliente estudia el proyecto preliminar y da su visto bueno o le solicita al gestor de proyectos, cambios posibles (7). Finalmente, el gestor de proyectos se encarga de las actividades iniciales para gestionar el proyecto o sistema programado a desarrollar (8).

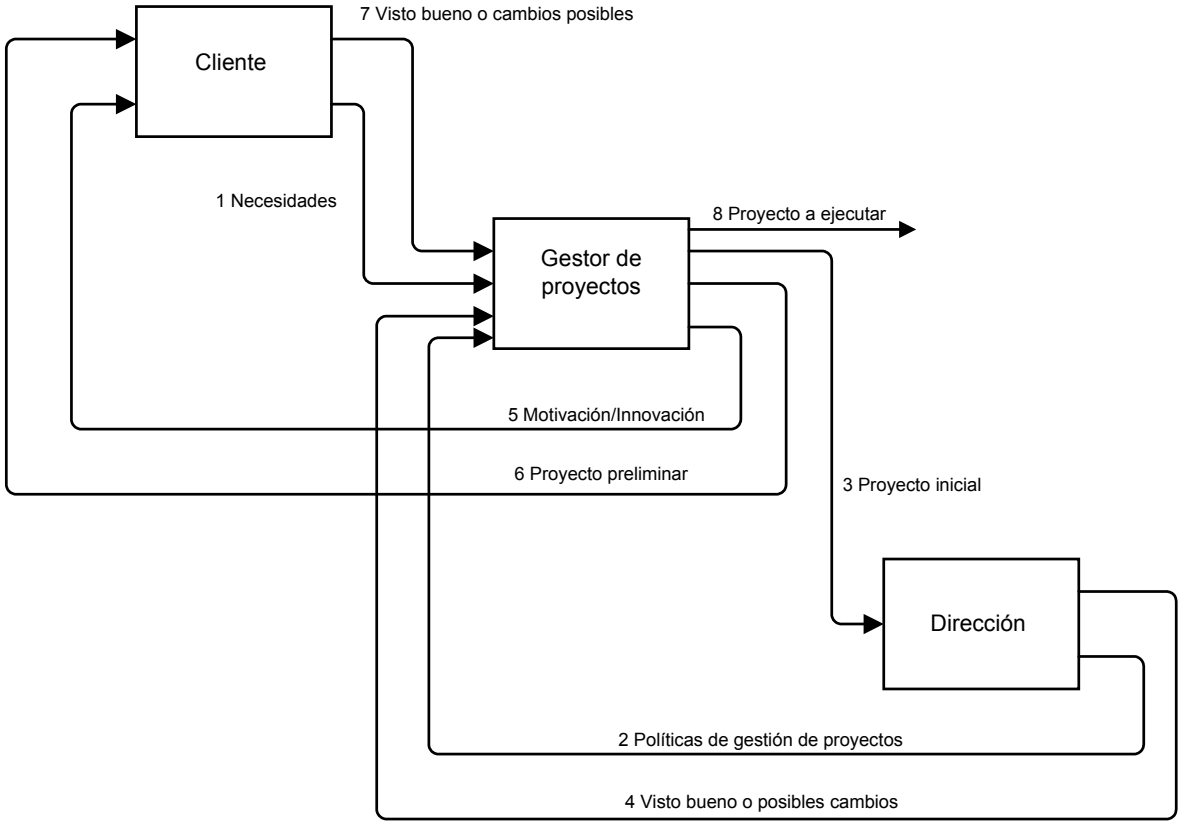


Figura 4.11. Relaciones entre los actores Cliente, Gestor de Proyectos y Dirección

Las reglas lógicas correspondientes a las interacciones de estos actores o agentes, se presentan en los anexos 9, 10 y 12.

La Figura 4.12, muestra las relaciones entre los actores Evaluador de Proyectos, Dirección y Cliente.

El evaluador de proyectos recibe como entrada el proyecto inicial a ejecutar, propuesto por el gestor de proyectos (1), y las políticas de evaluación por parte de Dirección (2). El evaluador de proyectos hace una evaluación de ese proyecto inicial y envía los resultados obtenidos a Dirección (3) y además, le brinda asesoramiento al cliente (4). El cliente le hace saber al evaluador de proyectos sus comentarios (5) y finalmente, el evaluador redefine los planes para el desarrollo del sistema programado (6).

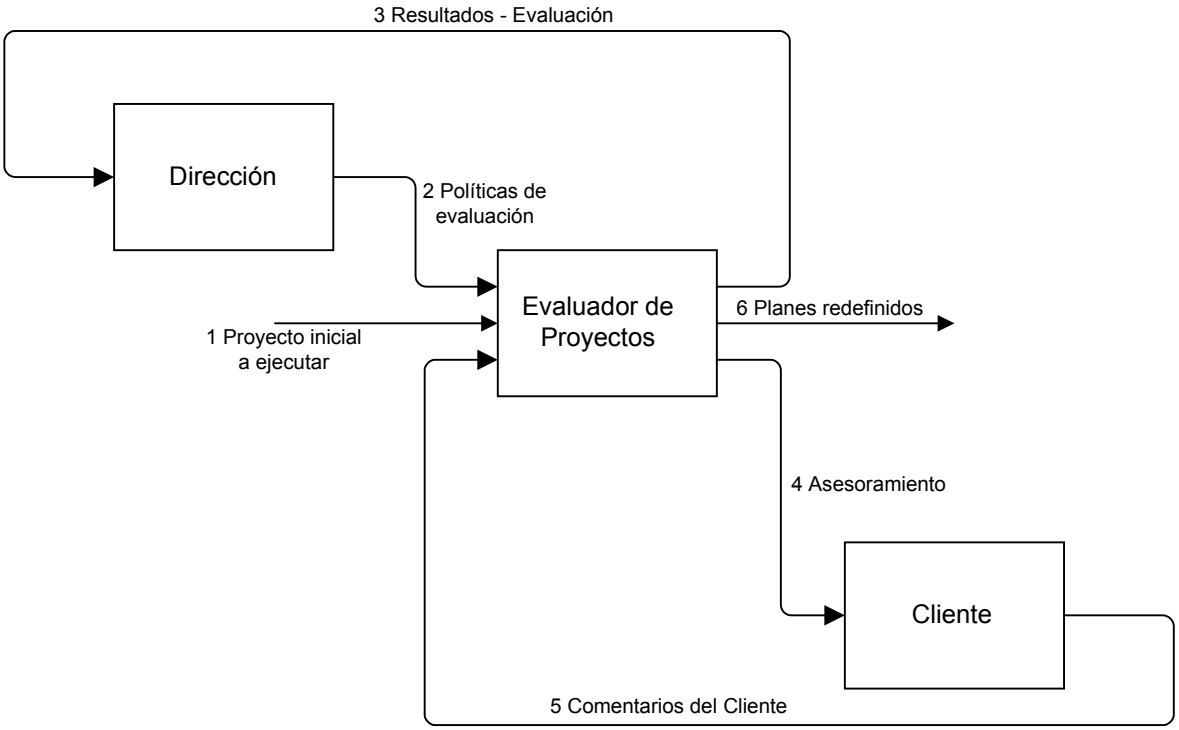


Figura 4.12. Relaciones entre los actores Dirección, Evaluador de Proyectos y Cliente

4.3 Descripción del flujo secuencial de un proyecto desarrollado por la empresa

Una vez presentado el modelo de una empresa para desarrollo de software, se procedió a implementar en el programa de simulación GLIDER [12], el flujo secuencial

correspondiente al proceso de ejecución de un proyecto desarrollado dentro de dicha empresa (Ver anexo 14); en la mencionada implementación se ha incorporado, tal como se muestra en el modelo, la Gestión de Calidad.

Para simular el comportamiento secuencial de un sistema programado o proyecto desarrollado en la empresa de software, se han considerado cada una de las etapas funcionales del proceso de ejecución, como nodos tipo "Recurso" en la semántica GLIDER. De igual manera, el sistema de gestión de calidad (QMS) ha sido considerado como un nodo tipo "Recurso" (Ver Figura 4.14). Para el programa de simulación GLIDER, un nodo tipo recurso representa un subsistema en el que cierto tipo de recurso es consumido durante un cierto tiempo.

Para simular el flujo secuencial de los proyectos desarrollados por la empresa en GLIDER, los proyectos son representados por mensajes que han sido generados por medio de un nodo tipo entrada, el cual se define en la semántica GLIDER como un nodo que crea mensajes (proyectos) y los envía a los nodos sucesores (etapas subsecuentes). En el caso de la figura 4.13, los mensajes son generados por el nodo de entrada "Necesidades".

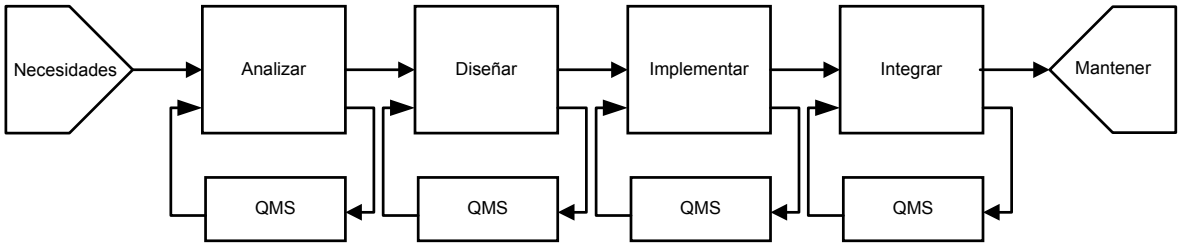


Figura 4.13. Modelo Glider de la empresa para desarrollo de software

La primera etapa funcional a la que entra todo proyecto generado, es la etapa de Analizar y siguiendo el comportamiento secuencial señalado en el modelo de la Figura 3.4, las etapas subsecuentes son Diseñar, Implementar, Integrar y finalmente Mantener.

Como se ha señalado anteriormente, en el sistema de gestión de calidad, se han establecido siete áreas de proceso clave que sirven como puntos de referencia para evaluar

el desarrollo de la empresa de software y proporcionar, en caso de ser necesario, mejoras en el proceso de software. Como se puede observar en la figura anterior, cada etapa funcional tiene contacto directo con el sistema de gestión de calidad (QMS) y, para cada una de ellas se han determinado las áreas de proceso clave involucradas en la gestión de calidad. De esta manera se tiene:

- Etapa Analizar está gestionada por las KPA's de Gestión de Requisitos y Planificación del Proyecto de software;

- Etapa Diseñar está involucrada o gestionada por la KPA de Seguimiento y Control del Proyecto de software;

- Etapas Implementar e Integrar están involucradas o trabajan con las KPA's de Seguimiento y Control del Proyecto de Software, Gestión de Subcontratación de Software, Gestión de Configuración de software y Peer Review.

- Etapa Mantener para efectos de este trabajo representa la finalización o culminación del sistema programado.

Definiendo el marco experimental:

Para efectos de la simulación, se han considerado tres tipos de proyectos que se generan:

- Proyectos Tipo 1; tienen un tiempo promedio de duración en el sistema de un año.

- Proyectos Tipo 2; tienen un tiempo promedio de duración en el sistema de dos años.

- Proyectos Tipo 3; tienen un tiempo promedio de duración en el sistema de cinco años.

Además, se va a considerar que un año de actividad equivale a once meses de actividad de 160 horas por mes y que cada unidad de tiempo en el programa de simulación GLIDER equivale a una hora de actividad o desarrollo del proyecto. También se ha asumido que las áreas de proceso clave involucradas en cada una de las etapas, son evaluadas o desempeñadas secuencialmente por el sistema de gestión de calidad.

Por ejemplo, si se genera un proyecto de algún tipo específico, este pasa a la etapa de Analizar donde permanece un tiempo consumiendo ese recurso hasta generar el informe inicial de requisitos. Este informe es enviado al sistema de gestión de calidad (QMS por sus siglas en inglés) para hacer la evaluación respectiva; si el informe inicial de requisitos no requiere de cambios, entonces el QMS da el visto bueno del informe a la etapa Analizar, en donde se comienza a trabajar con la siguiente KPA la cual está relacionada con la planificación del proyecto de software.

Por otra parte, si el QMS establece que es necesario hacer cambios al informe inicial de requisitos, entonces la etapa Analizar debe trabajar con los cambios requeridos y enviarlo nuevamente al QMS para su evaluación. Este ciclo se repite hasta que el QMS da el visto bueno al informe y de esta manera la etapa Analizar comienza a trabajar con el plan de proyecto (segunda KPA de esta etapa). Se pasa a la etapa siguiente, cuando el QMS haya dado la aprobación de las salidas generadas por cada KPA en la etapa Analizar. De igual forma se trabaja con las demás etapas funcionales involucradas en el proceso de ejecución.

Cálculo de los tiempos promedios de permanencia en cada etapa funcional:

Los tiempos promedios que el proyecto permanece en cada una de las etapas funcionales consumiendo recursos, han sido calculados de la siguiente manera:

Aprovechando la experiencia acumulada en la Ingeniería del Software [19], se han establecido, para el desarrollo de un proyecto, ciertos porcentajes para el tiempo de diseño, implementación y pruebas. Estos porcentajes son: 40% para el diseño; 20% para la implementación y 40% para las pruebas. Para efectos del trabajo, el diseño del proyecto es

realizado conjuntamente por las etapas funcionales de “Analizar” y “Diseñar”; la implementación es realizada por la etapa funcional “Implementar” y las pruebas son realizadas por la etapa funcional “Integrar”. En este trabajo, la etapa funcional “Mantener” hace una revisión de la versión final del producto. Entiéndase una etapa funcional como un proceso de la empresa, necesario para el desarrollo y posterior culminación de un proyecto.

Como el diseño del proyecto está comprendido por las etapas “Analizar” y “Diseñar” y además, requiere del 40% del tiempo total de su desarrollo, entonces se ha considerado un 20% del tiempo total para cada una de las etapas, alcanzando conjuntamente un total de 40% del tiempo. Los porcentajes para “Implementar” e “Integrar” se han dejado iguales de 20% y 40% respectivamente.

Se ejemplificará el cálculo de los tiempos para cada etapa funcional en el proceso de ejecución, para proyectos tipo 1 y, aún cuando no se realizan los cálculos para los proyectos tipo 2 y tipo3, es necesario señalar que se trabaja de manera similar. Los proyectos tipo1, como ya se mencionó, tienen una duración promedio de un año, que en horas equivales a 1760 horas (160 horas/mes * 11 meses), de esta manera se tiene la siguiente asignación en horas para cada una de las etapas:

Tabla No 4.1. Porcentajes y horas asignados (as) a cada etapa funcional

Etapas	**Porcentaje**	**Horas**
Analizar	20%	352
Diseñar	20%	352
Implementar	20%	352
Integrar	40%	704
		Total = 1760

Por otra parte, para cada tipo de proyecto se han establecido tiempos constantes para cada una de las áreas de proceso clave. Estos tiempos para cada tipo son

Tabla No 4.2. Tiempo en horas para cada tipo de proyecto en cada KPA.

Area de proceso clave (KPA_i)	**Proyecto Tipo 1** **(tiempo en horas)**	**Proyecto Tipo 2** **(tiempo en horas)**	**Proyecto Tipo 3** **(tiempo en horas)**
Gestión de Requisitos	50	80	120
Planificación del proyecto sw	75	100	200
Seguimiento y Control del proyecto	75	100	200
Gestión de subcontratación de sw	50	80	120
Gestión de configuración de sw	25	40	80
Peer Review	50	80	120

Para obtener el tiempo que un proyecto permanece en una etapa, se debe definir previamente, la notación que se va a utilizar para realizar los cálculos. Cada letra se corresponde con alguna etapa funcional del proceso de ejecución o con el sistema de gestión de calidad, las líneas representan la transición de una etapa a otra y sobre la línea se especifica el área de proceso clave (KPA) bajo la cual se está gestionando el proyecto. Recordemos que anteriormente se especificó con cuáles áreas de proceso clave están involucradas cada etapa funcional.

A : Etapa “Analizar”

Q : Sistema de Gestión de Calidad

KPA_1 : KPA Gestión de Requisitos

KPA_2 : KPA Planificación del Proyecto de Software

$VBKPA_i$: Visto Bueno por parte del QMS del área de proceso clave i; i = 1...2

Figura 4.14. Secuencia para la etapa Analizar.

Como ya se ha dicho, para los proyectos tipo 1, el tiempo en la etapa Analizar corresponde a 352 horas. El gráfico anterior muestra el flujo del proyecto entre esa etapa y el sistema de gestión de calidad, antes de pasar a la siguiente etapa que es "Diseñar". Se debe hacer notar que para determinar el tiempo que el proyecto permanece en la etapa de Analizar se ha asumido que para cada área de proceso clave dentro de esta etapa, el QMS da su visto bueno, sin considerar si es necesario realizar correcciones tanto al informe inicial de requisitos (KPA1), como al plan del proyecto (KPA2).

De acuerdo a los porcentajes, en esta fase los proyectos tipo 1 deben tardar 352 horas. A este número de horas le restamos el tiempo total que el proyecto permanece en el QMS, que para esta etapa es 125 horas, obteniéndose 227 horas. El proyecto pasa dos veces por Analizar para cubrir los dos KPA's involucradas en esta etapa, así que el tiempo promedio en Analizar viene dado por: $(227/2)$ horas, es decir, 113,5 horas.

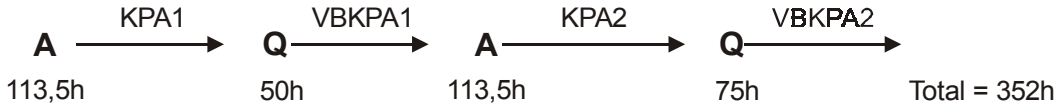


Figura 4.15. Tiempo promedio en la etapa Analizar.

El gráfico anterior (Figura 4.15) muestra que la etapa Analizar tarda 113,5 horas para generar el informe inicial de requisitos, ese informe se pasa al sistema de gestión de calidad, el cual hace una evaluación del informe durante 50 horas, dando como resultado el visto bueno de dicho informe. Seguidamente, la etapa analizar comienza a trabajar en el plan de proyecto en un tiempo de 113,5 horas, este plan se pasa de nuevo al QMS para su respectiva evaluación, dando como resultado el visto bueno del mismo y, de esta manera el proyecto pasa a la siguiente etapa correspondiente a Diseñar.

Los tiempos de cada una de las etapas, se han representado en el lenguaje de simulación GLIDER, a través de distribuciones Gaussianas (distribuciones normal). Los tiempos promedios para estas distribuciones se muestran en la tabla siguiente (Tabla No. 3) y, para

establecer las desviaciones estándares de las mismas, se ha planteado un escenario para el error cometido en los tiempos promedios de las distribuciones.

De forma análoga se han determinado los tiempos que permanecen los proyectos en cada una de las etapas funcionales restantes, obteniéndose los siguientes resultados:

Tabla No 4.3. Tiempo promedio para cada tipo de proyecto en cada etapa funcional

Etapa	**Proyecto Tipo 1** **(tiempo promedio en h)**	**Proyecto Tipo 2** **(tiempo promedio en h)**	**Proyecto Tipo 3** **(tiempo promedio en h)**
Analizar	113,5	262	720
Diseñar	277	604	1560
Implementar	38	101	310
Integrar	126	277	750

El escenario establecido, es para un error total del 15%; por haber cuatro etapas funcionales, se ha asumido un error equivalente para cada etapa de 3,75%. Las desviaciones se han calculado usando un procedimiento basado en las referencias de los libros de estocástica para una distribución normal [24]. Generalmente, el valor promedio μ es un valor conocido, tal como sucede en este trabajo para cada tipo de proyecto y para cada una de las etapas funcionales. Sin embargo, la desviación es un valor que se puede determinar o calcular conociendo el error permitido con respecto al promedio, para los valores generados por la distribución Gaussiana. En una distribución Gaussiana, el 95% de los valores generados están en el intervalo $(\mu - 2\sigma, \mu + 2\sigma)$, donde los valores extremos de ese intervalo dependen del error mencionado anteriormente.

De esta manera, si en la etapa Analizar se desea un error máximo de 3,75% con respecto a su valor promedio, entonces $\mu + 2\sigma = (0{,}0375 * \mu) + \mu \cong 117{,}76$. Por lo tanto, $\sigma \cong 2{,}13$.

En la Figura 4.16, se representa gráficamente lo señalado anteriormente.

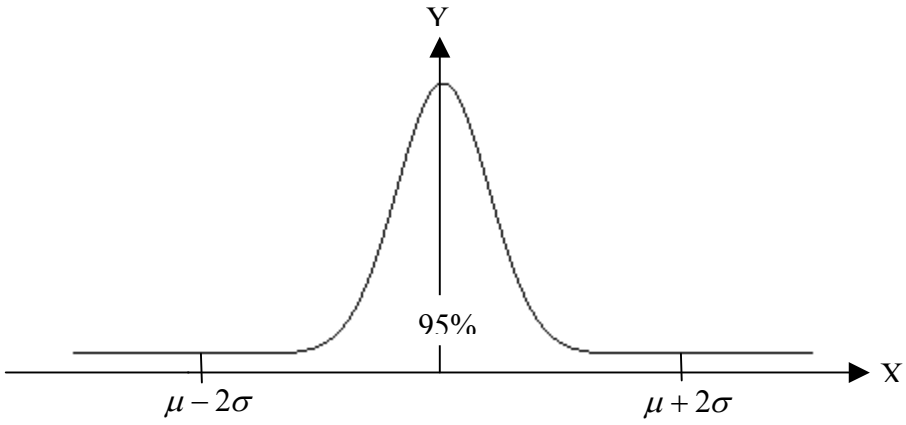


Figura 4.16. Distribución Gaussiana.

A continuación, se muestran en una tabla (Tabla No. 4..4), las desviaciones estándares para cada uno de los tipos de proyectos en cada una de las etapas funcionales.

Tabla No. 4.4. Desviaciones estándar para cada tipo de proyecto en cada etapa funcional

Etapa	**Proyecto Tipo 1** **(desviación estándar)**	**Proyecto Tipo 2** **(desviación estándar)**	**Proyecto Tipo 3** **(desviación estándar)**
Analizar	2,13	4,91	13,5
Diseñar	5,20	11,33	29,25
Implementar	0,72	1,89	5,81
Integrar	2,37	5,20	14,10

A continuación, se muestran en una tabla (Tabla No. 4.5), las desviaciones estándares para cada uno de los tipos de proyectos en cada una de las áreas de proceso clave.

Tabla No.4.5 Desviaciones estándar para cada tipo de proyecto en cada KPA.

Area de proceso clave (KPA_i)	**Proyecto Tipo 1 (desv. estándar)**	**Proyecto Tipo 2 (desv. estándar)**	**Proyecto Tipo 3 (desv. Estándar)**
Gestión de Requisitos	3,75	6,00	9,00
Planificación del proyecto sw	5,63	7,50	15,0
Seguimiento y Control del proyecto	5,63	7,50	15,0
Gestión de subcontratación de sw	3,75	6,00	9,00
Gestión de configuración de sw	1,88	3,00	6,00
Peer Review	3,75	6,00	9,00

4.4 Simulaciones y Estadísticas Obtenidas

Se realizará en GLIDER la simulación del flujo secuencial de un sistema programado desarrollado por la empresa, con la finalidad de obtener resultados que permitan concluir acerca de la incorporación del sistema de gestión de calidad (QMS) en la culminación exitosa del mismo. Para ello, una vez calculados los tiempos promedios y las desviaciones estándares necesarias, se realizaron algunos experimentos o pruebas que permitieron obtener estadísticas acerca del tiempo en el sistema, para cada uno de los proyectos generados. En los primeros experimentos se van a considerar sólo ocho proyectos de entrada, de los cuales cinco son del tipo 1, dos del tipo 2 y uno del tipo 3. El valor "tipo" de cada proyecto generado es obtenido por medio de una función de distribución discreta, que asigna al valor 1 una probabilidad de 0,45, al valor 2 le asigna 0,4 y al valor 3 le asigna 0,15.

Es importante señalar, que cada una las etapas funcionales del proceso de ejecución, así como el sistema de gestión de calidad, presentan una característica común llamada "Recurso". Esta característica representa el servicio que presta cada una de estas etapas, incluyendo al QMS, para el desarrollo de un proyecto dentro de la empresa. Esta característica también puede ser vista como la demanda o el requerimiento que hace el

proyecto en desarrollo a cada una de las etapas funcionales del proceso de ejecución y al QMS y será, una variable clave para los experimentos.

El servicio que presta una etapa puede ser modificado, es decir, si a "Recurso" se le asigna el valor 1, significa que esta etapa puede atender (prestar servicio) a un sólo proyecto a la vez; si a "Recurso" se le asigna el valor 2, significa que esta etapa puede atender a dos proyectos a la vez, y así sucesivamente. De esta manera, la característica o atributo "Recurso" puede ser considerado como la capacidad tanto de las etapas como del QMS.

Experimento 1: Las capacidades de cada etapa funcional y del QMS se han considerado iguales a uno. El escenario de error es de 15% y el tiempo de simulación para cada experimento es de 17600 horas, que equivale a 10 años de actividad por parte de la empresa. Los resultados obtenidos fueron:

Tabla No. 4.6. Tiempos de generación, culminación y en el sistema de los proyectos

Orden de Culminación	Tipo de proyecto en orden de generación	Tiempo de generación (h)	Tiempo de culminación	Tiempo en el sistema (h)	En años
1	Tipo 1	0	2710	2710	1,54
2	Tipo 1	960	3467	2507	1,43
3	Tipo 2	1920	6638	4718	2,68
4	Tipo 1	2880	6735	3855	2,20
6	Tipo 3	3840	15123	11283	6,41
7	Tipo 1	4800	15417	10617	6,03
8	Tipo 2	5760	16109	10349	5,88
5	Tipo 1	6720	14152	7432	4,22

Para la tabla que se muestra a continuación (Tabla No. 7), se tiene la siguiente leyenda:

El almacenamiento de mensajes se hace en dos tipos de listas asociadas al nodo:

- **Lista de Entrada (EL)** ("Entry List"), es donde están los mensajes que esperan ser procesados por el nodo.

- **Lista Interna (IL)** ("Internal List"), es donde están los mensajes que quedan almacenados en el nodo, los cuales pueden ser quitados luego y enviados a cualquier nodo que tenga EL.

 # Entradas: Número de entradas
Tiempo Libre Listas: Tiempo en que estuvo la lista vacía
Lgth: Largo actual de la lista
Max Cola: Largo máximo de la cola
EL: Lista de entrada; proyectos esperando para hacer uso del recurso de una etapa.
IL: Lista interna; corresponde al número de proyectos usando el recurso de una etapa.

Tabla No. 4.7: Número de proyectos que entran en cada etapa, tiempo libre de las listas de entrada (EL) e interna (IL) de las etapas y, longitud de estas listas.

Etapa	# Entradas	Tiempo Libre Listas (TLL)	%TLL	Lgth	Max Cola
Necesidades	8	_	_	_	_
Analizar (EL)	20	16733,1	95,07	0	1
Analizar (IL)	20	13556,5	77,03	0	1
Diseñar (EL)	9	15504,2	88,09	0	3
Diseñar (IL)	9	13117,1	74,53	0	1
Implement(EL)	34	16468,3	93,57	0	3
Implement (IL)	34	14729,7	83,69	0	1
Integrar (EL)	44	10452,8	59,39	0	3
Integrar (IL)	44	6995,52	39,75	0	1
QMS (EL)	107	15845,8	90,03	0	3
QMS (IL)	107	10368,0	58,91	0	1
Mantener (EL)	8	17600	100	0	1

Análisis de Resultados

En este primer experimento se observa que aún cuando la mayor parte del tiempo, las listas de entrada e interna de cada una de las etapas se encuentran vacías (ver los porcentajes %TTL en la tabla 4.7), el tiempo promedio en el sistema de cada uno de los proyectos generados, es mayor que el estipulado al comienzo de la simulación para cada tipo de proyecto (ver Tiempo en el sistema en la tabla 4.6). De esta manera se tiene que los proyectos no logran culminarse en el tiempo previsto, lo cual podría generar un incumplimiento en su entrega.

Se observa también en los datos que, a medida que el tiempo transcurre y se generan proyectos de un mismo tipo, el tiempo promedio en el sistema de estos proyectos aumenta. Por otra parte, como se ha dicho, el tiempo que las listas de entrada e interna permanecen vacías es bastante elevado, sin embargo, el largo máximo de la cola para Diseñar, Implementar, Integrar y el QMS (ver Max Cola en la tabla 4.7), llega a tomar un valor de tres proyectos en espera. Es decir, las demoras de los proyectos en el sistema, se deben a que ciertas actividades son cuellos de botella.

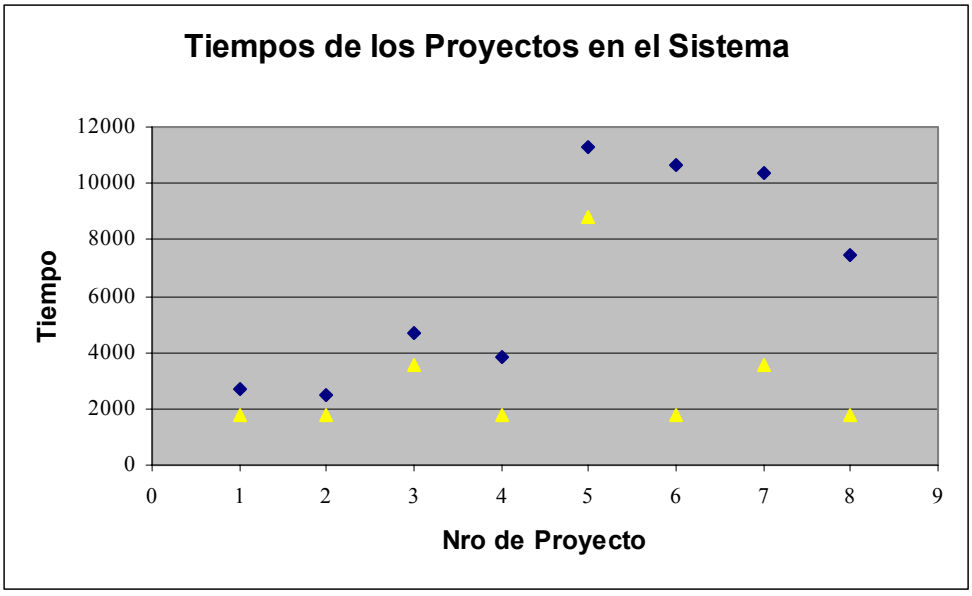


Figura 4.17. Tiempo de los Proyectos en el Sistema

Los rombos representan el tiempo de cada proyecto en el sistema obtenido mediante la simulación y, los triángulos representan los tiempos estipulados para cada proyecto al inicio de la simulación.

Lo expuesto anteriormente nos lleva a concluir que los proyectos tardan en el sistema un tiempo mayor al previsto, porquen tienen que estar cierto tiempo esperando en las listas de entrada de las etapas cuyo valor alcanza un máximo de tres proyectos en espera, para lograr usar el recurso de la etapa respectiva.

Se observa también, que la etapa que permanece mayor tiempo ocupada, es el sistema de gestión de calidad (QMS). Para verificar la conclusión a la cual se ha llegado, se realizó un nuevo experimento en donde se aumentó las capacidades de las etapas Diseñar, Implementar e Integrar, ya que en éstas es donde los proyectos tardan más tiempo esperando ser atendidos.

Es importante hacer notar, que para un tiempo de simulación de 10 años (17600 horas), se logra terminar con los ocho proyectos aceptados por el sistema o empresa.

A continuación, se muestra gráficamente el número de proyectos en cola de las listas de entrada de cada una de las etapas funcionales del proceso de ejecución y del sistema de gestión de calidad (QMS), durante el período de simulación. Estos datos también se pueden observar en la columna Max Cola de la tabla 4.7.

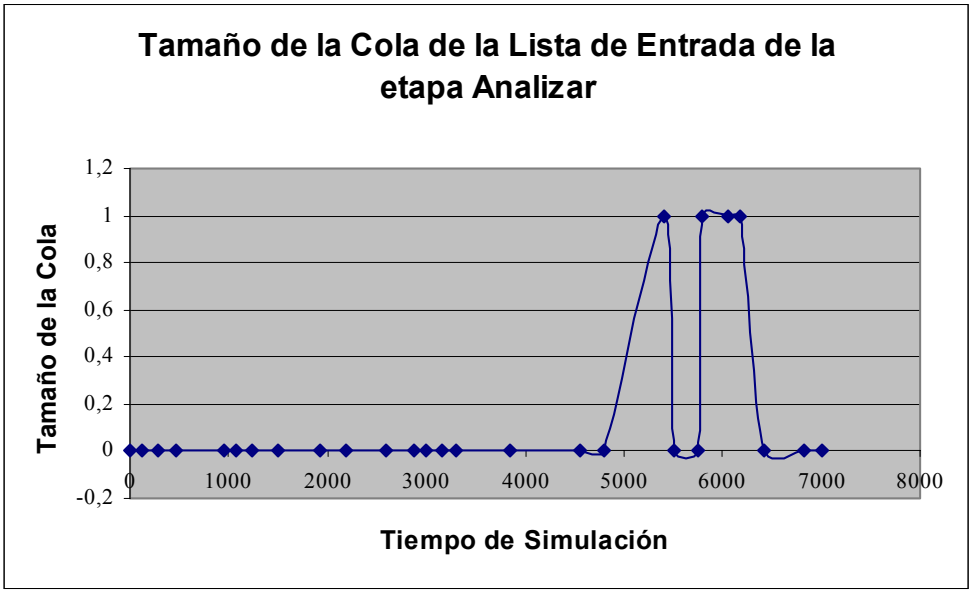


Figura 4.18. Tamaño de la Cola de la Lista de Entrada de la etapa Analizar

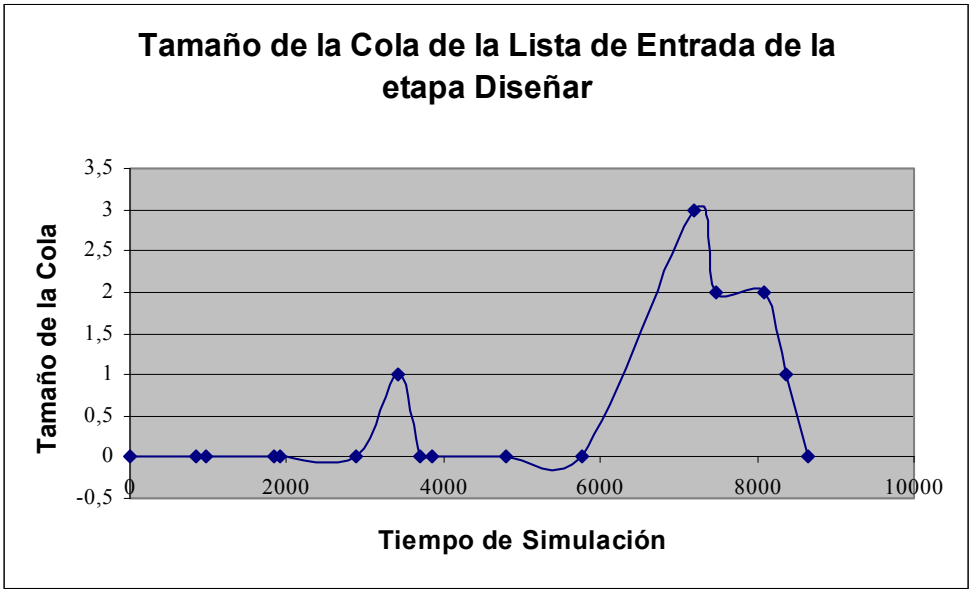


Figura 4.19. Tamaño de la Cola de la Lista de Entrada de la etapa Diseñar

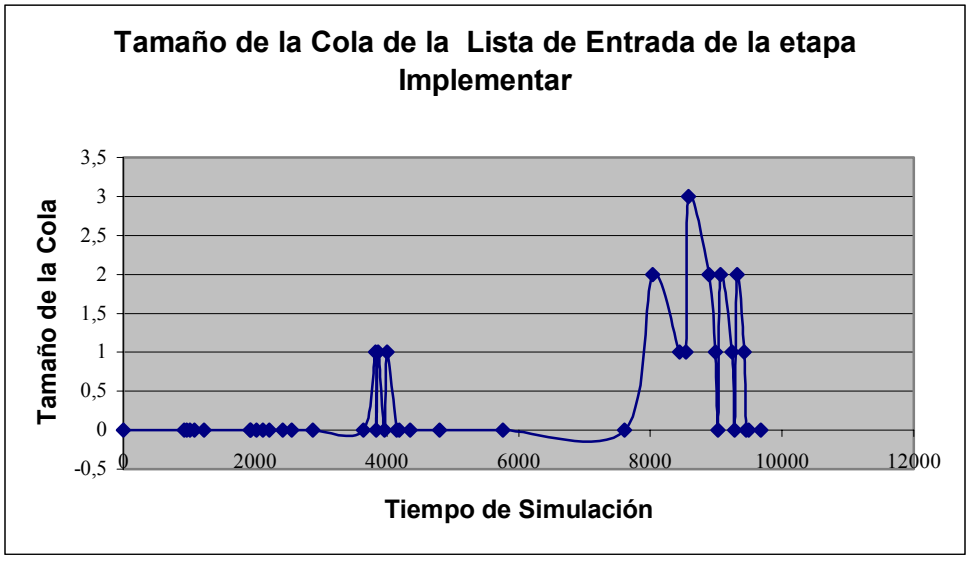


Figura 4.20. Tamaño de la Cola de la Lista de Entrada de Implementar

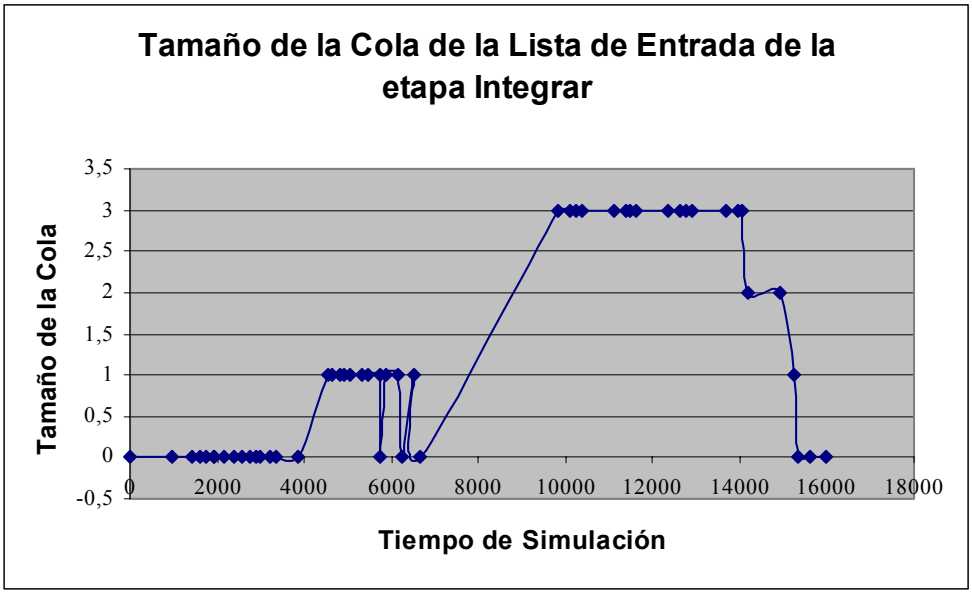


Figura 4.21. Tamaño de la Cola de la Lista de Entrada de la etapa Integrar

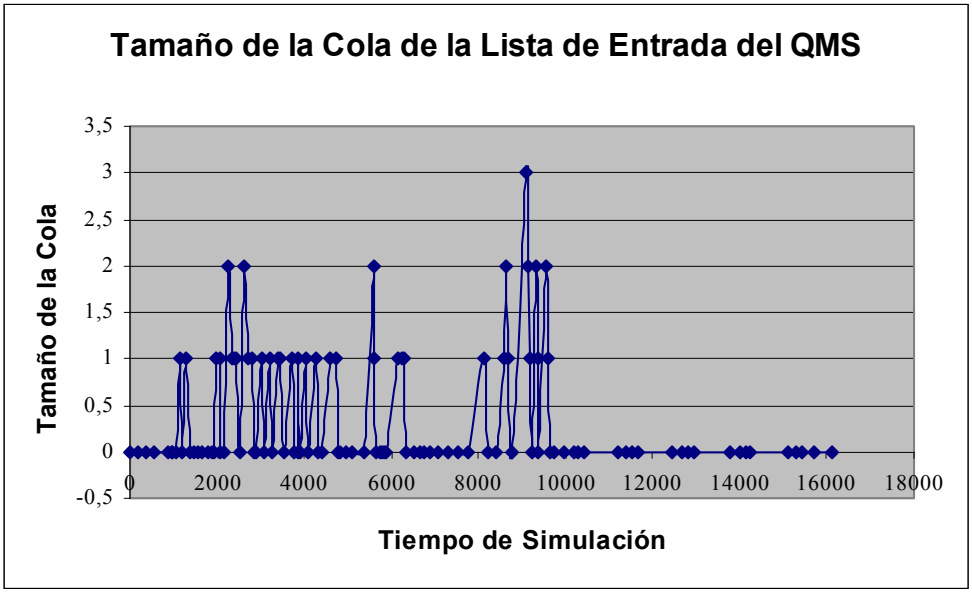


Figura 4.22. Tamaño de la Cola de la Lista de Entrada del Sistema de Gestión de Calidad

Experimento 2: Capacidad de Analizar = 1

Capacidad de Diseñar = 2

Capacidad de Implementar = 2

Capacidad de Integrar = 2

Capacidad del QMS = 1

Los resultados obtenidos se muestran en las Tablas No. 8 y 9. De los datos obtenidos, se puede observar que el número máximo en cola de la lista de entrada de la etapa Analizar ha aumentado, aún cuando da tiempo de que la etapa Analizar atienda a un proyecto de cualquier tipo antes de ser generado el siguiente para ser atendido por esta etapa. Esto se debe a que los proyectos son generados cada 960 horas y el proyecto tipo 3, que es el que más tarda siendo atendido por la etapa Analizar, dura consumiendo sus recursos 720 horas. Sin embrago, esto es posible porque los proyectos generados pueden ser cíclicos en cualquiera de las áreas de proceso clave. Los proyectos se dicen cíclicos en una KPA, en caso de que se requieran correcciones referentes a esa KPA y tengan que pasar de nuevo por alguna etapa específica.

Tabla No. 4.8. Tiempos de generación, culminación y en el sistema de los proyectos

Orden de Culminación	**Tipo de proyecto en orden de generación**	**Tiempo de generación (h)**	**Tiempo de culminación**	**Tiempo en el sistema (h)**	**En años**
1	Tipo 1	0	2710	2710	1,54
2	Tipo 1	960	3467	2507	1,43
4	Tipo 2	1920	5789	3869	2,20
3	Tipo 1	2880	5014	2134	1,21
8	Tipo 3	3840	16606	12766	7,25
5	Tipo 1	4800	9442	4642	2,64
7	Tipo 2	5760	11810	6050	3,44
6	Tipo 1	6720	11347	4627	2,63

Tabla No. 4.9. Número de proyectos que entran en cada etapa, tiempo libre de las listas de entrada (EL) e interna (IL) de las etapas y tamaño de estas listas.

Etapa	# Entradas	Tiempo Libre Listas (TLL)	%TLL	Lgth	Max Cola
Necesidades	8	_	_	_	_
Analizar (EL)	24	14158,7	80,45	0	3
Analizar (IL)	24	11284,0	64,11	0	1
Diseñar (EL)	8	17600,0	100	0	1
Diseñar (IL)	8	13777,8	78,28	0	2
Implement(EL)	39	17600,0	100	0	1
Implement (IL)	39	14115,2	80,20	0	2
Integrar (EL)	37	17600,0	100	0	1
Integrar (IL)	37	9911,00	56,31	0	2
QMS (EL)	108	15068,3	85,62	0	3
QMS (IL)	108	9638,58	54,76	0	1
Mantener (EL)	8	17600,0	100	0	1

Análisis de Resultados

Para las etapas del proceso de ejecución (sin incluir al QMS), los tiempos en que sus listas de entrada e interna permanecen vacías han aumentado un poco, excepto para la etapa Analizar. Este resultado se debe a que al aumentar la capacidad de las etapas indicadas, se tienen menos proyectos esperando en las listas de entrada y como se pueden atender más proyectos a la vez, es mayor el número de proyectos que van a salir de estas etapas, quedando éstas mayor tiempo libres.

Sin embargo, el sistema de gestión de calidad (QMS), permanece más tiempo activo; quizás la razón de esto es porque está saliendo mayor número de proyectos de las etapas Diseñar, Implementar e Integrar al QMS, por tener las anteriores una capacidad mayor. En las etapas donde se aumentó la capacidad, la longitud máxima de la cola en la lista de

entrada disminuyó, es decir, que ahora los proyectos permanecen menos tiempo esperando para lograr usar el "Recurso" de estas etapas. La longitud máxima de la cola en la lista interna, llega a alcanzar el valor de la capacidad asignada, significando esto que estas etapas pueden atender a más de un proyecto a la vez.

Otro aspecto importante que podemos notar en este segundo experimento es que los tiempos en el sistema de cada uno de los proyectos han disminuido, acercándose un poco más a los tiempos establecidos para cada uno de los tipos de proyectos. Comparando los tiempos de los proyectos en el sistema entre el experimento 1 y el experimento 2, se tiene que de los ocho proyectos generados, siete logran disminuir sus tiempos en el sistema en el experimento 2, a excepción de un proyecto. Sin embargo, la diferencia de tiempo para este proyecto no es significativa. Se hace referencia a este aspecto para aclarar que esto puede pasar y en todo caso depende de las áreas de proceso clave donde el proyecto sea cíclico, ya que mientras mayor sea el número de KPA's donde el proyecto sea cíclico, mayor tiempo permanece en el sistema.

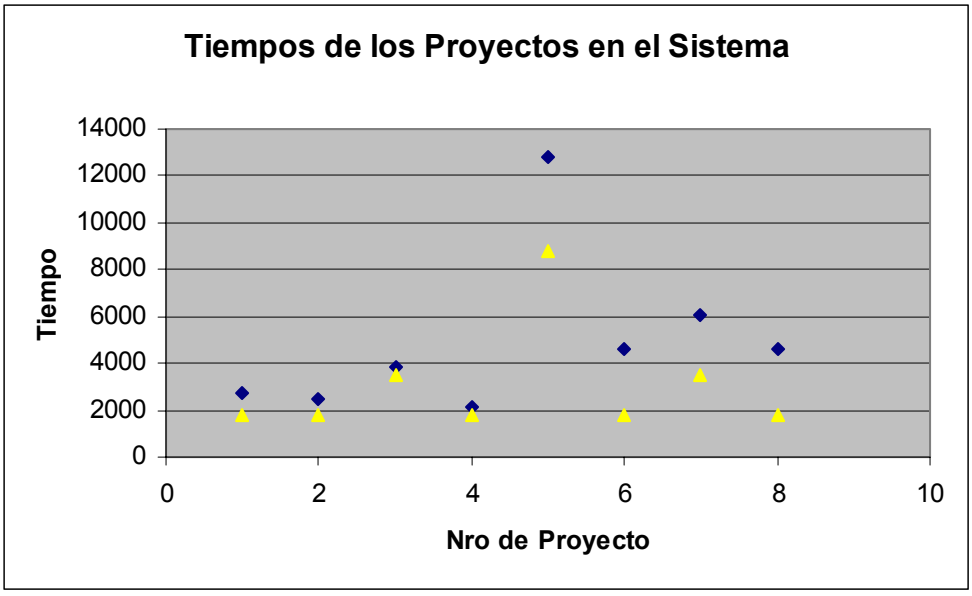


Figura 4.23. Tiempos de los Proyectos en el Sistema

Los rombos representan el tiempo de cada proyecto en el sistema obtenido mediante la simulación y, los triángulos representan los tiempos estipulados para cada proyecto al inicio de la simulación.

Para concluir con respecto a los dos experimentos anteriores, se observa que la mayor parte del tiempo, las etapas del proceso de ejecución presentan sus listas de entrada e interna vacías, a excepción del QMS. Aún cuando la mayor parte del tiempo no hay cola en las etapas funcionales, éstas podrían estar sirviendo a algún proyecto. Esta medida del tiempo en que las listas de las etapas permanecen vacías, es con respecto al tiempo total de simulación y probablemente se debe a que es muy pequeño el número de proyectos generados.

Experimento 3: Capacidad de Analizar = 2
Capacidad de Diseñar = 2
Capacidad de Implementar = 2
Capacidad de Integrar = 2
Capacidad del QMS = 1
Se generan 8 proyectos

En los resultados obtenidos en este experimento, se observa que los tiempos en el sistema disminuyen y que las listas de entrada en interna de las etapas permanecen mayor tiempo vacías, sin embargo, como ya se ha dicho, las etapas pueden estar sirviendo a algún proyecto. Este resultado se podía esperar ya que las capacidades de las etapas fueron aumentadas.

Experimento 4: Capacidad de Analizar = 1
Capacidad de Diseñar = 1
Capacidad de Implementar = 1
Capacidad de Integrar = 1
Capacidad del QMS = 2
Se generan los mismos ocho proyectos

Al comparar los resultados obtenidos en este experimento con los resultados obtenidos en el experimento1, donde la diferencia entre ambos es la capacidad del QMS, se observa que la mayoría de los proyectos disminuyeron su tiempo de permanencia en el sistema; sólo

dos aumentaron este tiempo. Sin embargo, como ya se ha dicho, esto puede pasar porque el tiempo de un proyecto en el sistema depende de si es cíclico o no en alguna de las KPA's.

Tomando en cuenta sólo los proyectos que disminuyen su tiempo en el sistema en el experimento 4, el porcentaje promedio de tiempo que disminuye es aproximadamente igual a 10,6%. Este valor se obtuvo calculando para cada uno de esos proyectos el porcentaje de tiempo que disminuyó y luego promediando los valores obtenidos. Podemos decir entonces, que con un pequeño aumento en la capacidad del QMS, los tiempos en el sistema de la mayoría de los proyectos disminuyen significativamente en alguno de los proyectos. A partir de estos resultados se puede decir que es importante la dedicación por parte del sistema de gestión de calidad para el desarrollo calificado del sistema programado, es decir, se justifica el sistema de gestión de calidad para producir software de calidad.

Los tiempos que las listas de entrada e interna permanecen vacías para cada etapa, presentan cambios insignificativos entre éste experimento y el experimento 1. Además, la longitud máxima de la cola para la lista de entrada en el QMS, ha disminuido a dos en el experimento 4, por lo que ahora los proyectos tardan menos tiempo esperando para ser atendidos por el QMS. Sin embargo, para las demás etapas, la longitud máxima de la cola se mantiene aproximadamente igual y es por eso que los tiempos en el sistema aún cuando han disminuido, son mayores que los estipulados al comienzo de la simulación. Y al igual que en el experimento 1, a medida que transcurre el tiempo de simulación, aumenta el tiempo en el sistema de los proyectos de un mismo tipo.

Experimento 5: Capacidad de Analizar = 1
Capacidad de Diseñar = 1
Capacidad de Implementar = 1
Capacidad de Integrar = 1
Capacidad del QMS = 3

Los resultados obtenidos en este experimento no arrojan cambios significativos en comparación con el inmediato anterior, respecto a los tiempos en el sistema de los

proyectos y a los tiempos en que las listas de entrada e interna de cada etapa permanecen vacías. Los tiempos en el sistema para los proyectos siguen siendo mayores que los establecidos al principio de la simulación. Se observa que esos tiempos ya no dependen de la capacidad del QMS sino de las capacidades de las demás etapas funcionales, ya que en ellas se presenta una longitud máxima en la cola de la lista de entrada igual a tres, lo que significa que los proyectos permanecen cierto tiempo en espera para ser atendidos por estas etapas.

Experimento 6: Capacidad de Analizar = 1
Capacidad de Diseñar = 1
Capacidad de Implementar = 1
Capacidad de Integrar = 1
Capacidad del QMS = 4

Los resultados obtenidos en este experimento han sido exactamente iguales a los del experimento inmediato anterior. De nuevo, los tiempos mencionados para el experimento 5 ya no dependen de la capacidad del QMS.

Experimento 7: Capacidad de Analizar = 2
Capacidad de Diseñar = 2
Capacidad de Implementar = 2
Capacidad de Integrar = 2
Capacidad del QMS = 2

En términos generales se observa por medio de los resultados obtenidos, que en este último experimento los tiempos en el sistema de los proyectos han disminuido, los tiempos en que las listas permanecen vacías han aumentado (al aumentar las capacidades de las etapas). Además, por los valores obtenidos en la longitud máxima de la cola en la lista de entrada, los proyectos permanecen menor tiempo esperando para ser atendidos por cada etapa.

Experimento 8: Capacidad de Analizar = 2
Capacidad de Diseñar = 2
Capacidad de Implementar = 2
Capacidad de Integrar = 2
Capacidad del QMS = 3

No se observan cambios significativos con respecto al anterior. Con respecto a todos los experimentos realizados, se nota una característica común y es que las listas, tanto de entrada como interna de cada una de las etapas, permanecen la mayor parte del tiempo vacías. Probablemente se debe a que el número de proyectos generados es pequeño para los 10 años promedio de servicio de la empresa. Es por ello, que en el siguiente experimento 9, se aumenta el número de proyectos generados y se hace la simulación dejando las capacidades de las etapas funcionales y del QMS iguales a uno. Bajo estas condiciones se observa si el tiempo en que las etapas permanecen inactivas disminuye.

Experimento 9: Capacidad de Analizar = 1
Capacidad de Diseñar = 1
Capacidad de Implementar = 1
Capacidad de Integrar = 1
Capacidad del QMS = 1
Se generan 10 proyectos: 6 tipo 1; 2 tipo 2; 2 tipo 3.
Se ha dejado el tiempo de simulación igual.

En este experimento se obtiene que tres de los proyectos generados no logran ser desarrollados por completo en la empresa, es decir, no logran culminar. Al igual que en los experimentos anteriores, los tiempos en el sistema aumentan a medida que transcurre el tiempo y además, están por encima de los estipulados para cada tipo de proyecto al inicio de la simulación; esto se debe a que la longitud máxima de la cola de la lista de entrada de las etapas ha aumentado con respecto a los experimentos anteriores.

Se desea ver cómo la capacidad del QMS afecta al número de proyectos que logra culminar el sistema; para ello se realiza un nuevo experimento.

Experimento 10: Capacidad de Analizar = 1
Capacidad de Diseñar = 1
Capacidad de Implementar = 1
Capacidad de Integrar = 1
Capacidad del QMS = 2
Proyectos generados por medio de una distribución Exponencial con μ = 960 horas
Se ha dejado el tiempo de simulación igual.

De los resultados obtenidos se observa que al aumentar la capacidad del QMS, se logra terminar 9 proyectos (2 más que en el experimento anterior). Los tiempos en el sistema disminuyen, aún cuando las capacidades de las demás etapas han permanecido iguales y en las etapas Diseñar, Implementar e Integrar, los proyectos permanecen cierto tiempo esperando para usar el recurso.

Se hizo un nuevo experimento donde la capacidad del QMS es igual a 3, las capacidades de las demás etapas fueron iguales a uno y se generaron los mismos 10 proyectos; los resultados obtenidos fueron exactamente iguales a los obtenidos en el experimento 10, esto se debe a que en el experimento anterior los proyectos permanecen más tiempo esperando en las listas de entrada de las etapas Diseñar, Implementar e Integrar que en la del QMS. Por ello, para tratar de disminuir los tiempos en el sistema de los proyectos, se pueden realizan nuevos experimentos cambiando las capacidades de las etapas del proceso de ejecución.

Experimento 11: Capacidad de Analizar = 1
Capacidad de Diseñar = 1
Capacidad de Implementar = 1
Capacidad de Integrar = 1

Capacidad del QMS = 1
Se generan 10 proyectos: 6 tipo 1; 2 tipo 2; 2 tipo 3.
Se ha dejado el tiempo de simulación igual.

En este último experimento, las listas de entrada e interna permanecen menos tiempo vacías y aumentan sus longitudes máxima en la cola. En términos generales, y para concluir con las pruebas o experimentos de esta simulación, se podría plantear un escenario en donde se establezca desde el principio una meta para la empresa acerca del número de proyectos que se deseen culminar en un tiempo promedio de 10 años (puede ser cualquier otro tiempo de simulación). Luego, se pueden establecer una serie de cambios que permitan disminuir el tiempo de los proyectos en el sistema, por ejemplo, modificar las capacidades de las etapas del proceso de ejecución y del QMS e incluso, se podrían cambiar los tiempos en cada una de las etapas y así verificar cuánto tiempo en promedio debe estar el proyecto en cada una de ellas con las capacidades deseadas, para el sistema ser capaz de culminar los proyectos en el tiempo previsto. Además, podría obtenerse resultados importantes sobre el impacto del sistema de gestión de calidad en el desarrollo del sistema programado.

Concluyendo acerca de los resultados obtenidos por medio de los experimentos realizados, se tiene que evidentemente vale la pena aumentar las capacidades tanto de las etapas funcionales como del QMS, ya que esto permite por una parte que los tiempos de los proyectos en el sistema disminuyan y espec;ificamente, en el caso del QMS, se logra que culminen exitosamente mayor número de proyectos o sistemas programados. Por otra parte, si el proyecto entra en un ciclo por alguna de la KPA’s, se emplea mayor tiempo en la ejecución del mismo pero se garantiza por medio del QMS su calidad y efectividad.

4.5 Restricciones del Modelo Propuesto

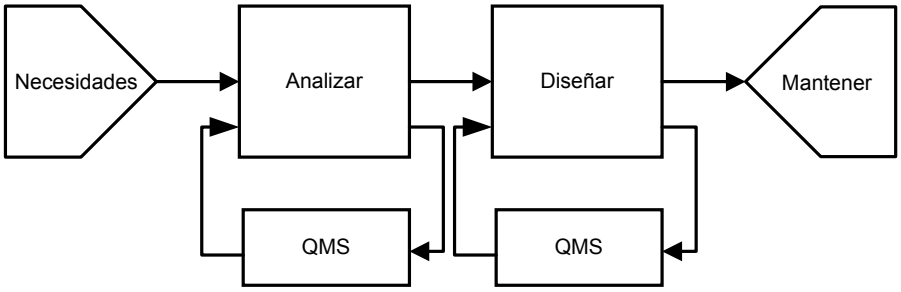
En el modelo Glider que se ha presentado, los proyectos en desarrollo consumen un cierto recurso en cada etapa funcional del proceso de ejecución y en el sistema de gestión de calidad (QMS). En las simulaciones que se han realizado, el número de proyectos consumiendo el recurso de cierta etapa o del QMS, se ha representado mediante la

capacidad de dicha etapa. Es decir, capacidad igual a uno en cualquiera de las etapas, significa que puede existir un único proyecto gestionándose en esa etapa y además, se está asumiendo que existe una persona dedicada ocho horas al día realizando las actividades correspondientes a la gestión de ese proyecto.

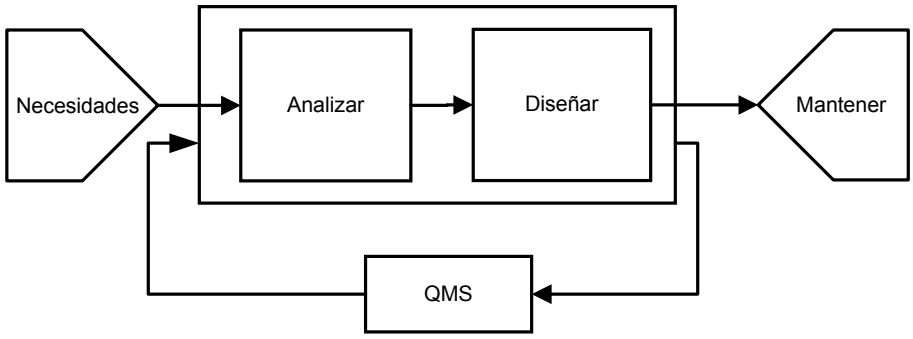
A través de los experimentos realizados bajo este modelo Glider, se observa que la empresa resulta un poco costosa en función del personal dedicado, ya que para desarrollar un mayor número de proyectos, es necesario aumentar el personal. Además de costosa, se observa que la empresa estaría siendo subutilizada, en virtud de que las etapas funcionales permanecen operativas por pequeños períodos, es decir, que en los resultados se observa que el porcentaje que permanecen libres las listas internas de las etapas funcionales (proyectos sin consumir el recurso) es bastante elevado.

Ya que el modelo propuesto permite que la empresa resulte costosa y subutilizada, se propone un nuevo modelo que permita la integración de las actividades realizadas en las etapas con alto porcentaje de tiempo libre en sus listas internas, en una sola etapa en donde sean atendidos estos proyectos por un número específico de personas. De esta manera sería más fácil poder determinar el número de personas necesarias en la empresa.

Lo expuesto anteriormente, se muestra gráficamente, suponiendo que las etapas Analizar y Diseñar permanecen la mayor parte del tiempo inactivas y por lo tanto, son agrupadas en una sola etapa que realiza las actividades de ambas y que atiende sus proyectos.



Al agrupar las etapas Analizar y Diseñar, se tiene:


Necesidades
Analizar
Diseñar
Mantener
QMS

Capítulo 5

Modelado y Simulación de un Centro de Entrenamiento

Un centro de entrenamiento es una organización de servicio, cuyo objetivo principal es la formación de personas en áreas o temas específicos. En el modelo que se presenta en este trabajo, se ha incorporado al centro de entrenamiento, el componente de calidad (sistema formado por un conjunto de personas que forma parte de la organización y que garantiza la calidad de los productos generados o de los servicios públicos proporcionados); para ello se ha tomado como referencia un documento británico [4] que habla sobre los centros de excelencia: qué hacen, por qué es importante tener un centro de excelencia y cómo establecer un centro de excelencia. Este documento británico presenta el modelo de referencia del sistema británico para calidad en organizaciones de servicio público.

El modelo que se propone para un centro de entrenamiento, no se presenta en notación IDEF0, sino que se resumen de una manera sencilla, las funciones o actividades principales llevadas a cabo en el centro y las relaciones entre cada una de ellas.

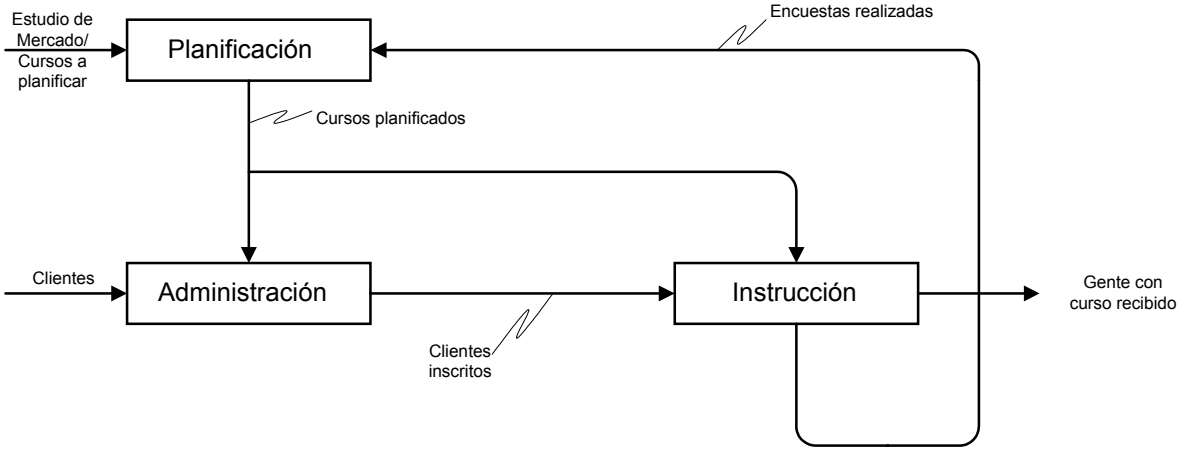


Figura 5.1. Un modelo simple de un centro de entrenamiento

Como podemos observar en el modelo, Planificación tiene como entrada un estudio de mercado que le proporciona información acerca de los cursos que actualmente tienen mayor demanda y, de esta manera decidir qué cursos dictar. Una vez que Planificación ha decidido qué cursos se van a dictar, procede a realizar la planificación de los mismos.

La planificación de los cursos a dictar es enviada tanto a Administración como a Instrucción. Administración inicia con sus actividades, correspondientes a la realización de los cobros respectivos por la apertura de inscripciones y también, a archivar los expedientes de los cursantes inscritos. Por su parte, Instrucción comienza con la preparación del curso, el dictado posterior dictado del mismo y al finalizar el curso realiza una encuesta a los cursantes; las encuestas realizadas son enviadas a Planificación para su evaluación y posteriormente, generar los resultados de dicha evaluación.

También podemos observar en el gráfico que los clientes llegan a Administración cuando se abren las inscripciones. Posteriormente, se especifican en la sección 5.2.1, cuáles son las actividades realizadas por los clientes.

5.1 ¿Qué es un Centro de Excelencia?

Un centro de excelencia proporciona junto a un conjunto de funciones esenciales, soporte para la culminación exitosa de programas y proyectos (servicios). Entre sus funciones, se tienen [4]:

- Reporte ascendente y coherente a la dirección, sobre sus programas y proyectos claves para soportar decisiones efectivas.

- Compartimiento oportuno de la información y lecciones aprendidas a través de relaciones externas con otras oficinas de gobierno.

- Soporte interno para ayudar a la culminación de programas y proyectos cuando el departamento lo necesite.

El propósito de un centro de excelencia (CDE por sus siglas en español) es proporcionar una revisión continua a través de todos los programas y proyectos del departamento; no sólo coordinando y reportando sobre los programas, sino cambiando lo que debe ser culminado o entregado y cómo debe ser culminado. La tarea del centro de excelencia es ayudar al departamento a alcanzar un mayor control sobre sus programas y proyectos día a día, mejorando continuamente su proyecto entregado.

Los resultados típicos esperados en la organización, donde un CDE efectivo ha sido establecido, son:

- Programas y proyectos enfocados sobre objetivos de culminación.
- La dirección está mejor informada del progreso y problemas potenciales.
- Las tasas de programas y proyectos exitosos son mejoradas.
- El plan de actividades de los programas y proyectos es continuamente prioritizado para alcanzar las necesidades del departamento.
- Las lecciones aprendidas pueden ser capturadas y explotadas.
- Las habilidades o destrezas básicas actuales del departamento pueden ser activamente gestionadas o administradas y más efectivamente empleadas. Esto se da porque el CDE verifica continuamente que las personas que están realizando ciertas actividades sean las más apropiadas y si las funciones que realizan son las más efectivas para la culminación exitosa de los proyectos.
- Las habilidades o destrezas requeridas actualmente y en el futuro pueden ser introducidas dentro de los planes de trabajo a través de una formación adecuada.

- El departamento desarrolla su propia habilidad para correr “Peer Review” de programas y proyectos y además, está mejor preparado para revisiones de interfaz. Recordemos que Peer Review es una actividad, cuyo propósito es remover los defectos de los programas o proyectos eficientemente y en una fase temprana de su desarrollo.

La Figura 5.2, muestra la estructura de un Centro de Excelencia.

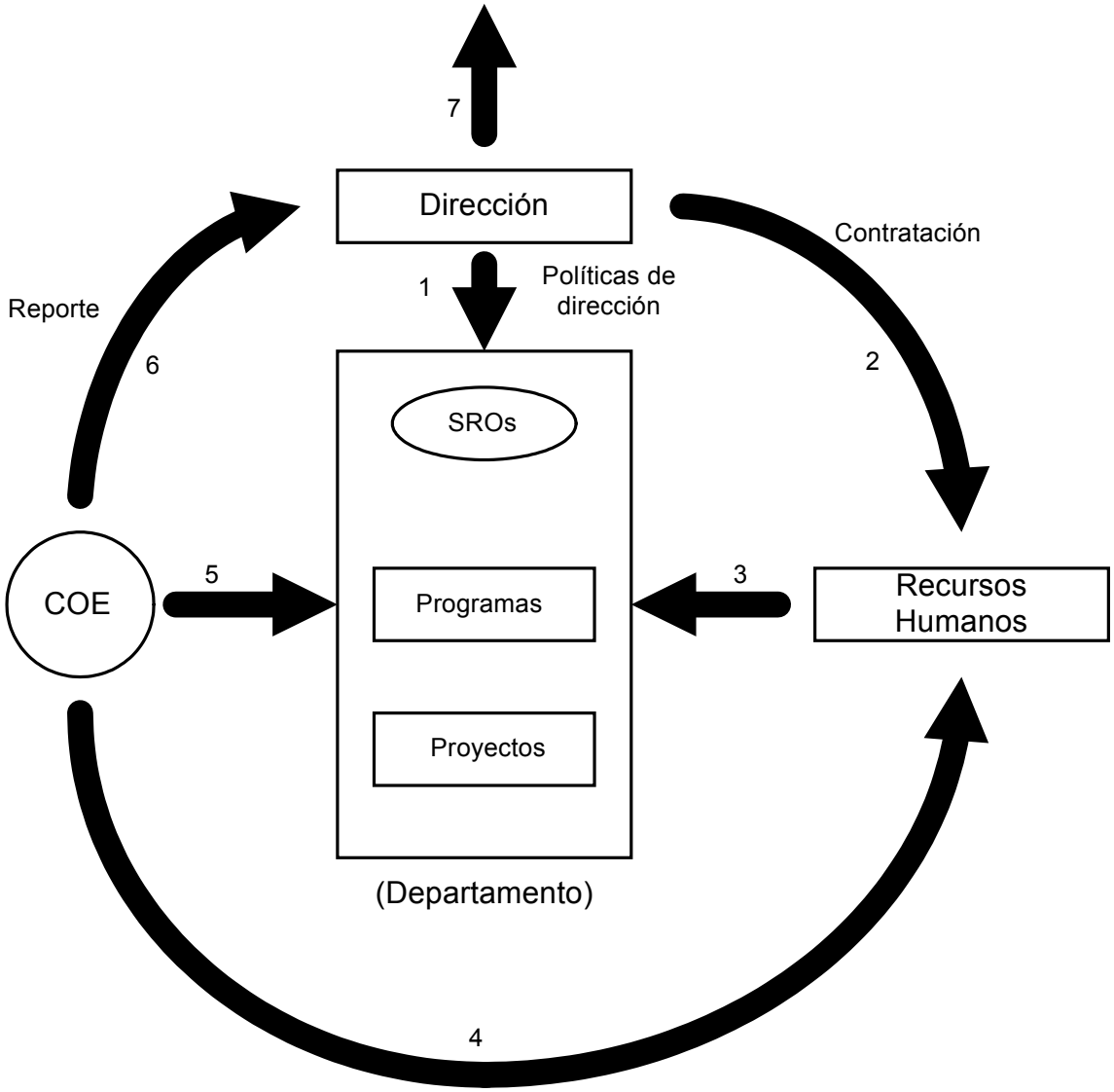


Figura 5.2. Estructura de un Centro de Excelencia (CDE)

En el gráfico anterior se tiene que Dirección transmite por medio del Responsable Principal del Proyecto (SRO), los estándares o las políticas para el desarrollo de los proyectos y establece los puntos de decisión claves, es decir, puntos críticos, que permitan verificar que el progreso del proyecto es como se esperaba, que el proyecto es aún necesario y que los datos de costos, riesgos y beneficios son como se esperaban (1). Por otra parte, Dirección determina las personas, recursos físicos (infraestructura, tecnología) y los fondos necesarios para los cambios requeridos por el centro durante su tiempo de vida en el desarrollo de un programa o proyecto (2). Las estimaciones de estos recursos pueden ser hechas en base a lo que otros han hecho, siempre que sea posible; buscando relaciones o dependencias de los programas proyectos actuales, que permitan llegar a establecer la asignación óptima de los recursos. También determina el origen de los recursos (el departamento, los socios, etc), confirma que el presupuesto esté disponible y asigna roles específicos,

Los recursos asignados son utilizados para el desarrollo de los programas y proyectos; a las personas se les han asignado roles específicos y actividades acordes con los objetivos de trabajo (3).

El CDE verifica que las personas realizando ciertas actividades sean las más apropiadas y que las funciones que realizan sean las más efectivas para la culminación exitosa de los programas y proyectos. Además, debe verificar que se esté dando el entrenamiento y la formación apropiadas, de acuerdo a las destrezas y habilidades requeridas para el desarrollo de los programas y proyectos (4). Por otra parte, el CDE verifica constantemente que se esté cumpliendo el plan de actividades para la culminación exitosa de los programas (5). Además, proporciona un reporte (plan de actividades) coherente a la Dirección sobre sus programas y proyectos claves, de tal manera que ésta pueda soportar decisiones efectivas y asegurar que el departamento tiene un claro entendimiento de los éxitos alcanzables por medio de la culminación de sus proyectos (6).

Finalmente, Dirección mantiene relaciones con entidades externas y garantiza el compartimiento oportuno de información acerca de las mejores prácticas (mecanismos que

permitan mejorar el desempeño de una actividad) y lecciones aprendidas a través de estas relaciones (7).

Cualquiera que sea la estructura interna del centro de excelencia (CDE), un CDE efectivo asegurará que los compromisos u obligaciones requeridas por los programas y proyectos, estén alineados con las capacidades y habilidades actuales para la culminación de los mismos.

A continuación, se especifica el modelado y simulación de un centro de entrenamiento basado en sistemas multiagente.

5.2 Modelado y Simulación de un Centro de Entrenamiento basado en Sistemas Multiagente

Un centro de entrenamiento proporciona formación a las personas con el propósito de que éstas puedan adquirir conocimientos o mejorar los que se tienen en ciertos tópicos de su interés. El modelo propuesto de un centro de entrenamiento (ver Figura 5.1), consta de los siguiente elementos o etapas: Planificación, Administración e Instrucción. Sin embargo, en ese modelo no se ha incorporado aún la gestión de calidad; para ello, se presenta a continuación un diagrama en donde se puede observar la incorporación del centro de excelencia al modelo propuesto (ver figura 5.3).

Las relaciones 1 – 6, que se muestran en la figura 5.3 a través de las flechas, se han explicado en la figura 5.2. La flecha designada por el número 7, indica que Planificación envía al Director del Proyecto los resultados de las encuestas realizadas a los participantes del curso. Estas encuestas sirven para evaluar el curso. El Director del Proyecto hace un análisis o estudio de esos resultados y genera soluciones posibles a los problemas encontrados y envía dichas soluciones al departamento entero, es decir, a Planificación, Administración e Instrucción (8).

Por medio de la interacción y cooperación de los principales procesos involucrados en el centro de entrenamiento, se logra el dictado de los cursos (10) y finalmente, el Director del Proyecto le envía al Líder del CDE un reporte de las diferentes actividades realizadas y de la evaluación de las medidas correctivas requeridas (9).

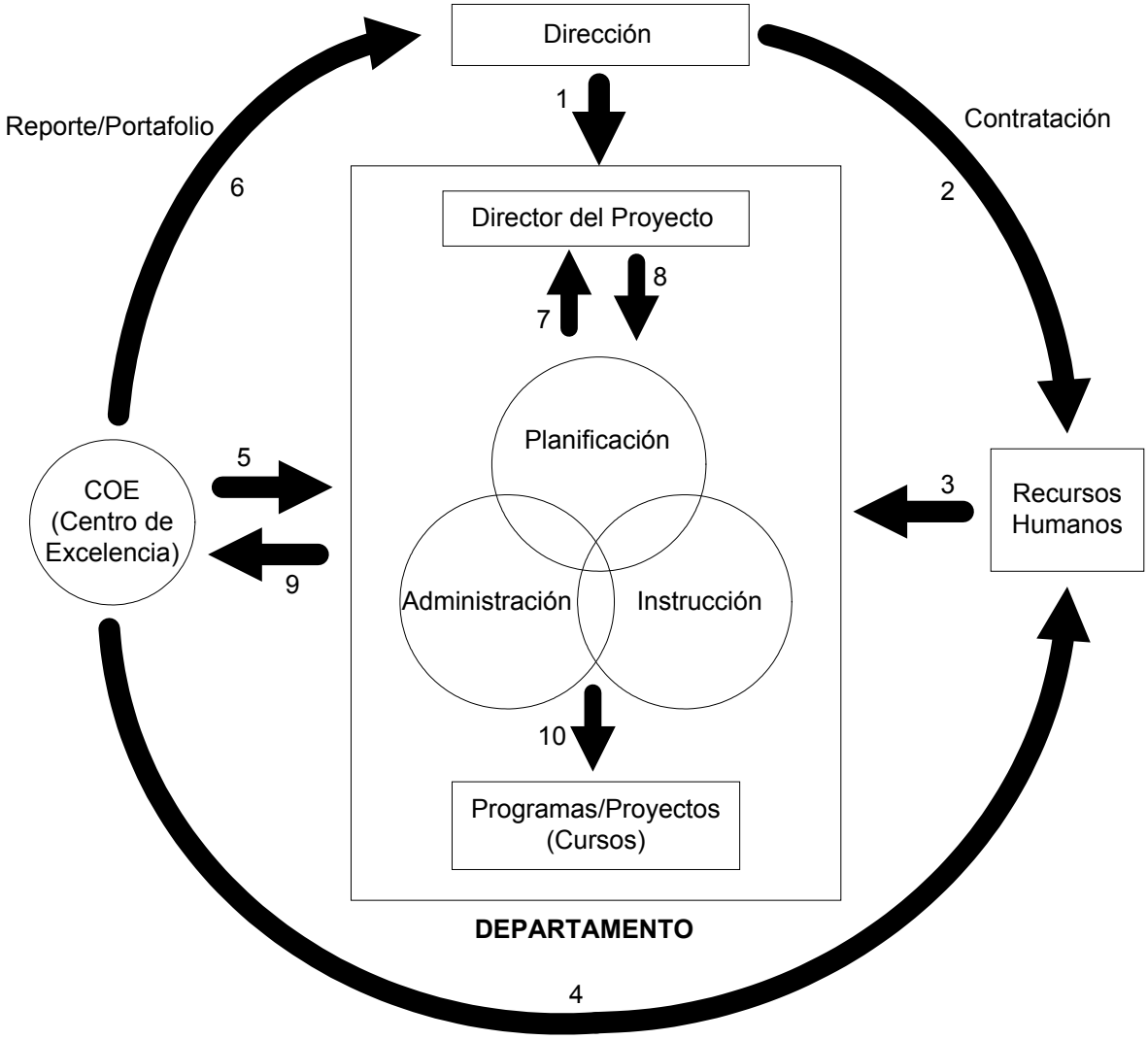


Figura 5.3. Estructura del CDE conjuntamente con el centro de entrenamiento

5.2.1 Análisis Funcional de los Agentes Involucrados en el Sistema Multiagente

El análisis funcional determina las responsabilidades de cada uno de los actores o agentes involucrados en el modelo simple de un centro de entrenamiento.

Actores del Centro de Entrenamiento

Planificación: Sus actividades son:

- Planificar cursos
- Corregir cursos (por medio de los resultados de las encuestas)
- Distribuir recursos
- Evaluar cursos
- Corregir calidad (aplicar medidas correctivas)

Administración: Sus actividades son:

- Hacer cobros
- Realizar inscripciones
- Archivar expedientes

Instrucción: Sus actividades son:

- Preparar curso
- Dictar curso
- Evaluar el curso (realizar encuestas)

Cliente: Sus actividades son:

- Inscribirse en el curso de su interés

- Participar en el curso
- Formar parte de la evaluación del curso (responder encuestas)
- Realizar pagos

Actores del Centro de Excelencia:

Reporte a la Dirección: Consiste en generar un listado de las actividades relacionadas con los programas y proyectos, de acuerdo a estándares definidos por la Dirección. Este listado requiere de:

- Establecer prioridades para los programas o proyectos.

- Establecer un lineamiento entre los compromisos de los programas y las capacidades y habilidades disponibles,

- Plasmar cómo es el desarrollo y la planificación de todos los programas (estimaciones de tiempo, costo total, recursos, entre otros).

- Tener enlaces explícitos de resultados estratégicos a proyectos individuales.

- Reflejar el costo del desarrollo del proyecto completo, incluyendo el costo de cambios en caso de ser requeridos.

- Proporcionar información clave de los proyectos, tales como: progreso alcanzado, principales resultados y riesgos, posibles problemas y planes para resolverlos y, acciones intermedias involucradas para el desarrollo de los proyectos.

- Evaluar el impacto de nuevas inversiones; cuando nuevos proyectos o programas son requeridos, su impacto sobre el plan actual de actividades necesita ser considerado. Algunos programas y proyectos ya iniciados pueden ser paralizados

o reducir su velocidad para tomar las capacidades y habilidades disponibles para el desarrollo de la nueva iniciativa.

- Asimilar el cambio de la relevancia y alcanzabilidad o logro de los proyectos actuales, a través de su ciclo de vida.

- Establecer el origen de los recursos y sus costos.

- Asignar roles claves para el desarrollo de cada curso.

Control del Desarrollo del CDE: Sus actividades son:

- Establecer qué podría afectar la culminación del proyecto, con variables medibles

- Establecer si es posible simplificar el proyecto, dividiéndolo en componentes manejables.

- Usar listas de chequeo para verificar si existen "causas comunes de fallas".

- Evaluar continuamente el progreso del proyecto, estableciendo puntos de revisión intermedios en su ciclo de actividades, para verificar que el desarrollo del proyecto y los datos de costos, beneficio y riesgo, son como se esperaba, de tal forma que se puedan ajustar o establecer objetivos de mejora continua.

Con respecto a los puntos de referencia o revisión, se deben identificar los procesos que el departamento requiere hacer bien o que necesita mejorar y hacer una comparación con el plan de actividades para verificar qué tan bien el departamento está llevando a cabo las cosas actualmente y qué nivel de realización él debería de alcanzar.

- Verificar que los programas o proyectos se estén desarrollando de acuerdo a las prioridades establecidas.

- Asegurar que la formación dada se lleva a cabo de acuerdo a las habilidades necesarias para cumplir los compromisos del proyectos.

- Verificar que el desarrollo del proyecto esté basado en los estándares proporcionados por la Dirección.

Difusión o propagación de lecciones aprendidas: Sus actividades son:

- Establecer o determinar las mejores prácticas y medidas correctivas de acuerdo a las lecciones aprendidas.

- Difundir e institucionalizar las mejores prácticas y medidas correctivas necesarias.

Soporte Interno para Director del Proyecto y Grupo del Proyecto (Soporte interno): Sus actividades son:

- Verificar que el portafolio de actividades completo muestra progreso para la culminación exitosa del proyecto; en un alto nivel para la Dirección y con un soporte más detallado para aquellos responsables del desarrollo de un proyecto.

- Ayudar a los grupos del proyecto en el arranque e iniciación del mismo, y en todo su ciclo de vida.

- Proporcionar soporte para el Director del proyecto, tales como:

    - Ayudarlo a evaluar o valorar su propia capacidad, proporcionandole un adiestramiento a la medida y apropiado.

- Proporcionar consejos adecuados durante el proyecto cuando sea necesario.

- Proporcionar soporte para generar un reporte coordinado y consistente de todos los proyectos del departamento.

- Verificar o revisar si las prioridades establecidas de los proyectos, están acorde con los estándares establecidos por la Dirección, para evitar: causas comunes de fallas, asegurar que no hay pérdida de dependencia entre proyectos y, asegurar enlaces consistentes con las prioridades y objetivos actuales del departamento.

- Verificar que se estén llevando a cabo las mejores prácticas o medidas correctivas y lecciones aprendidas recomendadas, asegurando la provisión de entrenamiento o formación apta apto(a) para el propósito del proyecto.

5.2.2 Análisis del Comportamiento

A partir del modelo anterior y tomando en cuenta las actividades realizadas por un centro de excelencia (CDE), se procede a identificar los actores o agentes involucrados en dicho modelo.

Actores del centro de entrenamiento:

- Planificación
- Administración
- Instrucción
- Cliente

Actores del centro de excelencia:

- Líder del centro de excelencia

- Soporte
- Control
- Dirección

Es importante señalar que los actores Soporte y Control, forman conjuntamente el actor Director del Proyecto (ver Figura 5.3). El Director del Proyecto realiza el control del desarrollo del CDE, la difusión o propagación de las lecciones aprendidas y el soporte interno. El reporte a Dirección lo hace el Líder del centro de excelencia.

Por otra parte, los puntos de decisión claves para el modelo propuesto de un centro de entrenamiento que se van a considerar, son los siguientes:

- **Gestión de Contratación:** Comprende contratación de instructores.

- **Entrenamiento o Formación Académica:** Comprende la formación de los instructores contratados.

- **Evaluación del curso:** Comprende: Evaluación del instructor, evaluación de los recursos disponibles (hardware, software, espacio físico, etc), Programación y contenido y, documentación (material de apoyo).

- **Procesos Administrativos:** Comprende: cobros, inscripciones, expedientes y publicidad.

Habiendo identificado los puntos de decisión claves, se establecen las reglas lógicas de los actores del centro de entrenamiento y del centro de excelencia. Al igual que para la Empresa para Desarrollo de Software, se hace uso de la ontología mereológica para representar el conocimiento de los agentes.

La Figura 5.5, muestra un esquema de las relaciones entre los agentes del CDE y del Centro de Entrenamiento. El departamento está constituido por los actores del Centro de

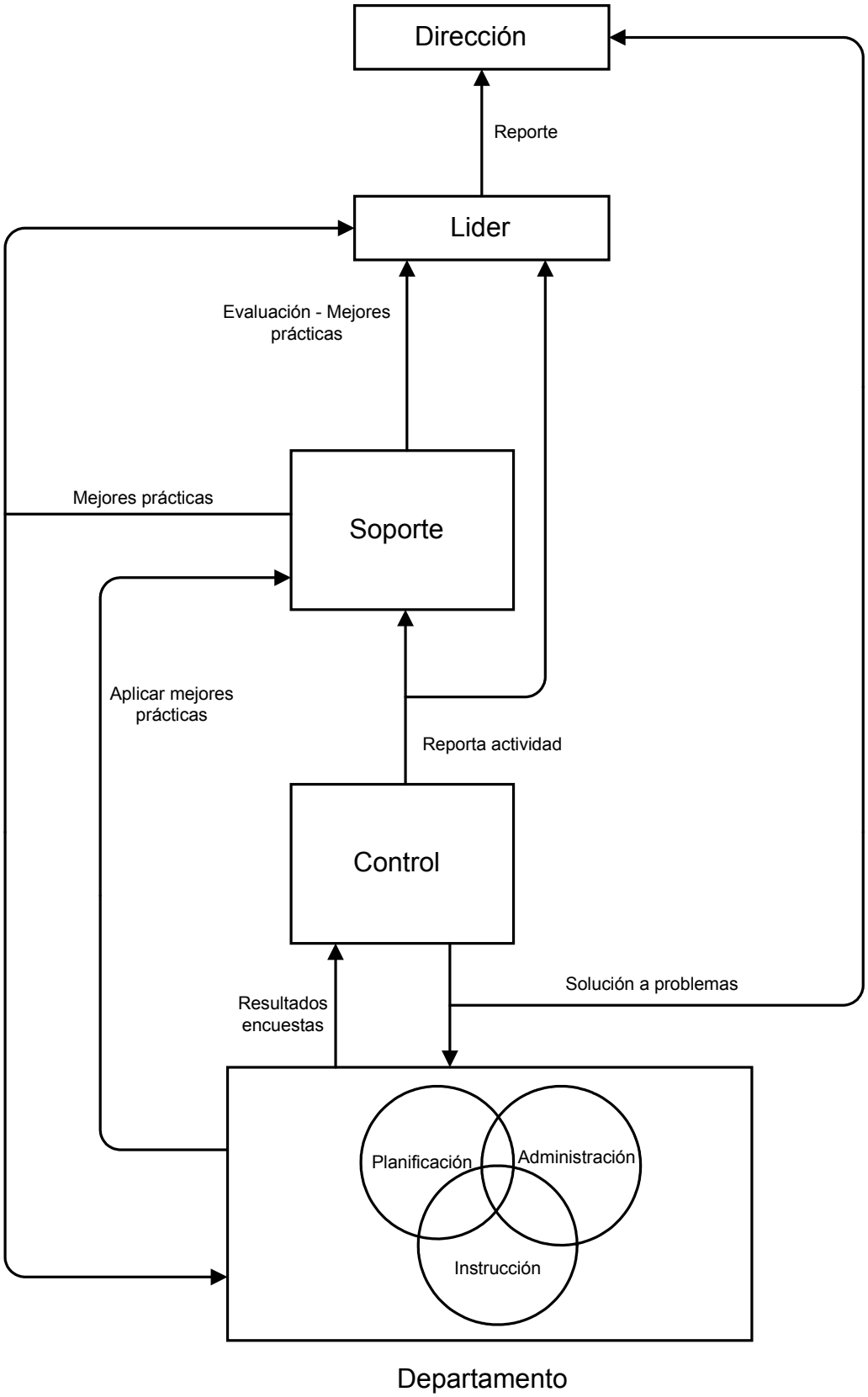


Figura 5.4. Relación entre los actores del COE y del Centro de Entrenamiento

Entrenamiento (Planificación, Administración e Instrucción). Los actores del centro de excelencia (CDE), tienen una jerarquía ascendente, tal como se observa en la figura, hasta que finalmente el actor Líder del CDE envía un reporte de las actividades realizadas a Dirección, quién hace una evaluación de dichas actividades y de los mecanismos utilizados para llevarlas a cabo.

La implementación del conocimiento de cada uno de los agentes, se ha realizado bajo el lenguaje de programación lógica Prolog, pero haciendo uso bajo su contexto, de un interpretador que permite la integración de dos lenguajes de programación lógica, como son: ACTILOG [15] y OPENLOG [16]. Este interpretador ha sido desarrollado por Jacinto Dávila (http://cesimo/ing.ula.ve/~jacinto). El ACTILOG activa o dispara la causa que posteriormente producirá un efecto y, el OPENLOG genera el efecto producido.

El uso de este interpretador en el lenguaje Prolog consiste en colocar, en cada archivo donde se especifican las reglas lógicas de un agente, dicho interpretador como encabezado. A continuación, se muestra la implementación de las reglas lógicas del actor Grupo de Mejora bajo Prolog, haciendo uso de la notación permitida por el interpretador de la integración, es decir, que estas reglas lógicas se han convertido al formato que requiere el cerebro de los agentes GALATEA.

A continuación, se va a ejemplificar el uso de ontologías a través de la descomposición de las actividades de un actor en sus roles específicos, hasta llegar a las reglas lógicas correspondientes. Se va a trabajar con el actor Planificación del Centro de Entrenamiento.

La base de conocimiento de un agente se especifica mediante las instrucciones "`observable`" y "`executable`". Por medio de la primera instrucción, se definen las percepciones del agente y a través de la segunda, se definen las acciones realizadas por el agente en base a sus percepciones.

Reglas Lógicas del Actor Planificación en Prolog

El correspondiente archivo en Prolog, donde se especifican las reglas lógicas del agente Planificación, se presenta a continuación:

```
observable(necesidades_del_mercado).
observable(apertura_de_inscripciones).
observable(finalizar_curso).
observable(resultados_encuesta_al_cliente).
observable(resultados_encuesta_al_instructor).
observable(proporcionan_soluciones_problemas_con_material_apoyo).
observable(proporcionan_soluciones_problemas_con_programacion_y_contenido
).
observable(proporcionan_soluciones_problemas_con_infraestructura).
observable(aplicar_medidas_correctivas).

executable(planificar_curso).
executable(promocionar_curso).
executable(encuesta_al_cliente).
executable(encuesta_al_instructor).
executable(enviar_al_COE_resultados_encuestas).
executable(enviar_certificados).
executable(ejecutar_soluciones_al_alcance).
executable(aplicadas_medidas_correctivas).
executable(soluciones_con_material_apoyo).
executable(soluciones_con_programacion_y_contenido).
executable(soluciones_con_infraestructura).
```

Una vez especificada la base de conocimiento, se tienen las reglas lógicas que definen el comportamiento del agente: "`Integrity constraints`". A continuación, debajo de algunas de las reglas lógicas se va a expresar cuál es su significado, sin embargo todas estas reglas lógicas se presentan en los anexos 15 - 22.

```
% Integrity constraints.

si necesidades_del_mercado entonces planificar_curso.
si apertura_de_inscripciones entonces promocionar_curso.
si finalizar_curso entonces encuestas.
```

Cuando el curso finaliza, Planificación envía a Instrucción un formato de encuestas que los participantes deben responder. Posteriormente Planificación hace una evaluación de los resultados obtenidos y de esta manera evalúa el curso dictado.

```
si resultados_encuestas entonces enviar_al_COE_resultados_encuestas,
enviar_certificados.
```

Una vez finalizado el curso y realizan encuestas a los participantes y los resultados de dichas encuestas son evaluados por Planificación y enviados al centro de excelencia para su verificación. Cuando los datos han sido verificados por el CDE, este centro envía los certificados de los cursantes aprobados a Planificación y proporciona soluciones a posibles problemas que se hayan presentado con el material de apoyo, con la programación y contenido o con la infraestructura .

```
si proporcionan_soluciones_problemas entonces
ejecutar_soluciones_al_alcance.
```

Si se presentaron problemas con el material de apoyo, con la programación y contenido o con la infraestructura, entonces Planificación ejecuta las soluciones respectivas que lleven a solventar dichos problemas.

```
si aplicar_medidas_correctivas entonces aplicadas_medidas_correctivas.

% Definitions

para encuestas haga
     encuesta_al_cliente,
     encuesta_al_instructor.

para resultados_encuestas haga
     resultados_encuesta_al_cliente,
     resultados_encuesta_al_instructor.

para proporcionan_soluciones_problemas haga
     proporcionan_soluciones_problemas_con_material_apoyo,
     proporcionan_soluciones_problemas_con_programacion_y_contenido,
     proporcionan_soluciones_problemas_con_infraestructura.

para ejecutar_soluciones_al_alcance haga
     soluciones_con_material_apoyo,
     soluciones_con_programacion_y_contenido,
     soluciones_con_infraestructura.
```

A continuación se muestra gráficamente (ver Figura 5.5) la ontología teleológica para el caso de la última regla lógica de este actor, en la sección "`% Integrity constraints`":

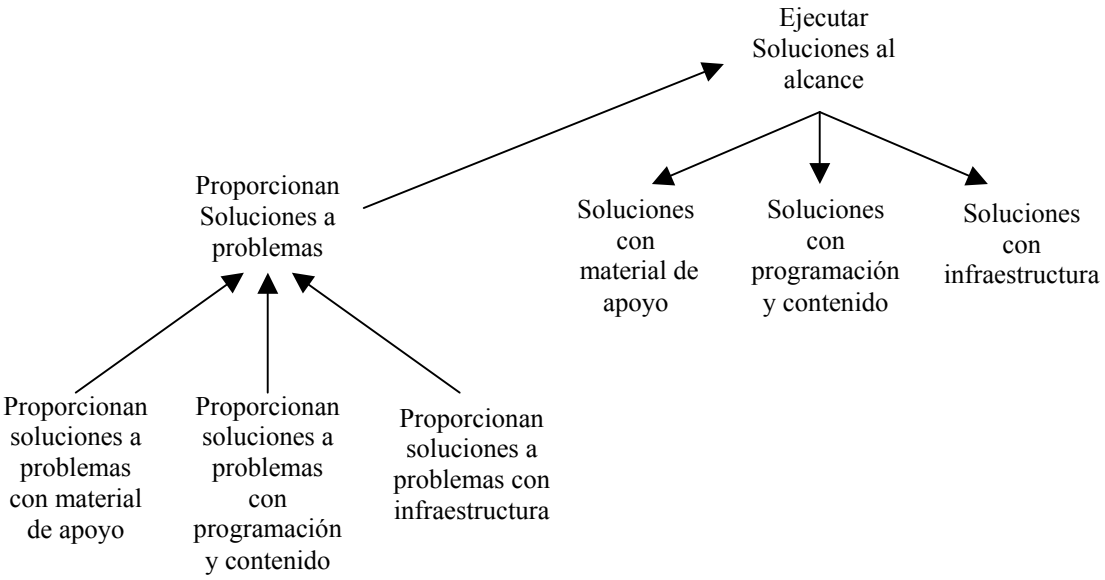


Figura 5.5. Arbol para ejemplificar el uso de ontología teleológica en el centro de entrenamiento

Una vez que el archivo ha sido compilado, el Prolog genera una salida, la cual muestra, las influencias del agente, que corresponden a las acciones tratándose de ejecutar en ese instante y las metas que quedan pendientes por hacer. Las influencias y metas de salida, dependen del escenario planteado. A continuación se muestra la salida obtenida.

```
% c:/windows/pl.ini compiled 0.00 sec, 392 bytes
Welcome to SWI-Prolog (Multi-threaded, Version 5.2.7)
Copyright (c) 1990-2003 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?-
% c:/Tesis/Agentes_en_Prolog/Actores del Centro de Entrenamiento/Actores
del Centro/Planificacion.pl compiled 0.05 sec, 17,060 bytes
1 ?- c(I,A).

I = [enviar_al_COE_resultados_encuestas, enviar_certificados,
ejecutar_soluciones_al_alcance]
```

La salida “I” representa el plan inmediato del agente cuando el escenario planteado es el siguiente:

```
observe resultados_encuestas.
observe proporcionan_soluciones_problemas.
```

5.3 Descripción del flujo secuencial de un curso dictado por el centro de entrenamiento

Este flujo ha sido simulado en el lenguaje de simulación GLIDER [12]; el sistema a simular corresponde al funcionamiento del Centro de Entrenamiento, incorporándole la gestión de calidad (CDE). Se asume que cada curso a dictar ha sido producto o resultado del estudio de mercado realizado previamente.

Cada entrada al sistema está referida a un curso a dictar por el centro; las llegadas se simulan por medio de una distribución exponencial y además, se trabaja con cualquiera de las unidades funcionales de Planificación, Administración e Instrucción como un solo Departamento, incluyendo al Director del Proyecto y los cursos o proyectos.

La Figura 5.6, muestra el flujo secuencial de un proyecto desarrollado por la empresa .

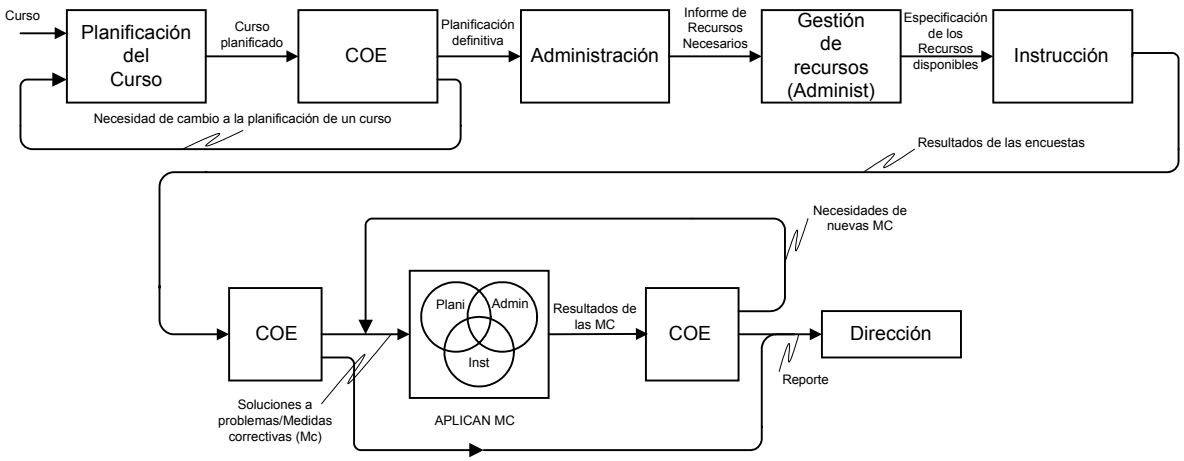


Figura 5.6. Flujo secuencial de un curso dictado por el Centro de Entrenamiento

Tal como se muestra en la figura 5.6, cuando un curso es generado, éste pasa a la etapa de Planificación, donde permanece cierto tiempo siendo planificado. Posteriormente, la planificación hecha es enviada al CDE para su respectiva evaluación y de ser necesario, es enviada por el CDE de nuevo a Planificación para realizarle los cambios correspondientes. En caso contrario, cuando ya se tiene la planificación definitiva, se envía a la etapa de Administración, donde se realizan las actividades relacionadas con la contactación de los instructores y con la búsqueda de disponibilidad de la infraestructura y demás requerimientos necesarios para el dictado del curso, así como también el envío al CDE de los expedientes de los instructores contactados (esto se especifica en el informe de recursos necesarios).

Al mismo tiempo, Administración realiza la apertura de inscripciones, archivando los expedientes de los inscritos, mientras que el CDE hace la selección del instructor apropiado, en base a los expedientes de los instructores contactados. Si el número de personas inscritas es menor que el número mínimo de cursantes, el curso no se abre, en caso contrario, el curso es abierto. Si el curso se abre, entonces Administración comienza a realizar la gestión operativa de los recursos necesarios para dictar el curso, incluyendo la contratación del instructor (Figura 5.6: especificación de los recursos disponibles). Mientras Administración hace efectivo el alcance de los recursos necesarios, el instructor realiza la preparación del curso a dictar, bajo las políticas de planificación y programación de contenido, emitidas por la Dirección del Centro de Entrenamiento.

Cuando se tienen disponibles los recursos necesarios para dictar el curso, la etapa Instrucción realiza el inicio del mismo, que comprende: el dictado del curso por parte del instructor y la asistencia por parte de los participantes, así como la realización de las evaluaciones necesarias. Una vez que el curso haya finalizado, la etapa Instrucción realiza encuestas tanto para los participantes como para el instructor; los resultados de las encuestas son enviados al grupo de gestión de calidad (CDE) para su respectiva evaluación y también, se determinan cuántas personas aprueban el curso y se actualizan los expedientes.

El CDE realiza la evaluación de las encuestas y en caso de encontrar problemas con infraestructura, con el material de apoyo, con la evaluación o con el instructor, genera o proporciona las soluciones posibles a estos problemas; estas soluciones o medidas correctivas son enviadas a las unidades funcionales de Planificación, Administración e Instrucción para ser aplicadas por ellas. Si no se encuentran problemas de los mencionados anteriormente, o una vez aplicadas las medidas correctivas en caso de existir estos problemas, se envía un reporte a Dirección de las eventualidades del curso y, Dirección refinas sus políticas en caso de ser necesario.

5.4 Pruebas y Resultados

Una vez implementado el centro de entrenamiento, realizamos ciertos experimentos para su posterior análisis:

Experimento 1: Capacidad de Planificación = 1
Capacidad de Administración = 1
Capacidad de Gestión de Recursos = 1
Capacidad de Instrucción = 1
Capacidad del Centro de Excelencia (CDE) = 1

El tiempo de simulación es de 5280 horas, lo que equivale a tres años de actividad y, el tiempo de generación entre cada curso es de 480 horas, es decir, tres meses de diferencia entre la generación de un curso y otro, esto significa que bajo un supuesto, nos solicitan en promedio, un curso cada tres meses.

Tabla No 5.1. Tiempos de generación, culminación y en el sistema de los cursos y, el tiempo desde la planificación del curso hasta su inicio

Orden de Culminación	Tipo de curso en orden de generac	Tiempo de generación (h)	Tiempo de culminación	Tiempo en el sistema (h)	Tiempo desde Planificación a Instrucción
1	Tipo 3	0	614	614	408
2	Tipo 2	480	1023	543	382
3	Tipo 1	1440	2062	622	361
4	Tipo 1	1920	2411	491	363
5	Tipo 2	2400	2903	503	370
6	Tipo 1	2880	3740	860	651
7	Tipo 1	3360	3989	629	347
8	Tipo 3	3840	4518	678	452
9	Tipo 1	4320	4873	553	451

Tabla No. 5.2. Número de proyectos que entran en cada etapa, tiempo libre de las listas de entrada (EL) e interna (IL) de las etapas y, tamaño de estas listas.

Etapa	# Entradas	Tiempo Libre Listas (TLL)	%TLL	Lgth	Max Cola
Cursos	12	_	_	_	_
Planifi (EL)	20	5123,28	97,03	0	1
Planifi (IL)	20	3921,08	74,26	1	1
Administ(EL)	16	5200,14	98,49	0	1
Administ (IL)	16	2423,21	45,89	0	1
Gest_R (EL)	10	5280,00	100	0	1
Gest_R (IL)	10	5139,46	97,34	0	1
Instruc (EL)	15	5280,00	100	0	1
Instruc (IL)	15	4524,07	85,68	1	1
COE(EL)	28	5194,90	98,39	0	1
COE (IL)	28	3923,99	74,32	0	1
Dirección (EL)	10	5280,00	100,0	0	1

 # Entradas: Número de entradas

Tiempo Libre Listas: Tiempo en que estuvo la lista vacía

Lgth: Largo actual de la lista

Max Cola: Largo máximo de la cola

EL: Lista de entrada; proyectos esperando para hacer uso del recurso de una etapa.

IL: Lista interna; corresponde al número de proyectos usando el recurso de una etapa.

En este primer experimento se obtiene que de los 12 cursos generados (ver # entradas en cursos), uno no se abre por falta de personas inscritas y, de los otros 11 cursos abiertos, 9 logran finalizar completamente, uno queda en la etapa Planificación (ver Lgth Planifi (IL)) y el otro en Instrucción (ver Lgth Instruc (IL)). Mediante la suma de los 9 cursos abiertos que logran finalizar, más el curso no abierto por falta de personas inscritas, se obtienen los 10 cursos que llegan a la EL del nodo de salida Dirección. Cada tipo de curso tiene un tiempo fijo de instrucción:

Tabla No.5.3. Tipo de curso y tiempo de duración de los mismos

Tipo curso	Tiempo de duración (horas)
1	40
2	72
3	80

En esta implementación, la variación del tiempo en el sistema de un curso, depende de las capacidades asignadas a cada etapa funcional. En este sentido, se observa que para las condiciones dadas en este experimento, el tiempo en el sistema de cada uno de los cursos generados y que logran ser finalizados totalmente, se encuentra en un promedio de 610,33 horas. El tiempo desde la planificación de un curso hasta su inicio, para el caso de los cursos abiertos, al igual que el tiempo anterior, depende de las capacidades de las etapas funcionales Planificación y Administración, que son las que influyen en el cálculo de este tiempo; con respecto a este tiempo, casi todos los cursos se encuentran en un promedio de 420,56 horas.

A diferencia de la implementación anterior, el tiempo en el sistema de los cursos no aumentan notablemente a medida que el tiempo transcurre, sino que como se ha señalado anteriormente, se mantiene alrededor de un promedio. Por otra parte, las listas tanto de entrada como interna de cada una de las etapas funcionales, incluyendo el Centro de Excelencia (COE), se mantienen la mayor parte del tiempo vacías, porque el porcentaje que permanecen libres es bastante elevado, excepto la lista interna de la etapa funcional de Administración, que permanece inactiva un poco menos del 50 por ciento del tiempo (ver %TTL Administ (IL)).

Algo importante que se observa, es que los cursos no tienen que esperar en cola para gestionar operativamente los recursos necesarios para su dictado y tampoco para ser dictados por el instructor asignado en la etapa de Instrucción (ver Max Cola de Gest_R (EL) y Instruc (EL)). En el siguiente experimento2, se procede a aumentar las capacidades de las etapas de Planificación y Administración, para tratar de disminuir el tiempo transcurrido desde la planificación hasta el inicio de un curso.

A continuación, se muestra gráficamente el número de cursos en cola de las listas de entrada de las etapas funcionales Planificación y Administración, durante el período de simulación. Estos datos también se pueden observar en la columna Max Cola de la tabla 5.2

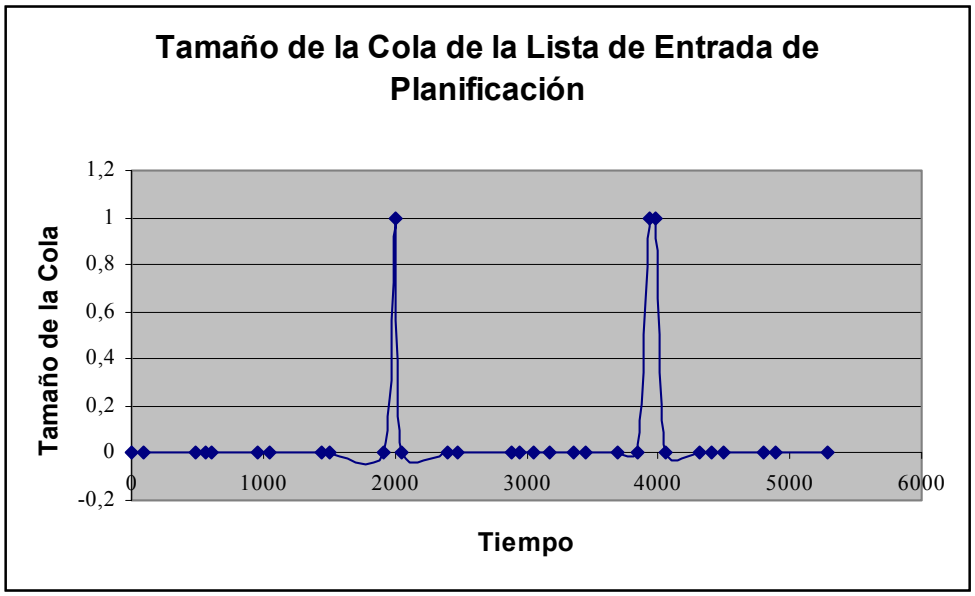


Figura 5.7. Tamaño de la Cola de la Lista de Entrada de Planificación

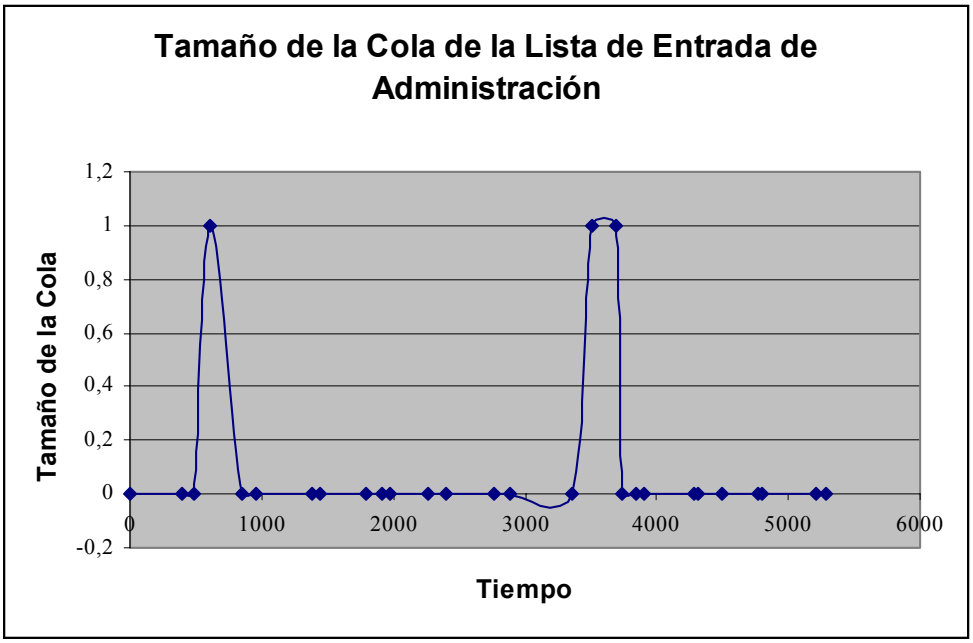


Figura 5.8. Tamaño de la Cola de la Lista de Entrada de Administración

Experimento 2: Capacidad de Planificación = 2
Capacidad de Administración = 2
Capacidad de Gestión de Recursos = 1
Capacidad de Instrucción = 1
Capacidad del Centro de Excelencia (COE) = 1

El tiempo de simulación sigue siendo de 5280 horas, es decir, tres años de actividad y, el tiempo entre generación de cursos es de 480 horas, lo que equivale a tres meses de diferencia. Los resultados obtenidos se muestran en las tablas 4 y 5.

En este experimento se obtiene que de los 12 cursos generados, 10 logran finalizar completamente; de los otros dos, uno queda en la etapa de Planificación y el otro en la etapa de Administración. Para este experimento, el tiempo promedio en el sistema es de 570,3 horas y el tiempo promedio de un curso desde su planificación hasta su inicio es de 397,6 horas; estos tiempos, con respecto al experimento anterior, han disminuido y las etapas funcionales permanecen mayor tiempo inactivas, es decir, sus listas tanto de entrada como interna, permanecen mayor tiempo desocupadas.

Se observa en este experimento, que la lista de entrada de la etapa funcional de Instrucción, permanece 100 por ciento vacía, lo que indica que los cursos no tienen que esperar para ser iniciados por el instructor asignado, una vez obtenidos los recursos necesarios para su dictado. Se desea ver cómo la capacidad del Centro de Excelencia (CDE) afecta a los tiempos mencionados anteriormente; para ello se realiza un nuevo experimento. Por supuesto, debe existir un límite superior en el número de cursos a dictar, porque de lo contrario, significaría que tenemos infinita capacidad de salones para dictar cualquier cantidad de cursos.

Tabla No. 5.4. Tiempos de generación, culminación y en el sistema de los cursos y, el tiempo desde la planificación del curso hasta su inicio

Orden de Culminación	**Tipo de curso en orden de generac**	**Tiempo de generación (h)**	**Tiempo de culminación**	**Tiempo en el sistema (h)**	**Tiempo desde Planificación a Instrucción**
1	Tipo 3	0	614	614	408
2	Tipo 2	480	950	470	298
3	Tipo 2	960	1498	538	361
4	Tipo 2	1440	2062	622	396
5	Tipo 1	1920	2411	491	363
6	Tipo 2	2400	2903	503	370
7	Tipo 1	2880	3686	806	650
8	Tipo 1	3360	3789	429	321
9	Tipo 1	3840	4399	559	348
10	Tipo 2	4320	4991	671	461

Tabla No. 5.5. Número de proyectos que entran en cada etapa, tiempo libre de las listas de entrada (EL) e interna (IL) de las etapas y, tamaño de estas listas

Etapa	# Entradas	Tiempo Libre Listas (TLL)	%TLL	Lgth	Max Cola
Cursos	12	_	_	_	_
Planifi (EL)	23	5280,00	100	0	1
Planifi (IL)	23	3872,67	73,35	1	2
Administ(EL)	18	5280,00	100	0	1
Administ (IL)	18	2665,89	50,5	1	2
Gest_R (EL)	10	5280,00	100	0	1
Gest_R (IL)	10	5119,25	96,96	0	1
Instruc (EL)	17	5280,00	100	0	1
Instruc (IL)	17	4400,00	83,33	0	1
COE(EL)	32	5213,20	98,73	0	1
COE (IL)	32	3857,98	73,07	0	1
Dirección (EL)	10	5280,00	100	0	1

Experimento 3: Capacidad de Planificación = 1
Capacidad de Administración = 1
Capacidad de Gestión de Recursos = 1
Capacidad de Instrucción = 1
Capacidad del Centro de Excelencia (CDE) = 2

El tiempo de simulación sigue siendo de 5280 horas, es decir, tres años de actividad y, el tiempo entre generación de cursos es de 480 horas, lo que equivale a tres meses de diferencia. Los resultados obtenidos son:

Tabla No. 5.6. Tiempos de generación, culminación y en el sistema de los cursos y, el tiempo desde la planificación del curso hasta su inicio

Orden de Culminación	Tipo de curso en orden de generac	Tiempo de generación (h)	Tiempo de culminación	Tiempo en el sistema (h)	Tiempo desde Planificación a Instrucción
1	Tipo 3	0	614	614	408
2	Tipo 2	480	975	495	340
3	Tipo 3	960	1597	637	425
4	Tipo 4	1440	1964	524	391
5	Tipo 1	1920	2448	528	401
6	Tipo 3	2400	3047	647	282
7	Tipo 1	2880	3471	591	422
8	Tipo 1	3360	3956	596	400
9	Tipo 1	3840	4251	411	320
10	Tipo 1	4320	4900	580	392

Tabla No. 5.7. Número de proyectos que entran en cada etapa, tiempo libre de las listas de entrada (EL) e interna (IL) de las etapas y, tamaño de estas listas

Etapa	# Entradas	Tiempo Libre Listas (TLL)	%TLL	Lgth	Max Cola
Cursos	12	_	_	_	_
Planifi (EL)	20	5100,00	96,60	0	1
Planifi (IL)	20	4076,22	77,20	1	1
Administ(EL)	19	5261,67	99,65	0	1
Administ (IL)	19	2315,09	43,85	0	1
Gest_R (EL)	10	5280,00	100	0	1
Gest_R (IL)	10	5126,73	97,10	0	1
Instruc (EL)	18	5280,00	100	0	1
Instruc (IL)	18	4368,00	82,70	0	1
COE(EL)	29	5280,00	100	0	1
COE (IL)	29	4096,36	77,00	0	2
Dirección (EL)	11	5280,00	100	0	1

Cuando la capacidad del CDE es aumentada a 2, los tiempos de los cursos en el sistema y desde la planificación de los mismos hasta sus inicios, han disminuido con respecto al primer experimento. Se sigue observando que las listas de entrada e interne de cada una de las etapas funcionales, incluyendo el CDE, están la mayor parte del tiempo vacías. Debido a esta característica, se realiza un nuevo experimento donde se aumenta el número de cursos a dictar, bajo el mismo tiempo de simulación.

Experimento 4: Capacidad de Planificación = 1
Capacidad de Administración = 1
Capacidad de Gestión de Recursos = 1
Capacidad de Instrucción = 1
Capacidad del Centro de Excelencia (CDE) = 1

El tiempo de simulación sigue siendo de 5280 horas y el tiempo entre generación de cursos es de 160 horas, lo que equivale a un mes. Los resultados obtenidos se muestran en las tablas 5.8 y 5.9.

Se observa que de los 34 cursos generados, sólo 13 logran finalizar completamente. Además, los tiempos de los cursos en el sistema y desde su planificación hasta su inicio, con respecto a los experimentos anteriores han aumentado lo suficiente para ser considerado significativo. Por otra parte, se observa que la mayoría de los cursos, permanecen esperando en la lista de entrada de la etapa funcional de Administración, ya que las listas tanto de entrada como interna de esta etapa son las que menos tiempo permanecen vacías; para evitar tanto tiempo de espera en esta etapa, se realiza un nuevo experimento donde se aumenta la capacidad de Administración.

Los cursos que se abren pero no logran finalizar, porque quizá el tiempo de simulación no ha permitido su ejecución completa, pueden quedar en la Lista Interna de cualquiera de las etapas consumiendo recursos. Por otra parte, los cursos que no se abren por falta de personas inscritas, para efectos de la simulación, son enviados directamente al nodo de salida Dirección, donde son sacados del sistema.

Tabla No. 5.8. Tiempos de generación, culminación y en el sistema de los cursos y, el tiempo desde la planificación del curso hasta su inicio

Orden de Culminación	Tipo de curso en orden de generac	Tiempo de generación (h)	Tiempo de culminación	Tiempo en el sistema (h)	Tiempo desde Planificación a Instrucción
1	Tipo 3	0	569	569	402
10	Tipo 2	160	4311	4151	495
2	Tipo 2	320	1120	800	593
3	Tipo 3	480	1265	785	609
4	Tipo 3	640	1503	863	662
5	Tipo 1	960	1892	932	793
8	Tipo 2	1120	3170	2050	863
6	Tipo 3	1280	2417	1137	993
7	Tipo 3	1600	2872	1272	1122
9	Tipo 1	1760	3746	1986	1227
11	Tipo 2	2080	4540	2460	1474
12	Tipo 2	2720	4732	2012	1828
13	Tipo 2	3040	5205	2165	2022

Tabla No. 5.9. Número de proyectos que entran en cada etapa, tiempo libre de las listas de entrada (EL) e interna (IL) de las etapas y, tamaño de estas listas

Etapa	# Entradas	Tiempo Libre Listas (TLL)	%TLL	Lgth	Max Cola
Cursos	34	_	_	_	_
Planifi (EL)	55	4149,93	78,60	0	2
Planifi (IL)	55	1367,68	25,90	1	1
Administ(EL)	48	437,73	8,29	20	21
Administ (IL)	28	139,611	2,64	1	1
Gest_R (EL)	17	5280,00	100	0	1
Gest_R (IL)	17	4994,77	94,6	0	1
Instruc (EL)	32	5222,19	98,91	0	1
Instruc (IL)	32	3544,00	67,12	0	1
COE(EL)	67	4396,50	83,27	0	2
COE (IL)	67	2169,67	41,09	0	1
Dirección (EL)	17	5280,00	100	0	1

Experimento 5: Capacidad de Planificación = 1
Capacidad de Administración = 2
Capacidad de Gestión de Recursos = 1
Capacidad de Instrucción = 1
Capacidad del Centro de Excelencia (COE) = 1

El tiempo de simulación es de 5280 horas y el tiempo de generación entre cursos es de 160 horas (1 mes). Los resultados obtenidos se muestran en las tablas 5.10 y 5.11.

En este experimento, con respecto al anterior, se logra dictar un mayor número de cursos, específicamente 5 cursos más. El tiempo en el sistema y el tiempo desde la planificación de un curso hasta su inicio, han disminuido y se debe a que se ha logrado disminuir el número de cursos esperando en la lista de entrada de Administración, se ha disminuido de 20 cursos esperando a cero, ningún curso en espera.

Tabla No. 5.10. Tiempos de generación, culminación y en el sistema de los cursos y, el tiempo desde la planificación del curso hasta su inicio

Orden de Culminación	Tipo de curso en orden de generac	Tiempo de generación (h)	Tiempo de culminación	Tiempo en el sistema (h)	Tiempo desde Planificación a Instrucción
2	Tipo 3	0	1056	1056	405
1	Tipo 2	160	664	504	368
4	Tipo 2	320	1432	1112	552
6	Tipo 1	480	1728	1248	691
3	Tipo 1	640	1172	532	339
8	Tipo 2	800	2063	1263	717
5	Tipo 2	960	1563	603	441
7	Tipo 2	1120	1887	767	604
10	Tipo 2	1280	2808	1528	583
11	Tipo 1	1760	2868	1108	831
9	Tipo 2	2080	2557	477	366
12	Tipo 3	2400	3054	654	487
13	Tipo 2	2720	3492	772	609
14	Tipo 1	2880	3822	942	596
16	Tipo 3	3200	4315	1115	470
18	Tipo 2	3360	4697	1337	772
15	Tipo 1	3520	4050	530	397
17	Tipo 1	3680	4595	915	667

Es importante notar que el hecho de que las listas de entrada de las etapas funcionales y del CDE, estén la mayor parte del tiempo vacías, no es un hecho relevante, ya que en todo caso expresa que los cursos tardan poco tiempo en espera para ser atendidos. Si embargo, en el caso de las listas internas si es importante, porque esto indica que las etapas funcionales permanecen mucho tiempo inactivas, es decir, ociosas.

Tabla No. 5.11. Número de proyectos que entran en cada etapa, tiempo libre de las listas de entrada (EL) e interna (IL) de las etapas y, tamaño de estas listas

Etapa	# Entradas	Tiempo Libre Listas (TLL)	%TLL	Lgth	Max Cola
Cursos	34	_	_	_	_
Planifi (EL)	76	2110,32	39,97	4	6
Planifi (IL)	72	451,72	8,56	1	1
Administ(EL)	53	3913,55	74,12	0	3
Administ (IL)	53	614,360	11,64	2	2
Gest_R (EL)	20	5280,00	100	0	1
Gest_R (IL)	20	4966,85	94,10	0	1
Instruc (EL)	44	5116,87	96,91	0	1
Instruc (IL)	44	3032,00	57,42	0	1
COE(EL)	91	3564,39	67,51	1	4
COE (IL)	90	1615,91	30,6	1	1
Dirección (EL)	25	5280,00	100	0	1

De esta manera, se puede ver que para que el centro de entrenamiento tenga un desempeño deseado, se puede establecer el número de cursos que se desean dictar en un cierto período de actividad y comenzar a cambiar las capacidades tanto de las etapas funcionales como del centro de Excelencia (COE) para lograr que cada unidad tenga en uso su recurso un cierto porcentaje y así, alcanzar el desempeño deseado.

Para finalizar, en el modelo Glider que se ha presentado, los cursos a dictar consumen un cierto recurso en cada etapa funcional y en el sistema de gestión de calidad (CDE). En las simulaciones que se han realizado, el número de cursos consumiendo el recurso de cierta etapa o del CDE, se ha representado mediante la capacidad de dicha etapa. Es decir, capacidad igual a uno en cualquiera de las etapas, significa que puede existir un único curso

gestionándose en esa etapa y además, se está asumiendo que existe una persona dedicada ocho horas al día realizando las actividades correspondientes a la gestión de ese proyecto.

A través de los experimentos realizados bajo este modelo Glider, se observa que el centro resulta un poco costoso en función del personal dedicado, ya que para dictar un mayor número de cursos, es necesario aumentar el personal. Además de costoso, se observa que el centro estaría siendo subutilizado, en virtud de que las etapas funcionales permanecen operativas por pequeños períodos, es decir, que en los resultados se observa que el porcentaje que permanecen libres las listas internas de las etapas funcionales (cursos sin consumir el recurso) es bastante elevado.

Ya que el modelo propuesto permite que el centro resulte costoso y subutilizado, se propone un nuevo modelo que permita la integración de las actividades realizadas en las etapas con alto porcentaje de tiempo libre en sus listas internas, en una sola etapa en donde sean atendidos estos cursos por un número específico de personas. De esta manera sería más fácil poder determinar el número de personas necesarias en algunas etapas del centro de entrenamiento. La integración de las etapas podría pensarse para cualquiera excepto para la etapa de Instrucción, en donde asumimos que existe un único instructor por cada curso que se dicta.

Conclusión

En este trabajo se han modelado e implementado dos tipos específicos de organizaciones: una empresa para Desarrollo de Software y un Centro de Entrenamiento, utilizando las tecnologías de agentes y haciendo las simplificaciones correspondientes, con el propósito de observar el comportamiento de estas organizaciones e incorporando en sus modelos, como parte de las mismas, un componente novedoso e importante, como lo es la gestión de la calidad en los productos y servicios que estas organizaciones proporcionan.

Uno de los objetivos fundamentales de este trabajo fue transmitir ciertos conocimientos básicos al lector, a fin de que lograra entender los beneficios que ofrecen los sistemas multiagente para el desarrollo y simulación de organizaciones. En las implementaciones de las organizaciones basadas en sistemas multiagente, se especificaron las relaciones entre los diferentes actores o agentes involucrados en cada organización. Para definir estas relaciones, se procedió de la siguiente manera: una vez identificados los actores, se definieron sus actividades o funciones realizadas dentro de la organización y, posteriormente se estableció, en base a estas actividades, cuál era la relación que se originaba entre las actividades de uno y otro actor, de esta manera se definió las relaciones que un actor presentaba, respecto a todos los demás actores; de esta manera, se permitió reforzar la toma de decisiones a través de toda la organización y además, facilitar la cooperación y coordinación, requeridas para que las organizaciones funcionaran como un todo.

Las ideas básicas presentadas a través de los modelos de las organizaciones estudiadas basados en sistemas multiagente, consisten en dar importancia a los individuos como agentes o actores, usar el paradigma de programación por objetos para intentar modelar con más detalle la organización y su ambiente y, usar la lógica simbólica para la representación del conocimiento de los agentes. También se presentaron, como aspectos resaltantes, el comportamiento de los agentes y la toma de decisiones de los mismos, en base a la ontología mereológica; de esta manera, se logró que los individuos fueran representados

explícitamente y además, se establecieron en este trabajo, como elementos importantes para entender el fenómeno organizacional, los agentes y los procesos.

Así como el modelado basado en sistemas multiagente de las organizaciones permite reforzar la toma de decisiones dentro de las mismas, la descripción del flujo secuencial de los productos desarrollados por ellas, permite determinar de manera aproximada el tiempo promedio empleado por un producto para su desarrollo.

En la descripción del flujo secuencial, tanto para la empresa de desarrollo de software, como para el centro de entrenamiento, se observa, a través de las diferentes pruebas realizadas para cada uno de ellos, la influencia del Sistema de Gestión de Calidad, tanto en la culminación de todos los productos o proyectos generados, como en el tiempo promedio de los mismos en el sistema. En términos generales, se observa como el cambio en la capacidad del Sistema de Gestión de Calidad afecta al tiempo promedio que los productos o proyectos permanecen en cola esperando ser atendidos por los diferentes procesos involucrados en los modelos de cada una de las organizaciones estudiadas. Por otra parte, estos tiempos no sólo dependen del Sistema de Gestión de Calidad sino también de las capacidades de los procesos, es decir, la disposición de gente de trabajo para realizar actividades específicas al mismo tiempo.

En los experimentos realizados en la simulación del flujo secuencial tanto de la empresa para desarrollo de software, como para el centro de entrenamiento, se observó que aún cuando la mayor parte del tiempo, las listas de entrada e interna de cada una de las etapas funcionales, se encuentran vacías, el tiempo promedio de cada uno de cada uno de los proyectos, logra aproximarse al tiempo promedio estipulado al inicio de la simulación, siempre que las capacidades tanto de las etapas del proceso de ejecución, como la del sistema de gestión de calidad, sean cambiadas; generalmente, si las capacidades aumentaban, el tiempo promedio de los proyectos en el sistema se acerca al tiempo promedio estipulado y además, el tamaño de las listas disminuía. Sin embargo, se pudo observar también en los experimentos, que se llegaba a un punto en el que aumentar las capacidades mencionadas, específicamente se comprueba con la capacidad del sistema de

gestión de calidad, no variaba el tiempo promedio de los proyectos en el sistema y tampoco, el tiempo de espera de los mismos en las etapas funcionales del proceso de ejecución y en el sistema de gestión de calidad.

Con respecto a la representación del conocimiento de los agentes, se observa como resultado de su implementación, las acciones a ejecutar en ese instante y los planes a ejecutar a futuro. Las influencias y acciones obtenidas en las simulaciones, dependen del escenario planteado para cada agente, es decir, de la especificación de sus percepciones, de su conocimiento y de las posibilidades generadas en base a su capacidad de razonamiento.

Finalmente, se puede decir que casi todos los objetivos específicos propuestos en el capítulo 1 fueron alcanzados, excepto la integración de todos los agentes de la Empresa para Desarrollo de Software y de los agentes del Centro de Entrenamiento. Este objetivo no fue alcanzado, debido a que el compilador GALATEA, en donde se realizaría la implementación del sistema multiagente, no ha sido culminado y depurado en su totalidad. La realización de este compilador corresponde a un trabajo de investigación que está siendo desarrollado actualmente por Jacinto Dávila.

Se propone como trabajo futuro, la integración del sistema multiagente para ambas organizaciones y, de esta manera, obtener resultados concluyentes acerca del comportamiento general de las mismas.

Bibliografía

[1] Carley, K., 'Computational and Mathematical Organization Theory: Perspectiva and Directions', *Computational Organization Theory,* 1995.

[2] Carley, Kathleen M. And Les Passer, 'Computational Organization Theory' in Weiss, Gerhard (ed.), *Multiagent Systems: a Modern Approach to Distributed Artificial Intelligence,* 1999.

[3] Carlos Iglesias, Mercedes Garijo y José González. *Metodologías orientadas a agentes.* Informe de Investigación. Esta investigación fue realizada durante una visita del primer autor al Departamento de Ingeniería de Sistemas Telemáticos de la Universidad Politécnica de Madrid, (sf).

[4] Centres of Excellence. *Programme/Project Management. 2003.*

[5] Chandrasekaran, B. , Josephson, J. R., & Benjamins, V. R. (1999): *What are ontologies, and why do we need them.* IEEE Intelligent Systems, 20 – 26.

[6] Cohen, M. D., J. P. Olsen, 'A Garbage Can Model of Organizational Choice', *Administrative Science Quarterly,* 17 (1), pp. 1 – 25, 1972.

[7] Duineveld, A. J., Stoler R., Weiden M. R., Kenepa, B. , & Benjamins. V. R (1999): *Wonder Tools?. A comparative study of ontological engineering tools.* International Journal of Human – Computer Studies 52:1111 – 1133.

[8] Elizabeth A. Kendall, Margaret T. Malkoun y Chong Jiang. *A methodology for Developing Agent Based Systems for Enterprise Integration.* In D. Luckose and Zhang C., editors, Proceedings of the First Australian Workshop on DAI, Lecture Notes on Artificial Intelligence. Springer – Verlag: Heidelberg, Germany, 1996.

[9] ESI: European Software Institute. *Introducción a la mejora de procesos de software.* Manual del estudiante, 2001.

[10] Fernández, J. T. , Paniagua, E. Martín, F. '*Análisis de los sistemas multiagente para representar organizaciones humanas*'. SEID 2000, Vol. 2, Mesas Redondas paralelas a SEID 2000. Ed. Jayanes L., García, A., Paniagua, E., Orense (2000).

[11] Gerhard Weiss, *Multiagent Systems.* A Modern Approach to Distributed Artificial Intelligence. The MIT Press Cambridge, Massachusetts London, England, 1999.

[12] Glider Development Group *GLIDER Manual de Referencia,* Versión 5.9, Cesimo e IEAC, Universidad de Los Andes, Mérida, Venezuela, Publicaciones del Cesimo IT – 02 – 00; versión en español del manual de referencia, draft, 1 -. Pp, febrero, 2000.

[13] Gruber T. R. : '*A translation approach to portable ontology specifications*'. Knowledge Acquisition Journal 5: 199 – 220, 1993.

[14] '*Integration Definition for Function Modeling* ' (IDEF0).

[15] ISO 9001:2000. Quality Management System Requirements <http://www.isoeasy.org/2000%20Sumary>

[16] Jacinto Dávila. ACTILOG: A logic programming language based on Abduction. Lecture Notes in Computer Science. 1702. Springer,1999.

[17] Jacinto Dávila. OPENLOG: An Agent Activation Language. Practical Aspects of Declarative Languages Dahl, V and Wadler, P. Editors. Lecture Notes in Computer Science, 2562. Springer, 2003.

[18] James Stoner, R. Edward Freeman y Daniel Gilbert. *Administración.* Prentice Hall Hispanoamericana, S. A. Sexta edición, 1996.

[19] J. Ferber. Reactive distributed artificial intelligence: Principles and applications. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence.* John Wiley & Sons, 1996.

[20] Jonás Montilva, *'Ingeniería de Software',* unidad 1. Universidad de Los Andes, Postgrado en Computación, Facultad de Ingeniería, 1997 – 1998.

[21] Marcelo Arnold, Ph. D. Y Francisco Osorio, M. A. *Introducción a los Conceptos Básicos de la Teoría General de Sistemas.* Facultad de Ciencias Sociales. Departamento de Antropología. Universidad de Chile, 1998. < http://rehue.csociales.uchile.cl/publicaciones/moebio/o3/frames45.htm>

[22] Masuch M., and LaPotin, 'Beyond Garbage Cans: An AI Model of Organizational Choice', *Administrative Sciences Quarterly*, 1989.

[23] Moss Scott, 'Critical Incident Management: An Empirically Derived Computational Model', Journal of Artificial Societies and Social Simulation Vol 1, No 4, 1998. <http://www.soc.surrey.ac.uk/JASSS/1/4/1.html>

[24] Oswaldo Terán. Monografía: *Modelado de Organizaciones.* Departamento de Investigación de Operaciones, Escuela de Ingeniería de Sistemas, Universidad de Los Andes, (sf).

[25] Patrick Billingsley. '*Probability and Measure*'. Second Edition. The University of Chicago, 1986. pp. 263

[26] Rational Software Corporation. *Unified Modelling Language (UML).* Cersión 1.0. Notation Guide, 1997. <http://www.rational.com>

[27] Shafritz, Jay M. (Editor), y J. Steven Ott (1997), *Classics of Organization Theory,* 4th Edition, Harcourt Brace.

[28] Studer. R., Benjamins, V. R., & Fensel, D. (1998): *Knowledge engineering principles and methods*. Data and Knowledge Engineering, 25: 161 – 197.

ANEXOS

Anexo 1

```
% ------------------------------------------- For Ingeniero de Software
observable(kpa_gestion_requisitos).
observable(necesidades_cliente).
observable(necesidades_cambios_en_informacion).
observable(kpa_planificacion_proyectos).
observable(planes_iniciales).
observable(necesidades_cambios_plan_inicial_proyecto).
observable(kpa_aseguramiento_calidad).
observable(aplicar_mejores_practicas).

executable(liberado_informe_inicial_requisitos).
executable(liberado_informe_requisitos_mejorado).
executable(liberado_plan_inicial_proyecto).
executable(liberado_plan_proyecto_mejorado).
executable(aplicadas_mejores_practicas).

% Integrity constraints.
```

% KPA GESTION DE REQUISITOS

```
si kpa_gestion_requisitos, necesidades_cliente entonces
liberado_informe_inicial_requisitos.
si kpa_gestion_requisitos, necesidades_cambios_en_informacion entonces
liberado_informe_requisitos_mejorado.
si kpa_gestion_requisitos, not(necesidades_cambios_en_informacion)
entonces liberado_informe_requisitos_mejorado.
```

% KPA PLANIFICACION DEL PROYECTO DE SOFTWARE

```
si kpa_planificacion_proyectos, necesidades_cliente, planes_iniciales
entonces liberado_plan_inicial_proyecto.
si kpa_planificacion_proyectos, necesidades_cambios_plan_inicial_proyecto
entonces liberado_plan_proyecto_mejorado.
si kpa_planificacion_proyectos,
not(necesidades_cambios_plan_inicial_proyecto) entonces
liberado_plan_proyecto_mejorado.
```

% KPA ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE

```
si kpa_aseguramiento_calidad, aplicar_mejores_practicas entonces
aplicadas_mejores_practicas.

% Escenario

observe liberado_informe_inicial_requisitos.
```

Anexo 2

```
% ------------------------------------------------------ For Diseñador
observable(kpa_planificacion_proyectos).
observable(encuesta_al_diseñador).
observable(kpa_seguimiento_proyecto).
observable(liberado_informe_requisitos_mejorado).
observable(liberado_plan_proyecto_mejorado).
observable(kpa_aseguramiento_calidad).
observable(aplicar_mejores_practicas).
observabel(medidas_correctivas).
observable(errores_version_final_del_producto).

executable(resultados_encuesta_al_diseñador).
executable(especificaciones_del_sistema).
executable(aplicadas_mejores_practicas).
executable(especificaciones_del_sistema_mejoradas).

% Integrity constraints.
```

% KPA PLANIFICACION DEL PROYECTO DE SOFTWARE

```
si kpa_planificacion_proyectos, encuesta_al_diseñador entonces
resultados_encuesta_al_diseñador.
```

% KPA SEGUIMIENTO DEL PROYECTO DE SOFTWARE

```
si kpa_seguimiento_proyecto, liberado_informe_requisitos_mejorado,
liberado_plan_proyecto_mejorado entonces especificaciones_del_sistema.
si kpa_seguimiento_proyecto, medidas_correctivas entonces
especificaciones_del_sistema_mejoradas.
```

% KPA ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE

```
si kpa_aseguramiento_calidad, aplicar_mejores_practicas entonces
aplicadas_mejores_practicas.
```

% RELACIONES ENTRE LOS ACTORES DISEÑADOR, PROGRAMADOR Y DEPURADOR

```
si errores_version_final_del_producto entonces
especificaciones_del_sistema_mejoradas.

% Definitions

% Escenario
observe especificaciones_del_sistema.
observe aplicar_mejores_practicas.
observe aplicadas_mejores_practicas.
```

Anexo 3

```
% ------------------------------------------------------ For Programador
observable(kpa_planificacion_proyectos).
observable(encuesta_al_programador).
observable(kpa_seguimiento_proyecto).
observable(especificaciones_del_sistema_mejoradas).
observable(liberado_plan_proyecto_mejorado).
observable(kpa_gestion_subcontratacion_software).
observable(recurso_contratado).
observable(kpa_aseguramiento_calidad).
observable(aplicar_mejores_practicas).
observable(kpa_gestion_configuracion_software).
observable(elementos_de_configuracion).
observable(actividades_gestion_configuracion).
observable(kpa_peer_review).
observable(errores_detectados_en_modulos).
observable(propuestas_cambios_a_documentacion_modulos).
observable(errores_version_final_del_producto).

executable(resultados_encuesta_al_programador).
executable(modulos).
executable(documentacion_modulos).
executable(aplicadas_mejores_practicas).
executable(especificacion_de_los_modulos).
executable(modulos_mejorados).
executable(documentacion_modulos_mejorada).
executable(especificacion_recurso_a_contratar).
executable(modulos_definitivos).
executable(partes_integradas).

% Integrity constraints.
```

% KPA PLANIFICACION DEL PROYECTO DE SOFTWARE

```
si kpa_planificacion_proyectos, encuesta_al_programador entonces
resultados_encuesta_al_programdor.
```

% KPA SEGUIMIENTO DEL PROYTECTO DE SOFTWARE

```
si kpa_seguimiento_proyecto, especificaciones_del_sistema,
liberado_plan_proyecto_mejorado entonces modulos, documentacion_modulos.
si kpa_seguimiento_proyecto, medidas_correctivas entonces
modulos_mejorados, documentacion_modulos_mejorada.
```

% KPA GESTION DE SUBCONTRATACION DEL SOFTWARE

```
si kpa_gestion_subcontratacion_software, especificaciones_del_sistema,
liberado_plan_proyecto_mejorado entonces
especificacion_recurso_a_contratar.
si kpa_gestion_subcontratacion_software, recurso_contratado entonces
modulos, documentacion_modulos.
si kpa_gestion_subcontratacion_software, not(recurso_contratado) entonces
modulos, documentacion_modulos.
```

% KPA ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE

```
si kpa_aseguramiento_calidad, aplicar_mejores_practicas entonces
aplicadas_mejores_practicas.
```

% KPA GESTION DE CONFIGURACION DEL SOFTWARE

```
si kpa_gestion_configuracion_software, especificaciones_del_sistema,
liberado_plan_proyecto_mejorado entonces especificacion_de_los_modulos.
si kpa_gestion_configuracion_software, elementos_de_configuracion,
actividades_gestion_configuracion entonces modulos,
documentacion_modulos.
```

% KPA PEER REVIEW

```
si kpa_peer_review, especificaciones_del_sistema,
liberado_plan_proyecto_mejorado entonces modulos_definitivos.
si kpa_peer_review, errores_detectados_en_modulos,
propuestas_cambios_a_documentacion_modulos entonces modulos_definitivos,
documentacion_modulos.
```

% RELACIONES ENTRE LOS ACTORES DEPURADOR, DISEÑADOR Y PROGRAMADOR

```
si errores_version_final_del_producto entonces modulos_definitivos,
documentacion_modulos.

% Definitions

% Escenario

observe modulos.
observe modulos_definitivos.
observe documentacion_modulos.
```

Anexo 4

```
% ------------------------------------------------------ For Depurador
observable(kpa_planificacion_proyectos_software).
observable(encuesta_al_depurador).
observable(kpa_seguimiento_proyecto).
observable(modulos_definitivos).
observable(documentacion_modulos).
observable(medidas_correctivas).
observable(kpa_gestion_subcontratacion_software).
observable(kpa_aseguramiento_calidad).
observable(aplicar_mejores_practicas).
observable(kpa_gestion_configuracion_software).
observable(liberado_plan_proyecto_mejorado).
observable(propuestas_version_final_del_producto).
observable(kpa_peer_review).
observable(errores_detectados_en_integracion).
observable(errores_detectados_en_pruebas_iniciales).
observable(recurso_contratado).

executable(resultados_encuesta_al_depurador).
executable(producto_de_software).
executable(producto_de_software_mejorado).
executable(errores_en_producto_software).
executable(aplicadas_mejores_practicas).
executable(integracion).
executable(pruebas_iniciales).
executable(partes_integradas).
executable(errores_version_final_del_producto).
executable(especificacion_recurso_a_contratar).

% Integrity constraints.
```

% KPA PLANIFICACION DEL PROYECTO DE SOFTWARE

```
si kpa_planificacion_proyectos, encuesta_al_depurador entonces
resultados_encuesta_al_depurador.
```

% KPA SEGUIMIENTO DEL PROYECTO DE SOFTWARE

```
si kpa_seguimiento_proyecto, modulos_definitivos, documentacion_modulos
entonces producto_de_software.
si kpa_seguimiento_proyecto, medidas_correctivas entonces
producto_de_software_mejorado.
```

% KPA GESTION DE SUBCONTRATACION DEL SOFTWARE

```
si kpa_gestion_subcontratacion_software, modulos_definitivos,
documentacion_modulos entonces especificacion_recurso_a_contratar.
si kpa_gestion_subcontratacion_software, recurso_contratado entonces
producto_de_software, errores_en_producto_software.
```

si kpa_gestion_subcontratacion_software, not(recurso_contratado) entonces producto_de_software, errores_en_producto_software.

% KPA ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE

```
si kpa_aseguramiento_calidad, aplicar_mejores_practicas entonces
aplicadas_mejores_practicas.
```

% KPA GESTION DE CONFIGURACION DEL SOFTWARE

```
si kpa_gestion_configuracion_software, modulos_definitivos,
documentacion_modulos, liberado_plan_proyecto_mejorado entonces
integracion_pruebas.
si kpa_gestion_configuracion_software,
propuestas_version_final_del_producto entonces
producto_de_software_mejorado.
si kpa_gestion_configuracion_software,
not(propuestas_version_final_del_producto) entonces
producto_de_software_mejorado.
```

% KPA PEER REVIEW

```
si kpa_peer_review, modulos_definitvos, documentacion_modulos entonces
partes_integradas.
si kpa_peer_review, errores_detectados entonces
producto_de_software_mejorado.
```

% RELACIONES ENTRE LOS ACTORES DISEÑADOR, PROGRAMADOR Y DEPURADOR

```
si modulos_definitivos, documentacion_modulos entonces
errores_version_final_del_producto.

% Definitions

para integracion_pruebas haga
     integracion,
     pruebas_iniciales.

para errores_detectados haga
     errores_detectados_en_integracion,
     errores_detectados_en_pruebas_iniciales.

% Escenario
observe producto_de_software_mejorado.
```

Anexo 5

```
% ------------------------------------For Mantenimiento y Actualizacion
observable(kpa_planificacion_proyectos).
observable(encuesta_a_mantenimiento_y_actualizacion).
observable(kpa_seguimiento_proyecto).
observable(producto_de_software_mejorado).
observable(medidas_correctivas).
observable(kpa_aseguramiento_calidad).
observable(aplicar_mejores_practicas).
observable(te_solicitan_apoyo_tecnico).
observable(sugerencias_de_actualizacion).
observable(errores_posibles).

executable(resultados_encuesta_a_mantenimiento_y_actualizacion).
executable(actualizaciones_posibles).
executable(apliacadas_mejores_practicas).
executable(version_final_del_producto).
executable(apoyo_tecnico).
executable(actualizaciones_hechas).

% Integrity constraints.
```

% KPA PLANIFICACION DEL PROYECTO DE SOFTWARE

```
si kpa_planificacion_proyectos, encuesta_a_mantenimiento_y_actualizacion
entonces resultados_encuesta_a_mantenimiento_y_actualizacion.
```

% KPA SEGUIMIENTO DEL PROYECTO DE SOFTWARE

```
si kpa_seguimiento_proyecto, producto_de_software_mejorado entonces
actualizaciones_posibles.
si kpa_seguimiento_proyecto, medidas_correctivas entonces
actualizaciones_hechas.
```

% KPA ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE

```
si kpa_aseguramiento_calidad, aplicar_mejores_practicas entonces
aplicadas_mejores_practicas.
```

% RELACIONES CON LOS ACTORES INGENIERO DE SOFTWARE Y USUARIO

```
si producto_de_software_mejorado entonces version_final_del_producto.
si te_solicitan_apoyo_tecnico entonces apoyo_tecnico.
si sugerencias_de_actualizacion, errores_posibles entonces
actualizaciones_hechas.
si not(sugerencias_de_actualizacion), not(errores_posibles) entonces
producto_de_software_mejorado.

% Definitions

% Escenario
```

```
observe actualizaciones_hechas.
observe producto_de_software_mejorado.
```

Anexo 6

```
% ------------------------------------------------- For Grupo de Mejora
observable(kpa_gestion_requisitos).
observable(liberado_informe_inicial_requisitos).
observable(establecido_compromiso_realizacion).
observable(kpa_planificacion_proyectos).
observable(liberado_plan_inicial_proyecto).
observable(kpa_seguimiento_proyecto).
observable(liberado_plan_proyecto_mejorado).
observable(kpa_gestion_subcontratacion_software).
observable(especificacion_recurso_a_contratar).
observable(kpa_aseguramiento_calidad).
observable(definidas_politicas_calidad).
observable(kpa_gestion_configuracion_software).
observable(especificacion_de_modulos).
observable(integracion).
observable(pruebas_iniciales).
observable(kpa_peer_review).
observable(modulos_hechos).
observable(partes_integradas).
observable(soy_peer).

executable(solicitar_a_direccion_compromiso_realizacion).
executable(definir_objetivos_mejora).
executable(definir_compromisos_mejora).
executable(definir_capacidad_mejora).


% Integrity constraints.
```

% KPA GESTION DE REQUISITOS

```
si kpa_gestion_requisitos, liberado_informe_inicial_requisitos entonces
solicitar_a_direccion_compromiso_realizacion.
si kpa_gestion_requisitos, establecido_compromiso_realizacion entonces
definir_plan_aseguramiento_mejora.
```

% KPA PLANIFICACION DEL PROYECTO DE SOFTWARE

```
si kpa_planificacion_proyectos, liberado_plan_inicial_proyecto entonces
solicitar_a_direccion_compromiso_realizacion.
si kpa_planificacion_proyectos, establecido_compromiso_realizacion
entonces definir_plan_aseguramiento_mejora.
```

% KPA SEGUIMIENTO DEL PROYECTO DE SOFTWARE

```
si kpa_seguimiento_proyecto, liberado_plan_proyecto_mejorado entonces
solicitar_a_direccion_compromiso_realizacion.
si kpa_seguimiento_proyecto, establecido_compromiso_realizacion entonces
definir_plan_aseguramiento_mejora.
```

% KPA GESTION DE SUBCONTRATACION DEL SOFTWARE

```
si kpa_gestion_subcontratacion_software,
especificacion_recurso_a_contratar entonces
solicitar_a_direccion_compromiso_realizacion.
si kpa_gestion_subcontratacion_software,
establecido_compromiso_realizacion entonces
definir_plan_aseguramiento_mejora.
```

% KPA ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE

```
si kpa_aseguramiento_calidad, definidas_politicas_calidad entonces
solicitar_a_direccion_compromiso_realizacion.
si kpa_aseguramiento_calidad, establecido_compromiso_realizacion entonces
definir_plan_aseguramiento_mejora.
```

% KPA GESTION DE CONFIGURACION DEL SOFTWARE

```
si kpa_gestion_configuracion_software, especificacion_de_modulos entonces
solicitar_a_direccion_compromiso_realizacion.
si kpa_gestion_configuracion_software, integracion, pruebas_iniciales
entonces solicitar_a_direccion_compromiso_realizacion.
si kpa_gestion_configuracion_software, establecido_compromiso_realizacion
entonces definir_plan_aseguramiento_mejora.
```

% KPA PEER REVIEW

```
si kpa_peer_review, modulos_hechos entonces
solicitar_a_direccion_compromiso_realizacion.
si kpa_peer_review, partes_integradas entonces
solicitar_a_direccion_compromiso_realizacion.
si kpa_peer_review, soy_peer, establecido_compromiso_realizacion entonces
definir_plan_aseguramiento_mejora.


% Definitions

para definir_plan_aseguramiento_mejora haga
          definir_objetivos_mejora,
          definir_compromisos_mejora,
          definir_capacidad_mejora.

% Escenario

observe solicitar_a_direccion_compromiso_realizacion.
observe establecido_compromiso_realizacion.
observe definir_plan_aseguramiento_mejora.
observe kpa_gestion_requisitos.
```

Anexo 7

```
% ------------------------------------------------ For Grupo de Solucion
observable(kpa_gestion_requisitos).
observable(definir_objetivos_mejora).
observable(definir_compromisos_mejora).
observable(definir_capacidad_mejora).
observable(resultados_encuesta_al_cliente).
observable(kpa_planificacion_proyectos).
observable(resultados_encuesta_al_diseñador).
observable(resultados_encuesta_al_programador).
observable(resultados_encuesta_al_depurador).
observable(kpa_seguimiento_proyecto).
observable(establecido_compromiso_interno).
observable(establecido_compromiso_externo).
observable(kpa_gestion_subcontratacion_software).
observable(proveedor_contratado).
observable(kpa_aseguramiento_calidad).
observable(kpa_gestion_configuracion_software).
observable(kpa_peer_review).
observable(resultados_encuesta_al_usuario).

executable(encuesta_al_cliente).
executable(medicion).
executable(analisis).
executable(encuesta_al_diseñador).
executable(encuesta_al_programador).
executable(encuesta_al_depurador).
executable(solicitar_a_direccion_reestablecer_compromiso_interno).
executable(solicitar_al_cliente_reestablecer_compromiso_externo).
executable(contratar_proveedor).
executable(encuesta_al_usuario).
executable(recurso_contratado).

% Integrity constraints.
```

% KPA GESTION DE REQUISITOS

```
si kpa_gestion_requisitos, definir_plan_aseguramiento_mejora entonces
encuesta_al_cliente.
si kpa_gestion_requisitos, resultados_encuesta_al_cliente entonces
medicion_analisis.
```

% KPA PLANIFICACION DEL PROYECTO DE SOFTWARE

```
si kpa_planificacion_proyectos, definir_plan_aseguramiento_mejora
entonces encuestas.
si kpa_planificacion_proyectos, resultados_encuestas entonces
medicion_analisis.
```

% KPA SEGUIMIENTO DEL PROYECTO DE SOFTWARE

```
si kpa_seguimiento_proyecto, definir_plan_aseguramiento_mejora entonces
solicitar_reestablecer_compromiso.
```

```
si kpa_seguimiento_proyecto, establecido_compromiso entonces
medicion_analisis.
```

% KPA GESTION DE SUBCONTRATACION DEL SOFTWARE

```
si kpa_gestion_subcontratacion_software,
definir_plan_aseguramiento_mejora entonces contratar_proveedor.
si kpa_gestion_subcontratacion_software, proveedor_contratado entonces
recurso_contratado.
```

% KPA ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE

```
si kpa_aseguramiento_calidad, definir_plan_aseguramiento_mejora entonces
medicion_analisis.
```

% KPA GESTION DE CONFIGURACION DEL SOFTWARE

```
si kpa_gestion_configuracion_software, definir_plan_aseguramiento_mejora
entonces medicion_analisis.
```

% KPA PEER REVIEW

```
si kpa_peer_review, definir_plan_aseguramiento_mejora entonces
encuesta_al_usuario.
si kpa_peer_review, resultados_encuesta_al_usuario entonces
medicion_analisis.

% Definitions

para medicion_analisis haga
    medicion,
    analisis.

para definir_plan_aseguramiento_mejora haga
     definir_objetivos_mejora,
     definir_compromisos_mejora,
     definir_capacidad_mejora.

para solicitar_reestablecer_compromiso haga
     solicitar_a_direccion_reestablecer_compromiso_interno,
     solicitar_al_cliente_reestablecer_compromiso_externo.

para establecido_compromiso haga
     establecido_compromiso_interno,
     establecido_compromiso_externo.

para encuestas haga
     encuesta_al_cliente,
     encuesta_al_diseñador,
     encuesta_al_programador,
     encuesta_al_depurador.

para resultados_encuestas haga
     resultados_encuesta_al_cliente,
```

```
     resultados_encuesta_al_diseñador,
     resultados_encuesta_al_programador,
     resultados_encuesta_al_depurador.

% Escenario

observe medicion_analisis.
```

Anexo 8

```
% ---------------------------------------------- For Gestor de la Mejora
observable(kpa_gestion_requisitos).
observable(medicion).
observable(analisis).
observable(kpa_planificacion_proyectos).
observable(kpa_seguimiento_proyecto).
observable(kpa_gestion_subcontratacion_software).
observable(kpa_aseguramiento_calidad).
observable(kpa_gestion_configuracion_software).
observable(kpa_peer_review).
observable(medicion_analisis_al_programador).
observable(medicion_analisis_al_depurador).

executable(necesidades_cambios).
executable(medidas_correctivas).
executable(resultados_evaluacion_desempeño).
executable(mejores_practicas).
executable(elementos_de_configuracion).
executable(actividades_gestion_configuracion).
executable(propuestas_version_final_del_producto).
executable(errores_detectados_en_modulos).
executable(propuestas_cambios_a_documentacion_modulos).
executable(errores_detectados_en_integracion).
executable(errores_detectados_en_pruebas_iniciales).

% Integrity constraints.
```

% KPA GESTION DE REQUISITOS

```
si kpa_gestion_requisitos, medicion_analisis entonces
necesidades_cambios.
```

% KPA PLANIFICACION DEL PROYECTO DE SOFTWARE

```
si kpa_planificacion_proyectos, medicion_analisis entonces
necesidades_cambios.
```

% KPA SEGUIMIENTO DEL PROYECTO DE SOFTWARE

```
si kpa_seguimiento_proyecto, medicion_analisis entonces
medidas_correctivas.
```

% KPA GESTION DE SUBCONTRATACION DEL SOFTWARE

```
si kpa_gestion_subcontratacion_software, medicion_analisis entonces
resultados_evaluacion_desempeño.
```

% KPA ASEGURAMEIENTO DE LA CALIDAD DEL SOFTWARE

```
si kpa_aseguramiento_calidad, medicion_analisis entonces
mejores_practicas.
```

% KPA GESTION DE CONFIGURACION DEL SOFTWARE

```
si kpa_gestion_configuracion_software, medicion_analisis_al_programador
entonces elementos_configuracion, actividades_gestion_configuracion.
si kpa_gestion_configuracion_software, medicion_analisis_al_depurador
entonces propuestas_version_final_del_producto.
```

% KPA PEER REVIEW

```
si kpa_peer_review, medicion_analisis_al_programador entonces
errores_propuestas.
si kpa_peer_review, medicion_analisis_al_depurador entonces
errores_detectados.

% Definitions

para medicion_analisis haga
     medicion,
     analisis.

para errores_propuestas haga
     errores_detectados_en_modulos,
     propuestas_cambios_a_documentacion_modulos.

para errores_detectados haga
     errores_detectados_en_integracion,
     errores_detectados_en_pruebas_iniciales.

% Escenario

observe medicion_analisis.
observe medidas_correctivas.
observe mejores_practicas.
```

Anexo 9

```
% ------------------------------------------------------ For Cliente
observable(kpa_gestion_requisitos).
observable(encuesta_al_cliente).
observable(kpa_planificacion_proyectos).
observable(kpa_seguimiento_proyecto).
observable(solicitar_al_cliente_reestablecer_compromiso_externo).
observable(motivacion).
observable(liberado_proyecto_preliminar).
observable(necesidades_cliente).
observable(asesoramiento_al_cliente).
observable(necesidades_cambios_al_proyecto_preliminar).

executable(visto_bueno_al_proyecto_preliminar).
executable(cambios_posibles_al_proyecto_preliminar).
executable(resultados_encuesta_al_cliente).
executable(establecido_compromiso_externo).
executable(resultados_del_asesoramiento_al_cliente).
executable(liberadas_necesidades_cliente).

% Integrity constraints.

si necesidades_cliente entonces liberadas_necesidades_cliente.
```

% KPA GESTION DE REQUISITOS

```
si kpa_gestion_requisitos, encuesta_al_cliente entonces
resultados_encuesta_al_cliente.
```

% KPA PLANIFICACION DEL PROYECTO DE SOFTWARE

```
si kpa_planificacion_proyectos, encuesta_al_cliente entonces
resultados_encuesta_al_cliente.
```

% KPA SEGUIMIENTO DEL PROYECTO DE SOFTWARE

```
si kpa_seguimiento_proyecto,
solicitar_al_cliente_reestablecer_compromiso_externo entonces
establecido_compromiso_externo.
```

% RELACIONES CON LOS DEMAS ACTORES DE GESTION

```
si motivacion, liberado_proyecto_preliminar,
not(necesidades_cambios_al_proyecto_preliminar) entonces
visto_bueno_al_proyecto_preliminar.
si motivacion, liberado-proyecto_preliminar,
necesidades_cambios_al_proyecto_preliminar entonces
cambios_posibles_al_proyecto_preliminar.
si asesoramiento_al_cliente entonces
resultados_del_asesoramiento_al_cliente.

% Definitions
```

```
% Escenario

observe encuesta_al_cliente.
observe resultados_encuesta_al_cliente.
observe establecido_compromiso_externo.
```

Anexo 10

```
% ------------------------------------------------------ For Direccion
observable(kpa_gestion_requisitos).
observable(solicitar_a_direccion_compromiso_realizacion).
observable(kpa_planificacion_proyectos).
observable(kpa_seguimiento_proyecto).
observable(solicitar_a_direccion_reestablecer_compromiso_interno).
observable(kpa_gestion_subcontratacion_software).
observable(kpa_aseguramiento_calidad).
observable(kpa_gestion_configuracion_software).
observable(kpa_peer_review).
observable(liberado_proyecto_inicial).
observable(necesidades_cambios_al_proyecto_inicial).

executable(establecido_compromiso_realizacion).
executable(establecido_compromiso_interno).
executable(liberadas_politicas_gestion_proyectos).
executable(cambios_posibles_al_proyecto_inicial).
executable(liberadas_politicas_evaluacion_proyectos).
executable(visto_bueno).

% Integrity constraints.
```

% KPA GESTION DE REQUISITOS

```
si kpa_gestion_requisitos, solicitar_a_direccion_compromiso_realizacion
entonces establecido_compromiso_realizacion.
```

% KPA PLANIFICACION DEL PROYECTO DE SOFTWARE

```
si kpa_planificacion_proyectos,
solicitar_a_direccion_compromiso_realizacion entonces
establecido_compromiso_realizacion.
```

% KPA SEGUIMIENTO DEL PROYECTO DE SOFTWARE

```
si kpa_seguimiento_proyecto, solicitar_a_direccion_compromiso_realizacion
entonces establecido_compromiso_realizacion.
si kpa_seguimiento_proyecto,
solicitar_a_direccion_reestablecer_compromiso_interno entonces
establecido_compromiso_interno.
```

% KPA GESTION DE SUBCONTRATACION DEL SOFTWARE

```
si kpa_gestion_subcontratacion_software,
solicitar_a_direccion_compromiso_realizacion entonces
establecido_compromiso_realizacion.
```

% KPA ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE

```
si kpa_aseguramiento_calidad,
solicitar_a_direccion_compromiso_realizacion entonces
establecido_compromiso_realizacion.
```

% KPA GESTION DE CONFIGURACION DEL SOFTWARE

```
si kpa_gestion_configuracion_software,
solicitar_a_direccion_compromiso_realizacion entonces
establecido_compromiso_realizacion.
```

% KPA PEER REVIEW

```
si kpa_peer_review, solicitar_a_direccion_compromiso_realizacion entonces
establecido_compromiso_realizacion.
```

% OTRAS RELACIONES

```
si liberado_proyecto_inicial entonces liberadas_politicas.
si necesidades_cambios_al_proyecto_inicial entonces
cambios_posibles_al_proyecto_inicial.
si not(necesidades_cambios_al_proyecto_inicial) entonces visto_bueno.

% Definitions

para liberadas_politicas haga
     liberadas_politicas_gestion_proyectos,
     liberadas_politicas_evaluacion_proyectos.

% Escenario

observe solicitar_a_direccion_compromiso_realizacion.
observe establecido_compromiso_realizacion.
```

Anexo 11

```
% ----------------------------------------- For Evaluador de Proyectos
observable(liberadas_politicas_evaluacion_proyectos).
observable(liberado_proyecto_a_ejecutar).
observable(resultados_asesoramiento_al_cliente).

executable(asesoramiento_al_cliente).
executable(planes_redefinidos).
executable(resultados_evaluacion_proyecto).

% Integrity constraints.

si liberadas_politicas_evaluacion_proyectos, liberado_proyecto_a_ejecutar
entonces asesoramiento_al_cliente.
si resultados_asesoramiento_al_cliente entonces planes_redefinidos,
resultados_evaluacion_proyecto.

% Definitions

% Escenario

observe liberadas_politicas_evaluacion_proyectos.
observe liberado_proyecto_a_ejecutar.
observe asesoramiento_al_cliente.
```

Anexo 12

```
% ---------------------------------------------- For Gestor de Proyectos
observable(necesidades_cliente).
observable(liberadas_politicas_gestion_proyectos).
observable(visto_bueno_al_proyecto_preliminar_por_parte_del_cliente).
observable(necesidades_cambios_al_proyecto_preliminar).
observable(visto_bueno_al_proyecto_preliminar_por_parte_de_direccion).
observable(cambios_al_proyecto_preliminar_por_parte_de_direccion).

executable(liberado_proyecto_preliminar).
executable(liberado_proyecto_a_ejecutar).
executable(cambios_posibles_al_proyecto_preliminar).

% Integrity constraints.

si liberadas_politicas_gestion_proyectos, necesidades_cliente entonces
liberado_proyecto_preliminar.
si visto_bueno, not(necesidades_cambios_al_proyecto_preliminar) entonces
liberado_proyecto_a_ejecutar.
si necesidades_cambios_al_proyecto_preliminar entonces
cambios_posibles_al_proyecto_preliminar, liberado_proyecto_a_ejecutar.

% Definitions

para visto_bueno haga
    visto_bueno_al_proyecto_preliminar_por_parte_del_cliente,
    visto_bueno_al_proyecto_preliminar_por_parte_de_direccion.

% Escenario

observe poryecto_a_ejecutar.
```

Anexo 13

```
% ------------------------------------------------------- For Usuario
observable(kpa_peer_review).
observable(encuesta_al_usuario).
observable(version_final_del_producto).
observable(problemas_de_operacion).
observable(problemas_de_manejo).
observable(recibido_apoyo_tecnico).

executable(resultados_encuesta_al_usuario).
executable(sugerencias).
executable(actualizaciones).
executable(posibles_errores_en_el_producto).
executable(resueltos_problemas_de_operacion).
executable(resueltos_problemas_de_manejo).
executable(solicitar_apoyo_tecnico).

% Integrity constraints.
```

% KPA PEER REVIEW

```
si kpa_peer_review, encuesta_al_usuario entonces
resultados_encuesta_al_usuario.
```

% OTRAS RELACIONES

```
si version_final_del_producto entonces plan_sugerencias.
si problemas_de_operacion, problemas_de_manejo entonces
solicitar_apoyo_tecnico.
si recibido_apoyo_tecnico entonces resueltos_problemas.

% Definitions

para plan_sugerencias haga
    sugerencias,
    actualizaciones,
    posibles_errores_en_el_producto.

para resueltos_problemas haga
    resueltos_problemas_de_operacion,
    resueltos_problemas_de_manejo.

% Escenario

observe plan_sugerencias.
observe resueltos_problemas.
```

Anexo 14

Código fuente del flujo secuencial de la empresa para desarrollo de software

```
DESARROLLO DE SOTWARE INCORPORANDO GESTION DE CALIDAD


NETWORK

NECESIDADES(I) ::

      IF (PACEPT<=9) THEN
        BEGIN
          IT:=960;
          PACEPT:=PACEPT+1;
        END
      ELSE BEGIN
            IF (PACEPT=10) THEN
              IT:=20000;
          END;
      TIPO:=FTIP; P:=0; KPA:=0; CONT:=1;
      N_PROYECTOS:=0; CICLO:=FALSE; CICLICO:=FALSE;
      WRITE('ENTRA UNO DE TIPO: ');
      WRITE(TIPO); PAUSE;
      WRITE(' EN EL TIEMPO:');
      WRITELN(GT:6:0);
      IF (TIPO=1) THEN
        BEGIN
           AUX1:=AUX1+1;
           N_PROYECTOS:=AUX1;
        END;
      IF (TIPO=2) THEN
        BEGIN
           AUX2:=AUX2+1;

           N_PROYECTOS:=AUX2;
        END;
      IF (TIPO=3) THEN
        BEGIN
           AUX3:=AUX3+1;
           N_PROYECTOS:=AUX3;
        END;

ANALIZAR(R) QMS ::
      RELEASE BEGIN
              CASE CONT OF
```

```
                1: IF (CICLO) AND (KPA=1) THEN
                     CICLO:=NOT CICLO
                   ELSE KPA:=1;
                2: IF (CICLO) AND (KPA=2) THEN
                     CICLO:=NOT CICLO
                   ELSE KPA:=2;
              END;
              SENDTO(QMS);
            END;
        P:=1;
        IF (TIPO=1) THEN STAY:= GAUSS(114,2.14);
        IF (TIPO=2) THEN STAY:= GAUSS(262,4.91);
        IF (TIPO=3) THEN STAY:= GAUSS(720,13.5);

DISENAR(R) IMP,QMS ::
        RELEASE BEGIN
              CASE CONT OF
                1: IF (CICLO) AND (KPA=4) THEN
                     CICLO:=NOT CICLO
                   ELSE KPA:=4;
              END;
              SENDTO(QMS);
            END;
        P:=2;
        IF (TIPO=1) THEN STAY:= GAUSS(277,5.2);
        IF (TIPO=2) THEN STAY:= GAUSS(604,11.33);
        IF (TIPO=3) THEN STAY:= GAUSS(1560,29.25);

IMP(R) QMS ::
        RELEASE BEGIN
              CASE CONT OF
                1: IF (CICLO) AND (KPA=4) THEN
                     CICLO:=NOT CICLO
                   ELSE KPA:=4;
                2: IF (CICLO) AND (KPA=5) THEN
                     CICLO:=NOT CICLO
                   ELSE KPA:=5;
                3: IF (CICLO) AND (KPA=6) THEN
                     CICLO:=NOT CICLO
                   ELSE KPA:=6;
                4: IF (CICLO) AND (KPA=3) THEN
                     CICLO:=NOT CICLO
                   ELSE KPA:=3;
              END;
              SENDTO(QMS);
            END;
        P:=3;
```

```
        IF (TIPO=1) THEN STAY:= GAUSS(38,0.72);
        IF (TIPO=2) THEN STAY:= GAUSS(101,1.89);
        IF (TIPO=3) THEN STAY:= GAUSS(310,5.81);

INTEGRAR(R) QMS ::
        RELEASE BEGIN
                CASE CONT OF
                  1: IF (CICLO) AND (KPA=4) THEN
                       CICLO:=NOT CICLO
                     ELSE KPA:=4;
                  2: IF (CICLO) AND (KPA=5) THEN
                       CICLO:=NOT CICLO
                     ELSE KPA:=5;
                  3: IF (CICLO) AND (KPA=6) THEN
                       CICLO:=NOT CICLO
                     ELSE KPA:=6;
                  4: IF (CICLO) AND (KPA=3) THEN
                       CICLO:=NOT CICLO
                     ELSE KPA:=3;
                END;
                SENDTO(QMS);
              END;
        P:=4;
        IF (TIPO=1) THEN STAY:= GAUSS(126,2.37);
        IF (TIPO=2) THEN STAY:= GAUSS(277,5.2);
        IF (TIPO=3) THEN STAY:= GAUSS(750,14.1);

QMS(R) ANALIZAR, DISENAR, IMP, INTEGRAR, MANTENER::
        RELEASE BEGIN
                IF BER(0.2) THEN
                  BEGIN
                    CICLO:=NOT CICLO;
                    CICLICO:=TRUE;
                  END;
                IF (NOT CICLO) THEN
                  CONT:=CONT+1;
                CASE P OF
                  1: IF (KPA=2) AND (NOT CICLO) THEN
                       BEGIN
                         KPA:=0; CONT:=1;
                         SENDTO(DISENAR);
                       END
                     ELSE SENDTO(ANALIZAR);
                  2: IF (KPA=4) AND (NOT CICLO) THEN
                       BEGIN
                         KPA:=0; CONT:=1;
                         SENDTO(IMP);
```

```
                    END
                  ELSE SENDTO(DISENAR);
                3: IF (KPA=3) AND (NOT CICLO) THEN
                    BEGIN
                      KPA:=0; CONT:=1;
                      SENDTO(INTEGRAR);
                    END
                  ELSE SENDTO(IMP);
                4: IF (KPA=3) AND (NOT CICLO) THEN
                    BEGIN
                      KPA:=0; CONT:=1;
                      SENDTO(MANTENER);
                    END
                  ELSE SENDTO(INTEGRAR);
              END;
            END;
        IF (TIPO=1) THEN
          CASE KPA OF
            1: STAY:= GAUSS(50,3.75);
            2: STAY:= GAUSS(75,5.63);
            3: STAY:= GAUSS(75,5.63);
            4: STAY:= GAUSS(50,3.75);
            5: STAY:= GAUSS(25,1.88);
            6: STAY:= GAUSS(50,3.75);
          END;
        IF (TIPO=2) THEN
          CASE KPA OF
            1: STAY:= GAUSS(80,6);
            2: STAY:= GAUSS(100,7.5);
            3: STAY:= GAUSS(100,7.5);
            4: STAY:= GAUSS(80,6);
            5: STAY:= GAUSS(40,3);
            6: STAY:= GAUSS(80,6);
          END;
        IF (TIPO=3) THEN
          CASE KPA OF
            1: STAY:= GAUSS(120,9);
            2: STAY:= GAUSS(200,15);
            3: STAY:= GAUSS(200,15);
            4: STAY:= GAUSS(120,9);
            5: STAY:= GAUSS(80,6);
            6: STAY:= GAUSS(120,9);
          END;

MANTENER(E) ::
    WRITELN('FIN DE EJECUCION DEL PROYECTO TIPO: ', TIPO);
    WRITELN('EN EL TIEMPO: ',TIME:6:0);
```

```
    WRITELN('NUMERO_PROYECTO DE ESTE TIPO: ',N_PROYECTOS);
    IF CICLICO THEN WRITE('PROYECTO CICLICO');
    WRITELN; PAUSE;

INIT
        TSIM:=17600;
        FTIP:=1, 45/2, 40/3, 15;
        AUX1:=0;
        AUX2:=0;
        AUX3:=0;
        PACEPT:=1;
        QMS:=3;
        ANALIZAR:=1;
        DISENAR:=2;
        IMP:=2;
        INTEGRAR:=2;
        ACT(NECESIDADES,0.0);
DECL

        MESSAGES NECESIDADES(P,KPA,CONT,TIPO,N_PROYECTOS:INTEGER;
                        CICLO,CICLICO:BOOLEAN);
        GFUNCTIONS FTIP FREQ(INTEGER):REAL:3;
        VAR AUX1,AUX2,AUX3,PACEPT:INTEGER;
        STATISTICS ALLNODES;

END.
```

Anexo 15

```
% ------------------------------------------------------ For Planificacion
observable(necesidades_del_mercado).
observable(apertura_de_inscripciones).
observable(finalizar_curso).
observable(resultados_encuesta_al_cliente).
observable(resultados_encuesta_al_instructor).
observable(proporcionan_soluciones_problemas_con_material_apoyo).
observable(proporcionan_soluciones_problemas_con_programacion_y_contenido
).
observable(proporcionan_soluciones_problemas_con_infraestructura).
observable(aplicar_medidas_correctivas).

executable(planificar_curso).
executable(promocionar_curso).
executable(encuesta_al_cliente).
executable(encuesta_al_instructor).
executable(enviar_al_COE_resultados_encuestas).
executable(enviar_certificados).
executable(ejecutar_soluciones_al_alcance).
executable(aplicadas_medidas_correctivas).
executable(soluciones_con_material_apoyo).
executable(soluciones_con_programacion_y_contenido).
executable(soluciones_con_infraestructura).

% Integrity constraints.

si necesidades_del_mercado entonces planificar_curso.
si apertura_de_inscripciones entonces promocionar_curso.
si finalizar_curso entonces encuestas.
si   resultados_encuestas   entonces   enviar_al_COE_resultados_encuestas,
enviar_certificados.
si               proporcionan_soluciones_problemas               entonces
ejecutar_soluciones_al_alcance.
si aplicar_medidas_correctivas entonces aplicadas_medidas_correctivas.

% Definitions

para encuestas haga
     encuesta_al_cliente,
     encuesta_al_instructor.

para resultados_encuestas haga
     resultados_encuesta_al_cliente,
     resultados_encuesta_al_instructor.

para proporcionan_soluciones_problemas haga
     proporcionan_soluciones_problemas_con_material_apoyo,
     proporcionan_soluciones_problemas_con_programacion_y_contenido,
     proporcionan_soluciones_problemas_con_infraestructura.

para ejecutar_soluciones_al_alcance haga
     soluciones_con_material_apoyo,
     soluciones_con_programacion_y_contenido,
     soluciones_con_infraestructura.
```

```
% Escenario

observe planificar_curso.
observe resultados_encuestas.
observe proporcionan_soluciones_problemas.
```

Anexo 16

```
% ------------------------------------------------- For Administracion
observable(planificar_curso).
observable(visto_bueno_a_planificacion).
observable(instructores_contactados).
observable(infraestructura_disponible).
observable(inscribirse_en_curso).
observable(numero_de_inscritos_menor_maximo).
observable(numero_inscritos_igual_a_maximo).
observable(cobro).
observable(instructor_seleccionado).
observable(instructor_contratado).
observable(finalizar_curso).
observable(enviar_certificados).
observable(nota_definitiva_mayor_o_igual_que_nota_aprobatoria).
observable(aplicar_medidas_correctivas).

executable(aumenta_numero_inscritos ).
executable(contactar_instructores).
executable(buscar_infraestructura_disponible).
executable(apertura_de_inscripciones).
executable(enviar_al_COE_expedientes_instructores_contactados).
executable(crear_expediente).
executable(contratar_instructor).
executable(iniciar_curso).
executable(otorgar_certificados).
executable(actualizar_expediente).
executable(aplicadas_medidas_correctivas).

% Integrity constraints.

si planificar_curso, visto_bueno_a_planificacion entonces
contactar_instructores, buscar_infraestructura_disponible.
si instructores_contactados, infraestructura_disponible entonces
apertura_de_inscripciones,
enviar_al_cOE_expedientes_instructores_contactados.
si inscribirse_en_curso, numero_inscritos_menor_maximo, cobro entonces
aumenta_numero_inscritos, crear_expediente.
si numero_inscritos_igual_a_maximo, instructor_seleccionado entonces
contratar_instructor.
si instructor_contratado entonces iniciar_curso.
si finalizar_curso, enviar_certificados,
nota_definitiva_mayor_o_igual_que_nota_aprobatoria entonces
otorgar_certificados, actualizar_expediente.
si not(nota_definitiva_mayor_o_igual_que_nota_aprobatoria) entonces
actualizar_expediente.
si aplicar_medidas_correctivas entonces aplicadas_medidas_correctivas.

% Definitions

% Escenario

observe nota_aprobatoria_mayor_o_igual_que_nota_aprobatoria.
observe finalizar_curso.
observe actualizar_expediente.
```

Anexo 17

```
% ------------------------------------------------------ For Instruccion
observable(contactar_instructores).
observable(contratar_instructor).
observable(curso_preparado).
observable(iniciar_curso).
observable(dictando_curso).
observable(tiempo_de_evaluacion).
observable(tiempo_duracion_del_curso).
observable(encuesta_al_instructor).
observable(proporcionan_soluciones_problemas_con_instructor).
observable(aplicar_medidas_correctivas).
observable(curso_finalizado).

executable(instructores_contactados).
executable(preparando_curso).
executable(dictar_curso).
executable(realizar_evaluacion_parcial).
executable(finalizar_curso).
executable(liberada_nota_definitiva).
executable(resultados_encuesta_al_instructor).
executable(ejecutar_soluciones_al_alcance).
executable(aplicadas_medidas_correctivas).
executable(instructor_contratado).

% Integrity constraints.

si contactar_instructores entonces instructores_contactados.
si contratar_instructor entonces instructor_contratado, preparando_curso.
si iniciar_curso entonces dictar_curso.
si curso_preparado, dictando_curso, tiempo_de_evaluacion entonces
realizar_evaluacion_parcial.
si curso_preparado, dictando_curso, tiempo_duracion_del_curso entonces
finalizar_curso.
si curso_finalizado entonces liberada_nota_definitiva.
si encuesta_al_instructor entonces resultados_encuesta_al_instructor.
si proporcionan_soluciones_problemas_con_instructor entonces
ejecutar_soluciones_al_alcance.
si aplicar_medidas_correctivas entonces aplicadas_medidas_correctivas.

% Definitions

% Escenario

observe finalizar_curso.
observe liberada_nota_definitiva.
observe aplicar_medidas_correctivas.
observe dictando_curso.
```

Anexo 18

```
% ------------------------------------------------------- For Cliente
observable(promocionar_curso).
observable(interes_en_curso).
observable(iniciar_curso).
observable(encuesta_al_cliente).
observable(otorgar_certificados).

executable(inscribirse_en_curso).
executable(asistir_a_curso).
executable(resultados_encuesta_al_cliente).
executable(entrenamiento_completado).

% Integrity constraints.

si promocionar_curso, interes_en_curso entonces inscribirse_en_curso.
si iniciar_curso entonces asistir_a_curso.
si encuesta_al_cliente entonces resultados_encuesta_al_cliente.
si otorgar_certificados entonces entrenamiento_completado.

% Definitions

% Escenario

observe inscribirse_en_curso.
observe asistir_a_curso.
observe iniciar_curso.
```

Anexo 19

```
% ------------------------------------------------------ For Direccion
observable(proporcionan_soluciones_problemas_con_instructor).
observable(proporcionan_soluciones_problemas_con_infraestructura).
observable(proporcionan_soluciones_problemas_con_material_apoyo).
observable(proporcionan_soluciones_problemas_con_programacion_y_contenido
).
observable(reporte_a_direccion).
observable(necesidad_de_cambios).

executable(gestionar_cambios_correspondientes).
executable(refinar_politicas).
executable(mismas_politicas).

% Integrity constraints.

si proporcionan_soluciones_problemas_con_instructor entonces
gestionar_cambios_correspondientes.
si proporcionan_soluciones_problemas_con_infraestructura entonces
gestionar_cambios_correspondientes.
si proporcionan_soluciones_problemas_con_material_apoyo entonces
gestionar_cambios_correspondientes.
si proporcionan_soluciones_problemas_con_programacion_y_contenido
entonces gestionar_cambios_correspondientes.
si reporte_a_direccion, necesidad_de_cambios entonces refinar_politicas.
si not(necesidad_de_cambios) entonces mismas_politicas.

% Definitions

% Escenario

observe reporte_a_direccion.
observe refinar_politicas.
observe gestionar_cambios_correspondientes.
observe necesidad_de_cambios.
```

Anexo 20

```
% ------------------------------------------------------- For Control
observable(enviar_al_COE_expedientes_instructores_contactados).
observable(enviar_al_COE_resultados_encuestas).
observable(problemas_con_instructor).
observable(problemas_con_infraestructura).
observable(problemas_con_material_apoyo).
observable(problemas_con_programacion_y_contenido).
observable(necesidades_del_mercado).
observable(planificar_curso).

executable(instructor_seleccionado).
executable(reporta_actividad_a_soporte).
executable(proporcionan_soluciones_problemas_con_instructor).
executable(proporcionan_soluciones_problemas_con_infraestructura).
executable(proporcionan_soluciones_problemas_con_material_apoyo).
executable(proporcionan_soluciones_problemas_con_programacion_y_contenido
).
executable(visto_bueno_a_planificacion).
executable(existe_problemas_con_instructor).
executable(existe_problemas_con_infraestructura).
executable(existe_problemas_con_material_apoyo).
observable(existe_problemas_con_programacion_y_contenido).

% Integrity constraints.

si enviar_al_COE_expedientes_instructores_contactados entonces
instructor_seleccionado, reporta_actividad_a_soporte.
si enviar_al_COE_resultados_encuestaS entonces existe_problemas.
si problemas_con_instructor entonces
proporcionan_soluciones_problemas_con_instructor,
reporta_actividad_a_soporte.
si problemas_con_infraestructura entonces
proporcionan_soluciones_problemas_con_infraestructura,
reporta_actividad_a_soporte.
si problemas_con_material_apoyo entonces
proporcionan_soluciones_problemas_con_material_apoyo,
reporta_actividad_a_soporte.
si problemas_con_programacion_y_contenido entonces
proporcionan_soluciones_problemas_con_programacion_y_contenido,
reporta_actividad_a_soporte.
si necesidades_del_mercado, planificar_curso entonces
visto_bueno_a_planificacion, reporta_actividad_a_soporte.

% Definitions

para existe_problemas haga
     existe_problemas_con_instructor.

para existe_problemas haga
     existe_problemas_con_infraestructura.

para existe_problemas haga
     existe_problemas_con_material_apoyo.
```

```
para existe_problemas haga
     existe_problemas_con_programacion_y_contenido.


% Escenario

observe existe_problemas.
observe reporta_actividad_a_soporte.
```

Anexo 21

```
% ------------------------------------------------------- For Soporte
observable(reporta_actividad_a_soporte).
observable(resultados_medidas_correctivas).

executable(aplicar_medidas_correctivas).
executable(evaluacion_de_medidas_correctivas).

% Integrity constraints.

si reporta_actividad_a_soporte entonces aplicar_medidas_correctivas.
si resultados_medidas_correctivas entonces
evaluacion_de_medidas_correctivas.

% Definitions

% Escenario

observe reporta_actividad_a_soporte.
observe resultados_medidas_correctivas.
observe evaluacion_de_medidas_correctivas.
```

Anexo 22

```
% ------------------------------------------------------ For Lider del COE
observable(aplicar_medidas_correctivas).
observable(evaluacion_de_medidas_correctivas).
observable(reporta_actividad_a_soporte).

executable(reporte_a_direccion).

% Integrity constraints.

si aplicar_medidas_correctivas, evaluacion_de_medidas_correctivas,
reporta_actividad_a_soporte entonces reporte_a_direccion.

% Definitions

% Escenario

observe reporte_a_direccion.
observe reporta_actividad_a_lider.
observe evaluacion_de_medidas_correctivas.
```

Anexo 23

CENTRO DE ENTRENAMIENTO INCORPORANDO GESTION DE CALIDAD

NETWORK

```
CURSOS(I) ::
        IT:=500;
        N:=0; P:=0; INSCRIP:=0; P_INSTRUCTOR:=0; P_INFRA:=0;
        P_MAT_APOYO:=0; P_CONTENIDO:=0; INSCRITOS:=0;
        APROBADOS:=0; T_INICIO:=0.0; CAMBIOS:=FALSE;
        NEC_M_CORRECT:=0; NUMERO:=0; CICLO:=0;

        AUXILIAR:=AUXILIAR+1;
        NUMERO:=AUXILIAR;

PLANIFI(R) COE,ENSAMBLE ::

      RELEASE BEGIN

        IF (NEC_M_CORRECT=1) THEN
          SENDTO(ENSAMBLE)
        ELSE
          SENDTO(COE);
      END;
      IF NEC_M_CORRECT=0 THEN
        BEGIN
          CAMBIOS:=FALSE;
          P:=1;
          STAY:=GAUSS(80,20);
        END;
      IF NEC_M_CORRECT=1 THEN
        STAY:=40;


ADMINIST(R) GEST_RECURSOS,ENSAMBLE,DIRECCION ::

      RELEASE BEGIN

        IF (NEC_M_CORRECT=1) THEN
          SENDTO(ENSAMBLE)
        ELSE
          BEGIN

            IF BER(0.2) THEN
              INSCRIP:=2
```

```
            ELSE INSCRIP:=1;

            IF INSCRIP=1 THEN
              BEGIN
                INSCRITOS:=UNIFI(20,25);
                SENDTO(GEST_RECURSOS);
              END
            ELSE SENDTO(DIRECCION);

          END;

        END;
        IF NEC_M_CORRECT=0 THEN
          BEGIN
            P:=2;
            STAY:=GAUSS(240,40);
          END;

        IF NEC_M_CORRECT=1 THEN
          STAY:=40;


GEST_RECURSOS(R) INSTRUC ::

        RELEASE SENDTO(INSTRUC);
        P:=3;
        STAY:=GAUSS(16,4);


INSTRUC(R) COE,ENSAMBLE ::

        RELEASE BEGIN

          IF (NEC_M_CORRECT=1) THEN
            SENDTO(ENSAMBLE)
          ELSE
            BEGIN
              APROBADOS:=UNIFI(0,INSCRITOS);
              SENDTO(COE);
            END;
        END;
        IF NEC_M_CORRECT=0 THEN
          BEGIN
            P:=4;
            T_INICIO:=TIME;
            STAY:=40;
          END;
```

```
      IF NEC_M_CORRECT=1 THEN
       STAY:=40;

COE(R) PLANIFI,ADMINIST,INSTRUC,DIRECCION ::

      RELEASE BEGIN

        IF P=1 THEN
         BEGIN
          IF BER(0.3) THEN
           BEGIN
            CAMBIOS:=NOT CAMBIOS;
            SENDTO(PLANIFI);
           END;
         END;

        IF (NOT CAMBIOS) AND (P=1) THEN
         SENDTO(ADMINIST);

        IF P=4 THEN
         BEGIN
          IF NEC_M_CORRECT=0 THEN
           BEGIN
            IF BER(0.3) THEN
             P_INSTRUCTOR:=1;
            IF BER(0.2) THEN
             P_INFRA:=1;
            IF BER(0.2) THEN
             P_MAT_APOYO:=1;
            IF BER(0.2) THEN
             P_CONTENIDO:=1;
           END;
          IF NEC_M_CORRECT=1 THEN
           NEC_M_CORRECT:=0;
          IF (CICLO=0) THEN
           BEGIN
            IF BER(0.6) THEN
             BEGIN
              CICLO:=1;
              NEC_M_CORRECT:=1;
              N:=N+1;
              SENDTO(PLANIFI);
              COPYMESS(1) SENDTO(ADMINIST);
              COPYMESS(1) SENDTO(INSTRUC);
             END;
           END;
```

```
          IF NEC_M_CORRECT=0 THEN
            SENDTO(DIRECCION);
         END;
      END;
      IF (P=1) THEN
        STAY:=GAUSS(40,8);

      IF (P=4) AND (NEC_M_CORRECT=0) THEN
        STAY:=GAUSS(80,20);

      IF (NEC_M_CORRECT=1) THEN
        STAY:=8;


ENSAMBLE(G) COE ::

      ASSEMBLE(ELIM,3,EQUFIELD(N)) SENDTO(COE);


DIRECCION(E) ::

      IF INSCRIP=2 THEN
        BEGIN
          WRITELN('VALOR DE INSCRIP= ',INSCRIP);
          WRITELN('CURSO NUMERO: ',NUMERO);
          WRITELN('CURSO NO ABIERTO');
          WRITELN; PAUSE;
        END;

      IF INSCRIP=1 THEN
        BEGIN
          WRITELN('VALOR DE INSCRIP= ',INSCRIP);
          WRITELN('CURSO NUMERO: ',NUMERO);
          WRITELN('CURSO ABIERTO');
          WRITELN('APROBADOS: ',APROBADOS);
          WRITELN('TIEMPO DESDE PLANIFI_INICIO=',T_INICIO-GT:6:0);

          IF P_INSTRUCTOR=1 THEN
           WRITELN('ESTE CURSO PRESENTO PROBLEMAS CON EL
INSTRUCTOR');

          IF P_INFRA=1 THEN
           WRITELN('ESTE CURSO PRESENTO PROBLEMAS CON LA
INFRAESTRUCTURA');

          IF P_MAT_APOYO=1 THEN
```

```
        WRITELN('ESTE CURSO PRESENTO PROBLEMAS CON MATERIAL DE
APOYO');

       IF P_CONTENIDO=1 THEN
        WRITELN('ESTE CURSO PRESENTO PROBLEMAS CON LA
PROGRAMACION');

       WRITELN; PAUSE;
      END;

INIT

       TSIM:=5000;
       AUXILIAR:=0;
       ACT(CURSOS,0.0);

DECL

       MESSAGES
CURSOS(N,P,INSCRIP,P_INSTRUCTOR,P_INFRA,P_MAT_APOYO,
                P_CONTENIDO,INSCRITOS,APROBADOS,NUMERO:INTEGER;

T_INICIO:REAL;NEC_M_CORRECT:INTEGER;CAMBIOS:BOOLEAN);
       VAR AUXILIAR,CICLO:INTEGER;
       STATISTICS ALLNODES;

END.
```