Universidad de las Ciencias Informáticas

”Facultad 5”

Título:“ Selección de un Lenguaje de Descripción Arquitectónica y modelado de las decisiones arquitectónicas del proyecto ERP Cuba”

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor(es): Yagnieris Montero Morales y Elier Carmenate Valero

Tutor(es): Ing. René Lazo Ochoa.

Consultante: Ing. Dorisbel Muro Fumero.

Junio del 2009

"La inteligencia consiste no sólo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica."

Aristóteles.

Agradecimientos

A mi mamá por siempre haber confiado en mí, por sus peleas, por sus consejos, por ser además de mi madre mi amiga, por caminar siempre juntas y por batallar tanto para lograr que este sueño se nos hiciera realidad. Mamita al fin lo logramos. Te quiero mucho.

A mi papá por tener la paciencia y delicadeza de siempre encontrar las palabras correctas en el momento preciso. Por ayudarme a superarme y guiarme siempre por el buen camino, gracias. Ya tienes a tu niña ingeniera. Te quiero papá.

A mis abuelos, especialmente a Miriam y Ñito por ser los rayos de sol que iluminan mis días grises. Por transmitirme toda la ternura del mundo. Por haberme enseñado a sacar de todo lo malo la parte buena. Lo que soy hoy se lo debo en gran parte a ustedes. Los quiero más que a mi vida.

A mis hermanos Luisi y Diana por alegrarme la vida y devolverme la inocencia que a veces me falta. Por hacer que me esfuerce cada día para que su tata sea un buen ejemplo a seguir y por molestarme y no dejarme dormir. Luisi esos besitos por la mañana me encantan. A Diani por todas las lágrimas derramadas por mí, no lo hagas más que no me gusta. Los adoro.

A todos mis tíos especialmente a Ramo, Oli, Reina, Lolo y Yaremi, por malcriarme tanto.

A Sol por aguantarme desde chiquita con todos mis venatismos. Gracias por tu paciencia y por todo lo que has hecho por mí. Un beso.

A Yoel por haber hecho tanto en estos últimos años y por querer tanto a mi mamá.

A Elier por ser más que mi compañero de tesis, mi apoyo durante estos dos años, por aguantar mis ataques y darme tanto amor. Por hacerme ver la vida de otro punto de vista, gracias.

A Irene, Eugenio y Amelia por haberme acogido en estos 2 años como si fuera de la familia. Gracias por todo.

A mis primos por hacer que mi infancia fuera feliz. Especialmente a Ramonin por habernos criado más que como primos, como hermanos.
A mis amigas Yanelys, Roxana Liarys y Ana que a pesar de la distancia siempre han estado ahí para mí. A Glendys por estar presente en todas mis locuras.
A Miguel Angel, Junier, Denis, William, Yandry, Junior, Lena, Heikel, Yadira por haber pasado los mejores años de la universidad juntos.
A mi prima Doris por habernos ayudado cuando más lo necesitábamos.
A mi tutor René Lazo por haber confiado en nosotros, por su ayuda incondicional y por habernos hecho correr, gracias, sin el no hubiese sido posible.

A todos mis compañeros de la universidad por los buenos y malos ratos.

A toda mi familia que siempre ha estado ahí para mí cuando los he necesitado.

A todas las personas que han estado presentes de una forma u otra en mi vida, a los buenos y a los no tan buenos, gracias, de todos he aprendido algo.

Yagnieris.

No resulta fácil agradecer a todas las personas que han estado conmigo brindándome su apoyo en cada momento en estos 5 cursos aquí en la UCI, pero de una manera u otra quiero agradecerles de todo corazón por haber estado conmigo en los momentos buenos y malos.

Especialmente quiero agradecer a mis padres por ser la luz de mi vida, mi máxima inspiración, por no haber dudado ni un segundo y haberme apoyado siempre, sin ellos no hubiese sido posible llegar hasta aquí. A mis abuelas por darme fuerzas con su amor y cariño. A mis abuelos, que aunque no hayan estado a mi lado yo nunca he olvidado sus enseñanzas. A mi tía Nury por haberse preocupado siempre por mi y no haber dudado ni un segundo en ayudarme. A Eglys, Yoel, Jorge y Sol por brindarme su apoyo.

A mi compañera de tesis y amiga Yagnieris por haber confiado en mí para esta ardua tarea, por todas las horas de estudio y dedicación que pasamos juntos, por no haberse rendido ni un instante hasta no ver materializado este trabajo. A nuestro tutor René Lazo Ochoa, por toda la confianza y el apoyo transmitido, por exigirnos cumplir nuestra tarea a cabalidad y no dejarnos solos ni un instante. A todos mis compañeros de aula, especialmente a: Dayán y Maikel por haberme apoyado en todo momento de la carrera. En fin a todos los que de una forma u otra han confiado en mí y me han animado en esta tarea tan difícil.

Elier

Dedicatoria

A mis padres por hacer de mi todo lo que soy, sin ustedes nada de esto sería realidad.

A mis abuelos Miriam y Ñito por criar de forma perfecta a una niña para que se convirtiera en la mujer que soy hoy.

A mis hermanos para que tomen ejemplo y se esfuercen para lograr todas sus metas en la vida.

A toda mi familia y amigos, cada uno ha aportado un granito de arena para conformar mis días.

A Sol y Yoel por llegar a mi familia y complementarla de forma perfecta.

Yagnieris.

Dedico esta tesis a toda mi familia, especialmente a mi madre, mi padre y mi abuela Amelia por brindarme su amor, su apoyo, sus consejos, su comprensión, su ejemplo y su confianza. Los quiero mucho.

Elier.

A nuestros familiares y amigos.

RESUMEN

La Universidad de las Ciencias Informáticas (UCI) tiene entre otras misiones la producción de software y proyectos productivos, los cuales hacen un gran aporte a la economía de país así como al desarrollo del la industria de software.

Uno de estos proyectos es el ERP[1] Cuba que por su alcance nacional, es considerado de gran importancia ya que representa un gran beneficio para el país. Este proyecto se encuentra actualmente en desarrollo y presentando problemas para modelar las decisiones arquitectónicas tomadas por los arquitectos de software ya que no cuentan con un lenguaje formal para describir dichas decisiones.

El propósito de este trabajo de diploma es seleccionar un ADL idóneo a partir del estudio del arte de los mismos y modelar en una vista de sistema la arquitectura de 9 subsistemas del proyecto ERP Cuba.

Palabras claves: Lenguajes de Descripción Arquitectónica (ADL), Arquitectura del Software(AS), Componentes, Conectores.

[1] ERP: Administración de Recursos Empresariales.

ÍNDICES

ÍNDICE DE CONTENIDO

Introducción

La Arquitectura de Software(AS) es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se le percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema (1). La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones.

Cuando el Arquitecto de Software obtiene sus decisiones arquitectónicas, es decir, la AS una vez conocidos los requerimientos, debe expresar las características de su sistema, o mejor dicho modelar estas decisiones arquitectónicas aplicando una convención gráfica o algún lenguaje avanzado de alto nivel de abstracción.

Actualmente existen varias modalidades arquitectónicas, una de ellas es la Arquitectura Estructural, basada en un modelo estático de estilos, lenguajes de descripción arquitectónica y vistas.

Los modelos estructurales sostienen que la AS está compuesta por componentes, conexiones entre ellos y (usualmente) otros aspectos tales como configuración, estilo, restricciones, semántica, análisis, propiedades, racionalizaciones, requerimientos, necesidades de los participantes. El trabajo en esta área está caracterizado por el desarrollo de ADL.

Los ADL permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento.

La definición de ADL que habrá de aplicarse en lo sucesivo es la de un lenguaje descriptivo de modelado que se focaliza en la estructura de alto nivel de la aplicación antes que en los detalles de implementación de sus módulos concretos (2).

Ocasionalmente, otras notaciones y formalismos se han utilizado y utilizan en sustitución de los ADL por su complejidad y poca expansión en la industria del software en función de la descripción de las arquitecturas de software. El caso más representativo es el Lenguaje Unificado de Modelado (UML 2.0) el cual no se presta a las mismas tareas que los lenguajes descriptivos de arquitectura. Ya que un caso de uso modelado en UML no especifica los requerimientos de interacción en situaciones en las que un sistema deba iniciar una interacción entre él mismo y un actor externo, puesto que proscribe asociaciones entre actores así como es imposible expresar relaciones secuenciales, paralelas o iterativas entre casos de uso. Además prohíbe la descomposición de casos de uso y la comunicación entre ellos. No puede

expresar una estructura entre casos de uso ni una jerarquía de casos de una forma fácil y directa; por ejemplo: “el caso de uso A requiere que el caso de uso B haya sido ejecutado previamente” no se puede expresar formalmente en ese lenguaje. Aunque en UML 2.0 existen algunos tipos de conexiones no consideran el concepto de conectores como elementos de primera clase, así como tampoco es posible representar sub-estructuras para las conexiones como es el caso de los roles en los ADL. Es deficiente para describir correspondencias tales como el mapeo entre elementos en diferentes vistas. Los diagramas de secuencia de UML no son tampoco adecuados para soportar la configuración dinámica de un sistema, ni para describir secuencias generales de actividades. Faltan también en UML los elementos para modelar comunicación peer-to-peer[2]. Le faltan recursos para detectar tempranamente errores de requerimiento y diseño, los componentes representan unidades de implementación y no de diseño, el concepto de puerto que está presente en UML 2.0 no representa una interfaz formal de un componente que describa sus características y funcionalidad en una arquitectura, ya que las conexiones se pueden hacer con o sin dicho puerto. Las agregaciones o extensiones que se les han desarrollado a UML 2.0, tienen limitaciones aun, por ejemplo la descripción de los fenómenos de transformación o dinamismo arquitectónico, por su dependencia taxonómica con las versiones anteriores, más condicionadas a las características del diseño de clases y sin prácticamente proyección sobre el diseño arquitectónico. Dada esta situación se comienzan a desarrollar los ADL.

Este trabajo surge a partir de la necesidad del desarrollo en la UCI del proyecto ERP Cuba, el cual viene al traste por la necesidad que tiene el país de desarrollar una tecnología base de gestión empresarial que permita garantizar la independencia tecnológica, para así poder perfeccionar y revolucionar la gestión de la empresa socialista cubana, además de elevar la calidad, productividad y competitividad mundial de los servicios y productos creados en la empresa cubana, lo que trae consigo el perfeccionamiento de la economía y por consiguiente del socialismo cubano, ya que se espera que el proyecto garantice como resultado final la obtención de una tecnología tipo basada en un marco legal, y una reducción en un 100% de las gastos por concepto de licencia en la gestión empresarial cubana, que puede representar niveles de ahorro del orden de los 14 millones de dólares anuales, solo por concepto de mantenimiento de

[2] peer-to-peer : Es un estilo arquitectónico aplicado a las Arquitecturas Basadas en Eventos , Arquitecturas Orientadas a Servicios (SOA) y Arquitecturas Basadas en Recursos.

licencias y contratos, en la adquisición de sistemas de este tipo adquiridos a empresas americanas monopólicas del ramo.

Este proyecto está presentando problemas para modelar las decisiones arquitectónicas ya que comenzó utilizando UML en su versión 2.0 con limitaciones para modelar las vistas de la arquitectura ya que no permite la herencia, extensión y transformación de flujos y procesamiento de datos o estructuras lógicas.

Este Proyecto de Software tiene 1321 casos de uso, lo cual implica un esfuerzo arquitectónico muy grande, y con características de integración extremas, algunos datos que reflejan la dimensión de la problemática radican en que el proyecto solo en su fase 1 de desarrollo cuenta ya con más de 110 componentes, con 176 integraciones internas y 150 externas, en solo 9 de sus subsistemas, en un equipo de desarrollo que cubren ya los 257 profesores y estudiantes lo que complejiza y pondera con mayor prioridad la necesidad de un eficiente y formal modelado del desarrollo arquitectónico propuesto para tan compleja solución de software. Ante las siguientes características observadas el trabajo plantea como **problema científico** ¿Cómo describir de manera formal[3] y entendible por los involucrados en el desarrollo, las decisiones arquitectónicas del proyecto ERP Cuba?.

Por consiguiente el **objeto de estudio** de este trabajo, consiste en la Arquitectura de Software asumiéndose como **objetivo general** Realizar la selección de un ADL idóneo a las características arquitectónicas y modelar la arquitectura de software del proyecto ERP Cuba. **El campo de acción** definido en la investigación radica en **Lenguajes Formales de Descripción Arquitectónica (ADL)**. Para dar cumplimiento al objetivo general se proponen los siguientes **objetivos específicos:**

1. Realizar marco de referencia bibliográfico a partir del estudio del estado del arte de la arquitectura de software y los lenguajes de descripción arquitectónica.
2. Realizar selección de un ADL específico y herramienta de modelado del mismo que cubra las necesidades arquitectónicas del proyecto ERP Cuba.
3. Realizar el modelado arquitectónico con el ADL propuesto de las decisiones arquitectónicas de los subsistemas Configuración, Multimoneda, Costos y Procesos, Logística, Inventario, Capital Humano, Estructura y Composición, Planificación y Contabilidad, del proyecto ERP Cuba.

[3] Descripción formal de una arquitectura de software: Se logra usando notaciones formales(ADL) para describir o modelar la arquitectura de un sistema software determinado.

Introducción

El documento está estructurado en tres capítulos, conteniendo todo lo referente al trabajo investigativo realizado así como los métodos de selección utilizados para dar cumplimiento a los objetivos del trabajo.

Capítulo I. Se encarga de abordar el tema referente al estado del arte de los Lenguajes de Descripción Arquitectónica así como hacer una adecuada conceptualización de los temas relevantes para el trabajo. Haciendo énfasis en la importancia de dichos lenguajes para el modelado arquitectónico y poniendo ejemplo de algunas aplicaciones prácticas realizadas con estos lenguajes. Además se ha hecho referencia a algunos ADL existentes tomando datos significativos de ellos como pueden ser las herramientas que utiliza, características base e institución a la que pertenece entre otras. Por último se han referenciado algunas deficiencias presentes el Lenguaje Unificado de Modelado (UML) ya que es objetivo de este trabajo sustituir el modelado con este último por uno utilizando ADL.

Capítulo II. En este capítulo se describen las características o requisitos a tener en cuenta para seleccionar un ADL, como pueden ser: Comunicación, Análisis y Validación, Propósito General, Abstracción, Derivación y Alternativas de Implementación entre otras. Posteriormente se realiza una comparación entre diferentes ADL, haciendo uso de una tabla con las características cualitativas referenciadas en los centros de investigación arquitectónicos y los ADL escogidos. Pero debido a que los procesos de selección de herramientas y arquitecturas, son procesos altamente subjetivos, y asociados a los escenarios de análisis y a las características de utilidad más significativas en el análisis, y por la complejidad de un análisis integral de la tabla de indicadores cualitativa desarrollada, se propone un instrumento de apoyo, que va a transformar en variables numéricas, las variables lingüísticas desarrolladas en la tabla de valores cualitativos y así hacer una adecuada selección del ADL a utilizar. Por último en este capítulo se especifican las características del ADL seleccionado, además de los pasos básicos de uso e interpretación, las herramientas y configuración de este.

Capítulo III.

Este capítulo le da conclusión a este trabajo de diploma, pues se realiza el modelado de la vista de sistema de los subsistemas anteriormente expuestos, siguiendo las orientaciones del equipo arquitectónico y la subdirección tecnológica del proyecto ERP Cuba.

1

CAPÍTULO I. “Conceptualización y estado del Arte de los lenguajes de descripción arquitectónica”.

Introducción

En este capítulo se hace alusión a aspectos importantes de los ADL como son: el surgimiento de estos así como su conceptualización. Además se toca el tema referente a la importancia de los ADL y de su aplicación práctica ya que estos lenguajes se hacen cada día más necesarios para la arquitectura de software. Seguidamente se hacen referencias de diferentes ADL, mostrando su autor, características base, institución a la que pertenecen así como las herramientas que automaticen el modelado.

1.1 Surgimiento de los ADL, Conceptualización de ADL, Importancia y aplicación práctica.

1.1.1 Surgimiento de los ADL

El primer estudio en que aparece la expresión “arquitectura de software”(AS) en el sentido que hoy se conoce es el de Dewayne E. Perry y Alexander L. Woolf; que ocurrió en 1992, aunque se venía indagando en el tema desde la década de los 60. En el texto fundacional de la AS, Perry y Woolf establecen el razonamiento sobre estilos de arquitectura como uno de los aspectos fundamentales de la disciplina ya que los estilos conjugan elementos (componentes), conectores, configuraciones y restricciones. Al estipular los conectores como elemento de juicio de primera clase, el concepto de estilo, incidentalmente, se sitúa en un orden de discurso y de método que el modelado orientado a objetos en general y UML en particular no cubren satisfactoriamente(3). Esto lo confirma unos años más tarde en una presentación, David Garlan, cuando señala deficiencias de UML en relación con lo que podría ser la representación de estilos(4).

Los ADL se remontan a los lenguajes de interconexión de módulos (MIL) de la década de 1970, pero se han comenzado a desarrollar con su denominación actual a partir de la década de 1990, poco después de fundada la propia arquitectura de software como especialidad profesional. Estos lenguajes surgen por la necesidad de satisfacer los requerimientos descriptivos de alto nivel de abstracción que las herramientas

basadas en objeto en general y UML en particular no cumplen satisfactoriamente. Ya que si es cierto que el modelado orientado a objetos de sistemas basados en componentes posee cierto número de rasgos muy convenientes a la hora de diseñar o al menos describir un sistema. No es menos cierto que la descripción de sistemas basados en componentes presenta también limitaciones serias, casi todas estructurales. Pues solo proporcionan una única forma de interconexión primitiva: la invocación de método, lo cual hace difícil modelar formas de interacción más ricas o diferentes. Además, el soporte de los modelos orientados a objetos para las descripciones jerárquicas es, en el mejor de los casos, débil. Tampoco se soporta la definición de familias de sistemas o estilos; y no hay recurso sintáctico alguno para caracterizar clases de sistemas en términos de las restricciones de diseño que debería observar cada miembro de la familia. Y por último, no brindan soporte formal para caracterizar y analizar propiedades no funcionales, lo que hace difícil razonar sobre propiedades críticas del sistema, tales como performance y robustez (5). Sin embargo los ADL permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento. La mayoría de los ADL existentes han sido de extracción académica, donde ha jugado un papel relevante la Universidad de Carnegie Mellon (CMU), en donde surgen ADL tales como Wright (1994) que es un ADL de propósito general con énfasis en comunicación, Aesop (1994) el cual tiene un propósito general también pero hace énfasis en estilos, UniCon (1995) que reúne características de los dos anteriores ya que hace énfasis en conectores y estilos y Armani(1998) que es un ADL asociado a Acme, los dos primeros creados por David Garlan y el tercero por Mary Shaw quienes son miembros de la facultad de ingeniería de software de dicha universidad y los cuales han brindado significativos aportes al campo de la AS en general y al de los ADL en particular. Ya que fueron ellos quienes en 1994, el cual podría ser considerado como “el año de oro de la Arquitectura de Software”, quienes lanzan el concepto de Lenguajes de Descripción Arquitectónica (ADL), en su libro “Characteristics of Higher Level Languages for Software Architecture”. En el estudio se señalan las alternativas que hasta el momento se poseen para la definición de la AS de un sistema. En primer lugar, a través de la modularización de las herramientas existentes de programación y de los módulos de conexión entre ellas y, en segundo lugar, describir sus diseños usando diagramas informales y frases idiomáticas. A partir de estas reflexiones, definen un conjunto de regularidades y propiedades específicas, que constituyeron las bases de los ADL(6). Hasta hace pocos años existía relativamente poco consenso respecto a qué es un ADL, cuáles lenguajes de modelado califican como tales y cuáles no, y qué aspectos

de la arquitectura se deben modelar con dichos lenguajes. Sin embargo actualmente los ADL son bien conocidos en los estudios universitarios de arquitectura de software(AS), aunque son muy pocos los arquitectos de industria que los utilizan como instrumento en el diseño arquitectónico de sus proyectos a pesar que los ADL difieran sustancialmente de los UML y que cubran necesidades que este último no es capaz de satisfacer.

Viendo la trayectoria que han seguido los lenguajes de descripción arquitectónica desde su surgimiento hasta hoy en día se puede decir que no han logrado abrirse mucho paso en la industria a pesar que la arquitectura de software sigue avanzando, y que queda aun mucho por descubrir e indagar en cuanto a los ADL.

1.1.2 Conceptualización de ADL.

Para poder adentrarse en el mundo de los ADL y entenderlos en su totalidad es necesario conocer una serie de conceptos que ayudaran además a un mejor desenvolvimiento y entendimiento de este trabajo. Uno de estos conceptos es el de Arquitectura de Software expuesto anteriormente.

Otra definición que debe quedar bien esclarecida es la de ADL, que como todos los componentes de la arquitectura no es rígida ni unívoca sino que está dada por el criterio de diferentes arquitectos según su experiencia o necesidades. Una de estas definiciones (además bastante general) sostiene que un ADL es una entidad consistente en cuatro “Cs”: componentes, conectores, configuraciones y restricciones, sin embargo hay ADL que pueden modelar o soportar los siguientes conceptos: Componentes, Conexiones, Composición jerárquica (en la que un componente puede contener una sub-arquitectura completa), Paradigmas de computación, es decir, semánticas, restricciones y propiedades no funcionales, Paradigmas de comunicación, Modelos formales subyacentes, Soporte de herramientas para modelado (análisis, evaluación y verificación) y Composición automática de código aplicativo. Debe quedar claro que no todos los ADL son capaces de soportar estos conceptos, generalmente para soportarlos todos se deben unir más de uno. Entre las características de estos lenguajes podemos considerar las siguientes: Composición, que permiten la representación del sistema como la composición de una serie de partes. Configuración, Abstracción, mediante la cual se describen los *roles* o papeles abstractos que juegan los componentes dentro de la arquitectura, Flexibilidad ya que permiten la definición de nuevas formas de interacción entre componentes. Reutilización pues permiten la reutilización tanto de los componentes como de la propia arquitectura, Heterogeneidad ya que pueden combinar descripciones heterogéneas, y

por último Análisis. Permiten diversas formas de análisis de la arquitectura y de los sistemas desarrollados a partir de ella.
Los conceptos que aparecen a continuación se ha decidido tratarlos por separado ya que es necesario que queden bien esclarecidos para un mayor entendimiento y mejor desenvolvimiento del trabajo de diploma, ellos son:

Componentes: Representan los elementos computacionales primarios de un sistema. Intuitivamente, corresponden a las cajas de las descripciones de caja-y-línea de las arquitecturas de software. Ejemplos típicos serían clientes, servidores, filtros, objetos, pizarras y bases de datos. En la mayoría de los ADL los componentes pueden exponer varias interfaces, las cuales definen puntos de interacción entre un componente y su entorno.

Conectores: Representan interacciones entre componentes. Corresponden a las líneas de las descripciones de caja-y-línea. Ejemplos típicos podrían ser tuberías (pipes), paso de mensajes, llamadas a procedimientos, protocolos cliente-servidor o conexiones entre una aplicación y un servidor de base de datos. Los conectores también tienen una especie de interfaz que define los roles entre los componentes participantes en la interacción.

Configuraciones o sistemas: Se constituyen como grafos de componentes y conectores. En los ADL más avanzados la topología del sistema se define independientemente de los componentes y conectores que lo conforman. Los sistemas también pueden ser jerárquicos: componentes y conectores pueden subsumir la representación de lo que en realidad son complejos subsistemas.

Restricciones: Representan condiciones de diseño que deben acatarse incluso en el caso que el sistema evolucione en el tiempo. Restricciones típicas serían restricciones en los valores posibles de propiedades o en las configuraciones topológicas admisibles. Por ejemplo, el número de clientes que se puede conectar simultáneamente a un servicio.

Propiedades: Representan información semántica sobre un sistema más allá de su estructura. Distintos ADL ponen énfasis en diferentes clases de propiedades, pero todos tienen alguna forma de definir propiedades no funcionales, o pueden admitir herramientas complementarias para analizarlas y determinar, por ejemplo, la velocidad de transferencia de datos y la latencia probables, o cuestiones de

seguridad, escalabilidad, dependencia de bibliotecas o servicios específicos, configuraciones mínimas de hardware y tolerancia a fallas.

Propiedades no funcionales: La especificación de estas propiedades es necesaria para simular la conducta de runtime, analizar la conducta de los componentes, imponer restricciones, mapear implementaciones sobre procesadores determinados, etcétera.

Estilos: Representan familias de sistemas, un vocabulario de tipos de elementos de diseño y de reglas para componerlos. Ejemplos clásicos serían las arquitecturas de flujo de datos basados en grafos de tuberías (pipes) y filtros, las arquitecturas de pizarras basadas en un espacio de datos compartido, o los sistemas en capas. Algunos estilos prescriben un framework[4], un estándar de integración de componentes, patrones arquitectónicos o como se lo quiera llamar. A continuación se mencionan algunos estilos arquitectónicos: *Estilos de Flujo de Datos, Estilos Centrados en Datos, Estilos de Llamada y Retorno, Estilos de Código Móvil, Estilos heterogêneos y Estilos Peer-to-Peer.*

Cuando se habla de estilos no se pueden olvidar los patrones de arquitectura los cuales están claramente dentro de la disciplina arquitectónica, solapándose con los estilos. Dichos patrones arquitectónicos expresan esquemas de organización estructural fundamentales para los sistemas de software, los cuales con un empaquetado un poco distinto, no son más que estilos y a su vez proporcionan un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen guías y lineamientos para organizar las relaciones entre ellos.

Para que un lenguaje se considere ADL debe cumplir con otros aspectos los cuales se expresan a continuación:

Dinamismo: Un ADL debe ser capaz de describir arquitecturas dinámicas y cambiantes ya que así permite interpretar los flujos de transformación y adaptabilidad de las mismas.

Comunicación: El ADL debe comunicar todas las partes de una arquitectura así como definir todas las estructuras de la misma ya sean estáticas o dinámicas las cuales deben ser capaces de identificar los diversos tipos de conectores y componentes. Además la información que proporcionan las descripciones arquitectónicas se debe poder personalizar según el lector actual de la arquitectura.

[4] Framework: Estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado.

Verificación de Propiedades (Análisis y Validación): El ADL debe dar soporte a las tareas de creación de la arquitectura, así como refinamiento y validación de las mismas. Para esto definirá reglas sobre lo que se entiende por una arquitectura completa y consistente.

Abstracción: El ADL debe ser capaz de proporcionar estructuras del sistema que expresen información arquitectónica y que además enmascare toda la información sobre la implementación que no sea arquitectónica.

Derivación: El ADL debe proporcionar una base para fomentar la implementación como por ejemplo: la generación rápida de prototipos. Debe permitir la agregación de información fuera de la descripción arquitectónica obtenida con la misma manera que permita que una especificación final del sistema se derive de dicha descripción arquitectónica.

Alternativas de Implementación: El ADL debe ser capaz de soportar la especificación de familias de implementaciones que satisfagan toda una arquitectura común.

1.1.3 Importancia y aplicación práctica de los ADL

Los lenguajes de descripción de arquitecturas, ocupan una parte importante del trabajo arquitectónico desde la fundación de la AS. Ya que contando con un ADL, un arquitecto puede razonar sobre las propiedades del sistema con precisión, pero a un nivel de abstracción convenientemente genérico. Algunas de esas propiedades podrían ser, por ejemplo, protocolos de interacción, anchos de banda y latencia, localización del almacenamiento, conformidad con estándares arquitectónicos y previsiones de evolución ulterior del sistema.

Además suministran construcciones para especificar abstracciones arquitectónicas y mecanismos para descomponer un sistema en componentes y conectores, especificando de qué manera estos elementos se combinan para formar configuraciones y definiendo familias de arquitecturas o estilos. Precisamente lo que necesita una arquitectura para tener éxito, y con ella el proyecto de software en sí, ya que un proyecto será bueno en la medida que lo sea su arquitectura.

Todos los ADL existentes tienen un objetivo específico por lo que cada uno tiene su importancia, uno de los que sobresale debido a su importancia es UniCon que se destaca por su capacidad de manejo de métodos de análisis de tiempo real a través de RMA (Rate Monotonic Analysis).

Conociendo la importancia de los ADL, se podría pensar que existe gran número de ellos, y que son utilizados para el modelado de toda arquitectura de software, sin embargo, contrario a lo que se piensa, no existen tantas herramientas de modelado de arquitectura, existen en el mundo alrededor de unos veinte ADL de primera magnitud y quizás una cifra mayor propuestos en ponencias pero que no han resistido el paso del tiempo o que no han encontrado su camino en el mercado.

En la práctica quienes no utilizan UML para diagramar la arquitectura, utilizan diagramas de cajas y líneas, a pesar de sus deficiencias, aunque esto no quiere decir que los ADL no se utilicen, ejemplo de ello es LILEANNA, un ADL (o más estrictamente un MIL) que se utilizó para producir software de navegación de helicópteros. También está MetaH un ADL que modela arquitecturas en los dominios de guía, navegación y control (GN&C) y en el diseño aeronáutico. Pero son muy pocos los que se han utilizado en la práctica, por lo que es misión de este trabajo llevar a la práctica un ADL que supla las necesidades arquitectónicas del proyecto ERP Cuba.

1.2 “Referencias de ADL.”

Acme

Acme fue desarrollado por Robert Monroe (profesor en la Universidad Carnegie Mellon) y David Garlan (miembro del directorio del área de Ingeniería de Carnegie Mellon University) en la Universidad Carnegie Mellon (CMU) específicamente en la Escuela de Ciencias de la Computación de dicha universidad. Acme se define como una herramienta capaz de soportar el mapeo de especificaciones arquitectónicas entre diferentes ADL, o en otras palabras, como un lenguaje de intercambio de arquitectura. Sus desarrolladores lo consideran como un ADL de segunda generación. El mismo define cuatro tipos dentro de la arquitectura: la estructura (organización de un sistema en sus partes constituyentes), las propiedades de interés (información que permite razonar sobre el comportamiento local o global, tanto funcional como no funcional), las restricciones (lineamientos sobre la posibilidad del cambio en el tiempo), los tipos y estilos. Además define 7 tipos elementos básicos que son: Componentes, conectores, sistemas, puertos, roles, representaciones y mapas de representación. Su código se encuentra disponible tanto en C++ como en Java, y puede ser invocada por lo tanto desde cualquier lenguaje la plataforma clásica de Microsoft o desde el framework de .NET. En el caso de java requiere JRE [5] que permite la

[5] JRE : Java Runtime Environment en español significa Entorno en Tiempo de Ejecución Java.

ejecución de programas java sobre todas las plataformas soportadas. Acme soporta una variedad de front-ends[6] de carácter gráfico, que se componen por: AcmeStudio que es un entorno gráfico basado en Windows y Linux, susceptible de ser configurado para soportar visualizaciones específicas de estilos e invocación de herramientas auxiliares , y un ambiente diseñado en ISI [7] que usa el editor de PowerPoint para manipulación gráfica acoplado con analizadores que reaccionan a cambios de una representación DCOM[8] (en español Modelo de Objetos de Componentes Distribuidos que es una tecnología COM) de los elementos arquitectónicos y de sus propiedades asociadas .

Aesop

Aesop[9] fue desarrollado por David Garlan en el Proyecto ABLE de la CMU. La definición también oficial de Aesop es “una herramienta para construir ambientes de diseño de software basada en principios de arquitectura”. Es un ADL de propósito general cuyo ambiente de desarrollo se basa en el estilo de tuberías y filtros propio de UNIX [10].Su naturaleza es Orientada a Objetos y soporta desarrollos realizados en C++ además de generar código en el mismo lenguaje. En Aesop, conforme a su naturaleza, el vocabulario relativo a estilos arquitectónicos se describe mediante la definición de sub.-tipos de los tipos arquitectónicos básicos tales como: componente, conector, puerto, rol, configuración y bindings. En Aesop la semántica de una arquitectura puede ser arbitrariamente distinta para cada estilo, por ello, no incluye ningún soporte nativo para la descripción de la semántica de un estilo o configuración, sino que apenas presenta unos cuadros vacantes para colocar esa información como comentario. No está disponible en plataforma Windows pero puede utilizarse para modelar sistemas implementados en cualquier plataforma.

[6] Front-Ends :Parte del software que interactúa con los usuarios

[7] ISI : Editor de diseño visual

[8] DCOM: En español Modelo de Objetos de Componentes Distribuidos que es una tecnología COM(Modelo de Objetos de Componentes).

[9] Aesop : Software Architecture Design Environment Generator.

[10] UNIX : Sistema operativo portable, multitarea y multiusuario.

Darwin

Darwin fue desarrollado por Jeff Magee y Jeff Kramer .Este ADL está orientado al diseño de arquitecturas dinámicas y cambiantes. Describe un tipo de componente mediante una interfaz consistente en una colección de servicios que declarados por dicho componente que se espera ocurran en el entorno. Las configuraciones se desarrollan instanciando las declaraciones de componentes y estableciendo vínculos entre ambas clases de servicios. Soporta la descripción de arquitecturas que se reconfiguran dinámicamente a través de dos tipos de construcciones: **la instanciación tardía**[11] y la **instanciación dinámica explícita**[12]. Darwin proporciona herramientas para desarrollar configuraciones ejecutables de caja negra y posee un número fijo de tipos de interacción, por otra pare el soporte para estilos arquitectónicos se limita a la descripción de configuraciones parametrizadas, para delinear un estilo es necesario construir un algoritmo capaz de representar a los miembros de un estilo. Queda claro que en Darwin Los dos conceptos básicos que se manejan son el de componente y el de servicio, esto se debe a que el mismo carece de la posibilidad de ponerle nombre, sub-tipear o reutilizar un conector así como tampoco se pueden describir patrones de interacción independientemente de los componentes que interactúan. Se puede usar sobre Windows utilizando la herramienta gráfica Software Architect’s Assistant y para ello requiere JRE. Este ADL ha sido extendido recientemente para incorporar la posibilidad de analizar propiedades de seguridad y viveza de las especificaciones y además es capaz generar código a partir de las mismas. Es importante señalar que este ADL no es eficiente ya que no proporciona una base adecuada para el análisis de la conducta de una arquitectura, debido a que el modelo no dispone de ningún medio para describir las propiedades de un componente o de sus servicios más que como comentario. Los componentes de implementación se interpretan como cajas negras, mientras que la colección de tipos de servicio es una colección dependiente de plataforma cuya semántica tampoco se encuentra interpretada en el framework de Darwin.

[11] La instanciación tardía**:** Se describe una configuración y se instancian componentes sólo en la medida en que los servicios que ellos proveen sean utilizados por otros componentes, es decir no se instancia hasta que un usuario intenta acceder a dicho servicio.

[12] La instanciación dinámica explícita**:** Se ejecuta en tiempo de ejecución, antes que una declaración estática de la estructura.

UniCon

UniCon[13] fue desarrollado por Mary Shaw en la CMU .Es un ADL de propósito general con énfasis en conectores y estilos. Su objetivo es servir de vínculo de representación de las abstracciones utilizadas en la práctica por los diseñadores de software, para lo cual los mecanismos de conexión ofrecidos por el lenguaje deben ser variados. En UniCon los componentes son las unidades de compilación de los lenguajes de programación y otros objetos del nivel del usuario, por ejemplo un fichero. Pero los conectores no pueden identificarse tan fácilmente en el nivel del usuario por ejemplo: entradas en una tabla, directivas de ensamblado, estructuras de datos utilizadas en tiempo de ejecución, protocolos estándares de comunicación, etc. Tanto a los componentes como a los conectores se les asocia un tipo o clase, que es la que determina cual será su interfaz, además estos están descritos mediante una serie de atributos y tienen una implementación determinada. Los atributos de los componentes describen características tales como su funcionalidad, rendimiento, etc., mientras que su implementación será el código en un lenguaje de programación. Los atributos de un conector incluyen diversas características por ejemplo la garantía de entrega de paquetes en una red de comunicación, restricciones sobre el orden de envió o recepción de eventos, reglas sobre la instanciación de parámetros, restricciones sobre el número de componentes que conectan y los papeles que estos componentes juegan en la conexión, entre otros. En el caso de los conectores su implementación es también diversa, se realiza a través de mecanismos proporcionados por los lenguajes de programación y no son más que variables globales o llamadas a procedimiento, por medio de entradas en tablas de tareas o de encaminamiento y llamadas a funciones del sistema operativo o una plataforma subyacente, como por ejemplo para la lectura y escritura de sockets o para el envió y recepción de mensajes etc. Con este ADL la descripción de la interfaz se lleva a cabo con el propósito de definir las posibles conexiones entre componentes y conectores para formar sistemas más grandes. En los componentes la interfaz está constituida por varios actores mientras que en los conectores está formada por roles. Los actores y roles son de un tipo específico que se describe a través de atributos que dependen del tipo de que se trate, de tal forma que los actores y roles conectados entre sí deben ser de un mismo tipo. En UniCon estos tipos están predeterminados y forman parte del propio lenguaje, por lo que se tiene control sobre los tipos componentes y conectores que pueden utilizarse y cuáles son las posibilidades de combinarlos, y es precisamente esto lo que permite a UniCon

[13] UniCon : Universal Connector Support. En español: Conector universal de soporte

la generación de un sistema ejecutable en C a partir de una descripción arquitectónica , pero por otra parte no es posible definir sub.-tipos y por lo tanto carece de capacidad de evolución. Este ADL se destaca además por su capacidad de manejo de métodos de análisis de tiempo real a través de RMA[14]. Es importante señalar este ADL no tiene una base formal que posibilite la validación de propiedades de las especificaciones, por otra parte no es posible la reutilización de componentes o arquitecturas, ni su adaptación a cambios en los requisitos y la descripción de sistemas dinámicos.

Wright

Wright fue desarrollado por David Garlan en la CMU específicamente en la Escuela de Ciencias Informáticas de esta universidad, como parte del proyecto mayor de ABLE. Es un ADL de propósito general con énfasis en la comunicación y además una herramienta de formalización de conexiones arquitectónicas. Sus objetivos principales son la integración de una metodología formal con una descripción arquitectónica y la aplicación de procesos formales tales como el álgebra de proceso y el refinamiento de procesos a una verificación automatizada de las propiedades de las arquitecturas de software. En Wright la interfaz de los componentes se representa a través de una serie de puertos y la de los conectores a través de una serie de roles, donde los puertos describen el comportamiento de los componentes y los roles el comportamiento que deben tener los conectores una vez se conecten los componentes. Los puertos y roles que se relacionen entre sí deben ser compatibles como ocurre en UniCon, con la diferencia que no existen tipos predefinidos para los componentes y conectores, aquí las interfaces indican cual es el protocolo que rige el comportamiento del componente o conector que se está especificando a través del algebra de procesos CSP[15], de esta forma se logra describir la interacción (el protocolo de coordinación) que tiene lugar entre el elemento y su entorno, mediante la sincronización de eventos que permiten modelar tanto la invocación de operaciones, como el intercambio de mensajes. Al usar CSP como modelo es posible automáticamente analizar conectores y verificar (a través de verificaciones de consistencia) de forma estática que estos están libres de bloqueo, esta característica le aporta gran confiabilidad a este ADL. En Wright existe la posibilidad de usar cualquier herramienta de análisis o técnica que pueda usarse para CSP por ejemplo el FDR[16], el cual proporciona a Wright la

[14] RMA : Rate Monotonic Analysis.

[15] CSP: Communicating Sequential Process. En español: Comunicación de procesos secuencial

[16] FDR : Verificador de modelos comercial para CSP.

garantía de que un conector esté libre de bloqueo en todos los escenarios posibles .Es importante señalar que no genera código a través de una descripción arquitectónica, así como tampoco permite reutilizar componentes y conectores.

Rapide

Fue propuesto por la ARPA[17] como el diseño de un lenguaje para la producción de prototipos de sistemas, con herramientas de soporte para la simulación y el análisis. Fue desarrollado por Luckha (Stanford). Es un ADL de propósito general que permite modelar interfaces de componentes y la conducta de los mismos. Se centra en obtener prototipos ejecutables en VHDL, C. C++, Ada y Rapide [18]a partir de las especificaciones arquitectónicas donde las deben contener explícitamente reflejadas las propiedades de flujo de información, sincronización, concurrencia y temporización de la arquitectura que representan. Presenta una estructura compleja la cual se compone de 5 lenguajes comparten un único modelo de ejecución: el lenguaje de tipos (describe las interfaces de los componentes), el lenguaje de arquitectura (describe el flujo de eventos entre componentes), el lenguaje de especificación (describe restricciones abstractas para la conducta de los componentes), el lenguaje ejecutable (describe módulos ejecutables) y el lenguaje de patrones (describe patrones de los eventos). El modelo de ejecución adoptado por este ADL está basado en los posets[19] los cuales son identificados a través de patrones de eventos. El objetivo de estos posets es describir las secuencias válidas de eventos que pueden ocurrir en un determinado sistema, los cuales se corresponden con la invocación de las operaciones entre los componentes, estos eventos se dividen en acciones que modelan comunicaciones asíncronas y funciones que modelan comunicaciones síncronas; de esta forma es posible simular las especificaciones realizadas con este ADL, así como verificar si las trazas de eventos generados cumplen determinadas restricciones. En Rapide toda la arquitectura es simulada junto con la ejecución de la implementación, lo cual es muy útil para detectar alternativas de ejecución.

Es importante señalar que carece de la definición de conector como parte del lenguaje, y lo que se hace es modelar los sistemas de software como un conjunto de componentes conectados a través del envío de mensajes o eventos, así que la descripción de la arquitectura se lleva a cabo por medio de una lista de

[17] ARPA: Advanced Research Projects Agency. En español: Red Avanzada de Agencias para Proyectos.

[18] (VHDL, C. C++, Ada y Rapide):Todos excepto Rapide(ADL) son lenguajes de programación.

[19] Posets: Conjunto de eventos parcialmente ordenados.

componentes interconectados entre sí a través de las operaciones importadas y exportadas en las interfaces de dichos componentes. Una característica significativa es que este presenta un mecanismo de herencia para la reutilización de especificaciones y su adaptación a cambios de los requisitos. La disponibilidad de herramientas de soporte así como su documentación se puede obtener de forma gratuita, por otra parte hasta el momento el conjunto de herramientas se encuentra disponible para Solaris 2.5, SunOS 4.1.3. y Linux[20], dichas herramientas son: **RPDC**[21] y **POV**[22] .

C2

C2 (Chiron-2) fue desarrollado en la Universidad de California en Irvine .Se le considera más un estilo arquitectónico que un ADL. El mismo define que un sistema de software está compuesto por componentes ordenados en capas o niveles que se coordinan y comunican mediante peticiones de servicios y notificaciones de eventos), y por conectores los cuales tienen como única función conectar a los componentes y llevar a cabo la comunicación entre ellos, en cada nivel o capa hay un solo conector el cual actúa de delimitador para su nivel correspondiente. En el caso de los componentes solo se pueden vincular con un conector, mientras un conector se puede vincular con más de un componente y más de un conector. A la hora de la comunicación por medio de dos tipos de mensajes, los de requerimiento que sólo se pueden enviar “hacia arriba” y los de notificación sólo “hacia abajo”. En C2 no existe limitación para los lenguajes de la implementación, aunque normalmente soporta desarrollos en C++, Ada, C# y Java. Hasta el momento existen extensiones de Microsoft Visio para C2, así como se puede usar una herramienta llamada SAAGE la cual requiere Visual J++, COM y ya sea Rational Rose o DRADEL como front-ends. Entre las herramientas independientes de plataforma que dispone C2 está el entorno gráfico ArchStudio implementado en estilo C2.

[20] Solaris 2.5, SunOS 4.1.3. y Linux : Todos son Sistemas Operativos de licencia GPL.

[21] RPDC: Compilador de código fuente *RAPIDE*, el cual reporta errores de sintaxis y semánticos. Genera un ejecutable que al correr produce un archivo “*.log*“ que es un registro de los eventos producidos durante la ejecución.

[22] POV : Las siglas significan Partial Order Viewer y permite observar en forma gráfica la traza de eventos producidos por los ejecutables.

Jacal

Jacal fue desarrollado por un Grupo de investigación del Departamento de Computación de la Facultad de Ciencias Exactas y Naturales de la Universidad de Buenos Aires entre ellos Nicolás Kicillof y Daniel Yankelevich. Es un ADL de propósito general se basa en la notación de alto nivel para descripción y prototipado. El objetivo principal de Jacal es lo que actualmente se denomina “animación” de arquitecturas, que no es más que poder visualizar una simulación de cómo se comportaría en la práctica un sistema basado en la arquitectura que se ha representado ya que la notación principal de Jacal es gráfica. Este ADL posee un conjunto predefinido y extensible de conectores, cada uno con una representación distinta. Se utiliza para expresar arquitecturas de distintos estilos. No ofrece una forma de restringir una configuración a un estilo específico, ni de validar la conformidad. Cada componente cuenta con puertos (ports) que constituyen su interfaz y a los que pueden adosarse conectores. Jacal tiene una semántica denotacional que asocia a cada arquitectura una red correspondiente que está dada en función de las redes de Petri la cual justifica la animación de las arquitecturas. Jacal ofrece además del nivel de interfaz un nivel de comportamiento en el cual se describe la relación entre las comunicaciones recibidas y enviadas por un componente, usando diagramas de transición de estados con etiquetas en los ejes que corresponden a nombres de puertos por los que se espera o se envía un mensaje. Las herramientas que actualmente están disponible para editar y animar arquitecturas en Jacal son una aplicación Win32 que no requiere instalación, basta con copiar el archivo ejecutable para comenzar a usarla.

LEDA

Es un lenguaje de especificación y validación de arquitecturas de software que fue desarrollado por el grupo de ingeniería de software de la universidad de Málaga (GISUM). Posee dos niveles fundamentales, uno para la definición de componentes los cuales representan partes o módulos del sistema, proporcionando cada uno de ellos una determinada funcionalidad al mismo; y otro para la definición de roles los cuales describen el comportamiento observable de los componentes y los protocolos de interacción que siguen con el resto de los componentes, esto es utilizado para el prototipado, validación y ejecución de la arquitectura. Los roles se logran especificar aplicando el cálculo *pi,* un álgebra de procesos que expresa de forma natural la movilidad lo que permite la especificación de arquitecturas cuya topología de comunicación varía en el tiempo es decir las arquitecturas dinámicas. Los componentes están compuestos por otros componentes. La estructura o arquitectura de los mismos se indica mediante

las relaciones establecidas entre sus subcomponentes, lo cual es expresado por un conjunto de *conexiones* entre los roles de dichos subcomponentes. En LEDA los conectores se especifican como otro tipo de compones permitiendo que el lenguaje sea más simple y regular, a la vez que no impone un modelo composicional concreto para la descripción de arquitecturas de software. Como se dijo anteriormente la semántica de LEDA está escrita en términos del cálculo *pi*, gracias a esto es posible que las especificaciones sean analizadas y ejecutadas, lo que permite el prototipado de la arquitectura así como analizar si sus componentes presentan comportamientos compatibles entre sí y pueden ser combinados correctamente para construir el sistema. Con LEDA se puede adaptar un componente a otro donde la interfaz que no sea compatible con la suya a través de los *adaptadores*, lo que permite la reutilización del componente. Los *adaptadores* al igual que lo *roles* se describen usando el cálculo *pi*. Una descripción arquitectónica realizada usando este lenguaje es marco de trabajo genéricos que puede ser instanciados y reutilizados tantas veces como sea necesario.

Conclusiones del capitulo

En este capítulo se hace una recopilación sobre el mundo de los ADL, específicamente se profundiza en el surgimiento de los mismos así como su importancia y aplicación práctica. Además se realiza un estudio detallado del estado del arte haciendo énfasis en 9 ADL específicamente, donde se puede apreciar que además de que cada uno posee características propias todos suministran construcciones para especificar abstracciones arquitectónicas y mecanismos para descomponer un sistema en componentes y conectores, especificando de qué manera estos elementos se combinan definiendo familia de configuraciones y estilos.

.

2

CAPITULO 2. ”Selección del ADL y herramienta . Caracterización del lenguaje de descripción y la herramienta seleccionada”.

Introducción

Este capítulo se compone de dos epígrafes. En el epígrafe 1 se hace un análisis comparativo entre los ADL tratados en el capítulo 1, y se selecciona el adecuado que da respuesta a los objetivos trazados. Para la comparación se hace uso de una tabla cualitativa que expresa las características de estos lenguajes. Y como método auxiliar para la selección se desarrolla un instrumento de análisis cuantitativo. Por último, el epígrafe 2 se expresan detalladamente las características del ADL seleccionado así como los pasos lógicos de uso, y las características de la herramienta de modelado.

2.1 “Análisis comparativo de los ADL. Selección del lenguaje y su herramienta.”

Como parte de las estrategias que se llevaron a cabo para seleccionar el ADL adecuado entre los expuestos en el capitulo1 para la descripción de las decisiones arquitectónicas del proyecto software ERP Cuba, se realizaron dos tablas comparativas las cuales constan de los mismos aspectos a comparar solo que en una se dan valores cualitativos y en la otra cuantitativos. El primer epígrafe del capítulo anterior sirvió de base para conformar dichos aspectos comparativos. Se partió de las características generales, las más significativas según las necesidades del equipo de desarrollo, y se incluyeron nuevas características asociadas al tipo de proyecto, y al entorno.

Se creó una tabla con valoraciones cualitativas teniendo en cuenta las características de análisis seleccionada y se realizó una cuantificación de las variables lingüísticas para poder realizar un análisis más objetivo de las características cualitativas expresadas en la comparación, esta técnica expresa el análisis de las mismas características pero en una notación lingüística transformada a una especificación cuantitativa.

Los atributos seleccionados para el instrumento son los siguientes:

Atributos	**Significado**

Herramienta Gráfica Windows	Debe existir al menos una herramienta para el sistema operativo Windows.
Herramienta Gráfica Linux	Debe existir al menos una herramienta para el sistema operativo Linux.
Varios Tipos de Componentes	El ADL debe dar la posibilidad de crear más de un tipo de componente.
Varios tipos de Conectores	El ADL debe dar la posibilidad de crear más de un tipo de conector.
Restricciones	El ADL debe permitir denotar las restricciones que se hayan trazado en las decisiones arquitectónicas.
Sub-estructuras	El ADL debe dar la posibilidad de crear sub-estructuras para obtener la mejor representación y abstracción posible.
Dinámica	El ADL debe dar la posibilidad de modelar arquitecturas cambiantes en el tiempo.
Herramienta con Licencia GPL	La herramienta debe ser de licencia GNU/GPL, para poder trabajar sobre plataforma libre.
Verificación de propiedades	El ADL debe proporcionar algún medio para verificar las propiedades.
Desarrollo y reutilización (Herencia)	El ADL debe ser capaz de dar soporte a la herencia, proporcionando la posibilidad de reutilizar los modelados arquitectónicos.
Patrones y Estilos Arquitectónicos.	El ADL debe ser capaz de modelar vistas donde se pueda apreciar los patrones arquitectónicos a utilizar en la arquitectura de software.

Tabla 1 Significado de las variables o atributos seleccionados por la subdirección técnica del proyecto ERP Cuba.

A continuación se muestra la comparación cualitativa :

ADL	**UniCon**	**Wright**	**Rapide**	**Jacal**	**Darwin**	**LEDA**	**C2**	**Acme**	**Aesop**

Herramienta Gráfica Windows	Si	Si	No	Si	Si	Si	Si	Si	No
Herramienta Gráfica Linux	No	No	Si	No	No	No	No	Si	No
Varios Tipos de Componentes	Si	Si	Si	Si	Si	Si	Si	Si	Si
Varios tipos de Conectores	Si	Si	No	Si	No	No	No	Si	Si
Restricciones	Si	Si	Si	Si	Si	Si	Si	Si	Si
Sub-estructuras	Si	Si	Si	Si	Si	Si	Si	Si	Si
Dinámica	No	No	No	No	Si	Si	Si	No	No
Herramienta con Licencia GPL	No	No	No	No	No	No	No	No	No
Verificación de propiedades	No	Si	No	No	Si	Si	No	No	No
Desarrollo y reutilización (Herencia)	No	No	Si	No	No	Si	Si	No	No
Patrones y Estilos Arquitectónicos.	No	No	Si	No	Si	Si	Si	Si	Si

Tabla 2 Comparación cualitativa de los ADL.

Como resultado de esta comparación se llegó a la conclusión que la mayoría de estos ADL desarrollan los mismos conceptos, centrándose en los elementos más relevantes de la arquitectura de software tales como componentes, conectores, restricciones y sub estructuras; no obstante en el caso de los ADL que no definen variedad de conectores, no quiere decir que esté ausente este concepto sino que para realizar la

comunicación entre los componentes existe un único tipo de conector que en la mayoría de los casos representa un tipo de mensaje y lo negativo en este caso es que no tienen a los conectores como elemento de primera clase. Hay que señalar el caso particular de LEDA el cual usa componentes que se comportan como conectores. Otro es el caso de Rapide donde las conexiones entre los componentes se realizan a través de mensajes en un tiempo dado por instanciación tardía o temprana, aunque en realidad el rol de conector se define en un componente una vez que queda incluido en la descripción del protocolo de su interfaz y por último Darwin donde los componentes se conectan a través de servicios, es decir un componente solicita un servicio de otro componente y es entonces cuando se realiza la conexión ya que estos servicios llevan asociado un tipo, a través del que se realizan las comprobaciones de compatibilidad de las conexiones entre componentes.

En el caso de las herramientas gráficas, se puede apreciar que de los ADL que cuentan con notaciones gráficas la mayoría son en plataforma Windows, solo tres de los analizados cuentan con herramientas en Linux donde existen componentes que se comportan como un tipo de conector.
Para la expresión cuantitativa del mismo análisis cualitativo se desarrolla una conversión de los atributos definidos en la comparación mediante la transformación en variables lingüísticas de los mismos, que permite cuantificar las evaluaciones del mismo y realizar un mejor análisis en la toma de decisiones comparativa desarrollada.

Es importante señalar que no es un instrumento matemático cartesiano que permita discriminar o definir una selección ya que los procesos de selección de herramientas de arquitecturas son procesos altamente subjetivos y asociados a los escenarios del análisis y las características de utilidad más significativas en el mismo.
Para ponderar a las variables en el instrumento cuantitativo a cada una se le asignó un valor o peso en una escala de 0 a 10. Estos valores fueron definidos por el equipo de arquitectura del ERP, la subdirección tecnológica responsable del desarrollo tecnológico y arquitectónico del proyecto, esta selección solo es aplicable en este caso o en casos similares, si el ADL cumple con el atributo se le pondera con el valor del peso de dicho atributo, sino ocurre esto entonces se le pondera con 0.

A continuación se expone una tabla explicativa con cada atributo y su peso correspondiente, en la que se refleja el argumento arquitectónico analizado para la ponderación del peso que ha sido realizado por el equipo de arquitectura del ERP.

Atributo	Peso	Argumentos
Herramienta Gráfica Windows	3	En este trabajo no es de interés modelar en este sistema operativo, pero es válido poder analizar los resultados del modelado en Windows.
Herramienta Gráfica LINUX	7	Es de interés trabajar en plataforma LINUX el modelado arquitectónico.
Varios tipos de Componentes	10	Por ser uno de los elementos más significativos de los conceptos de una arquitectura de software y de vital importancia en el modelado arquitectónico del mismo.
Varios tipos de Conectores	10	Por ser uno de los elementos más significativos de los conceptos de una arquitectura de software y de vital importancia en el modelado arquitectónico del mismo.
Restricciones	2	A pesar de ser un elemento arquitectónico representativo y substancial en la toma de decisiones arquitectónicas, exige un nivel de madurez en el equipo de desarrollo que se prefiere expresar este elemento arquitectónico en una descripción formal textual de los mismos.
Sub-estructuras	7	Por ser significativo como elemento descriptivo, pero no imprescindible en los conceptos de una arquitectura de software se le asigna un valor alto, pero no el máximo
Dinámica	2	Es importante porque es una característica descriptiva que sirve de apoyo al equipo técnico, para interpretar los flujos de transformación y adaptabilidad de una arquitectura, requiere alto nivel de especialización el análisis de estas características, por lo que no se le asigna un índice numérico alto.
Herramienta con Licencia	4	Porque es de interés trabajar en plataforma libre, pero no es indispensable para realizar la tarea de modelado.

GPL		
Verificación de propiedades	2	El ADL debe dar soporte a las tareas de creación de la arquitectura, así como refinamiento y validación de las mismas. Pero no es una característica indispensable para este trabajo por lo que no se le asigna un índice alto.
Desarrollo y reutilización	5	Porque esta característica aunque no es indispensable, permite reutilizar descripciones en arquitecturas y aumentar la productividad del modelado arquitectónico.
Patrones y Estilos Arquitectónicos	7	Porque es importante poder representar los patrones y estilos arquitectónicos en un modelado arquitectónico. Son de prioridad en la normalización del lenguaje en la comunicación de los arquitectos y el equipo de desarrollo.

Tabla 3 Asignación de pesos a los atributos cualitativos.

A continuación la tabla cuantitativa ilustra los ADL de muestra con los valores asignados según las características de cada uno.

ADL	UniCon	Wright	Rapide	Jacal	Darwin	LEDA	C2	Acme	Aesop
Herramienta Gráfica Windows	3	3	0	3	3	3	3	3	0
Herramienta Gráfica Linux	0	0	7	0	0	0	0	7	0
Varios tipos de Componentes	10	10	10	10	10	10	10	10	10
Varios tipos de Conectores	10	10	0	10	0	0	0	10	10
Restricciones	2	2	2	2	2	2	2	2	2
Sub-estructuras	7	7	7	7	7	7	7	7	7
Dinámica	0	0	2	0	2	2	2	0	0
Herramienta con licencia GPL	0	0	0	0	0	0	0	0	0
Verificación de propiedades	0	2	0	0	2	2	0	0	0

Desarrollo y reutilización (Herencia)	0	0	5	0	0	5	5	0	0
Patrones y Estilos Arquitectónicos	0	0	7	0	7	7	7	7	7
Valor Final	**32**	**34**	**40**	**32**	**33**	**38**	**36**	**46**	**36**

Tabla 4 Instrumento cuantitativo para la selección del ADL.

Para tener una mejor visión de los resultados obtenidos se propone el siguiente gráfico de barras donde se puede observar el nivel de cubrimiento obtenido por cada ADL.

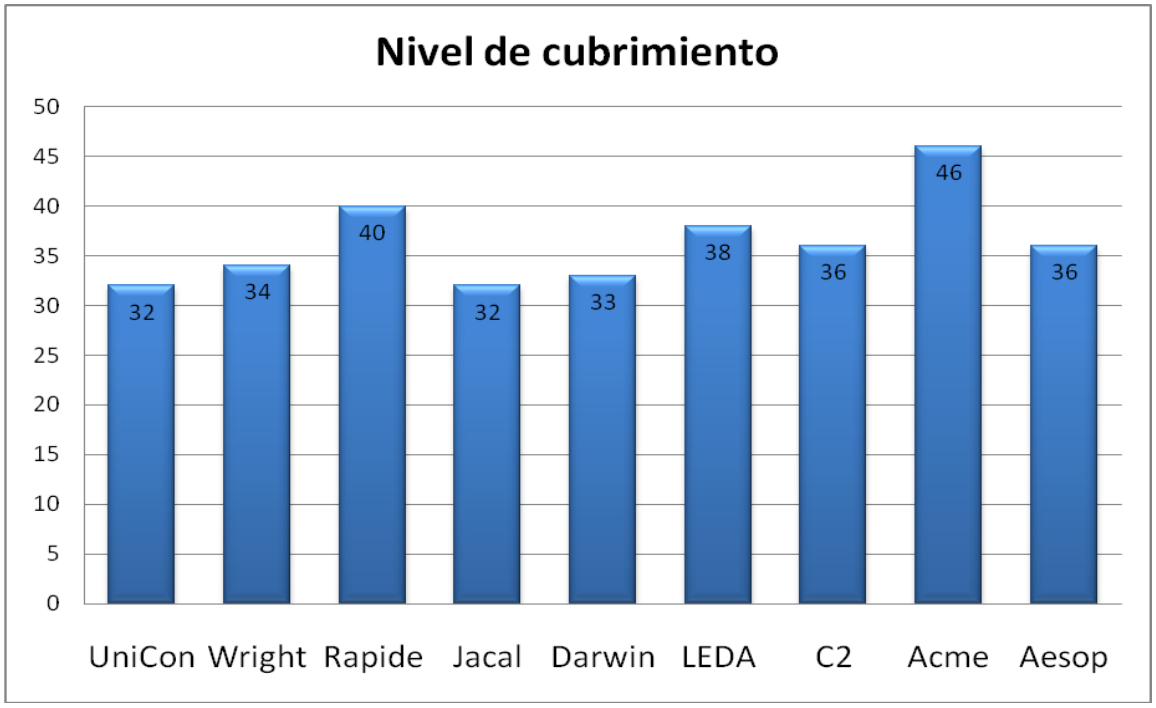


Gráfico 1 Representación del resultado obtenido con el instrumento cuantitativo

Luego de realizada la comparación y analizándose los resultados arrojados por el instrumento utilizado, se observa que el valor más alto lo obtiene el ADL Acme debido a que el mismo cumple con la mayoría de las características evaluadas, siendo así el más adecuado para modelar las decisiones arquitectónicas del proyecto ERP Cuba. Por lo tanto se selecciona para dar cumplimiento a dicha tarea.

2.2 “Características del ADL seleccionado. Pasos básicos de uso. Herramientas y configuración”.

2.2.1 Características del ADL seleccionado. Pasos básicos de uso e interpretación.

Como se evidenció en el epígrafe anterior, el ADL escogido es Acme, en el primer capítulo se hace referencia al mismo en el ***Epígrafe 2. “Referencias de ADL”*** y se exponen algunas de sus características, no obstante, con el objetivo de ofrecer un mayor nivel de comprensión, en este capítulo se lleva a cabo un análisis más detallado del mismo omitiéndose algunos elementos que pueden verse en el epígrafe 2 del capítulo 1 mencionado anteriormente.

Acme es un lenguaje genérico utilizado para describir arquitecturas de software y familias de arquitecturas. Está destinado a ser uniforme en cuanto a representación de herramientas. El mismo proporciona construcciones para describir la arquitectura como sistemas gráficos de componentes interactuando mediante los conectores, cuenta además con un mecanismo de representación de descomposición jerárquica de los componentes y conectores en subsistemas y una manera de describir familias y tipos de elementos en términos de estas construcciones.

Acme está concebido como un lenguaje para describir la arquitectura de un sistema. Si bien no hay acuerdo sobre una definición concreta de diseño arquitectónico, todos los enfoques arquitectónico tienen en común el objetivo de representar un sistema con un alto nivel de abstracción, pero que sea comprensible.

Acme es uno de varios lenguajes de descripción de la arquitectura que se han desarrollado durante los últimos años. Las herramientas de este lenguaje proporcionan una forma de describir arquitecturas formales que ayudan a un diseñador a tomar decisiones de diseño y a razonar acerca de las implicaciones de esas decisiones.

El tipo de modelo utilizado depende de qué tipo de diseño se pretende obtener según las necesidades del diseñador.

Es importante reconocer que aunque Acme está basado en la idea de representar una arquitectura como estructura estática, también puede ser utilizado para representar arquitecturas reconfigurables expresando la posible reconfiguración en términos de las estructuras Acme. Por ejemplo, un sistema podría incluir propiedades que describen componentes que pueden ser añadidos en tiempo de ejecución y cómo adjuntarlos en el sistema actual.

Este lenguaje ofrece una forma de documentar explícitamente la comunicación en un sistema, y la naturaleza exacta para realizar una descomposición en Acme depende del modelo particular que se esté usando.

Acme también puede ser utilizado para representar alternativas de diseño. Una descripción con Acme puede ser utilizada para describir conjuntos de arquitecturas, no sólo una única arquitectura fija. El primer paso para utilizar Acme para cualquier aplicación es comprender los conceptos básicos definidos por la herramienta Acme.

Estos conceptos son:

Componentes:

En Acme los componentes representan a los elementos computacionales y de almacenamientos de un sistema y siempre se definen dentro de una familia. En Acme los componentes son los elementos básicos en la descripción de un sistema. Representan centros de cómputo: los elementos de un sistema responsable de hacer el "trabajo" en un sistema. Esto es en contraste con los conectores que son el modelo de comunicación y la interacción entre los componentes de un sistema.

Interfaz de los Componentes:

Los componentes pueden tener múltiples puntos de interfaces los cuales se llaman puertos, los mismos definen interfaces simples o complejas.

Los componentes exponen su funcionalidad por medio de sus puertos. Un puerto representa un punto de contacto entre el componente y su medio ambiente. Un componente puede tener varios puertos correspondientes a diferentes interfaces para el componente. Un puerto puede utilizarse para representar lo que tradicionalmente se considerada como una interfaz: un conjunto de operaciones disponible en un componente. Pero puede representar una fuente de peticiones o un destino de datos.

Conectores:

Son entidades de primera clase representando interacciones entre los componentes. Como los componentes, los conectores pueden ser utilizados para modelar una variedad de diferentes tipos de interacciones bajo un número de diferentes modelos. Un conector típico podría definir una sincronización modelo para la comunicación, un protocolo de comunicación, un bloqueo modelo, o las características de un canal de comunicación como ancho de banda.

Si bien el conector puede utilizarse para construir, representar algo parecido a un procedimiento sincrónico de llamada entre partes de un programa, es frecuentemente utilizado para capturar una o más interacción entre los componentes.

Interfaces de los Conectores:

Un conector incluye un conjunto de interfaces en forma de funciones, cada una de las cuales define la interacción de un componente y capturados por el conector.
La interfaz de los conectores se definen a partir de los roles, que no son más que el papel que juegan cuando conectan a los componentes.

Estilos:

Acme posee manejo intensivo de familias o estilos. Es una capacidad que está construida como una jerarquía de propiedades correspondientes a los tipos. En Acme se consideran tres clase de tipos: tipos de propiedades, tipos estructurales y estilos. Los tipos estructurales representan conjuntos de elementos estructurales y una familia o estilo representa un conjunto de sistemas. Una familia en Acme se define especificando tres elementos: un conjunto de tipos de propiedades y tipos estructurales, un conjunto de restricciones y una estructura por defecto, que prescribe el conjunto mínimo de instancias que deben aparecer en cualquier sistema de la familia.
A continuación se muestra un ejemplo explicado donde se define una familia simple de tuberías y filtros. Este estilo se relaciona con las arquitecturas de flujo de datos y es muy usado en compiladores, flujos de datos, procesamiento de señales y programación funcional. Es un estilo primitivo que facilita el entendimiento de la sintaxis del código de Acme :

Una familia Acme incluye un conjunto de tipos de componentes, conectores, puertos (port) y rol que definen el vocabulario propio del estilo.
Family PipeFilterFam = {
Declara tipos de componentes. Una definición de tipo de componente en Acme permite establecer la estructura requerida por el tipo. Esta estructura se define mediante la misma sintaxis que la instancia de un componente.
Component Type FilterT = {

Todos los filtros definen por lo menos dos puertos.

Ports { stdin; stdout; };

Property throughput : int;

};

Extiende el tipo básico de filtro con una subclase (herencia).Las instancia de WindowsFilterT tendrán todas las propiedades y puertos de las instancias de FilterT, más un puerto stderr y una propiedad implementationFile.

Component Type WindowsFilterT extends FilterT with {

Port stderr;

Property implementationFile : String;

};

Declara el tipo de conector de tubería. Igual que los tipos de componente, un tipo de conector también describe la estructura requerida.

Connector Type PipeT = {

Roles { source; sink; };

Property bufferSize : int;

};

Declara algunos tipos de propiedad que pueden usarse en sistemas del estilo PipeFilterFam

Property Type StringMsgFormatT = Record [size: int; msg:String;];

Property Type TasksT = enum {sort, transform, split, merge};

};

Tabla 4 Representación del estilo de tuberías y filtros en Acme.

Representación

Los elementos en Acme pueden tener más de una representación, cada uno de los cuales corresponde a una vista conceptual diferente o uno de varios modos alternativos de descomponer un elemento. La representación en Acme no define explícitamente la naturaleza de la relación entre un elemento y sus representaciones, en su lugar, al igual que otros conceptos de Acme, la naturaleza exacta de la representación depende del contexto particular en que se esté trabajando.

2.2.2 Herramienta gráfica escogida para el modelado con Acme. Configuración, instalación y uso.

Para el modelado de arquitecturas con Acme está disponible la herramienta gráfica AcmeStudio, la cual ha sido desarrollada en la plataforma Eclipse en el lenguaje de programación Java. La misma tiene soporte tanto en Linux como en Windows en su versión 3.4.0 que será la usada en este trabajo.

A continuación se muestran dos imágenes con la interfaz gráfica de esta herramienta, la primera es su entorno de trabajo en la forma de diseño o de modelado y la segunda el código correspondiente de la descripción arquitectónica :

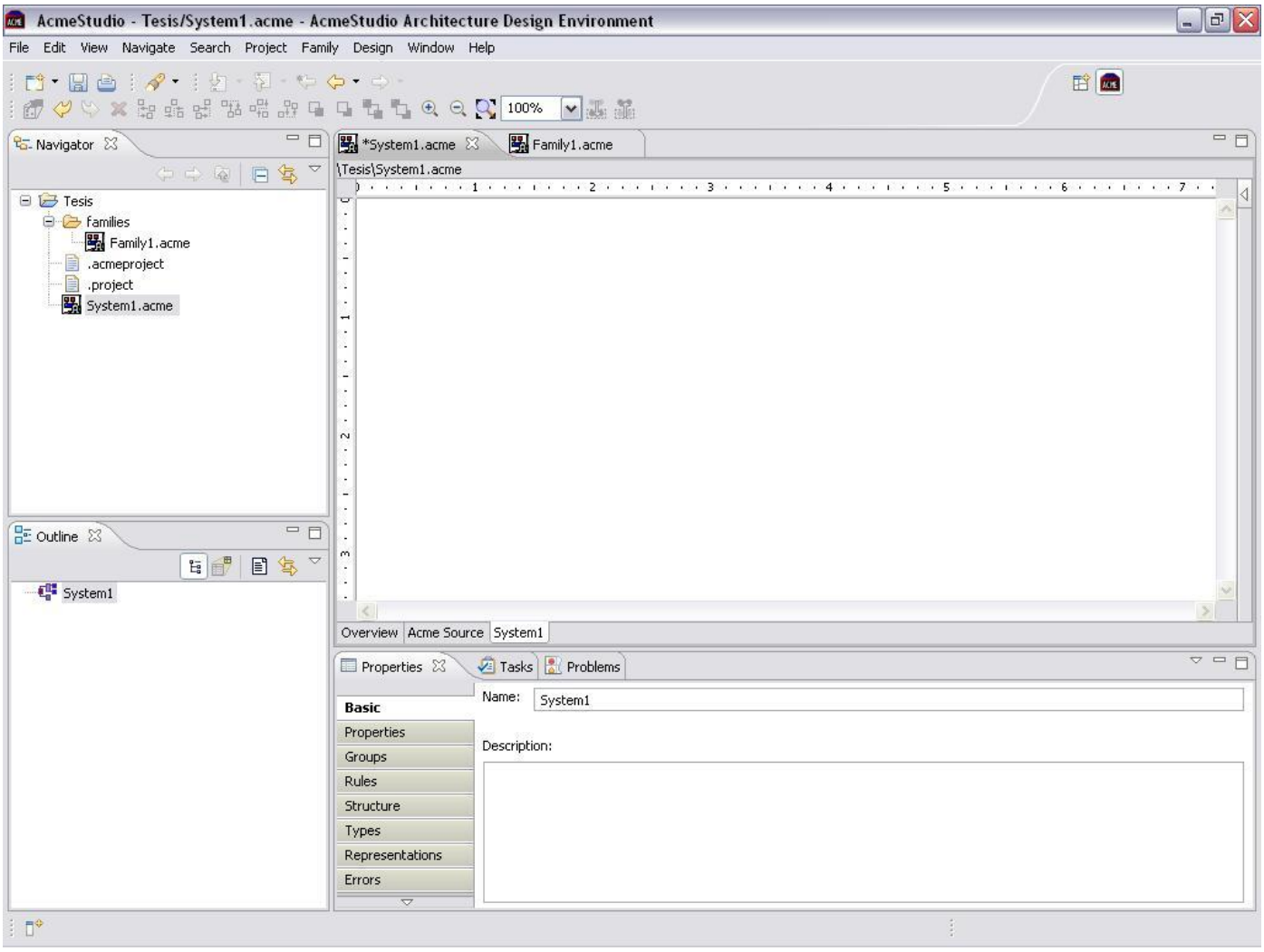


Ilustración 1Entorno de Modelado del Acme Studio 3.4.0

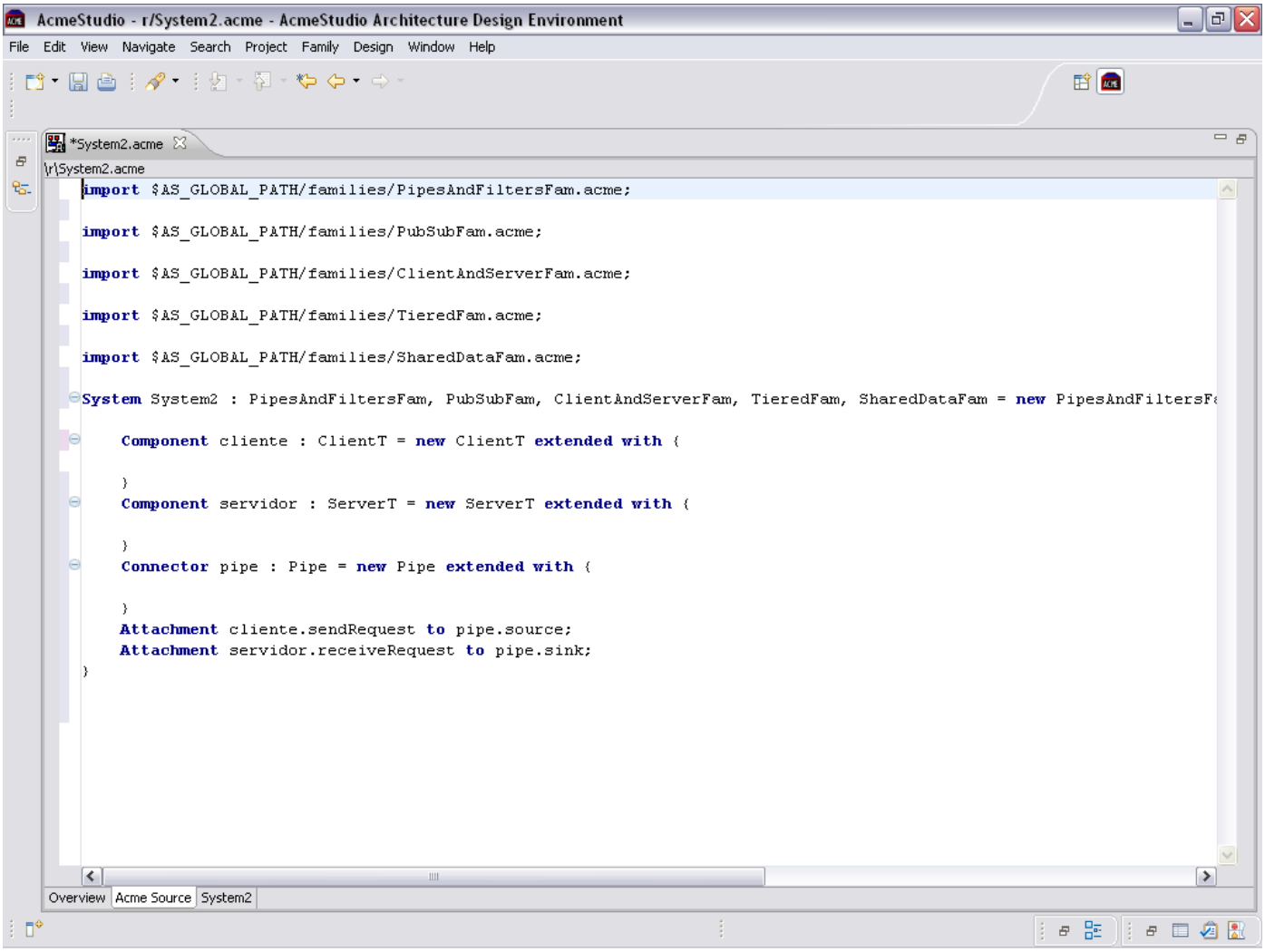
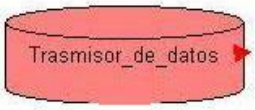


Ilustración 2 Ejemplo de código detrás del modelado en Acme

AcmeStudio permite definir todos los conceptos del ADL Acme de forma correcta, es una herramienta fácil de usar y entender ya que proporciona una ayuda muy sustancial y además existe documentación para esta herramienta que se puede descargar de forma gratuita en su página oficial : http://acme.able.cs.cmu.edu/acmeweb/

A continuación se muestra una figura con la interfaz de los componentes, puertos, conectores y roles que propone la herramienta:

Componentes	Definición
	Componente General
	Nodo Cliente
	Cliente T
	Filtro
	Nodo Lógico T
	Nodo de Datos
	Emisor de datos
	Receptor de datos
	Filtro binario
	Componente de Seguridad

	Participante T

Tabla 5 Representación de componentes en AcmeStudio 3.4.0

Puertos	Definición
	Puerto de Uso
	Puerto Cliente
	Puerto de Anuncio
	Puerto de Entrada
	Puerto de Salida
	Puerto de Recibo
	Puerto Proveedor
	Puerto de Uso Remoto
	Puerto Proveedor Remoto T
	Puerto proveedor T
	Puerto Estándar

Tabla 6 Representación de los puertos de los componentes en AcmeStudio 3.4.0

Conectores	Definición
	Conector de Acceso a Datos
	Conector de Bus de Eventos
	Conector de llamada de retorno
	Conector de Tuberías
	Conector Estándar
	Conector Local
	Conector Tipo CS
	Conector Tipo RMI

Tabla 7 Representación de conectores en AcmeStudio 3.4.0

Roles	Definición
	Rol Cliente
	Rol Envío
	Rol Estándar
	Rol Fuente
	Rol Proveedor
	Rol Proveedor Remoto
	Rol Proveedor T
	Rol Publicador
	Rol Servidor
	Rol Subscriptor
	Rol Usuario
	Rol Usuario Remoto T
	Rol Usuario T

Tabla 8 Representación de los roles(interfaz) de los conectores en AcmeStudio 3.4.0.

2.2.3 Pasos básicos para el uso de la Herramienta AcmeStudio 3.4.0

Para poder usar AcmeStudio 3.4.0 se necesita tener instalada la maquina virtual de java para luego ejecutar el archivo AcmeStudio.exe contenida en el paquete de la herramienta.

A continuación se explicarán los pasos básicos para el uso de AcmeStudio 3.4.0.

Una vez que se haya ejecutado el archivo AcmeStudio.exe, el siguiente paso será crear un nuevo proyecto, para esto se va a la barra de tarea de la herramienta y se escoge **File**, luego la opción **Acme**

Project, se crea un nuevo proyecto con el nombre deseado y se presiona el botón **Finish** para terminar**.** (Ver anexo 1).

Una vez que aparezca en el navegador de la herramienta el nuevo proyecto con el nombre seleccionado, el siguiente paso será crear un nuevo sistema, para esto se da clic derecho sobre el proyecto y se escoge la opción Acme System, posteriormente aparecerá una ventana para nombrar el sistema, se presiona el botón siguiente y en la nueva ventana se escoge la familia AcmeStudio que se usará según el tipo de componente, puerto, conectores y roles que sean útiles y necesarios para conformar el modelado. Una vez seleccionada la familia aparecerá en el navegador de la herramienta el nuevo sistema creado, el próximo paso será comenzar a modelar. (Ver Anexo 2).

Para crear un nuevo componente se presiona el clic derecho en el editor y se escoge la opción New Component, luego se selecciona el tipo de componente, se nombra y se presiona el botón Ok, posteriormente aparecerá el componente en el editor gráfico.(Ver Anexo 3).

Para crearle los puertos a los componentes se presiona clic derecho sobre el componente y en la ventana que sale se escoge el tipo de puerto y se nombra.(Ver Anexo 4).

Para crear los conectores se realiza la primera operación antes de crear un componente lo que se escoge New conector, luego en la ventana que aparece escoges el tipo de conector y se nombra.(Ver Anexo 5).

Para crearle los roles a los conectores se presiona clic derecho sobre el conector y se escoge la opción new rol, luego en la ventana que aparece se escoge el tipo de rolo y se nombra. Cuando se termine de crear el rol habrá que conectarlo al puerto compatible con el mismo, en el caso de no realizar correctamente la conexión AcmeStudio notificara un mensaje de error. (Ver Anexo 6).

Usualmente los componentes están compuestos por otros componentes, en AcmeStudio también es posible modelar esta situación donde un componente encapsula a otros de la siguiente forma: se presiona el clic derecho sobre el componente compuesto y se escoge la opción Add Representation, aparecerá la representación del componente donde se crearan los nuevos componentes.(Ver Anexo 7).

Cuando se dan casos como estos, los componentes internos se relacionan con el exterior a través del componente que lo encapsula por medio de un puerto externo; con esta herramienta también es posible modelar esta situación , la solución que da AcmeStudio, es crearle un puerto al componente interno con el mismo nombre del puerto externo y unirlos a través de una función que se llama bindings, esta operación hay que realizarla directamente en el código , para ello se presiona clic izquierdo sobre la

pestaña nombrada Acme Source y se busca la función que implementa la representación del componente y dentro se escribe la función bindings.(Ver Anexo 8).
A continuación se mostrará la estructura de un Bindings :

Bindings

{ *<Puerto Externo>* ***To*** *<Componente Interno>.<Puerto del Componente Interno> ;* ***}***

Para salir de la representación existen dos vías, la primera es presionar el clic derecho sobre la representación y escoger la opción **Navigate up Representation**(navegar fuera de la representación) o ir a la barra de herramientas y escoger la opción con ese mismo nombre. (Ver Anexo 9).

Existe también otra vía para crear los elementos necesarios para modelar una arquitectura, es a través una pestaña con el nombre de **Palette**, en la misma se pueden encontrar dos pestañas más, una con el nombre de **Architecture** donde están los componentes, conectores, puertos y roles, y otra denominada **Anotations**, esta última agrupa elementos útiles para hacer anotaciones. Para insertar los elementos en el modelado por esta vía solamente hay que escoger el elemento y trasladarlo hasta el editor gráfico. En el (anexo 10), se muestran tres imágenes donde se pueden apreciar los espacios anteriormente mencionados.

Conclusiones del capítulo

Después de realizar un estudio de los ADL expuestos en el capítulo 1 de este trabajo de diploma, y realizados los instrumentos cualitativos y cuantitativos a partir de los atributos seleccionados cuidadosamente que cumplen con las expectativas de las decisiones arquitectónicas de proyecto de software ERP Cuba, se logró obtener el ADL y la herramienta de modelado adecuados. De esta forma se culminó una de las tareas fundamentales de esta tesis que es la selección de un lenguaje de descripción arquitectónica y su herramienta de modelado para describir la arquitectura del proyecto ERP Cuba.

Capítulo 3. “Modelado de la arquitectura de ERP Cuba”

3

CAPITULO 3. “Modelado del proyecto ERP utilizando el ADL seleccionado”.

Introducción

Como se pudo evidenciar en el capítulo anterior, el ADL seleccionado fue Acme el cual cuenta con la herramienta de modelado AcmeStudio 3.4.0. Dada esta selección el próximo paso será modelar las decisiones arquitectónicas del proyecto ERP Cuba, el cual está conformado por 12 subsistemas principales, constituidos a su vez por varios componentes cada uno. Por la complejidad, la dimensión y el nivel de terminación de las decisiones arquitectónicas se modelaran solo 9 de estos subsistemas. Este capítulo esta estructurado por 3 epígrafes, en el epígrafe 1 se caracterizan cada uno de los subsistemas a modelar en cuanto al papel que cumplen así como los componentes que lo conforman, en el epígrafe 2 se explica como se aplicó el instrumento metodológico definido por el equipo de arquitectura el cual tiene como objetivos exponer la integración de los subsistemas seleccionados para el modelado y servir de apoyo para la descripción arquitectónica. Por último en el epígrafe 3 se expone como se realizó el modelado basado en Acme y se muestran los resultados obtenidos.

3.1 “Características generales de los subsistemas a modelar”

El sistema está compuesto por 12 subsistemas principales, cada uno responde a procesos de negocio, un conjunto de funcionalidades y requerimientos del cliente. Todos estos subsistemas interactúan y comparten datos de interés en dependencia de la funcionalidad de cada uno, y siguiendo un estricto régimen de seguridad de la información. Cada uno de ellos cuenta con componentes internos que lo hacen a su vez otro sistema.

Los subsistemas a modelar son: Estructura y Composición, Logística el cual contiene a Inventario, Contabilidad, Costos y Procesos, Capital Humano, Planificación y por último Configuración dentro del cual

se encuentra Multimoneda. A continuación se explican cada uno de ellos, teniendo en cuenta su nombre así como el papel que desempeñan, además de los componentes claves que lo conforman.

Subsistema Estructura y Composición

El módulo estructura y composición es el encargado de diseñar una estructura, el mismo debe gestionar estructuras dentro de ellas Niveles 1, Agrupaciones, Entidades y Unidades; gestionar áreas de cada unidad y gestionar cargos de esas áreas. El mismo brinda servicio a casi todos los subsistemas existentes. Su componente es Estructura y composición pues él se comporta como componente dentro del sistema en general pero debido a que es extenso se ha tratado como un subsistema.

Subsistema Logística/ Inventario

El subsistema Inventario es quien se encarga de los medios materiales.

Componentes:

Documento: Encargado de la gestión de documentos.

Producto: Encargado de la gestión de los productos.

Recepción: Es aquí donde se recepcionan los documentos de los productos que entran al almacén. Apertura: Encargado de realizar el inventario inicial de los productos que llegan.

Inventario: Encargado de realizar inventario parcial o general del almacén.

Baja: Si es necesario dar baja a un producto del almacén por alguna causa es aquí donde se realiza esta operación.

Conciliación: Es donde se legaliza que los productos que hayan sido transferidos lleguen correctamente a su destino.

Generación de salida: Permite crear el documento de salida de un producto.

Ajuste: Es donde se realizan los ajustes de inventarios en caso de que sea necesario(faltantes, sobrantes).

Despacho: Se realizan las ordenes de despacho.

Nomenclador: Es aquí donde se definen los nomencladores.

Recuperaciones: Permite realizar recuperaciones en caso de que sea necesario, permitiendo un mejor trabajo.

Configuración: Es aquí donde se configuran los productos que no sean estándar.

Submayor: Permite ver los movimientos de los productos(entrada/ salida).

Cierre: Se realizan los cierres de operaciones.

Subsistema Contabilidad

El Módulo de Contabilidad General está formado por un conjunto de componentes que encapsulan funcionalidades que representan el núcleo del sistema de planificación de recursos empresariales. Los mismos se interrelacionan entre sí y con otros componentes de otros subsistemas para dar respuesta a las necesidades que urgen a las empresas del país de contar con un único sistema que norme las actividades contables y financieras, y que permita tener un óptimo control de los recursos.

Componentes:

Recuperaciones: Este componente es el encargado de gestionar todos los reportes del módulo, dando la información necesaria y oportuna para a toma de decisiones.

Cierre: Es el encargado de cerrar los períodos y ejercicios contables, y a la vez darle reapertura a los próximos.

Comprobantes de Operaciones: Este componente es el encargado de contabilizar los comprobantes de operaciones de todos los subsistemas del ERP y los propios.

Nomenclador de cuenta: Este componente es el encargado de gestionar el árbol de cuentas en el mayor sentido de la palabra, entiéndase las cuentas y sus aperturas, sean del tipo que sean, controladas por otros subsistemas.

Configuración: En este componente se gestionan una serie de nomencladores que son la base para el componente nomenclador de cuentas, entiéndase grupos de cuentas, contenidos económicos de las cuentas y estructuras económicas que son los propietarios de esas cuentas.

Subsistema Costo y Proceso

La contabilidad general permite controlar de la forma más abarcadora las principales funciones de la entidad como son las finanzas, la administración, la producción y la venta, distribución, etc.; sin embargo es necesario no solo conocer los resultados generales, sino la eficiencia con que se logran dichos resultados. El estado de resultado como parte de los estados financieros brinda la información de la relación existente entre los ingresos y gastos de la actividad que realiza la entidad, del volumen de las ventas y de los gastos en un periodo determinado. El registro de los gastos y el análisis de su comportamiento es materia de la que se encarga la contabilidad de costos.

El costo constituye un elemento normativo y evaluador de la gestión de la entidad, de aquí su importancia como herramienta de dirección por lo que se requiere por parte del personal dirigente, el dominio de los aspectos que lo caracterizan, su contenido y características. Es importante destacar que la determinación de los costos no corresponde solo a las empresas, debiendo constituir actividades obligadas en aquellas unidades presupuestadas autofinanciadas o no, que tengan autorizadas actividades productivas y comerciales con peso económico significativo. El subsistema de costos por lo tanto es el encargado de automatizar toda la gestión de los costos en la entidad que lo utilice, pudiendo medir la suma de gastos de toda naturaleza expresada monetariamente, que se aplica a una producción o servicio determinado de la entidad que lo utilice.

Componentes:

Recuperación: Se realizan un serie de reportes necesarios de todas las operaciones realizadas por el subsistema contabilidad de costos para la información y la toma de decisiones.

Ajuste y Costo: Dadas las operaciones a veces es necesario ajustar los costos obtenidos por diversas razones, y es aquí donde se realizan esas acciones.

Cierre y Traspaso: Se realizan todos los cierres de período y de ejercicios contables de la entidad, además se realizan operaciones como traspasos de gastos.
Configuración y Nomencladores: Componente donde se definen las configuraciones y nomencladores necesarios para la adecuada contabilidad de costos.

Subsistema Capital Humano

El subsistema Capital Humano está compuesto por el módulo Gestión de trabajadores y el módulo Nómina. El primero de estos se encarga de adicionar, eliminar, modificar y consultar todos los datos referente a los trabajadores, además de gestionar los datos de los familiares de estos trabajadores entre otros. Mientras que el módulo Nómina se encarga principalmente de gestionar lo referente a conceptos de pago, y todas las demás funcionalidades que tributan de una forma u otra a la nómina de cada trabajador.

Componentes:

Persona: Administra todos los datos de la persona.

Trabajador: Administra datos contables así como los movimientos de alta, baja y/o reubicación en un puesto de trabajo.

Puesto de Trabajo: Gestiona los puestos de trabajo asociados a un cargo que pertenecen a un área determinada.

Pagos Adicionales: Administra los pagos adicionales de un trabajador y los asocia a un puesto de trabajo.

Incidencias: Registra las incidencias de un trabajador y por un período de pago.

Nómina: Procesa la nómina de un área de trabajo, teniendo en cuenta las incidencias y los pagos adicionales de los trabajadores.

Submayores: Actualiza los submayores de vacaciones y retenciones de un trabajador, y verifica que el saldo del submayor coincida con el saldo de contabilidad.

Carga: Gestiona los pagos adicionales y guarda los datos de un trabajador.

Subsistema Planificación

Capítulo 3. “Modelado de la arquitectura de ERP Cuba”

La arquitectura del sistema de Planificación está compuesta por un conjunto de componentes en los cuales se agrupan los requisitos funcionales para dar una solución de forma genérica a la planificación ya sea a nivel de empresa o a nivel de entidades presupuestadas.

Componentes:

Construcción Planes: Se definen ejercicios como, la configuración del plan y se ordenan las etapas asociadas a esa configuración. Además se crean los modelos.
Importar Exportar: Aquí es donde se realiza el calculo de necesidades.
Indicadores: Es el encargado de gestionar los indicadores.
Nomencladores: Es el encargado de gestionar los nomencladores: tipo plan, etapa, modelo, tipo celda, estado modelos y columna.
Plantilla: En este componente se realizan operaciones como Adicionar Indicador, Adicionar columna y Gestionar plantilla, donde se clasifican las celdas (editada (por defecto) o calculada).
Realización_de_Plan: Es el encargado de gestionar las restricciones y llenar el modelo Construcción de plan. Es aquí donde se cambia el estado de los modelos y se cambia de etapa si los modelos están confirmados.

Subsistema Configuración

Se encarga de centralizar aquellos componentes que son utilizados por todos los subsistemas del sistema ERP Cuba por lo que brinda servicio a la mayoría de los subsistemas existentes.

Componentes:

Comprobante Tipo, Subsistemas, Ejercicio_Período, Multimoneda(es además un Subsistema del ERP Cuba), Formato, Nomencladores, Fecha y Cierre.

3.2 “Integración entre los Subsistemas. Matriz de Integración”

Antes de modelar las decisiones arquitectónicas del proyecto ERP Cuba, el equipo de arquitectura decidió definir un instrumento metodológico donde se expone la integración de los subsistemas del proyecto.
Este instrumento consiste en una matriz donde se muestran las entradas y salidas de servicios así como el contrato de integración. Por cada subsistema se elaboraron dos de estas matrices, una para la

interacción externa con el resto de los subsistemas y otra para la interacción de sus componentes internos.

Para conformar cada matriz se utilizó la herramienta Microsoft Office Excel 2007 y como fuente de expresión tecnológica el XML[23] que genera el marco de trabajo del proyecto ERP Cuba con el nombre de IOC, en el mismo se encuentra cada subsistema con los servicios que brindan, estos servicios tienen el siguiente formato mostrado en la tabla 9:

Subsistema / Componente	Aquí aparece el nombre del subsistema seguido por el nombre del componente interno que brinda el servicio.
Nombre del Servicio	Este nombre es el identificador del servicio que se expone al reto de los subsistemas encapsulado dentro de su interfaz.
Interfaz	Esta es la interfaz que contiene el servicio Una interface puede contener varios servicios.
Método	Es la función que realiza el servicio , el mismo requiere una serie de atributos para realizar operaciones que retornan un valor o resultado.
Atributos	Son los atributos que se requieren en el método del servicio.
Resultado	Es el resultado que arroja el método del servicio.

Tabla 9 Formato de los servicios en el IOC.

Es importante señalar que estas matrices se realizaron con los datos existentes en el IOC hasta el momento, por lo cual puede ser tarea de trabajos posteriores dar continuidad o actualizar estas matrices, ya que el proyecto ERP está actualmente en desarrollo y no están terminadas todas las líneas que lo

[23] XML: Lenguaje de marcas extensibles que permite definir la gramática de lenguajes específicos.

conforman. Las mismas representan un artefacto significativo para este trabajo por lo tanto se decidió incluirlas en los anexos. (Ver anexos del 11 al 16).

3.3 "Modelado basado en ADL del Sistema ERP Cuba"

Para modelar las relaciones entre los subsistemas mencionados anteriormente usando el ADL seleccionado (Acme) y basándose en las matrices de integración referenciadas anteriormente se siguió la siguiente estrategia:

Por cada Subsistema se realizó una descripción arquitectónica externa donde se modelaron las relaciones de este con el resto de los subsistemas basándose en las entradas y salidas de los servicios, y una descripción interna donde se representó la interacción entre los componentes que integran al subsistema, igualmente basándose en las entradas y salidas de servicios.

Para denominar los componentes se usaron los nombres de los componentes que integran el proyecto ERP Cuba.

Para denominar los puertos de los componentes, se usó el nombre del componente que recibe o envía el servicio con los comienzos In y Out respectivamente. Es importante especificar que dentro de los puertos que brinda AcmeStudio, los usados fueron el puerto Input(entrada) y el puerto Output(salida), ya que son los más adecuados para modelar entradas y salidas de servicios.

Para denominar a los conectores (como la interacción entre los componentes es a través de servicios y los conectores son los encargados de relacionar a los componentes) se usó el nombre de la interfaz de la cual proviene el servicio.

Para denominar los roles se siguieron los siguientes pasos:

Para el rol que va conectado al puerto de salida se colocó el nombre del subsistema, seguido por el nombre del componente que brinda el servicio, la estructura sería :(***subsistema/componente que brinda el servicio***).

Para el rol que va conectado al puerto de entrada se colocó el nombre del subsistema que encapsula al componente que da el servicio seguido por el nombre de este, luego la palabra To (del Inglés que significa: a(destino)), y posteriormente el nombre del subsistema que encapsula al componente que recibe el servicio seguido por el nombre de este, la estructura sería (***Subsistema1/componente que brinda el servicio*** **To** ***Subsistema2/Componente que recibe el servicio***).

Capítulo 3. “Modelado de la arquitectura de ERP Cuba”

La herramienta AcmeStudio permite especificar textualmente las propiedades de cada elemento, aprovechando esta opción, en los conectores se especifican todos los servicios que brindan, y en cada rol los servicios que se envían por él.

A continuación se muestran imágenes con los resultados del modelado de las decisiones arquitectónicas del proyecto ERP Cuba basado en el ADL seleccionado. Como se puede apreciar solo se muestran las interacciones entre los subsistemas desde una vista de sistema externa ya que las interacciones entre los componentes internos de cada subsistema forman parte de los anexos (Ver anexo 17).

Capítulo 3. “Modelado de la arquitectura de ERP Cuba”

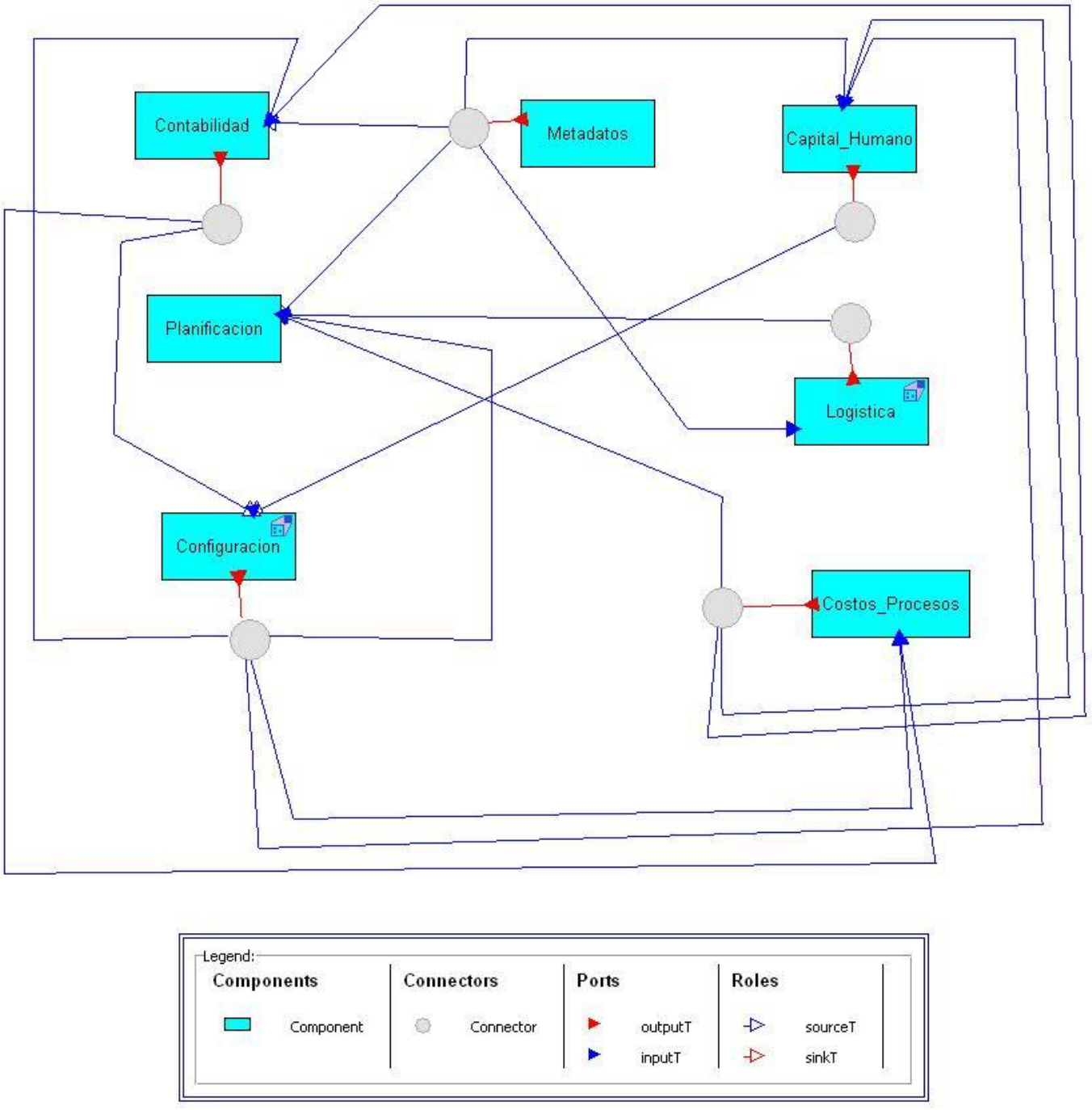


Ilustración 3 Mapa de Interacción general entre todos los Subsistemas

Capítulo 3. “Modelado de la arquitectura de ERP Cuba”

La figura anterior expone el Mapa de Interacción general entre todos los subsistemas donde cada uno representa un flujo de proceso funcional del ERP .El diagrama expresa como dato significativo que los subsistemas de mayor relevancia arquitectónica son Metadatos y Configuración con 4 dependencias en el nodo de integración del puerto de salida de cada uno, y Contabilidad y Costos_Procesos con 2 dependencias respectivamente. El menos significativo es Planificación del que no se visualiza dependencia arquitectónica del resto de los subsistemas arquitectónicos con respecto a este.

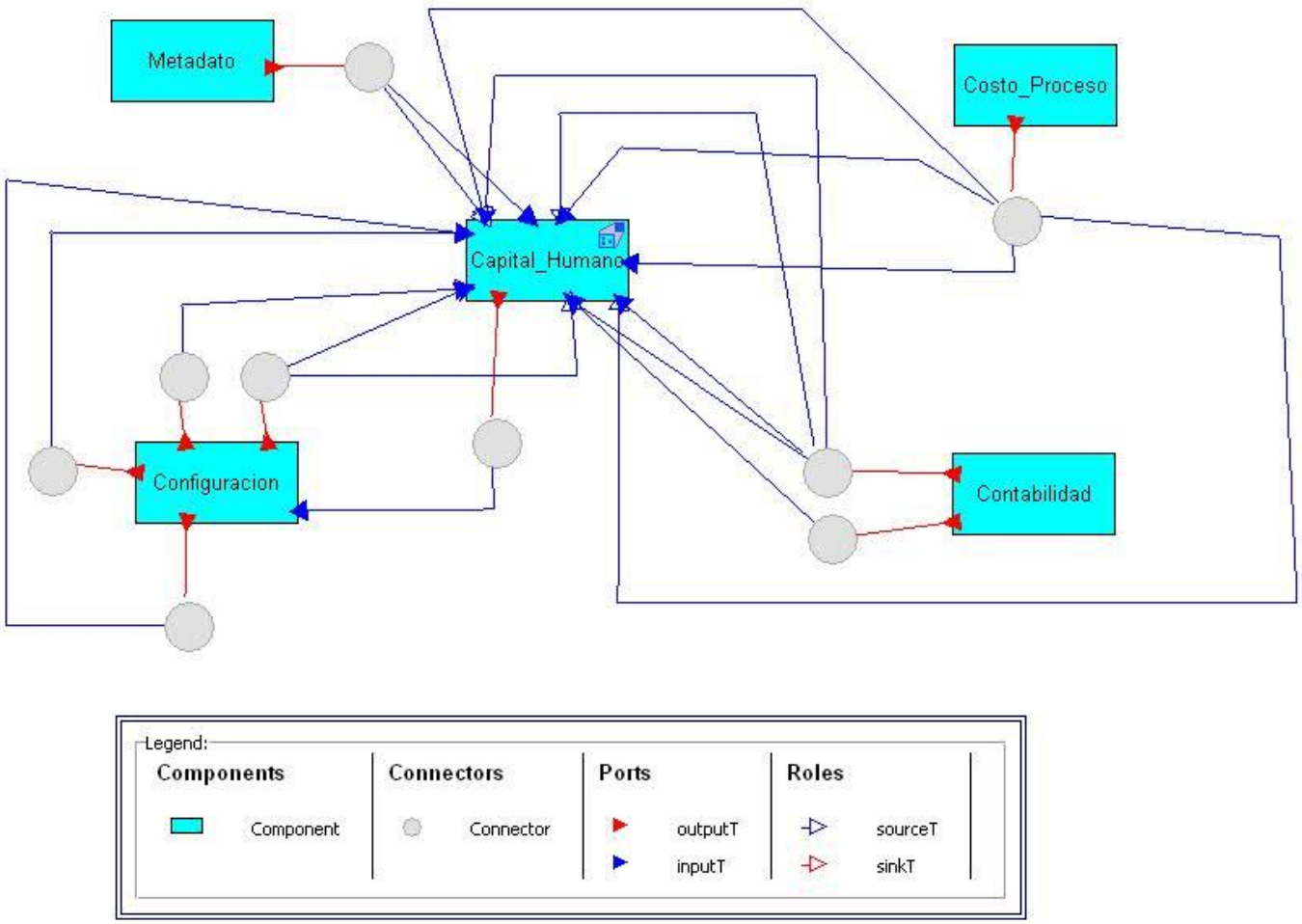


Ilustración 4 Mapa de interacción externa(Capital Humano)

La figura anterior expone el Mapa de interacción externa del subsistema Capital Humano donde se aprecia la relación de este con 4 subsistemas. Este diagrama expresa como dato significativo que los

subsistemas con mayor influencia sobre él son Configuración y Contabilidad con 5 dependencias en los nodos de integración de los puertos de salida de cada uno, y Costos_Procesos con 4 dependencias. El menos significativo es Metadatos con 2 dependencias solamente. Otro dato significativo observable es que existen subsistemas con más de un puerto de salida como es el caso de Configuración con 4 y Contabilidad con 2. Se puede visualizar además que el subsistema Capital Humano no presenta gran influencia sobre el resto de los subsistemas, solamente tiene una dependencia con Configuración desde el nodo de su único puerto de salida.

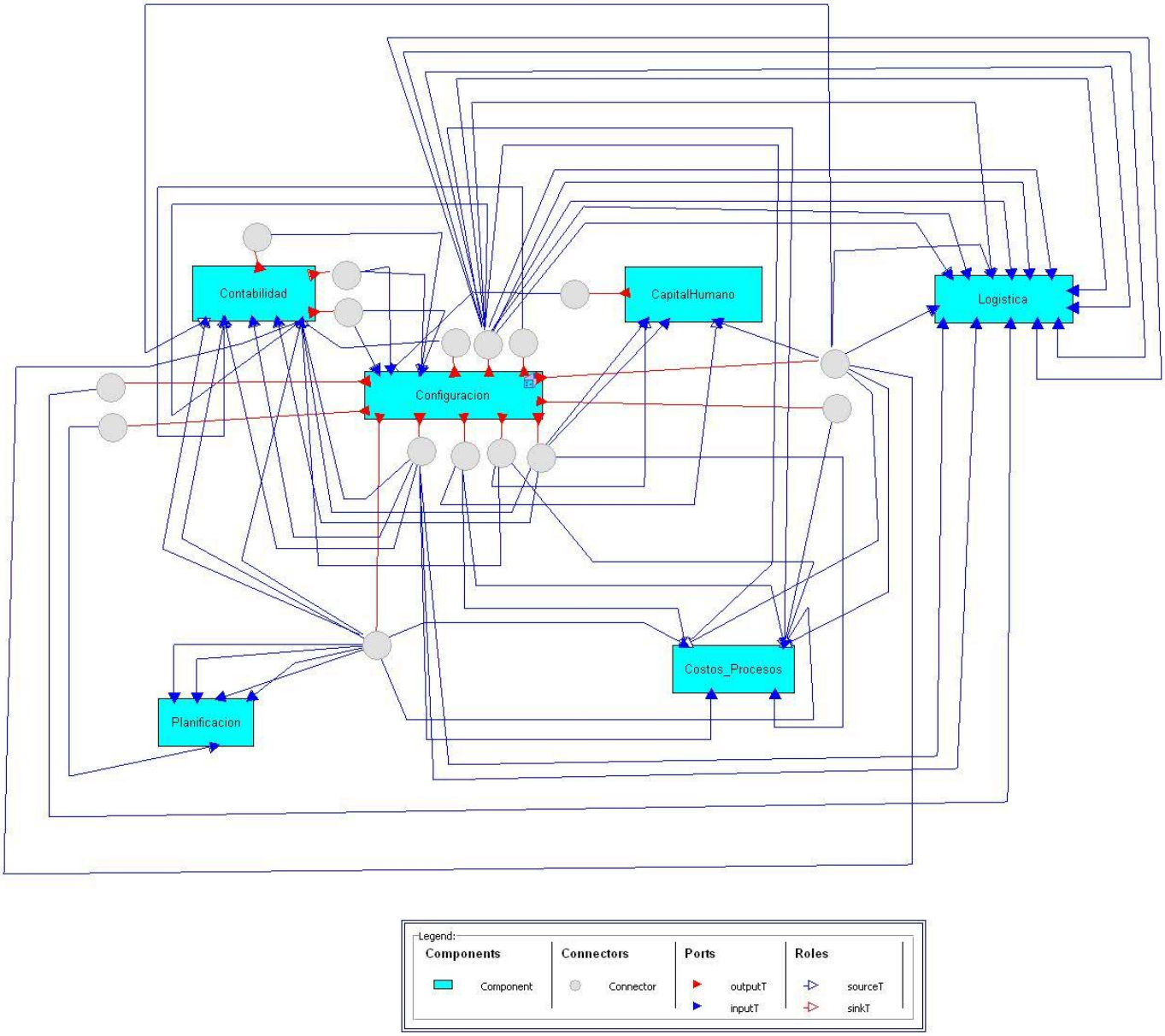


Ilustración 5 Mapa de interacción externo(Configuración)

La figura anterior expone el Mapa de interacción externa del subsistema Configuración donde se aprecia la relación de este con 5 subsistemas. En el mismo se puede observar que los únicos subsistemas que presentan influencia sobre él son Contabilidad con 5 dependencias en los nodos de integración de sus 3 puertos de salida y Capital Humano con una desde su único nodo de salida. Lo más significativo en este

diagrama es que Configuración tiene 8 puertos de salida y 8 nodos de los cuales tienen dependencia los 5 subsistemas(Logística con 16, Contabilidad con 11 al igual que Costos_Procesos, y Capital Humano y Planificación con 5).

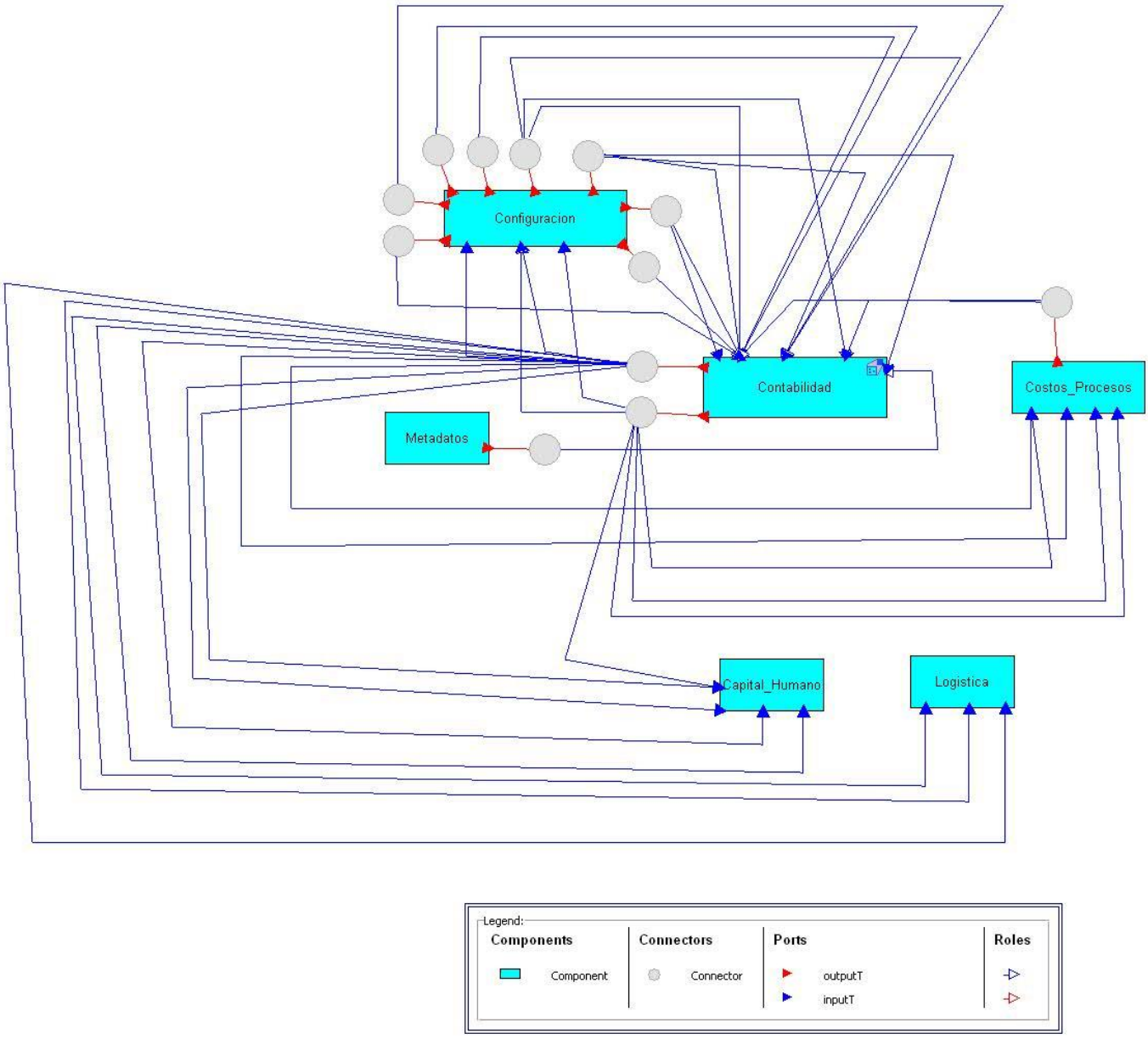


Ilustración 6 Mapa de interacción externo(Contabilidad)

La figura anterior expone el Mapa de interacción externa del subsistema Contabilidad donde se aprecia la relación de este con 5 subsistemas. En el mismo se puede observar como dato significativo que el

subsistema que más influye en el es Configuración ,con el cual tiene 11 dependencia de los nodos de sus 8 puertos de salida. Por otra parte presenta una única dependencia de Metadato y sucede lo mismo con Costos y Procesos. Además de este dependen Capital Humano, Logística y Costos y Procesos, con 5, 3 y 5 dependencias respectivamente.

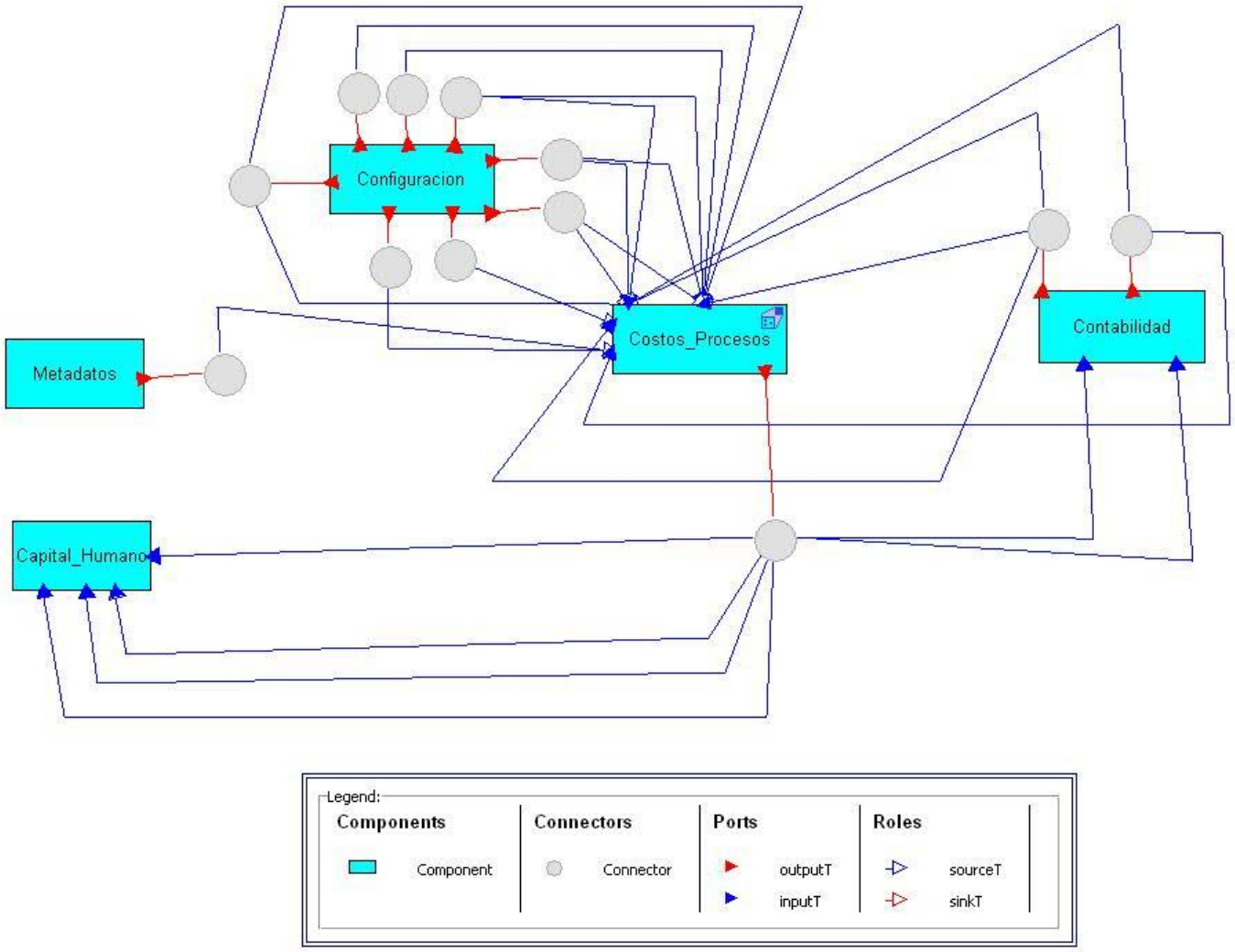


Ilustración 7 Mapa de interacción externo(Costos y Procesos)

En la figura anterior se muestra le interacción del subsistema Costos y Procesos con el resto. El mismo brinda 5 servicios por el nodo de su único puerto de salida, 2 a Contabilidad y 3 a Capital Humano.

Además como dato significativo se observa que tiene 11 dependencias de Configuración y 4 de Contabilidad, mientras de Metadatos solamente 1.

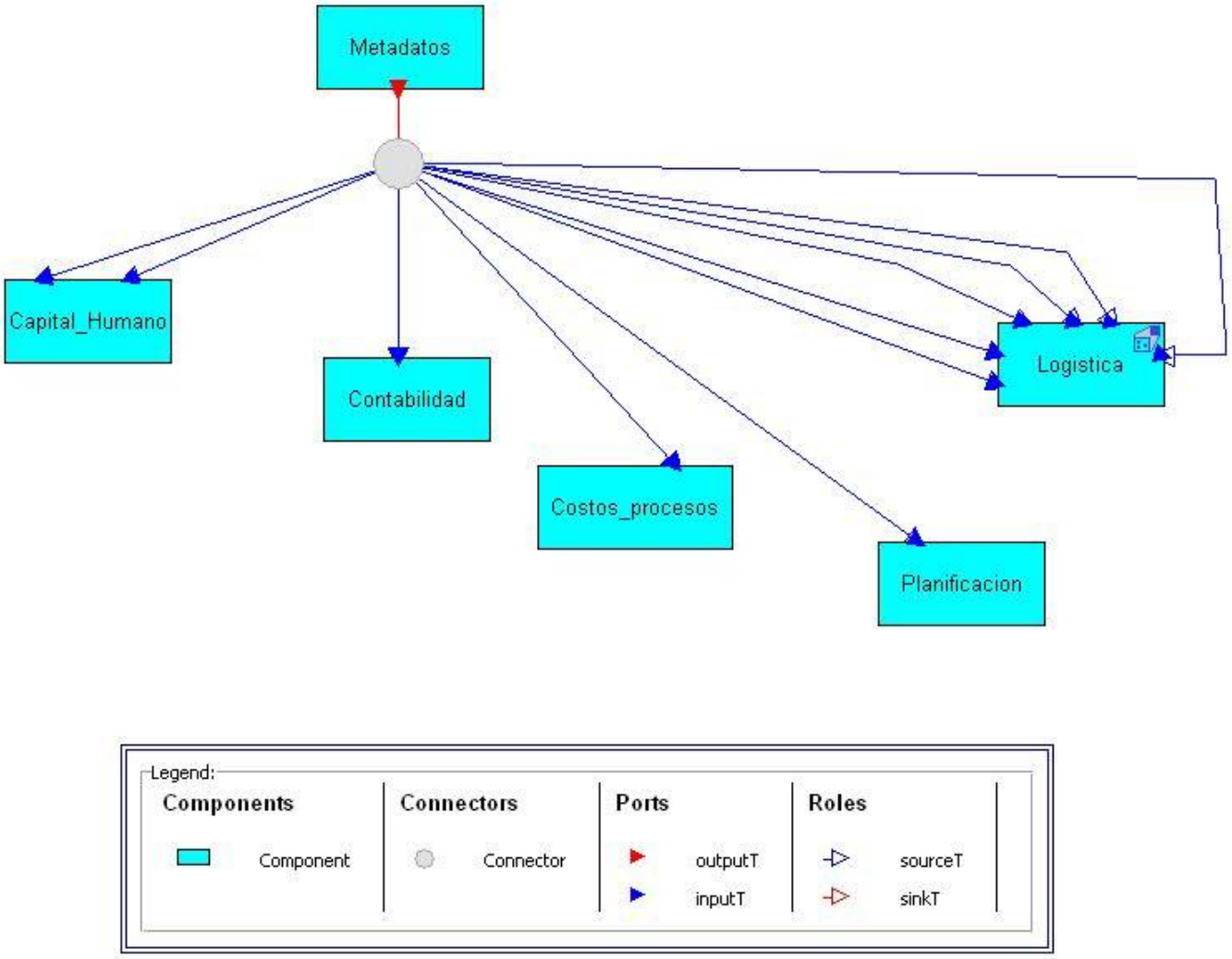


Ilustración 8 Mapa de interacción externo(Estructura y Composición(Metadatos)

En la figura anterior se puede apreciar la interacción del subsistema Metadatos con otros 5, donde como dato se observa que de este depende el resto mientras que él no tiene dependencias de ninguno. Como dato mas significativo se puede apreciar que el subsistema Logística tiene 6 dependencias de Metadatos.

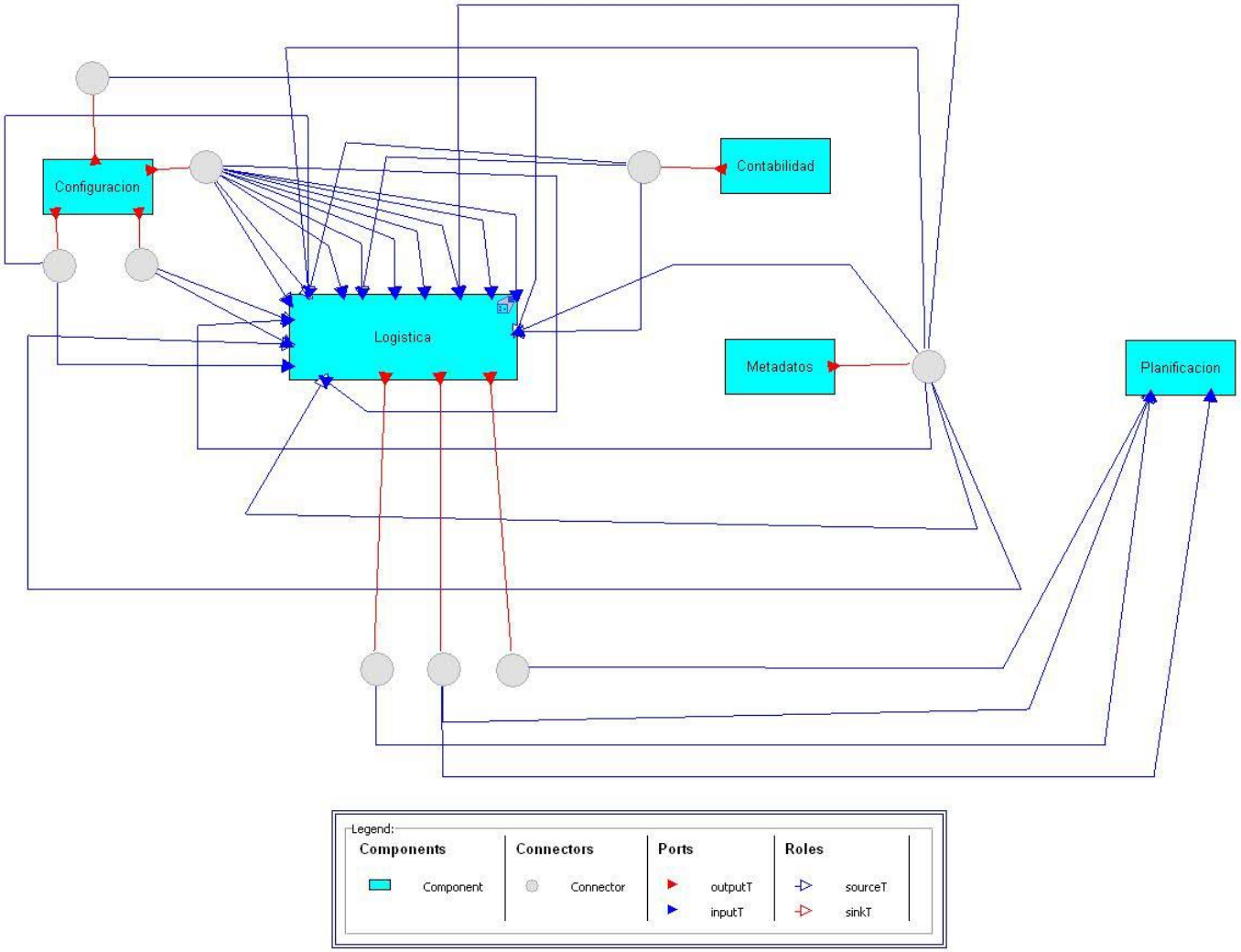


Ilustración 9 Mapa de interacción externo(Logística)

En la figura anterior se puede apreciar como dato significativo que el subsistema Logística, específicamente el subsistema interno Inventario tiene 14 dependencias del subsistema Configuración, 6 de Metadatos y 3 de Contabilidad, mientras que de él solamente depende Planificación, el cual consume 4 servicios de Logística.

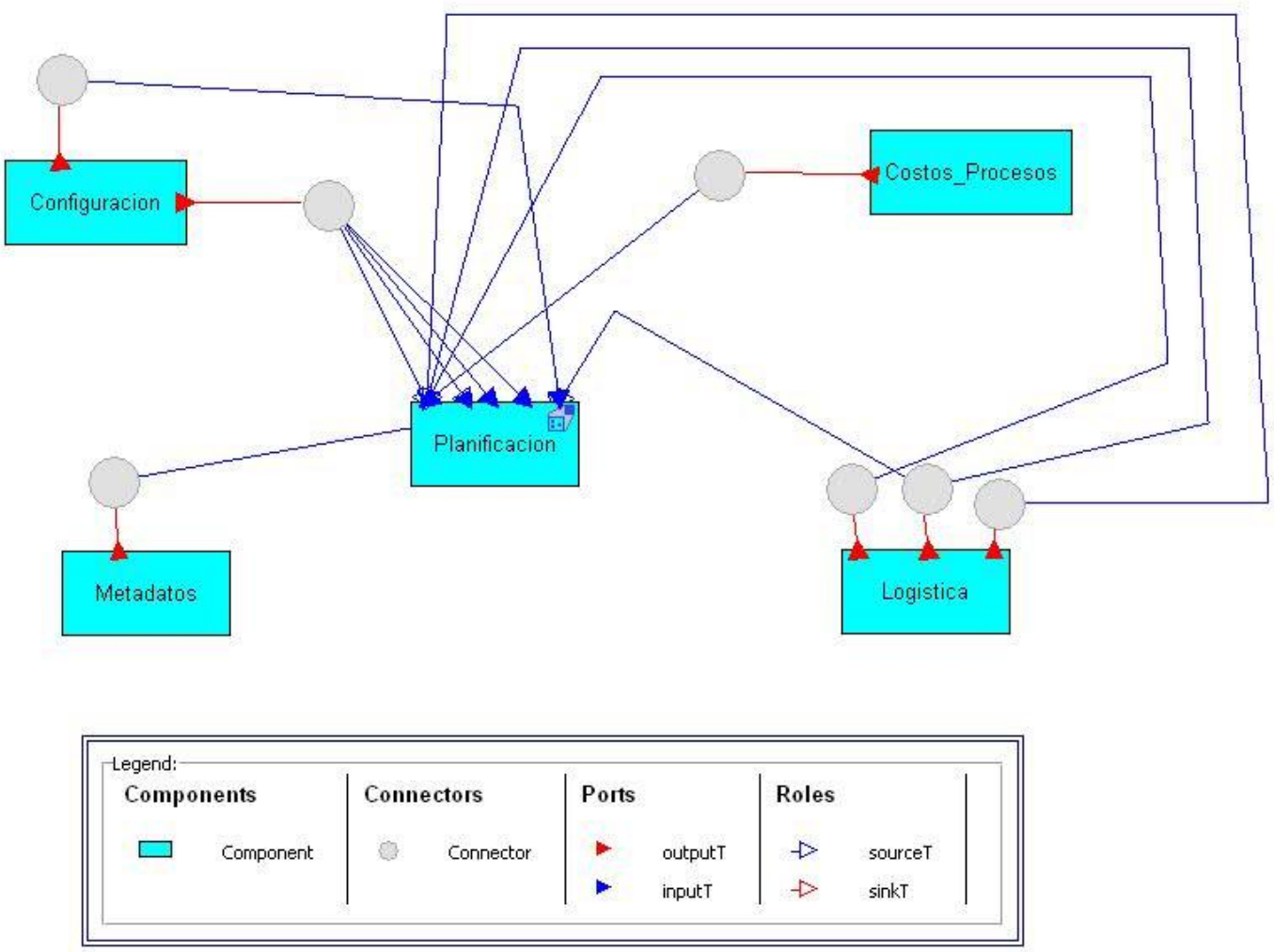


Ilustración 10 Mapa de interacción externo(Planificación).

En la figura anterior se puede apreciar que el subsistema Planificación tiene dependencias con todos los subsistemas mientras que de el no depende ninguno. Las dependencias son 5 con Configuración, 4 con Logística, 1 de Metadatos y 1 de Costos y Procesos.

Capítulo 3. “Modelado de la arquitectura de ERP Cuba”

Conclusiones del capítulo

Como resultado de este último capítulo se obtuvo exitosamente el modelado basado en el ADL Acme de las vistas de sistema, internas y externas de 9 subsistemas del ERP, basándose en los resultados obtenidos en el instrumento metodológico usado según las orientaciones del equipo de arquitectura y la subdirección tecnológica del ERP Cuba. Además se llega a la conclusión que los subsistemas más significativos son los que se exponen en la siguiente tabla:

Subsistemas Significativos	**Cantidad de Servicios Brindados**	**Cantidad de Subsistemas Dependientes.**
Configuración	62	4
Contabilidad	24	3
Metadatos	13	6
Costos y Procesos	10	3

Tabla 10. Subsistemas más significativos del ERP Cuba.

Conclusiones generales

Durante el desarrollo de este trabajo de diploma se realizaron estudios sobre los lenguajes de descripción arquitectónica dentro de la arquitectura de software, lo cual permitió crear las bases necesarias para desarrollar el mismo. A partir de las investigaciones realizadas se llegó a la conclusión que los ADL en general, a pesar de ser un campo nuevo en la industria del software mundial y nacional son los adecuados para modelar decisiones arquitectónicas, debido a que los mismos permiten representar elementos arquitectónicos de muy variados tipos aportando un alto nivel de abstracción a un modelado arquitectónico.

En el caso particular del ADL seleccionado(Acme) a partir de los instrumentos usados, por la satisfacción del equipo de arquitectura según los resultados obtenidos, se llegó a la conclusión de que el mismo está apto para representar correctamente una arquitectura de software ya que se adecúa a los conceptos arquitectónicos más importantes. Es significativo señalar que para la realización de este trabajo se hicieron varias entrevistas a los arquitectos de las diferentes líneas del proyecto, así como a los jefes de la arquitectura, lo que permitió un mayor entendimiento y la materialización de los objetivos trazados.

Recomendaciones

- ✓ Continuar el estudio del ADL propuesto e identificar nuevas características así como explorar las características presentes en este que no lograron ser dominadas por los autores del trabajo lo que imposibilito su explotación en el resultado que se presenta.
- ✓ Extender el resultado presentado en el trabajo en el resto de los proyectos de la Universidad de Ciencias Informáticas e incluir el mismo en algún programa o asignatura optativa relacionada con la disciplina de la arquitectura de software que se imparta en la Universidad.

Bibliografía

Referencias Bibliográficas

1. Clements, Paul. "A Survey of Architecture Description Languages". Proceedings of the international Workshop on Software Specification and Design. . Alemania : s.n., 1996.

2. Vestal, Steve. An Overview and Comparison of Four Architecture Description Languages. Informe tecnico. Honeywel Tecnology Centerl. febrero de 1993.

3. Wolf, Dewayne E. Perry y Alexander L. "Foundations for the study of the software architecture". octubre de 1992.

4. Garlan, David. "Software architectures". Presentación en transparencias. 2000.

5. David Garlan, Robert Monroe y David Wile. "Acme: Architectural des-cription of component-based systems".Foundations of Component-Based Sys-tems, Gary T. Leavens y Murali Sitaraman (eds), Cambridge University Press, pp. 47-68. 2000.

6. Garlan, Mary Shaw y David. "Characteristics of Higher Level Languages for Software Architecture". 1994.

Bibliografía

Bibliografía Consultada

Aynur Abdurazik. “Suitability of the UML as an Architecture Description Language with Applications to testing”. Reporte ISE-TR-00-0, George Mason University. Febrero de 2000.

Bradley Schmerl and David Garlan. AcmeStudio: Supporting Style-Centered Architecture Development (Research Demonstration) In *Proceedings of the 26th International Conference on Software Engineering*, Edinburgh, Scotland, 23-28 May 2004.

Christine Hofmeister, Robert Nord y Dilip Soni. “Describing Software Architecture with UML” En Proceedings of the First Working IFIP Conference on Software Architecture, IEEE Computer Society Press, pp. 145-160, San Antonio. Febrero de 1999.

David Luckham y James Vera. “An Event-Based Architecture Definition Language”. IEEE Transactions on Software Engineering, pp. 717-734, Setiembre de 1995.

David Garlan. “Software architectures”. Presentación en transparencias, disponible en http://www.sti.uniurb.it/events/sfm03sa/slides/garlan-B.ppt.2000.

David Garlan, William K. Reinholtz, Bradley Schmerl, Nicholas Sherman and Tony Tseng. Bridging the Gap between Systems Design and Space Systems Software. In *Proceedings of the 29th Annual IEEE/NASA Software Engineering Workshop (SEW-29)*, Greenbelt, MD, 6-7 April 2005.

David Garlan. Evolution Styles: Formal foundations and tool support for software architecture evolution. Technical report, CMU-CS-08-142, *School of Computer Science, Carnegie Mellon University*, June 2008.

Jeff Magee y Jeff Kramer. “Dynamic structure in software architectures”. En Proceedings of the Fourth ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 3–14, San Francisco, Octubre de 1996.

Klaus-Dieter Schewe. “UML: A modern dinosaur? – A critical analysis of the Unified Modelling Language”. En H. Kangassalo, H. Jaakkola y E. Kawaguchi (eds.), Proceedings of the 10th European-Japanese. Conference on Information Modelling and Knowledge Bases, Saariselk, Finlandia, 2000.

Luckham, D. C. et al. Specification and Analysis of System Architecture using Rapide. *IEEE Trans. on Software Engineering*, 21(4):336–355. 1995

Magee, J., Kramer, J., y Giannakopoulou, D. Behaviour Analysis of Software Architectures. En *Software Architecture*, pp. 35–49. Kluwer Academic Publishers. 1999

Martin Glinz. “Problems and deficiencies of UML as a requirements specification language”. En Proceedings of the 10th International Workshop of Software Specification and Design (IWSSD-10), pp. 11-22, San Diego, noviembre de 2000.

Bibliografía

Mary Shaw y David Garlan. “Characteristics of Higher-Level Languages for Software Architecture”. Technical Report CMU-CS-94-210, Carnegie Mellon University, Diciembre de 1994.

Mary Shaw, Robert DeLine, Daniel Klein, Theodore Ross, David Young y Gregory Zelesnik. “Abstractions for Software Architecture and Tools to Support Them”. IEEE Transactions on Software Engineering, pp. 314-335, Abril de 1995.

Mary Shaw y David Garlan. “Formulations and Formalisms in Software Architecture”. Springer-Verlag, Lecture Notes in Computer Science, Volumen 1000, 1995.

Marwan Abi-Antoun, Jonathan Aldrich, Nagi Nahas, Bradley Schmerl and David Garlan. Differencing and Merging of Architectural Views. In *Automated Software Engineering Journal*, Vol. 15(1), March 2008.

Marwan Abi-Antoun, Jonathan Aldrich, David Garlan, Bradley Schmerl, Nagi Nahas and Tony Tseng. Software Architecture with Acme and ArchJava (Research Demonstration) In *Proceedings of the 27th International Conference on Software Engineering*, St. Louis, MS, May 2005.

Medvidovic, N. y Taylor, R. (2000). A Classification and Comparison// Medvidovic y Taylor, 2000, Medvidovic y Rosenblum, 1997

Medvidovic, N. et al..Using Object-Oriented Typing to Support Architectural Design in the C2 Style. En *ACM Foundations of Software Engineering (FSE’96)*, pp. 24–32, San Francisco (Estados Unidos).1996.

Neno Medvidovic. “A classification and comparison framework for software Architecture Description Languages”. Technical Report UCI-ICS-97-02, 1996.

OOSPLA’99, Workshop #23: “Rigorous modeling and analysis with UML: Challenges and Limitations”. Denver, Noviembre de 1999.

Paul Kogut y Paul Clements. “Features of Architecture Description Languages”. Borrador de un CMU/SEI Technical Report, Diciembre de 1994.

Paul Kogut y Paul Clements. “Feature Analysis of Architecture Description Languages”.En Proceedings of the Software Technology Conference (STC’95), Salt Lake City, Abril de 1995.

RAPIDE Design Team. RAPIDE 1.0 Pattern Language Reference Manual. Program Analysis and Verification Group, Computer Systems lab. Stanford University 1997 .

Robert Allen. “A formal approach to Software Architecture”. Technical Report, CMU-CS-97-144, 1997.

Bibliografía

Shang-Wen Cheng, David Garlan, Bradley Schmerl and Vahe Poladian. Improving Architecture-Based Self-Adaption Through Resource Prediction. In *Software Engineering for Self-Adaptive Systems*, Chapter 15, LNCS, 2008. To Appear.

Steve Vestal. “A cursory overview and comparison of four Architecture Description Languages”. Technical Report, Honeywell Technology Center, Febrero de 1993.

Shang-Wen Cheng, David Garlan, Bradley Schmerl and Vahe Poladian. Improving Architecture-Based Self-Adaption Through Resource Prediction. In *Software Engineering for Self-Adaptive Systems*, Chapter 15, LNCS, 2008.

Páginas Web Consultadas

The ABLE Project Home Page. http://www-2.cs.cmu.edu/afs/cs/project/able/www/able.html , 2009.

The ABLE Project Home Page. http://www-2.cs.cmu.edu/afs/cs/project/able/www/wright/index.html. 2009.

The Acme Web Page. http://acme.able.cs.cmu.edu/acmeweb/.2009.

The Acme Web. *Modeling a System with Acme*. http://acme.able.cs.cmu.edu/html/WORKING-%20Modeling%20a%20System%20with%20Acme.html. 2009.

The Acme Web. An Overview of Acme. http://acme.able.cs.cmu.edu/docs/language_overview.html.2009.

The Nicolás Kicillof Home Page. http://www.dc.uba.ar/people/profesores/nicok/jacal.htm**.** 2009.

Anexos

Anexo 1 Creación de un nuevo proyecto con la herramienta AcmeStudio 3.4.0.

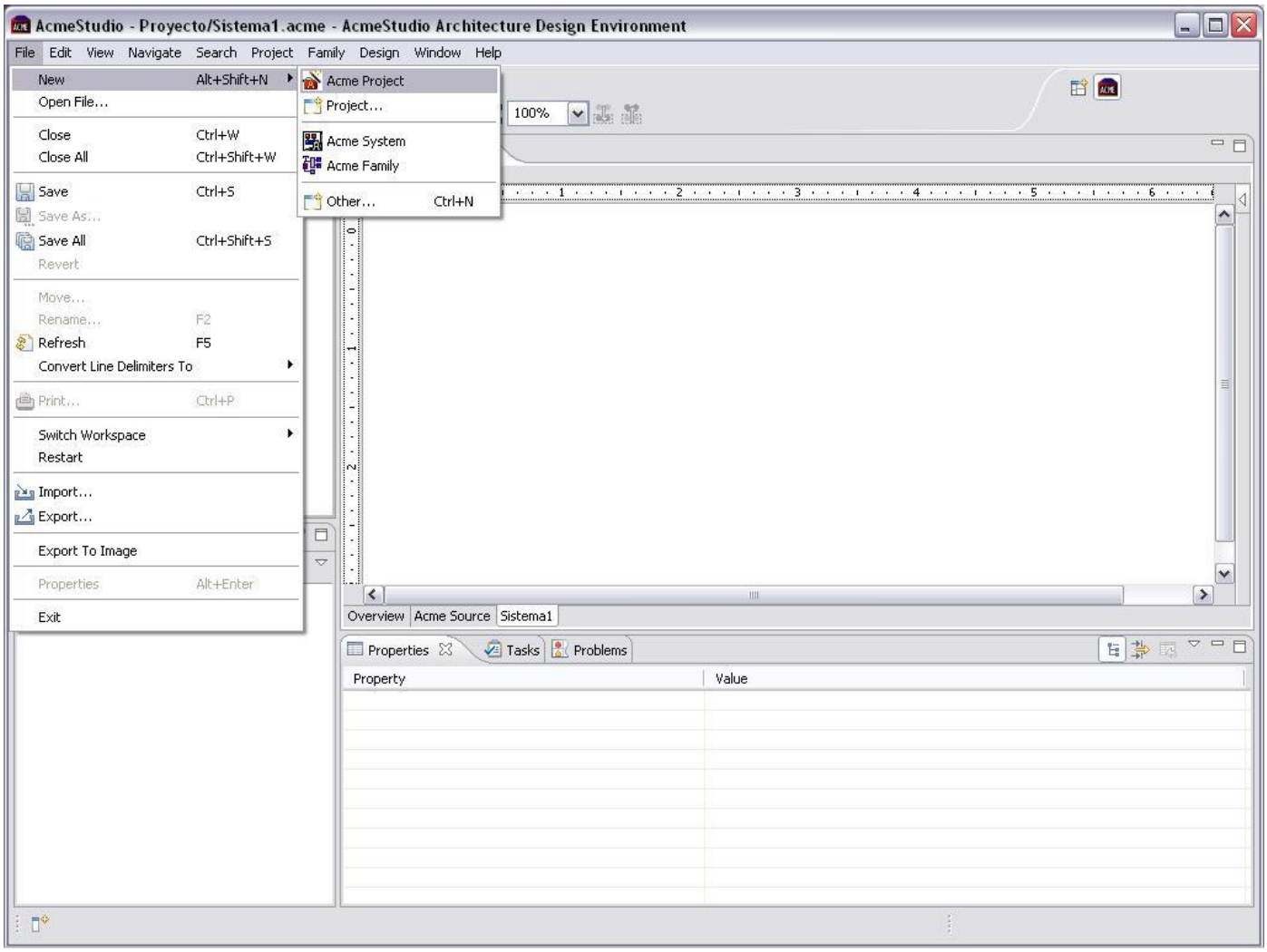


Ilustración 11. Creación de un nuevo proyecto en AcmeStudio 3.4.0.

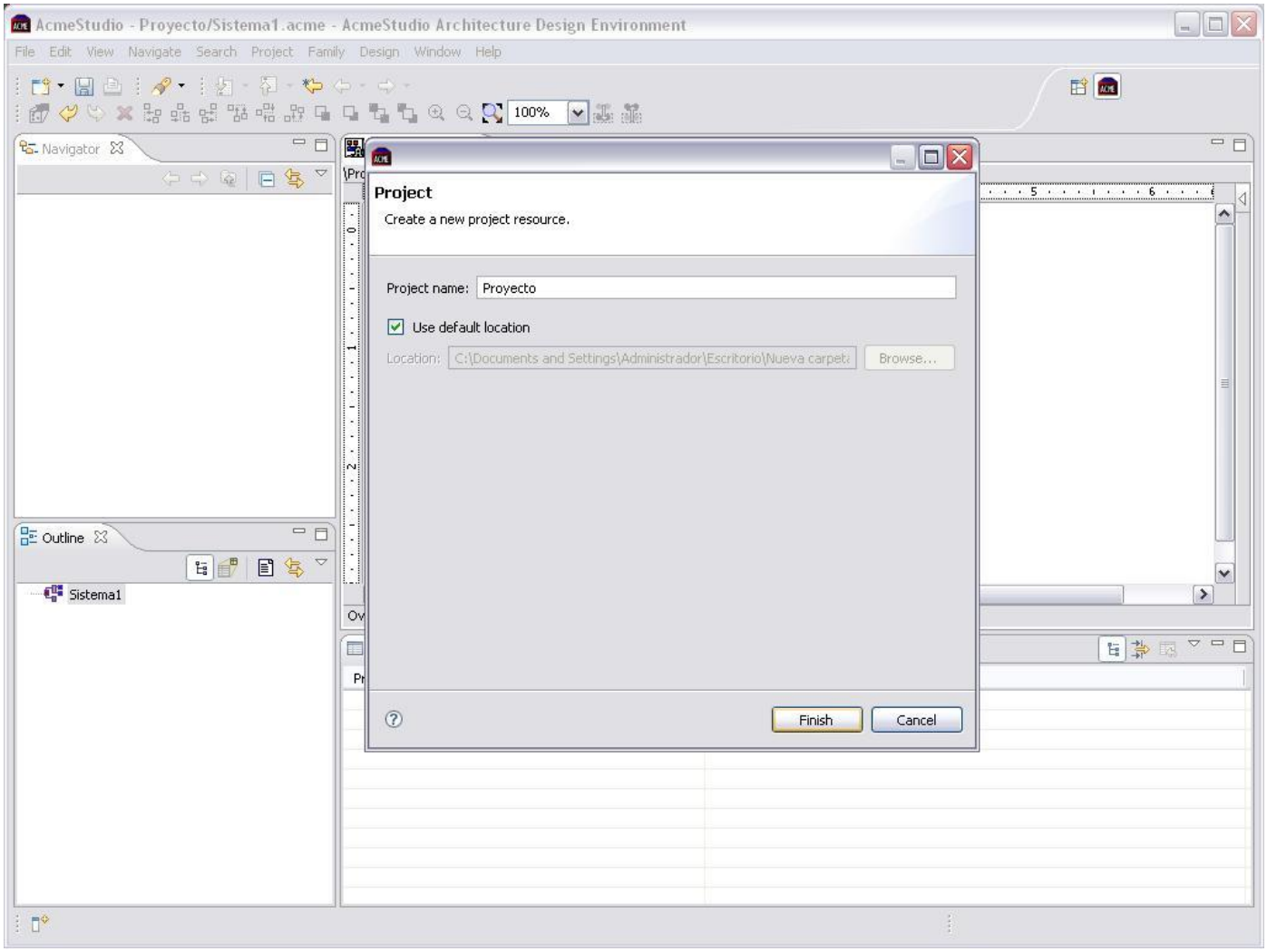


Ilustración 12 Creación de un nuevo proyecto en AcmeStudio 3.4.0.Nombar el proyecto.

Anexo 2 Creación de un nuevo sistema con la herramienta AcmeStudio 3.4.0.

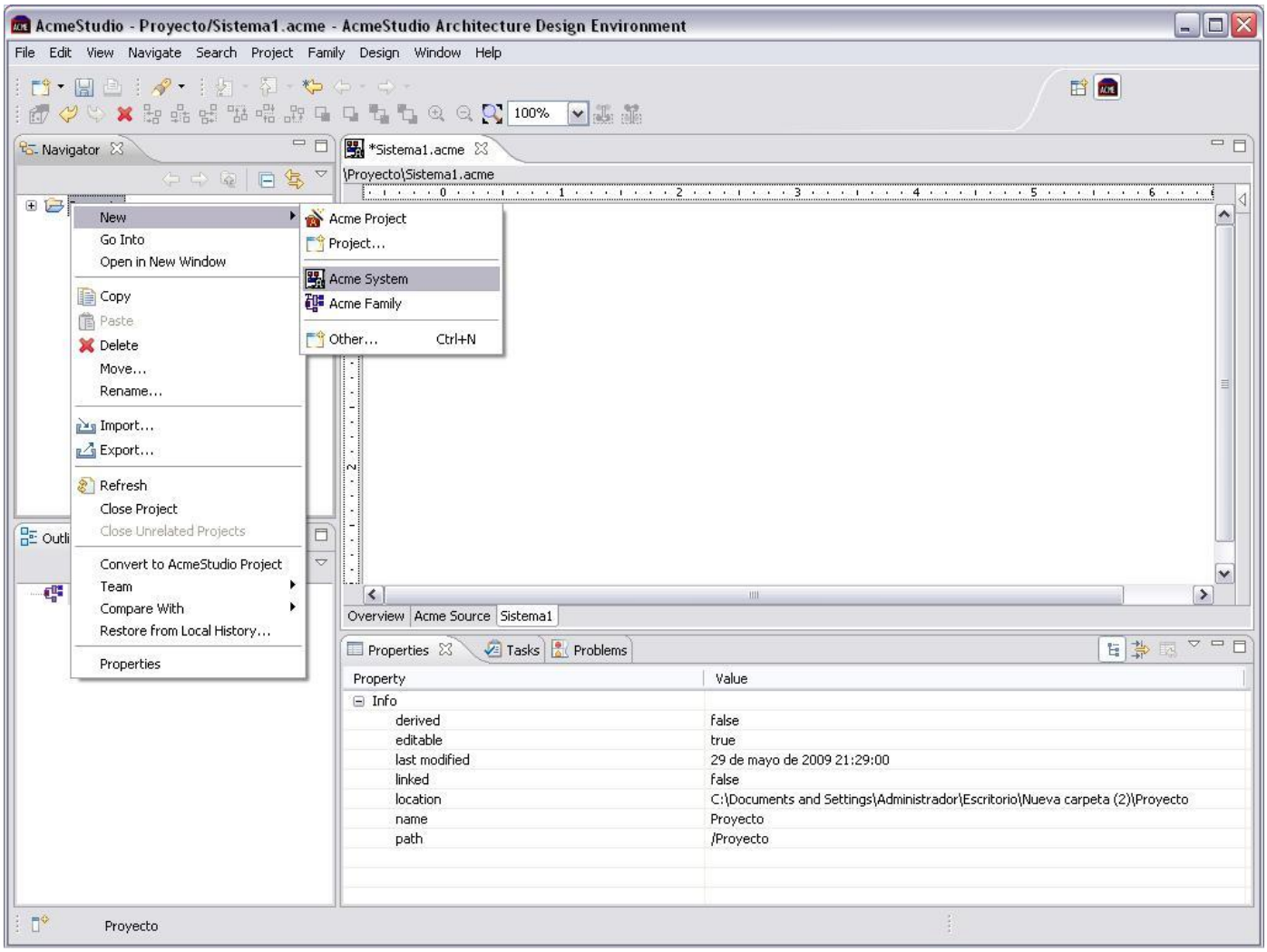


Ilustración 13 Creación de un nuevo sistema en AcmeStudio 3.4.0.

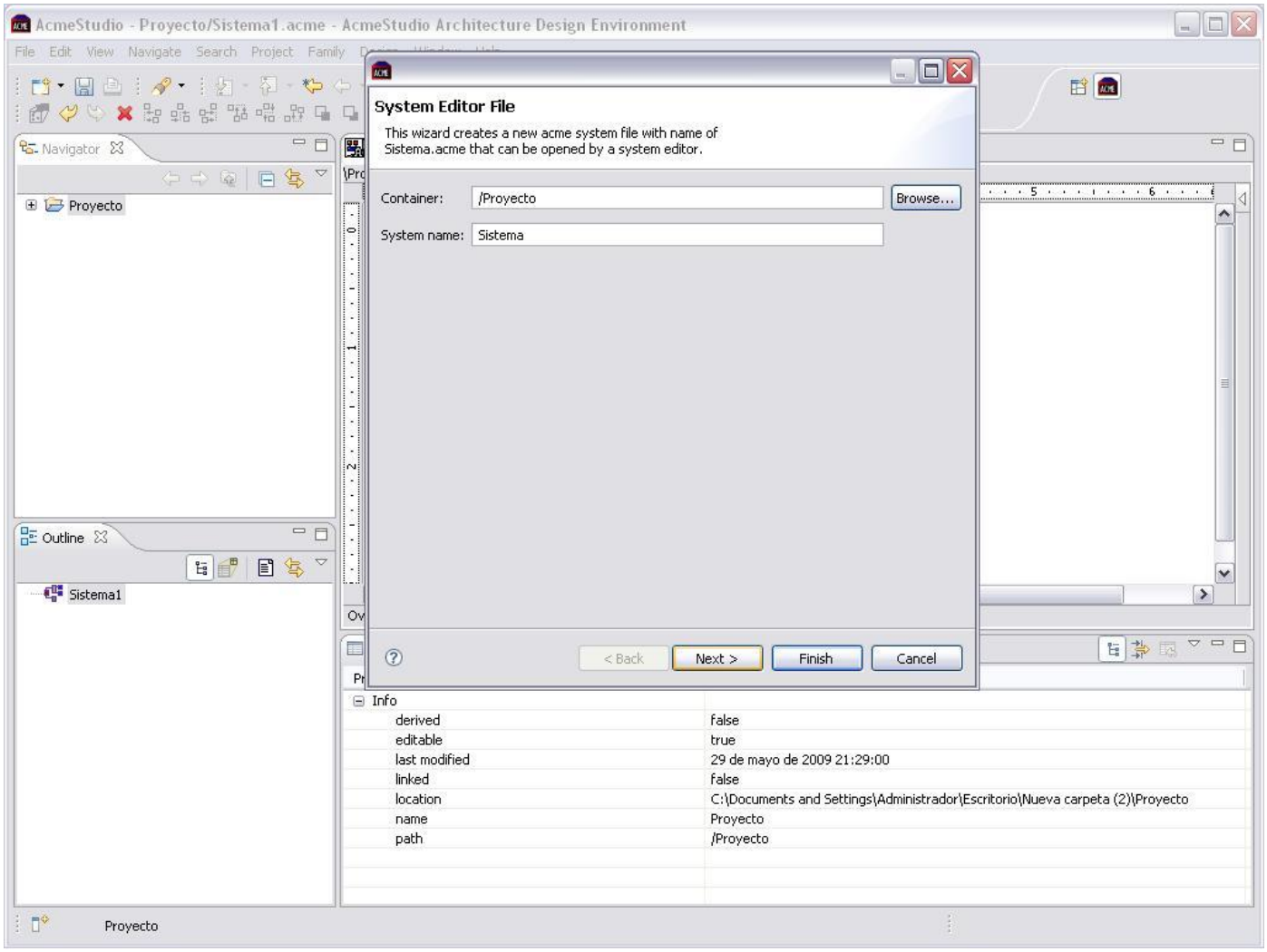


Ilustración 14. Creación de un nuevo sistema en AcmeStudio 3.4.0.Nombrar el sistema.

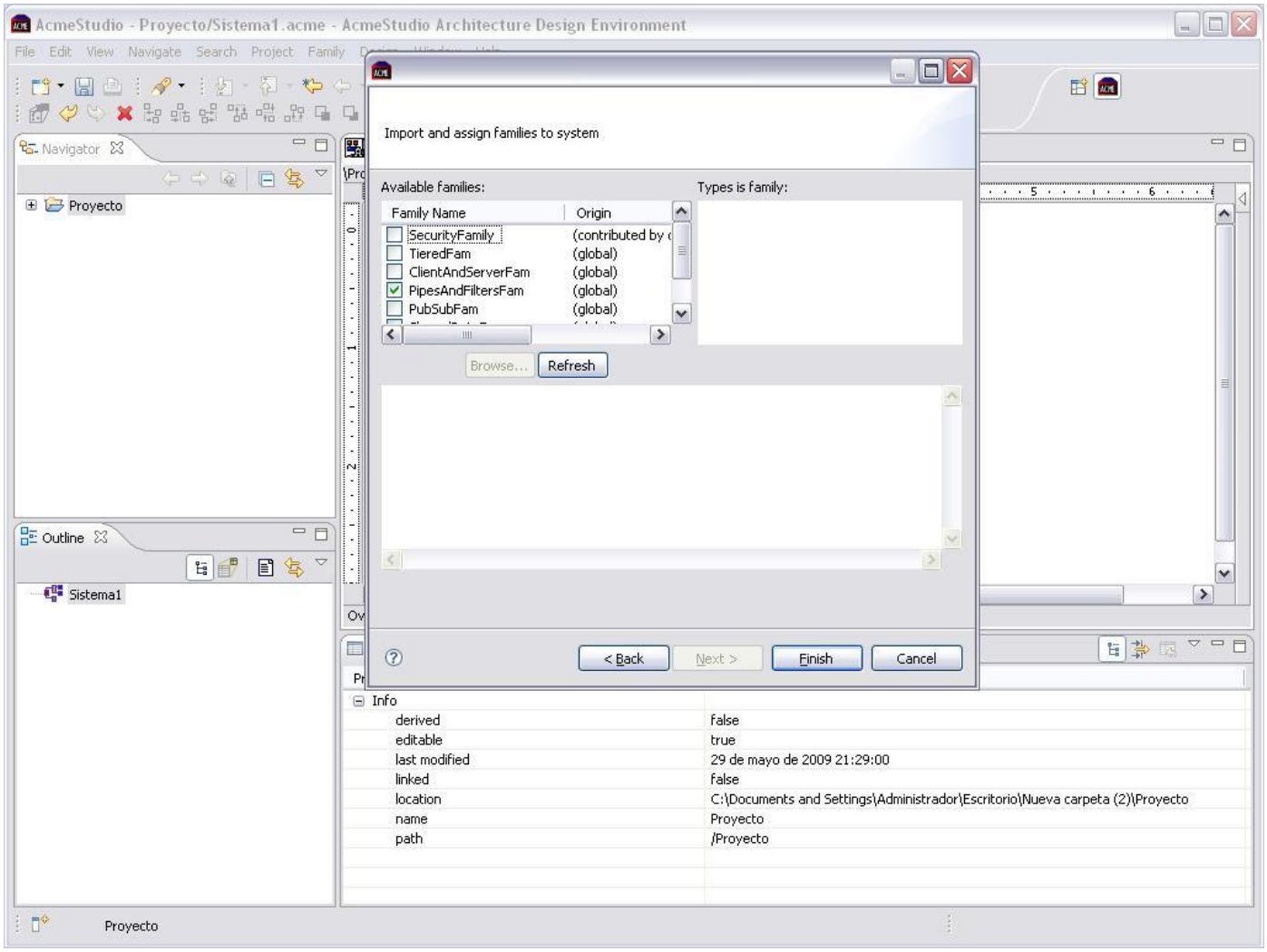


Ilustración 15. Creación de un nuevo sistema en AcmeStudio 3.4.0. Escoger la familia Acme.

Anexo 3. Creación de un nuevo componente con la herramienta AcmeStudio 3.4.0.

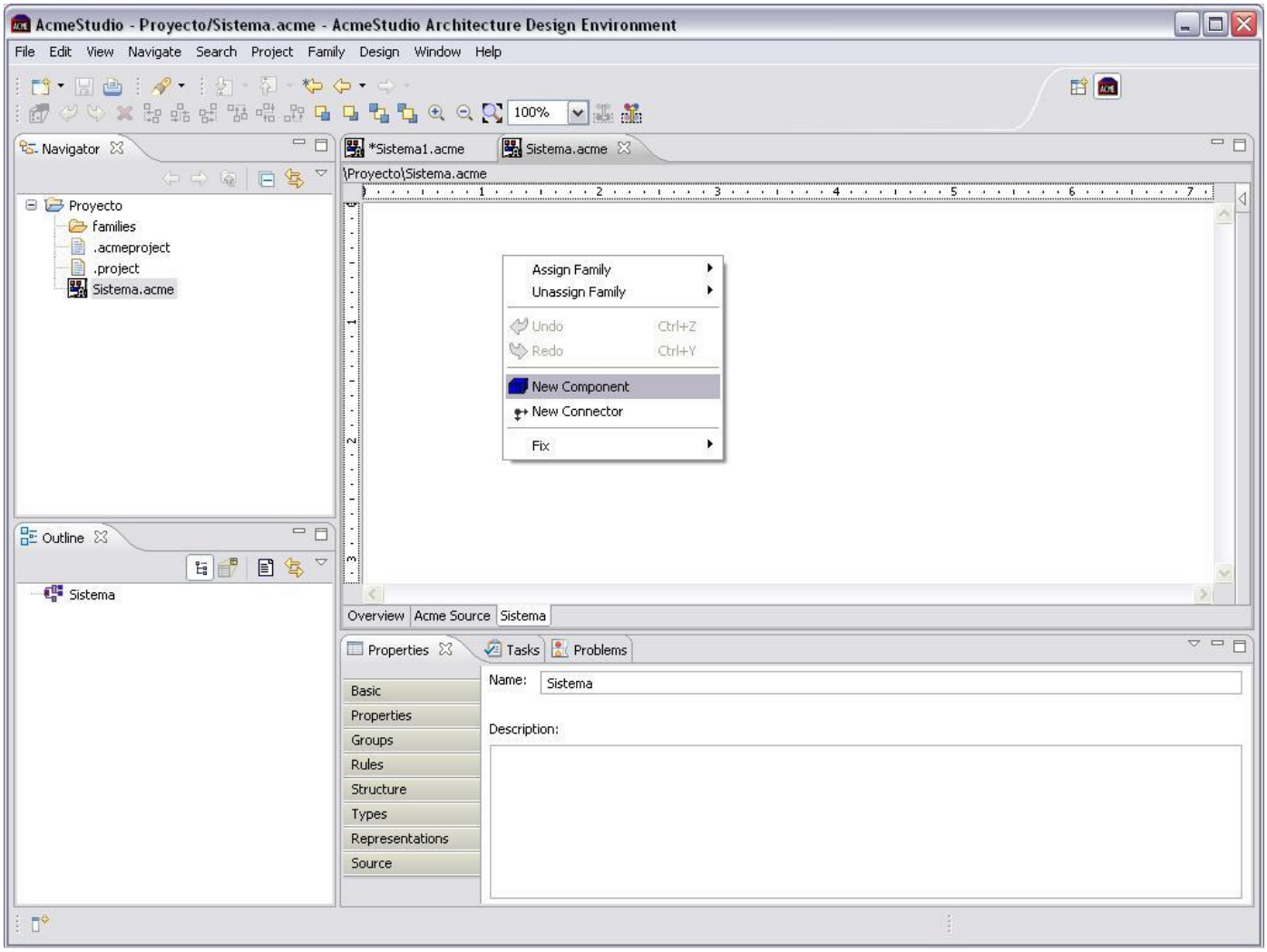


Ilustración 16 Creación de un nuevo componente.

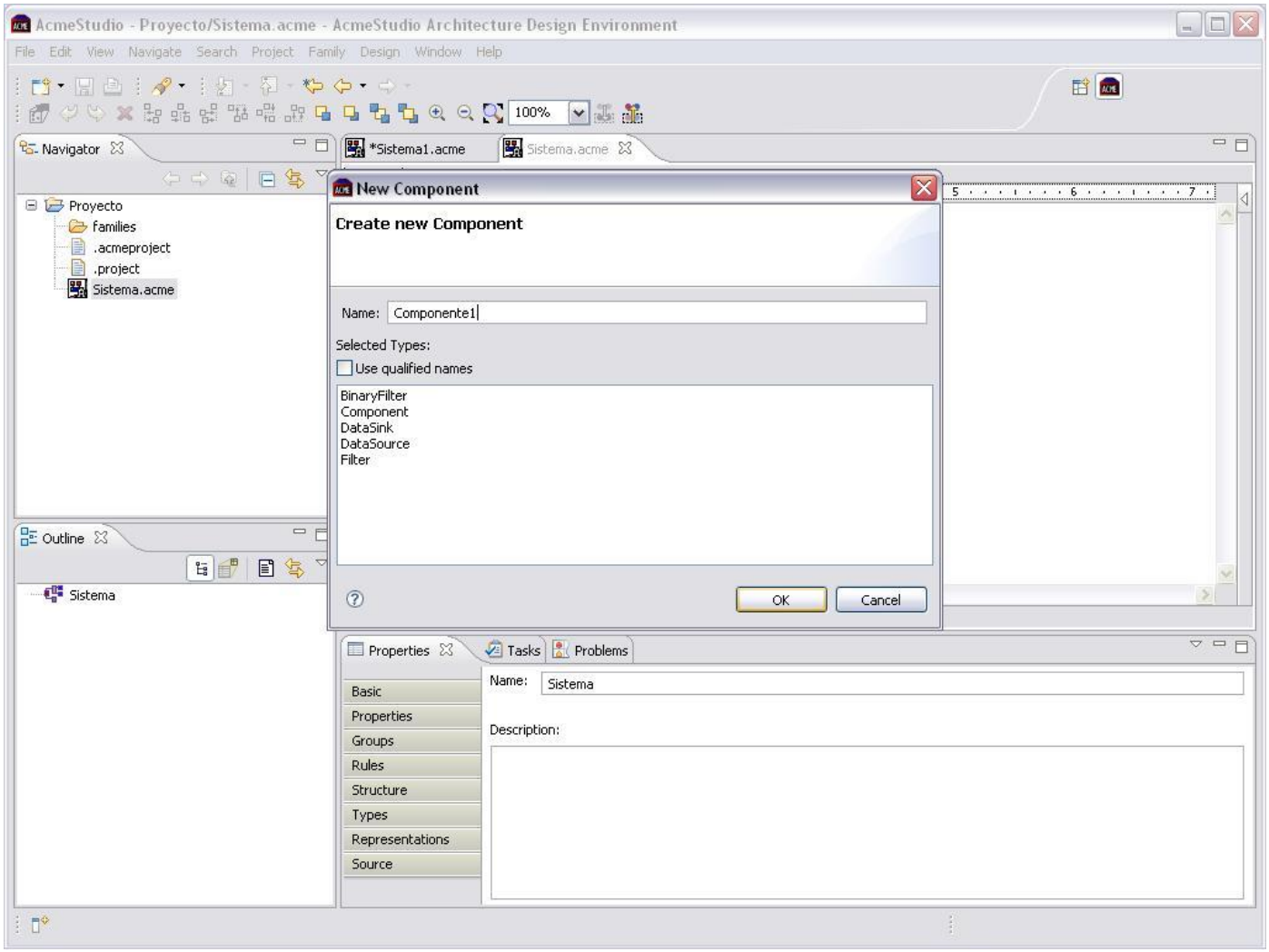


Ilustración 38 Creación de un nuevo componente. Escoger el tipo de componente y nombrarlo.

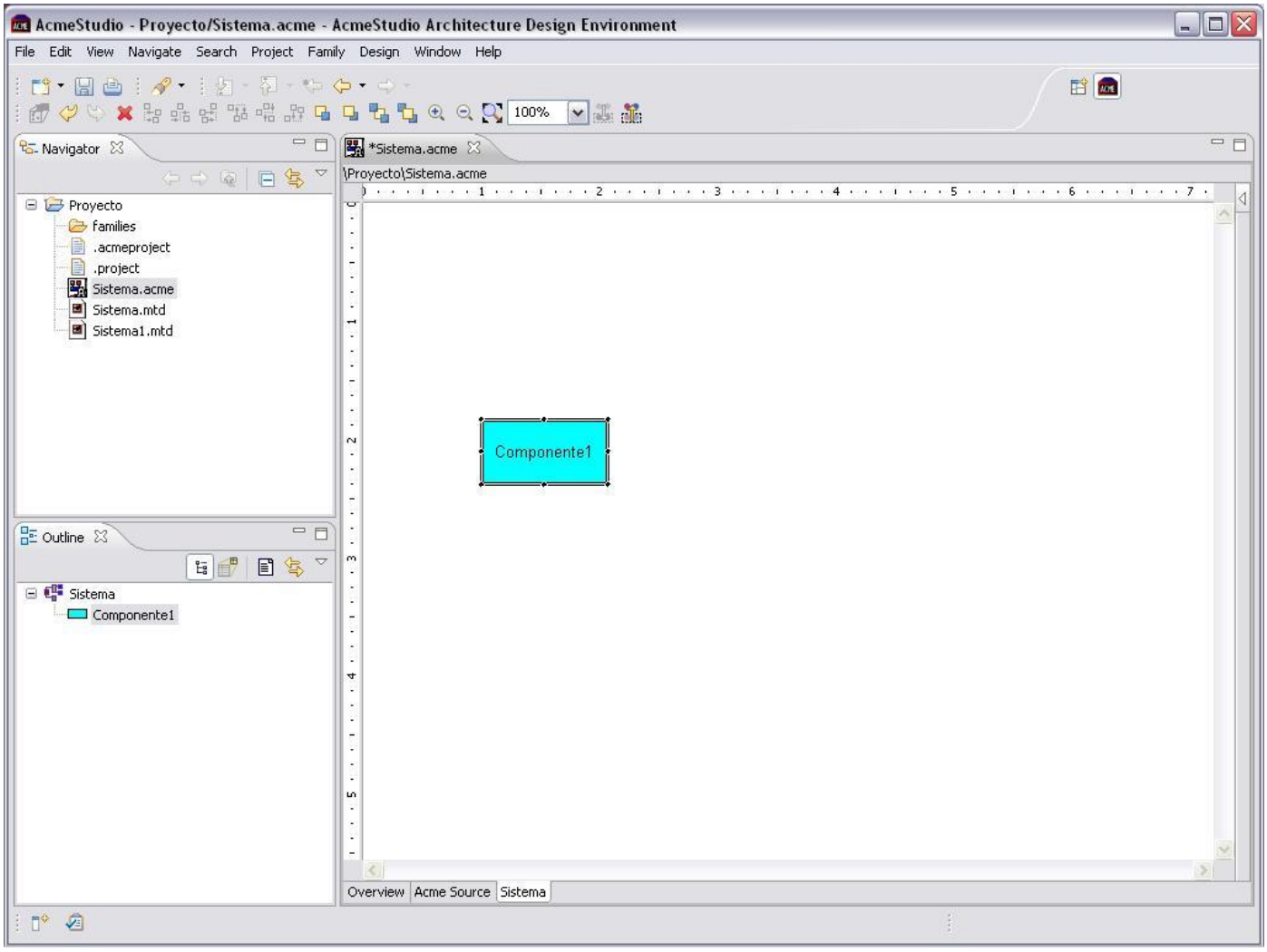


Ilustración 39 Nuevo componente creado.

Anexo 4. Creación de los puertos con la herramienta AcmeStudio 3.4.0

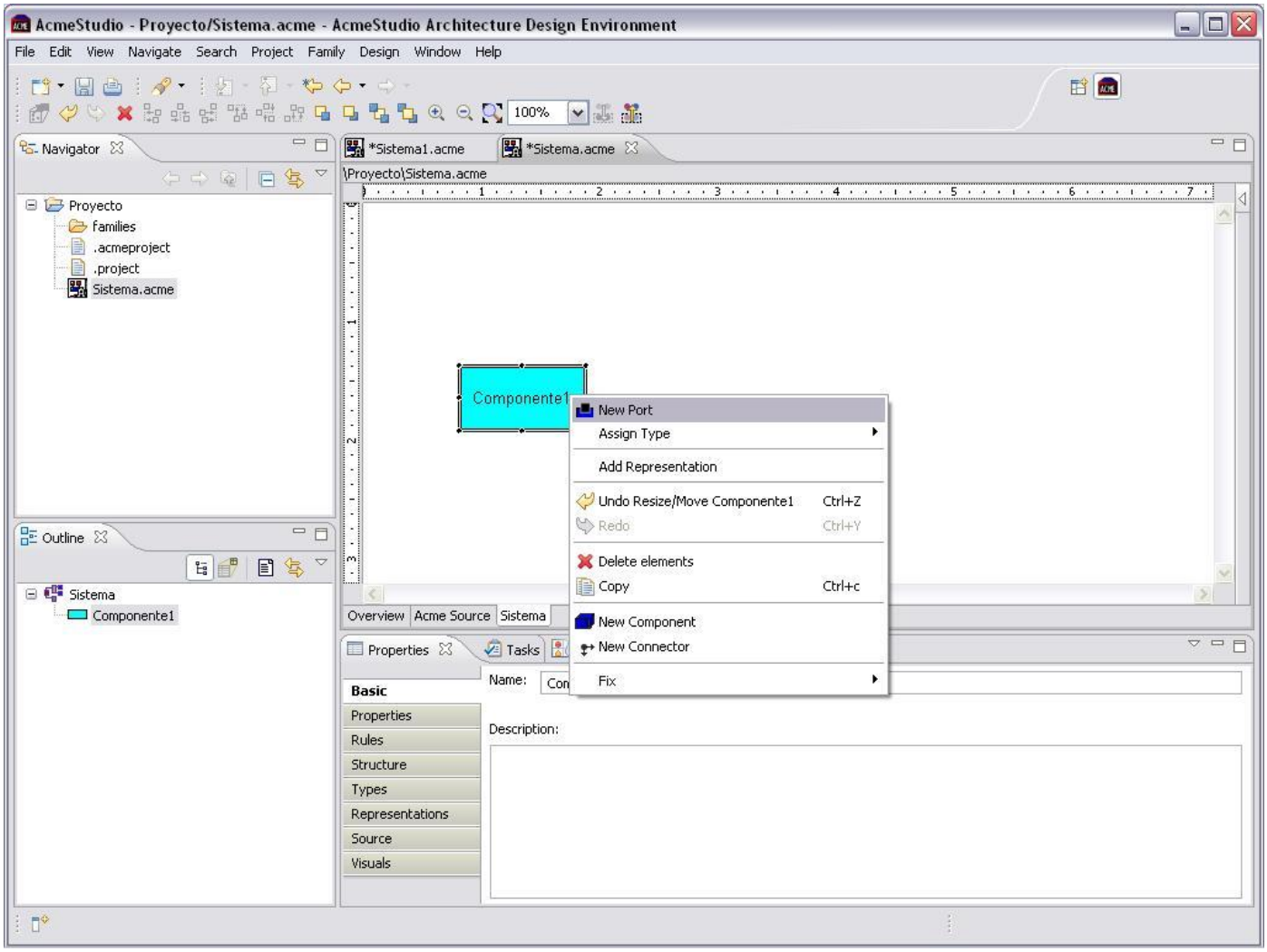


Ilustración 17 Crearle un nuevo puerto al componente.

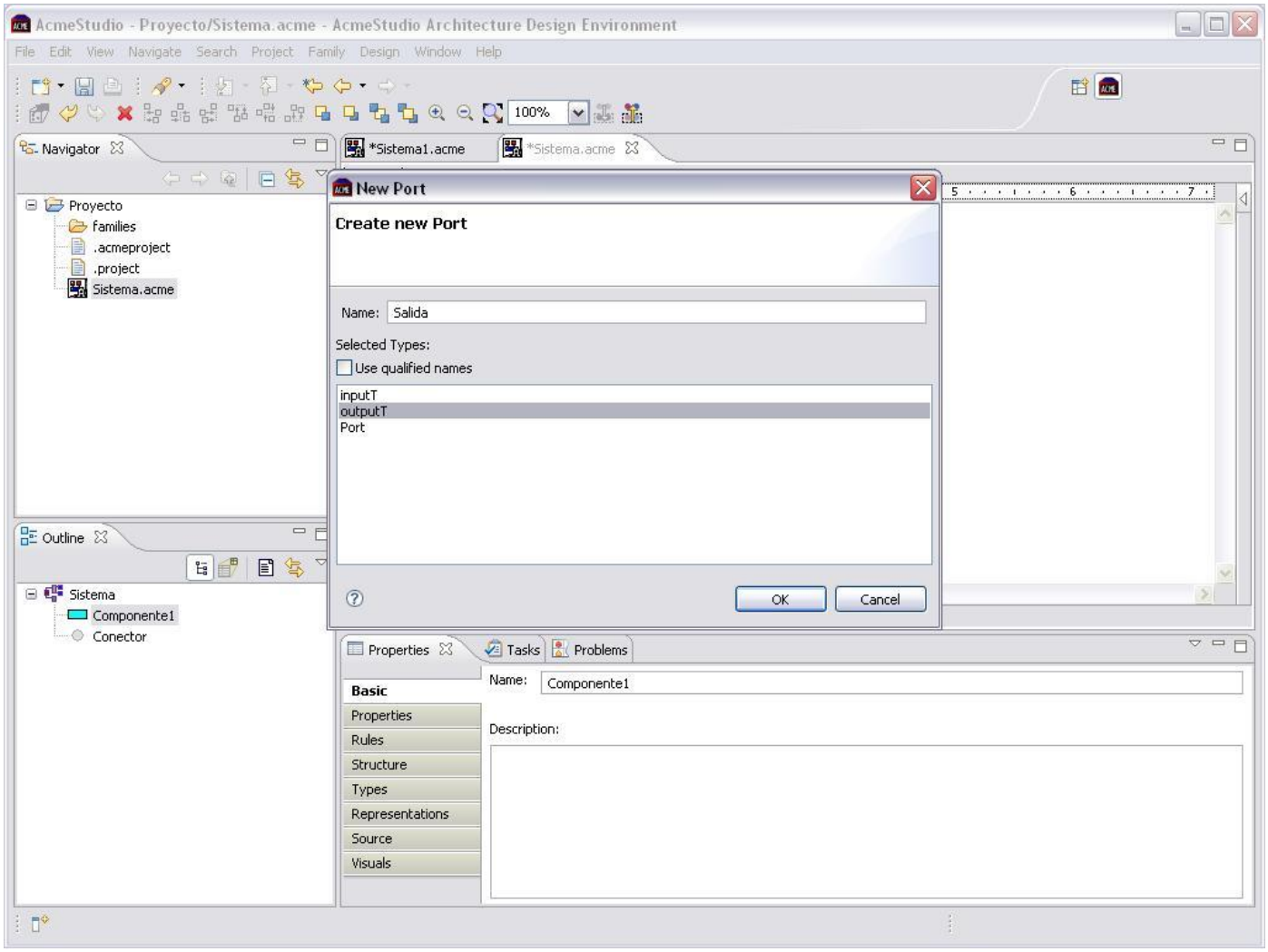


Ilustración 18 Crearle un nuevo puerto al componente. Escoger el tipo de puerto y nombrarlo.

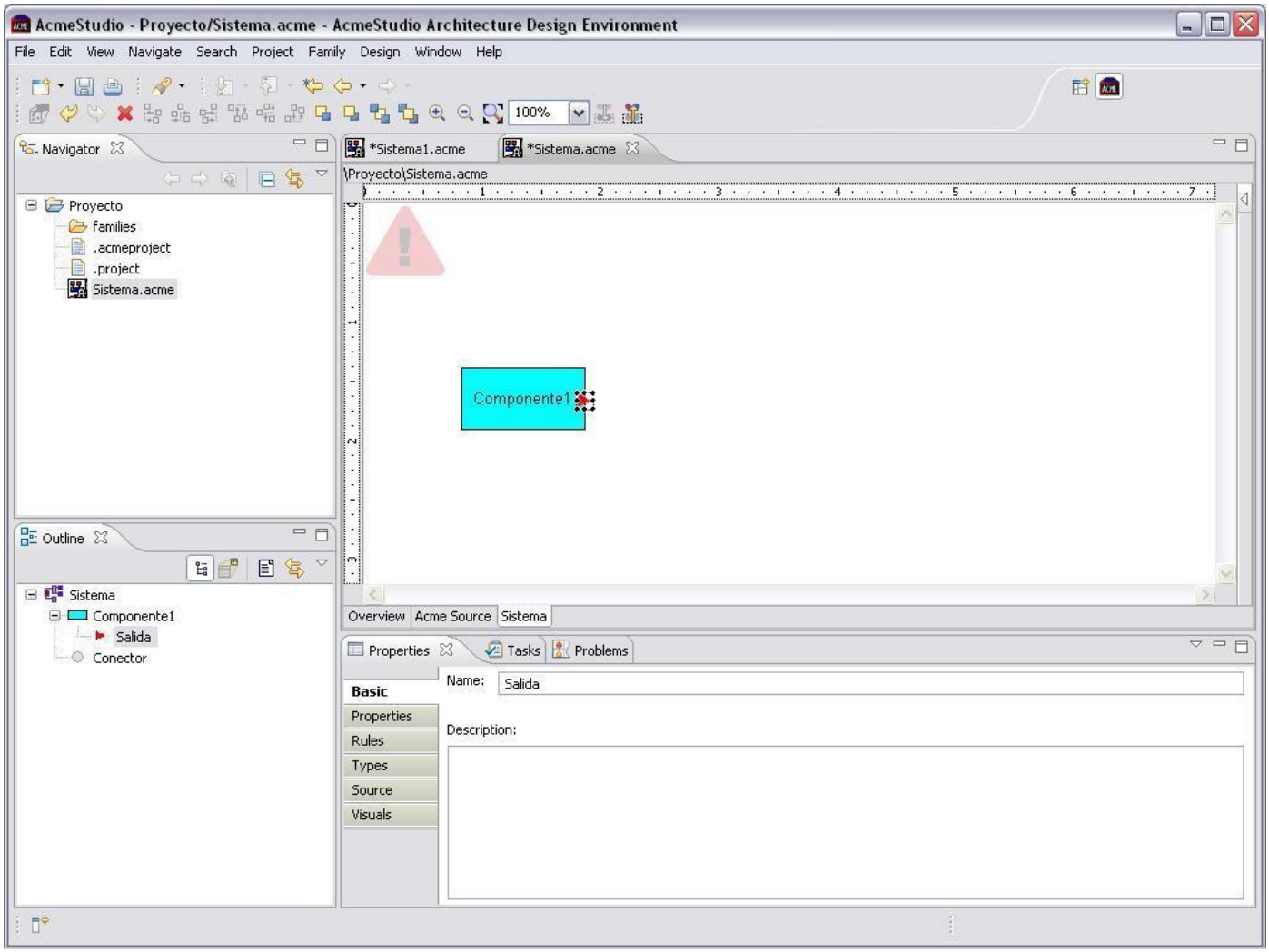


Ilustración 19 Nuevo puerto creado.

Anexo 5. Creación de los conectores con la herramienta AcmeStudio 3.4.0.

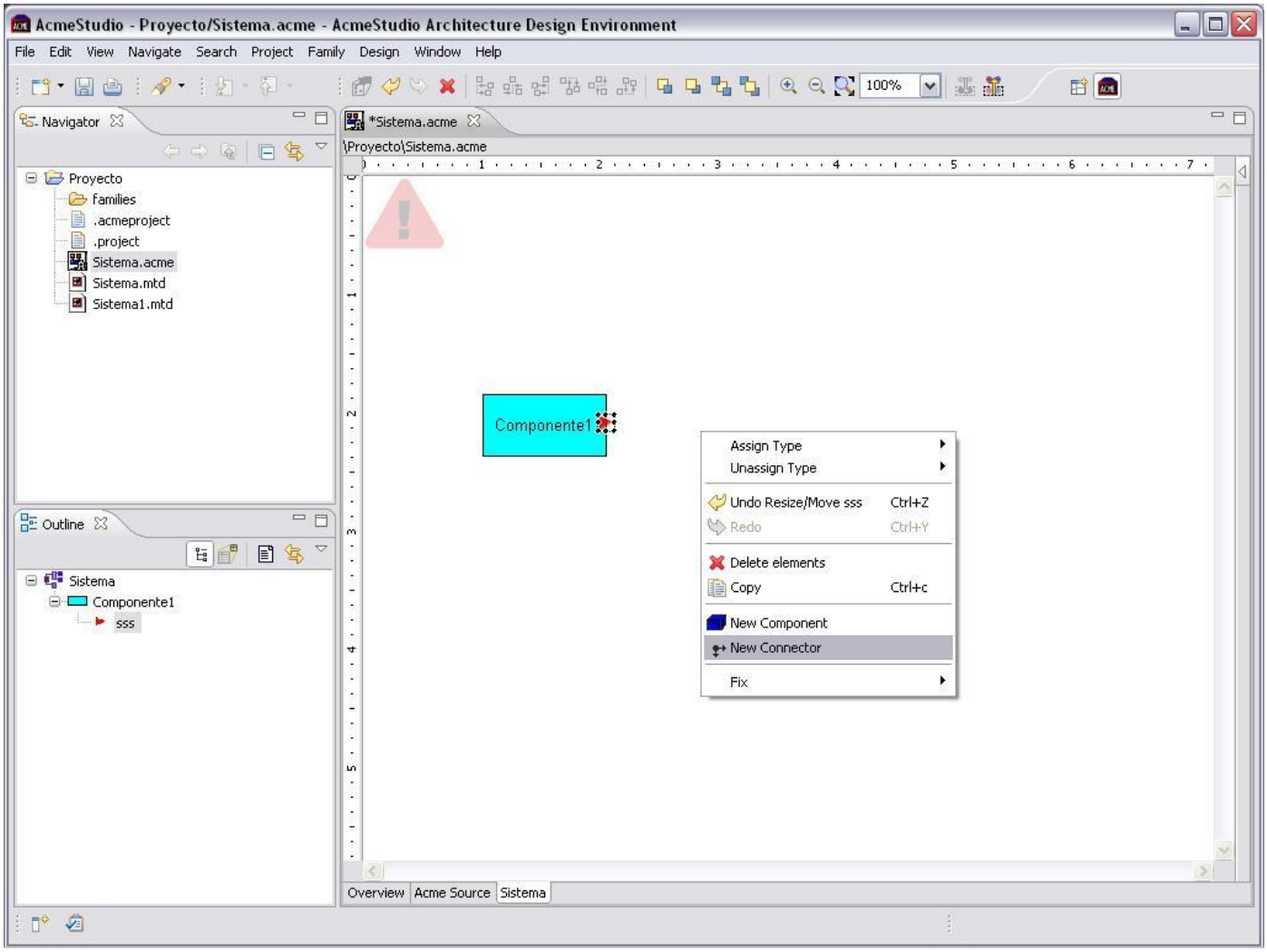


Ilustración 20 Creación de un nuevo conector.

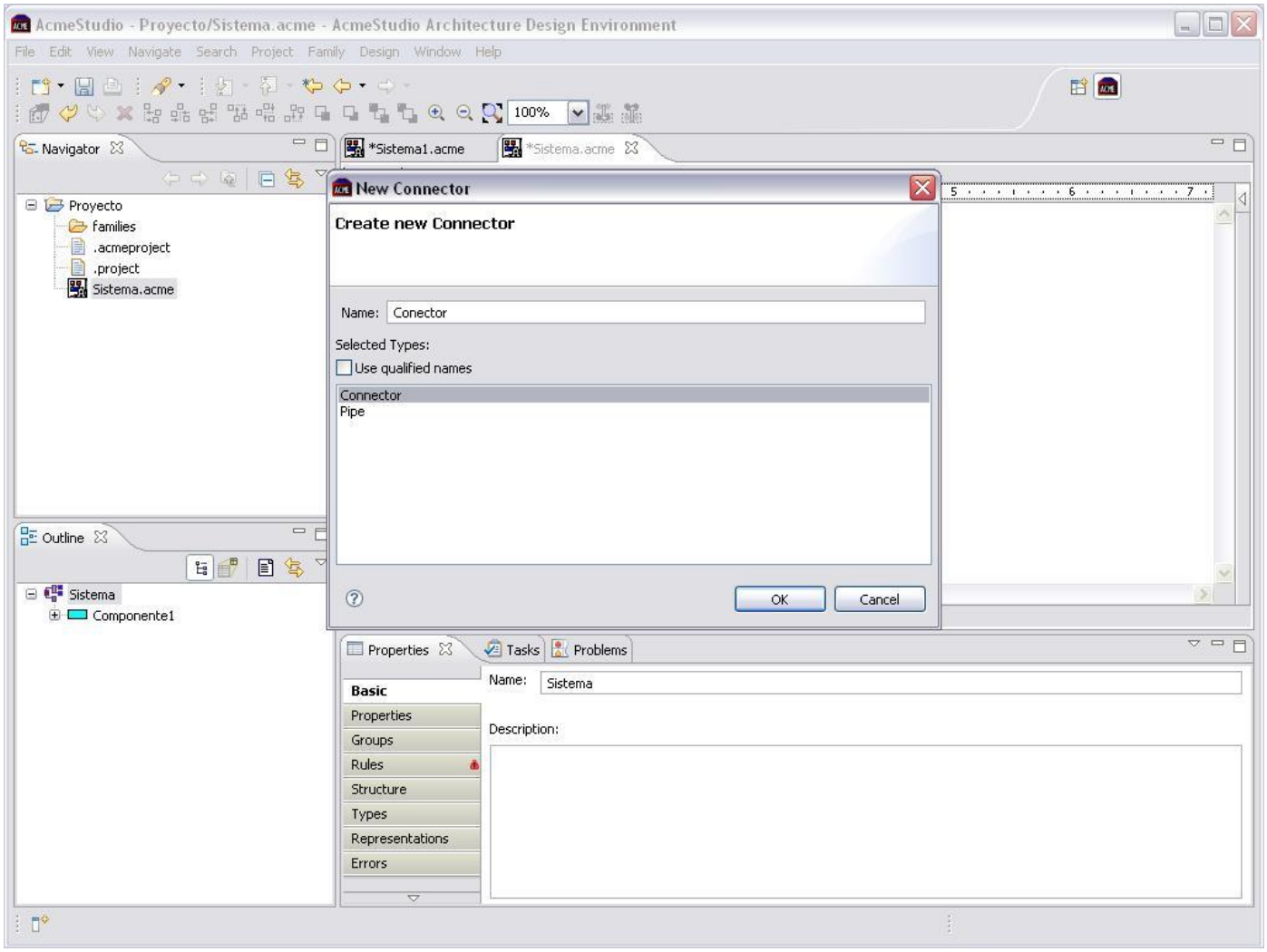


Ilustración 21 Creación de un nuevo conector. Escoger el tipo de conector y nombrarlo.

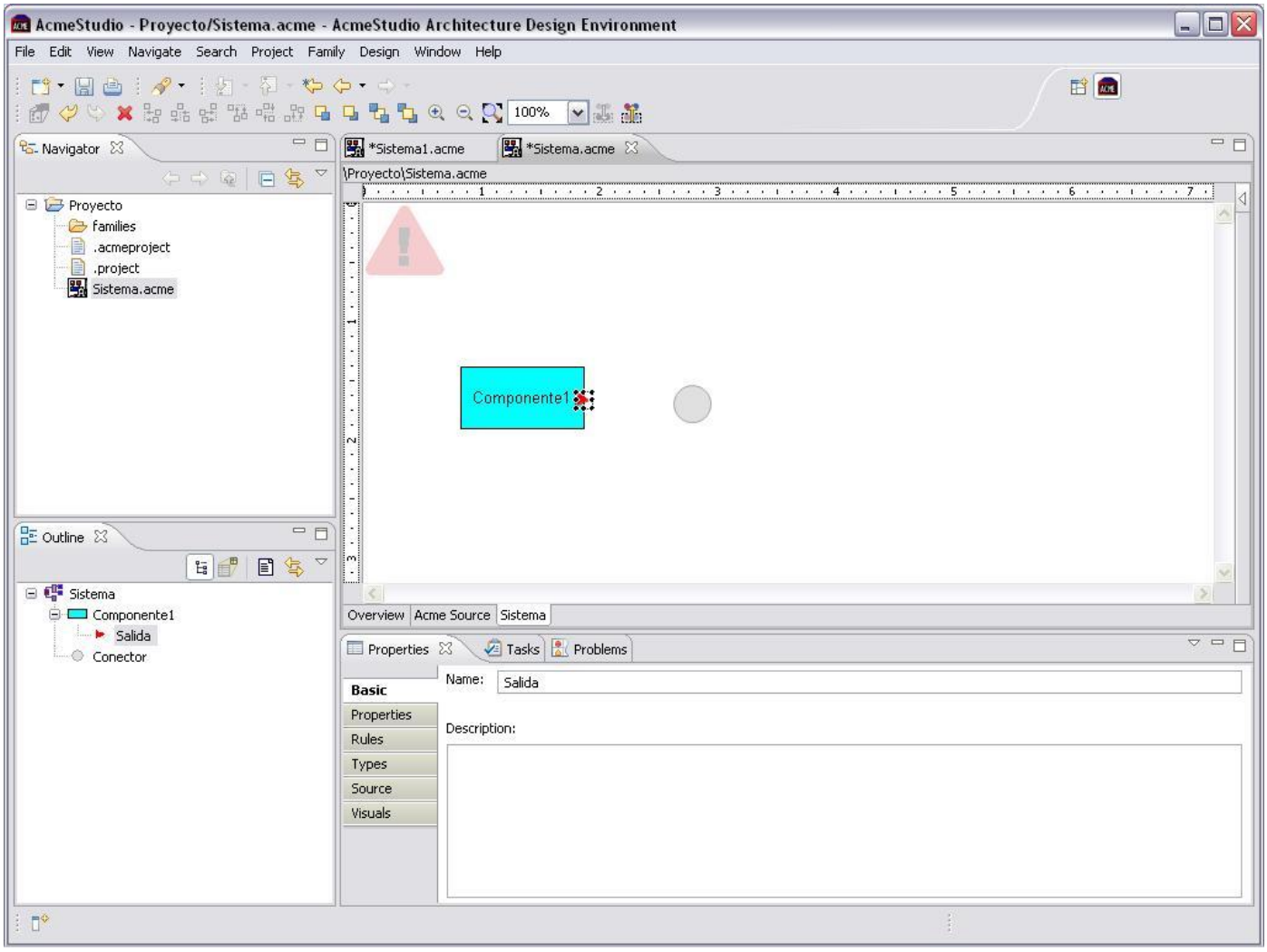


Ilustración 22 Nuevo conector creado.

Anexo 6. Creación de los roles con la herramienta AcmeStudio 3.4.0.

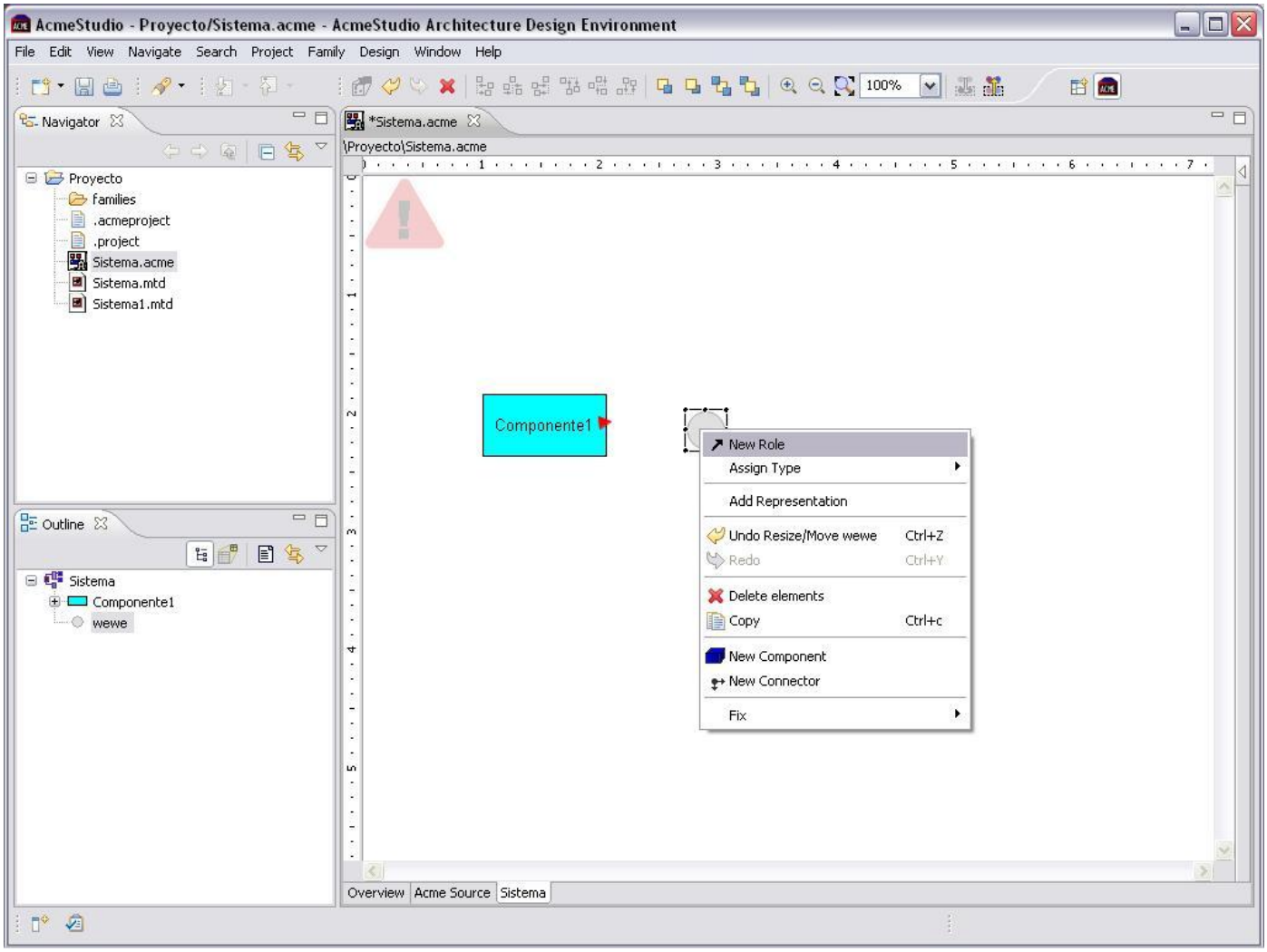


Ilustración 23 Creación de un nuevo rol al conector.

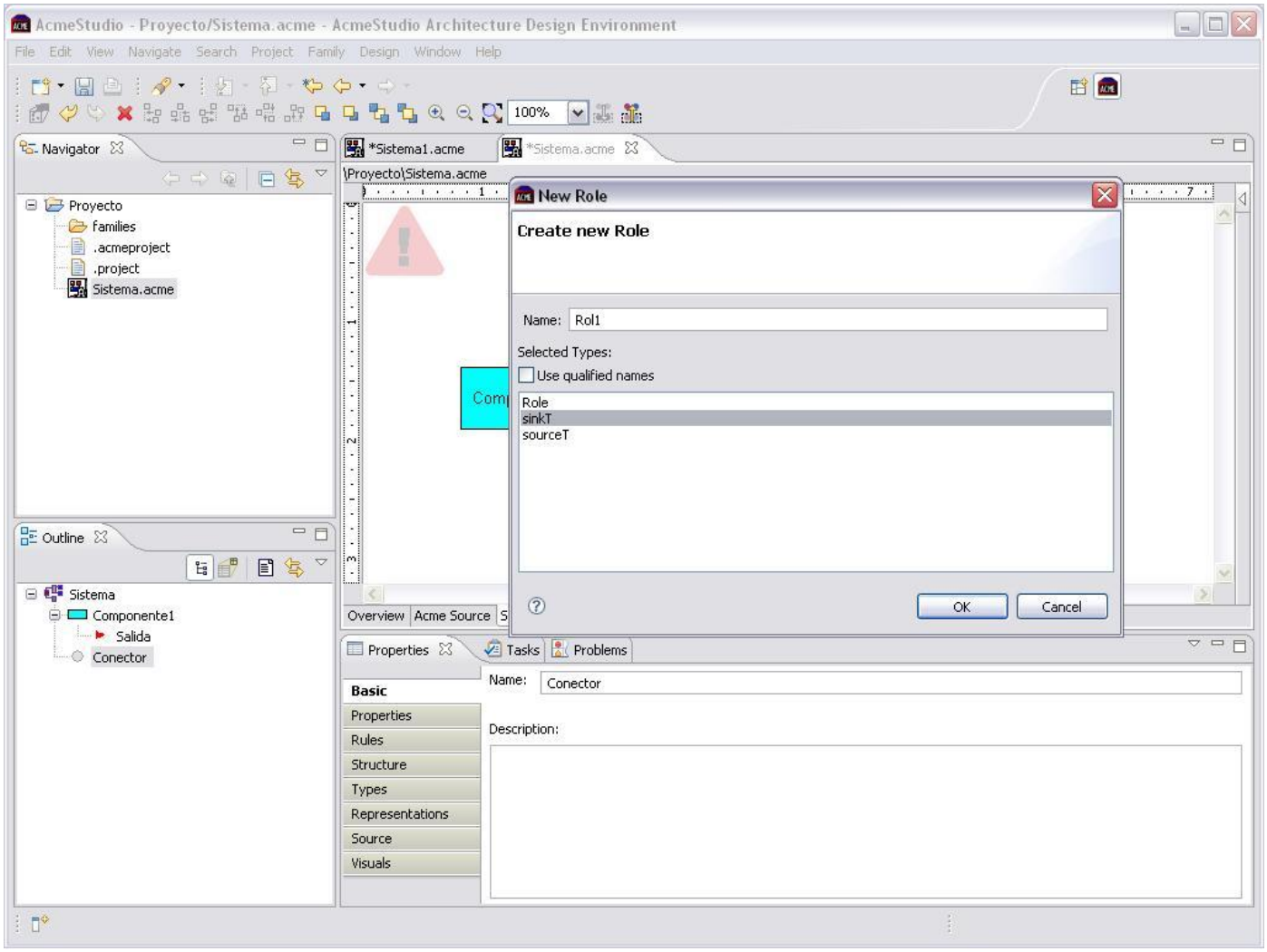


Ilustración 24 Creación de un nuevo rol al conector. Escoger el tipo de rol y nombrarlo.

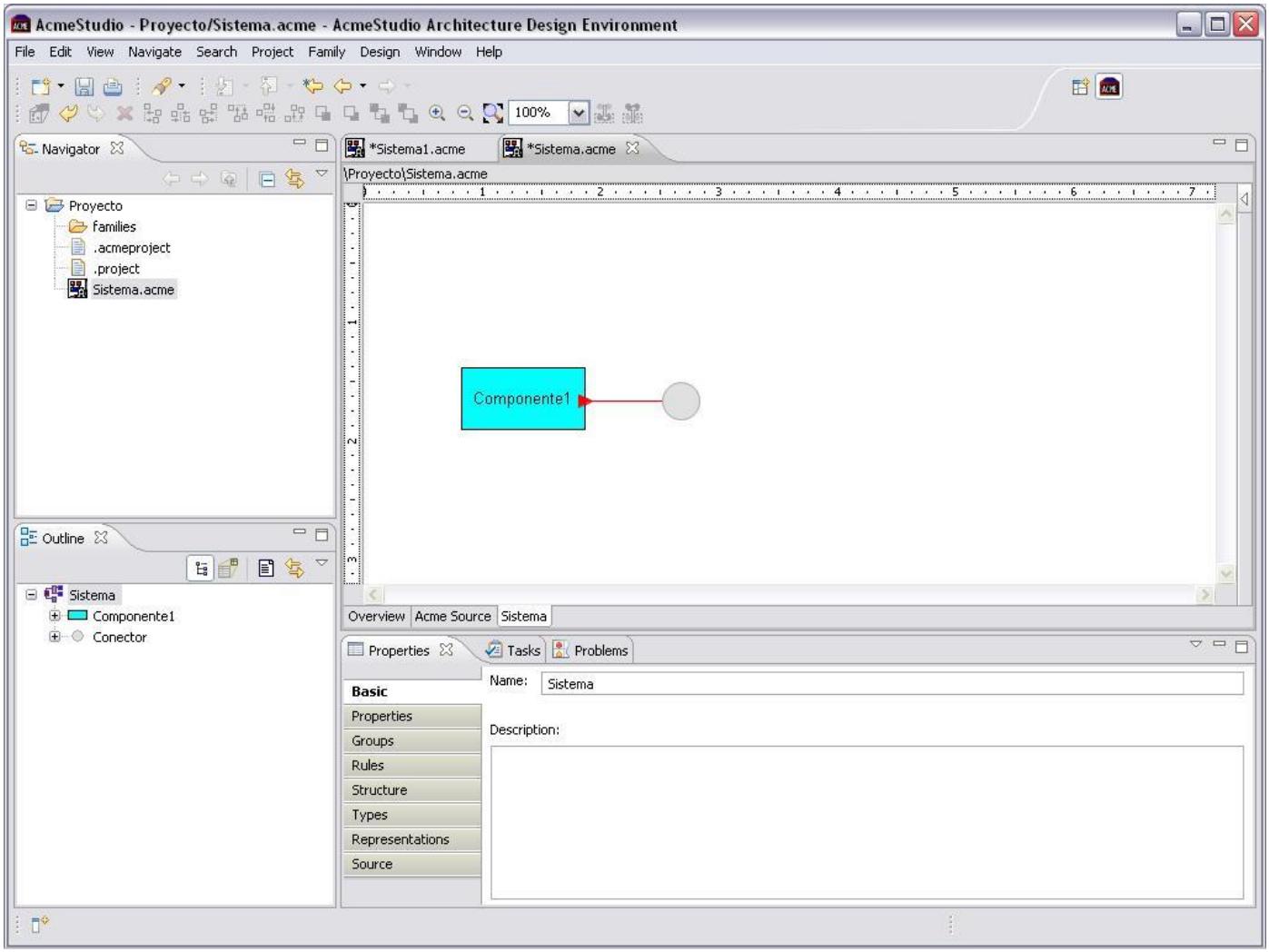


Ilustración 25 Conector conectado a través del rol al puerto del componente.

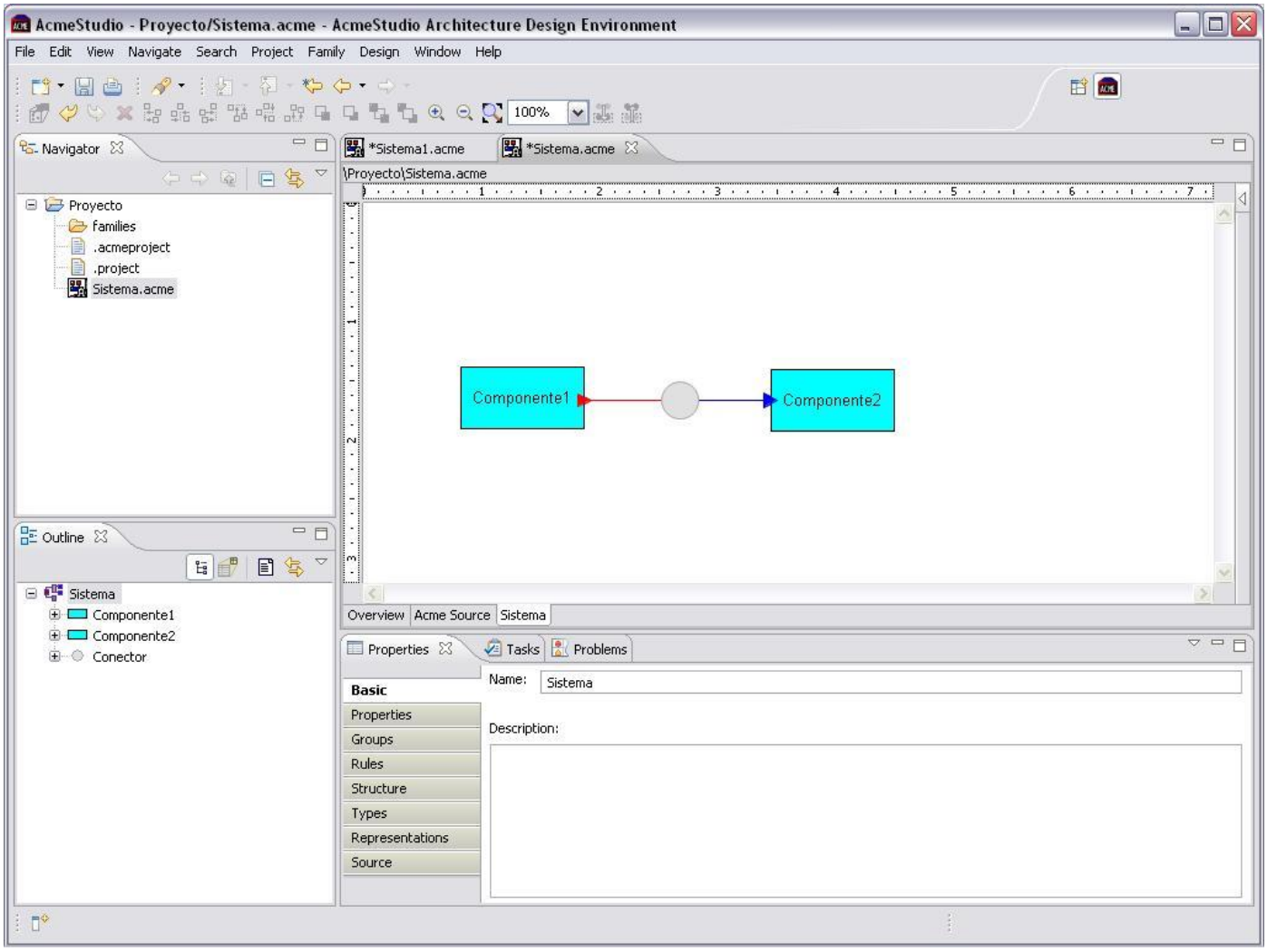


Ilustración 26 Dos componentes conectados a través de los roles del conector por sus respectivos puertos.

Anexo 7. Hacer una representación interna de los componentes con la herramienta AcmeStudio 3.4.0.

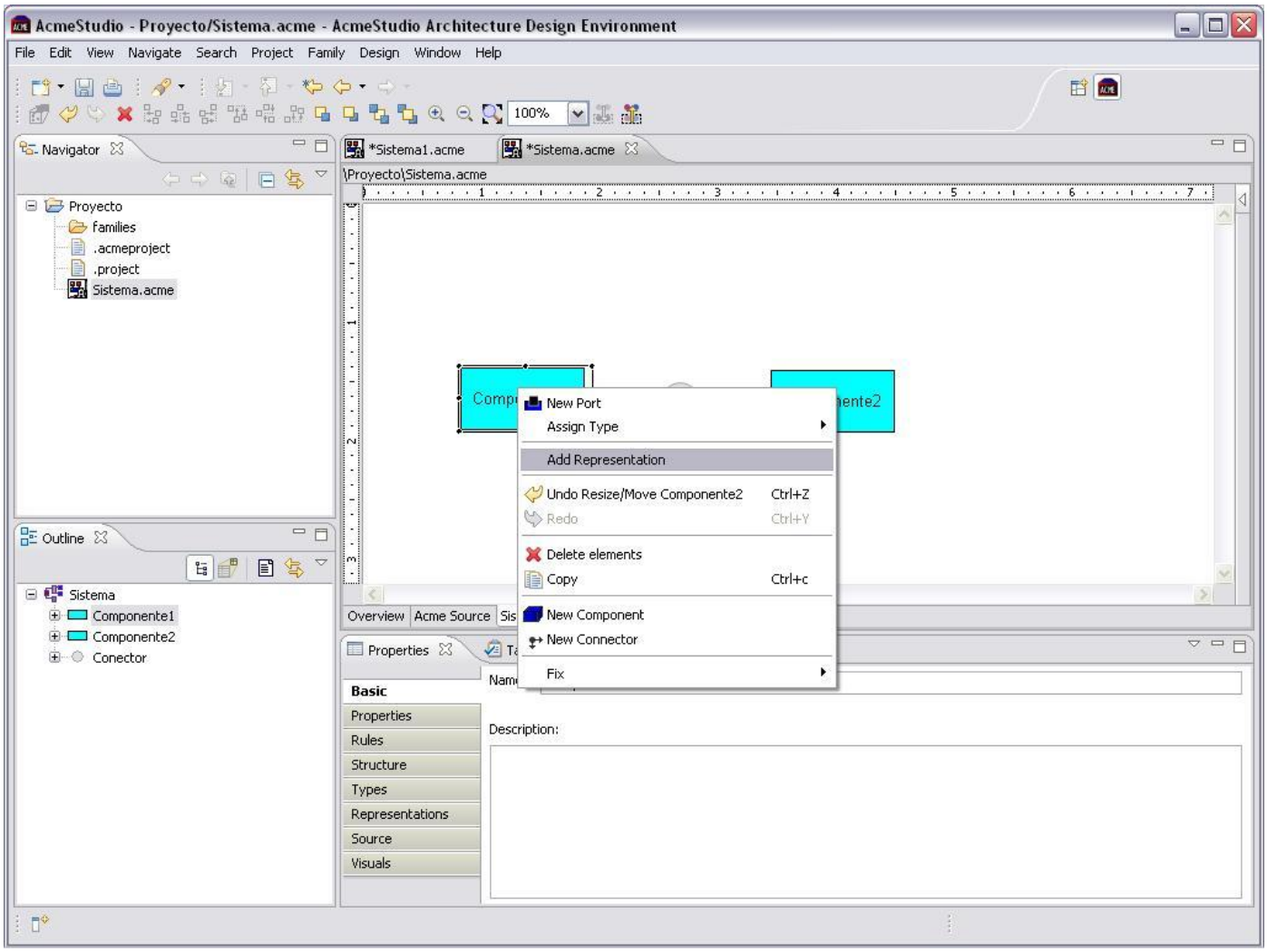


Ilustración 27 Creación de una representación.

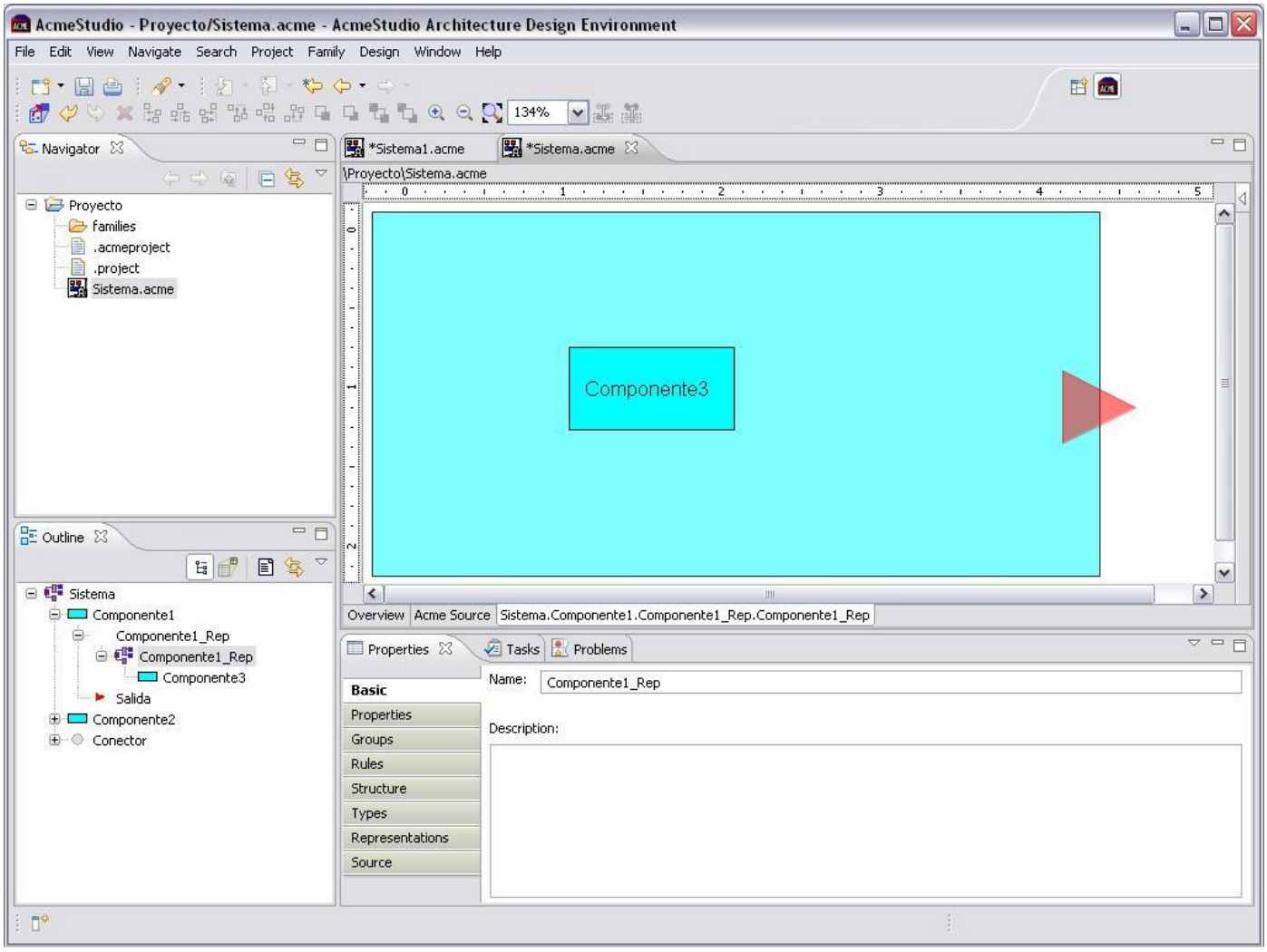


Ilustración 28 Creación de un componente dentro de la representación.

Anexo 8. Construcción de un bindings con la herramienta AcmeStudio 3.4.0.

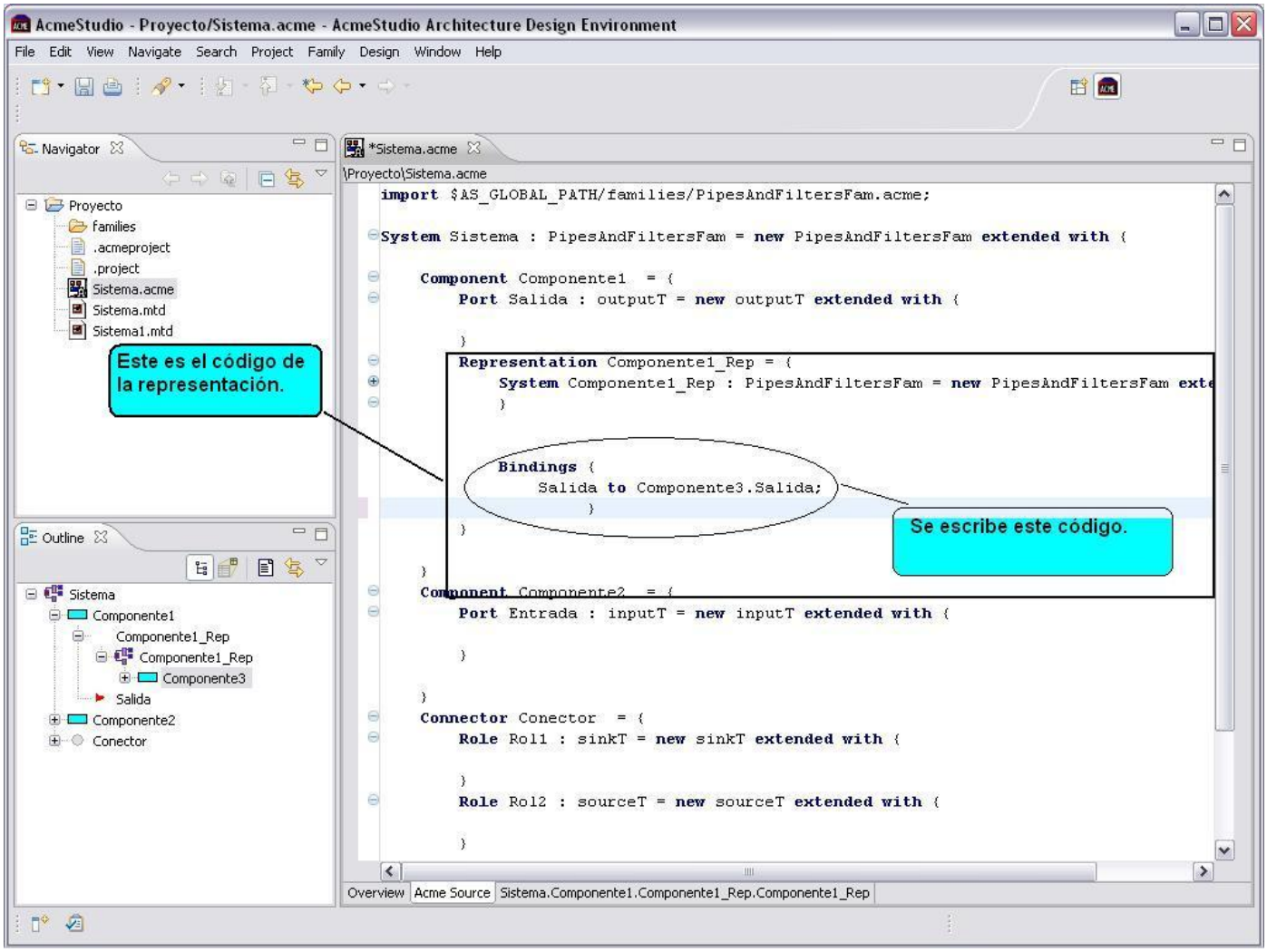


Ilustración 29 Creación de un bindings .

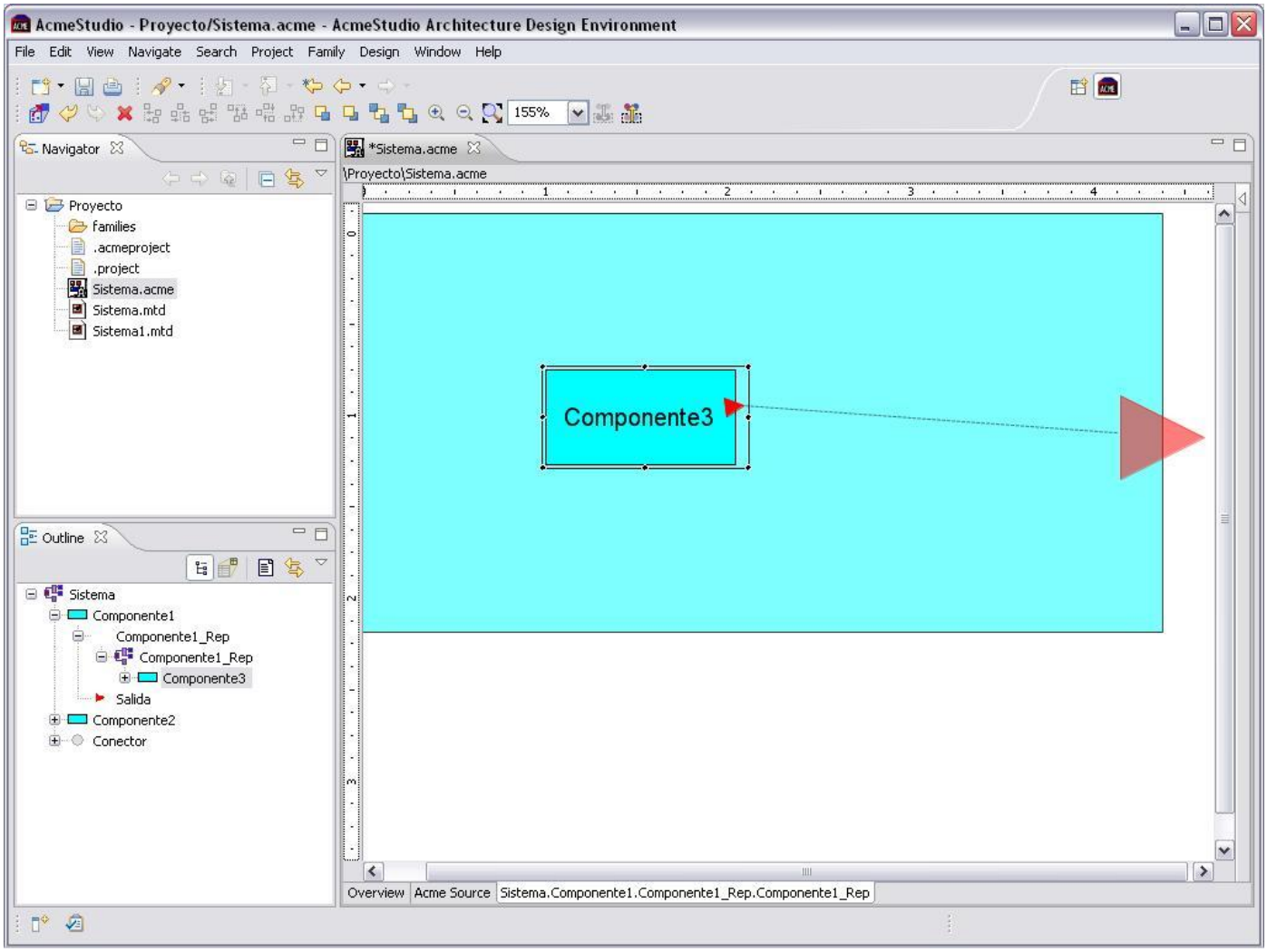


Ilustración 30 Visualización del bindings en el editor(es la línea que aparece desde el puerto del componente hasta el puerto externo).

Anexo 9. Salir de la representación interna de los componentes en la herramienta AcmeStudio 3.4.0.

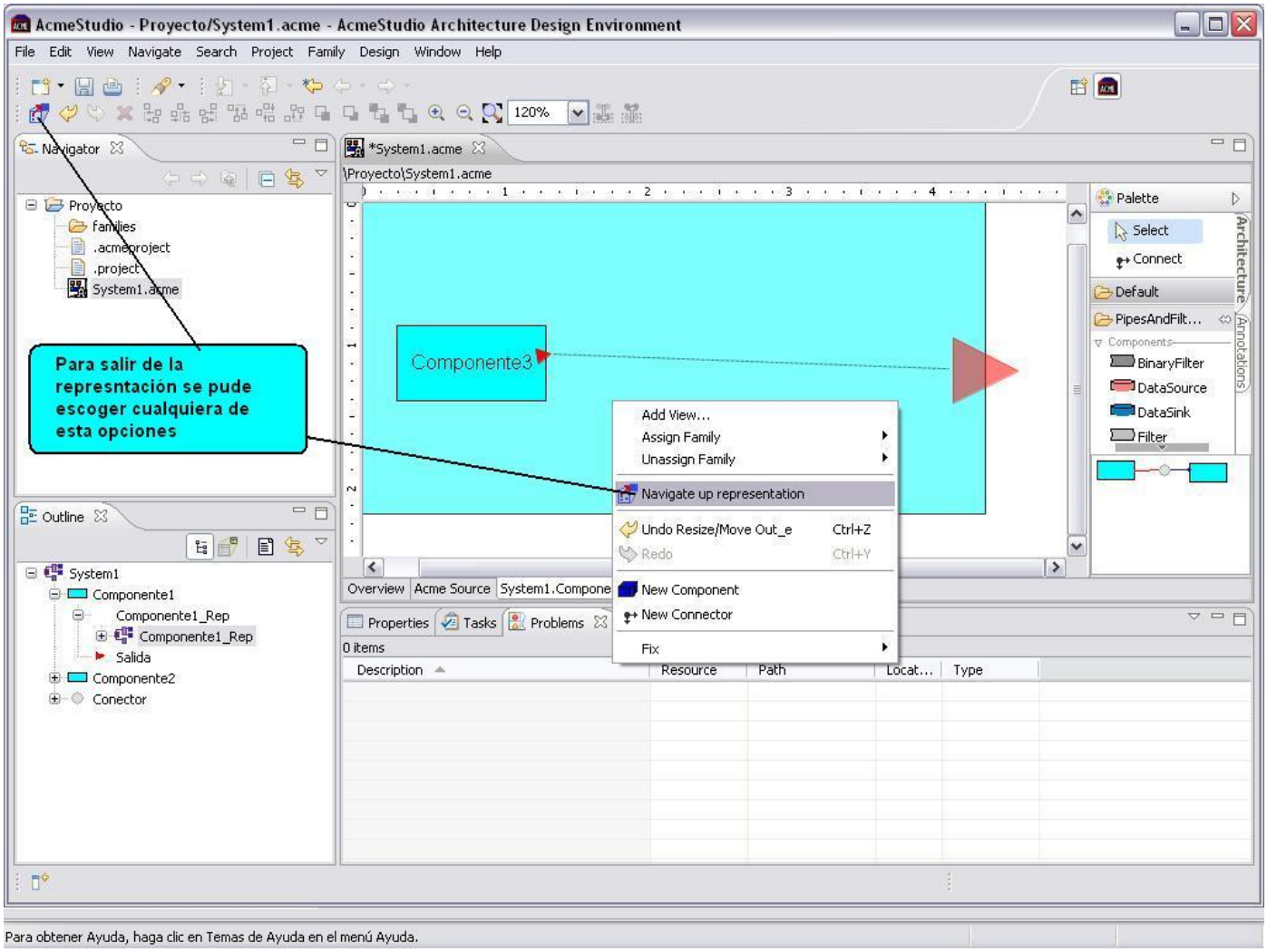


Ilustración 31 Salir de la representación.

Anexo 10. Pestaña con el nombre de Palette presente en la herramienta AcmeStudio 3.4.0.

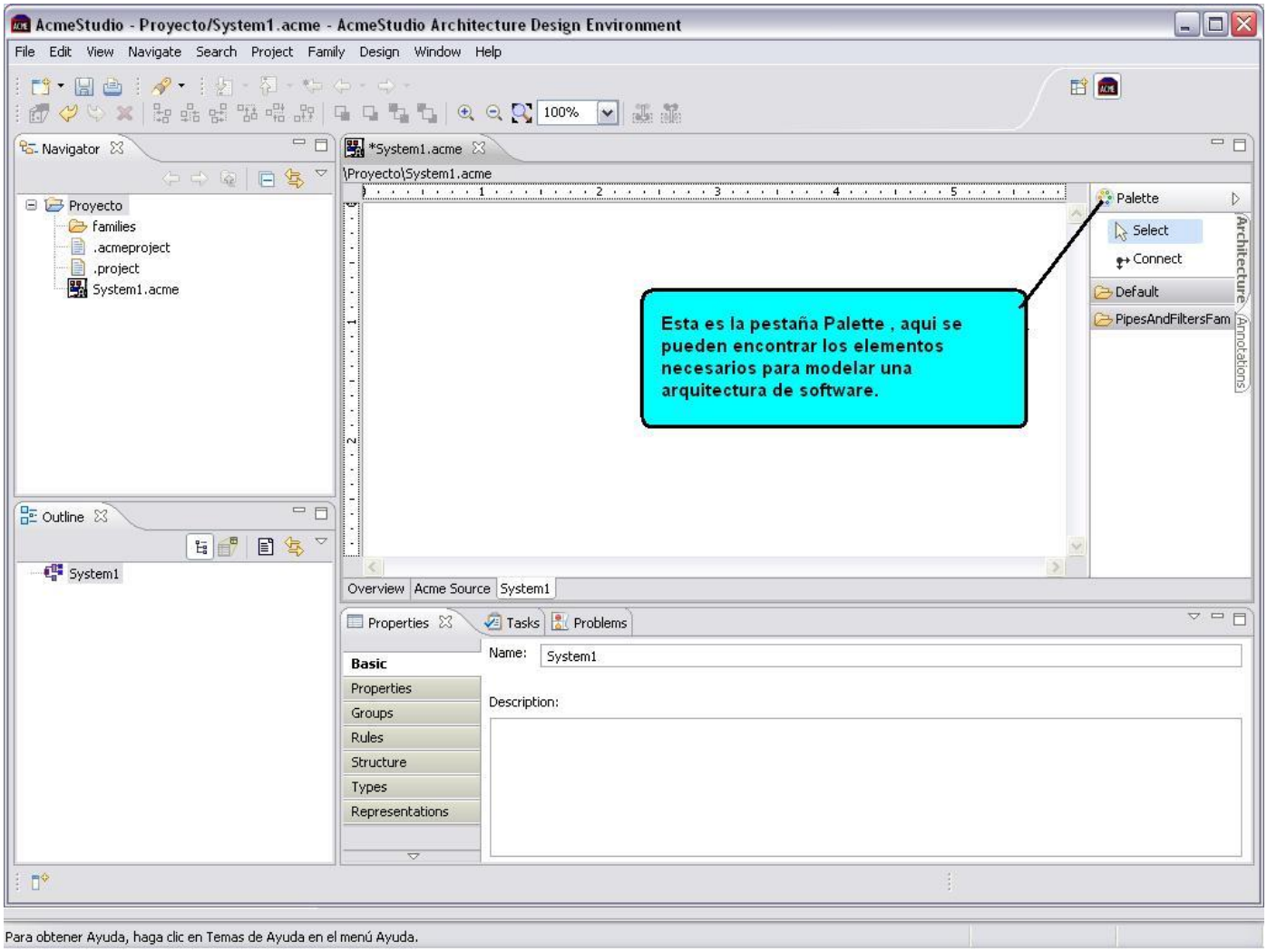


Ilustración 32 Visualización de la pestaña nombrada Palette.

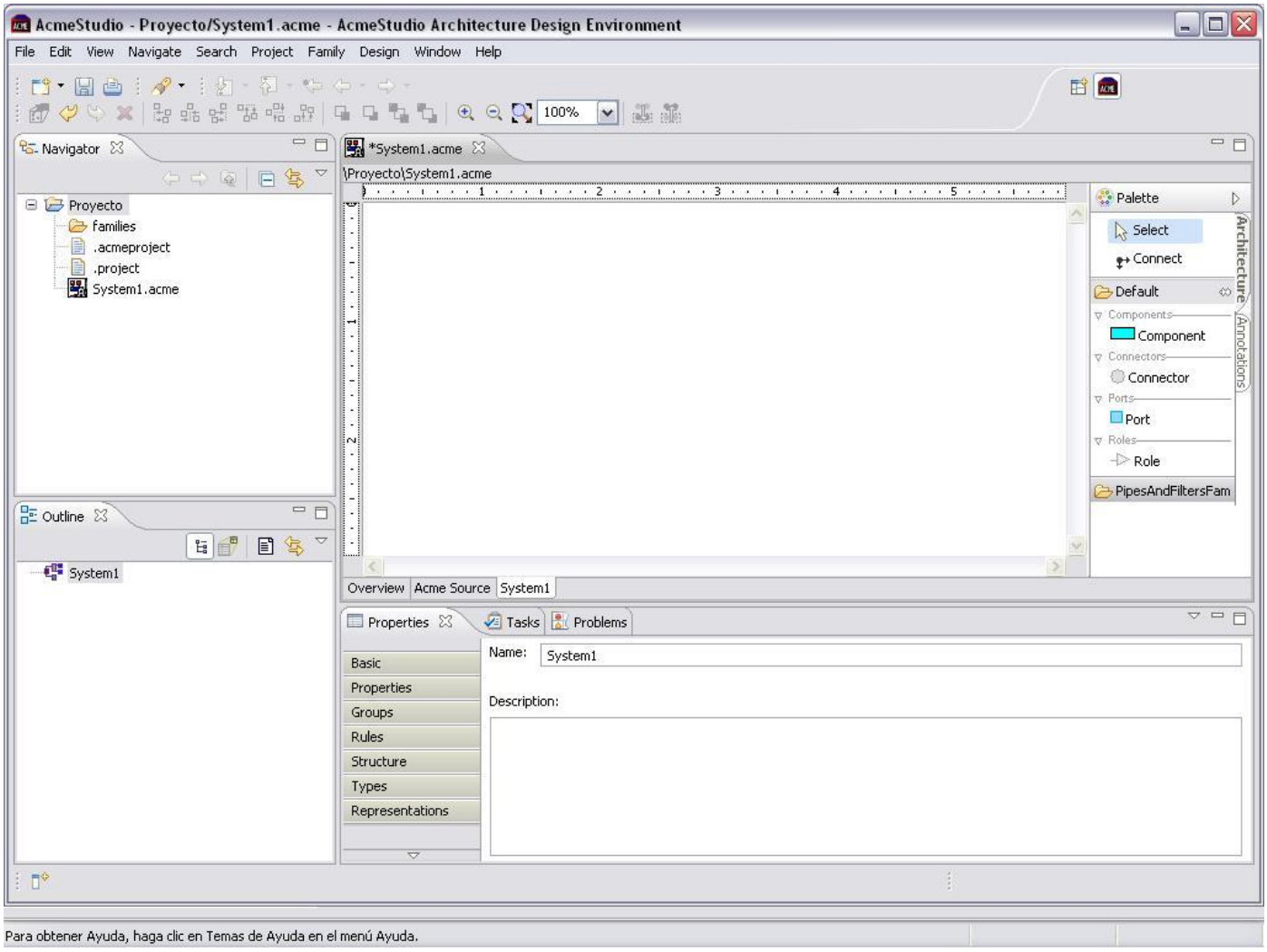


Ilustración 33 Pestaña nombrada Palette .Sección de arquitectura.

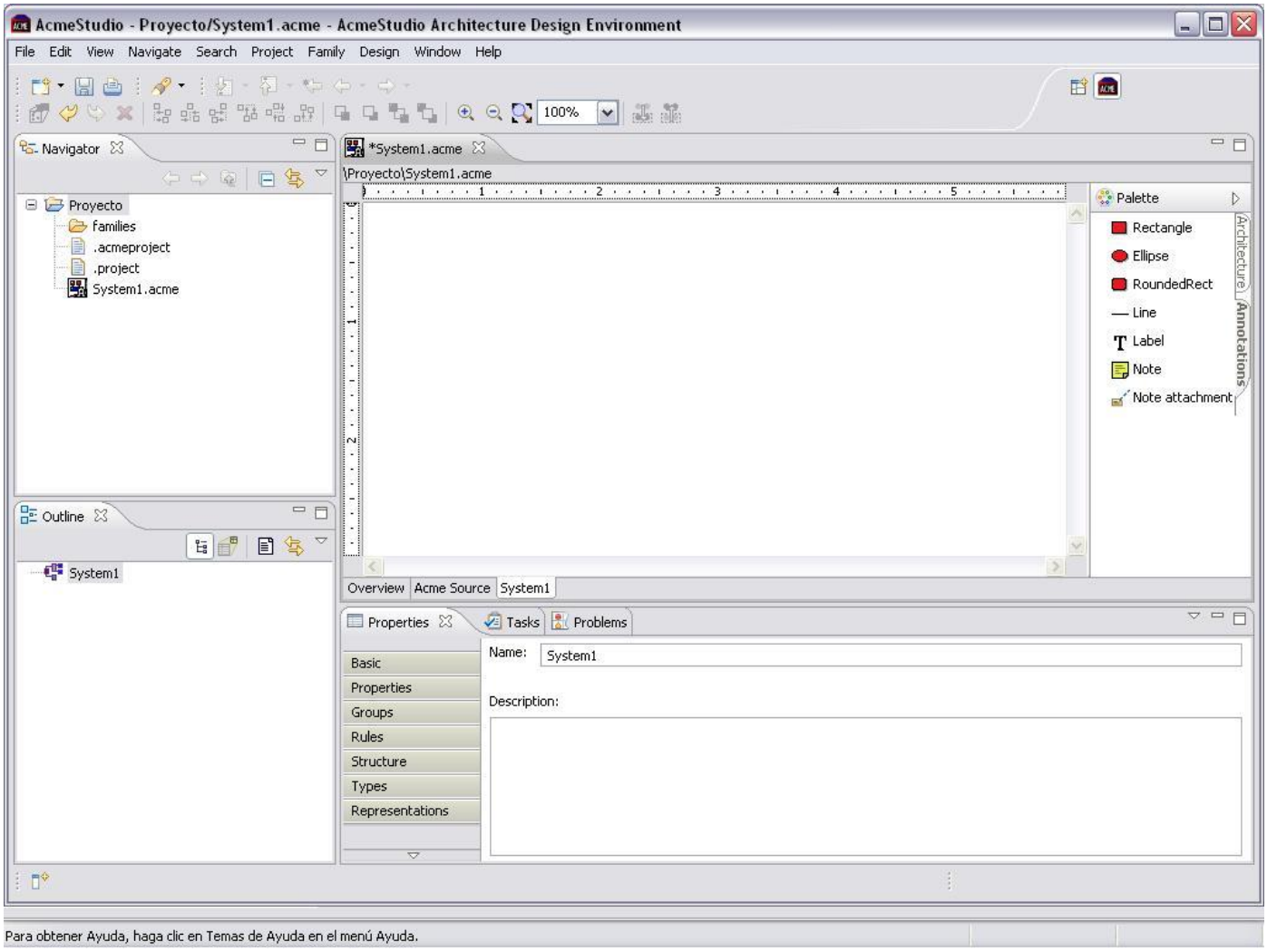


Ilustración 34 Pestaña nombrada Palette. Sección de anotaciones.

Anexo 11.Matrices de integración del Subsistema Capital Humano.

Entradas/Salidas	Configuración	Configuración/Multimon	Estructura y C	Contabilidad	Costo y Proceso
Capital Humamo	CerrarPeriodo	BuscarMonedaContable	listadoEstructurasInternas	ObtenerCuentasPorId	descripcionElemento
	PeriodoActualSubsistema	BuscarTasa	DameCargoPorId	ArbolCuentas	elementosHoja
	EjercicioActualSubsistema		DameEstructurasInternas	ObtenerCuentasHojas	centrosCosto
	FechaContableSubsistema		DameAreasPorId	ObtenerSaldoCuenta	obtenerDato
	ContabilizarNomina		BuscarCargosPorTipos	RegistrarCuentasControladas	
				ObtenerCuentaPorConcat	

Ilustración 35. Matriz de integración externa (Capital Humano).

Entradas/Salidas	administracion\incidencias	administracion\pagos_adicionales	administracion\trabajador
administracion\incidencias			BuscarTrab
			ObtenerDatosContablesPorTrabajador
administracion\pagos_adicionales			
administracion\trabajador	ObtenerImpuestos	ObtenerPagosAdicionales	
		ObtenerPagos	
		ObtenerPagosTodos	
		ObtenerPagosAdicionalesPorTrabajador	
		ObtenerPagosTrabajadores	
carga	DevolverId	ObtenerIdPagoAdic(no aparece en el ioc)	ObtenerIdcontrato
			Obtenersistemapago
			obtenerIdMotivomovnomina
			ObtenerIdreubicacion
			guardarTrabajador
			guardarPagoAdicTrab
organizacion_trabajo\puesto_trabajo		ObtenerPagosAdicionales	ObtenerDatosNomencladorTipoDeMovimiento
			ObtenerDatosNomencladorMotivoMovimiento
			InsertarModeloDeNomina
			BuscarIdModeloNomina

Ilustración 36 Matriz de integración interna (Capital Humano) parte 1.

Entradas/Salidas	organizacion_trabajo\puesto_trabajo	remuneracion_nomina\nomina	remuneracion_nomina\submayores	seleccion_integracion\persona
administracion\incidencias		ObtenerTipoNominas		
		ObtenerPeriodoPago		
administracion\pagos_adicionales				
administracion\trabajador	ObtenerIdPuestoTrabajo			BuscarPersonaPorParam
	ObtenerGruposTrabajo			BuscarPersonaPorID
	ObtenerPuestoTrabajoDandoId			BuscarPorArreglo
	ObtenerPuestoTrabajoPorGrupo			
	ObtenerPagosAdicionalesPuestoTrabajo			
	ObtenerIdCargo			
	ModificarEstadoPuestoTrab			
	ObtenerPuesto			
carga	obtenerIdPuestodetrabajo		ObtenerIdTipoRet(no parece en IOC)	guardarPersona
			ObtenerIdOperacion(no parece en IOC)	
			guardarSubRetTrab	
			guardarSubVacTrab	
organizacion_trabajo\puesto_trabajo				

Ilustración 37 Matriz de integración interna (Capital Humano)parte 2

Entradas/Salidas	administracion\incidencias	administracion\pagos_adicionales	administracion\trabajador
remuneracion_nomina\cierre			
remuneracion_nomina\nomina	TipoIncidenciaPorincidencias	DenominacionPorId	ObtenerDatosTrabajador
	ObtenerContribucionE		BuscarTrab
	ObtenerDenom		PagosAdicionalesTrab
	ObtenerIncidenciasPorTrabajador		
	ObtenerDenomporID		
remuneracion_nomina\submayores			BuscarTrab
			ObtenerTrabajadorPorIdTrabajador

Ilustración 38 Matriz de integración interna (Capital Humano)parte 3.

Entradas/Salidas	organizacion_trabajo\puesto_trabajo	remuneracion_nomina\nomina	remuneracion_nomina\submayores	seleccion_integracion\persona
remuneracion_nomina\cierre		ConfirmacionNominas	ConciliacionSubmayorVacaciones	
remuneracion_nomina\nomina	BuscarPagosAdicionalesPorPuestoDeTrabajo		ObtenerDenomRegistroRetenciones	BuscarPersonalID
			VerificarExistenciaNomina	
			VerificarExistenciaNominaR	
			Obtener_RegistroRetencion_Importe_Prioridad	
			ActualizarSubmayorRetenciones	
			ActualizarSubmayorVacacionesAcumulados	
			ObtenerDatosSubmayorVacaciones	
			DatosProgramas	
remuneracion_nomina\submayores		DenominacionPorId		BuscarPersonaPorID
		DenominacionIdNominas		

Ilustración 39 Matriz de integración interna (Capital Humano)parte 4.

Anexo 12. Matrices de integración del Subsistema Configuración.

Entradas/Salidas	Configuracion	Configuracion/Multimoneda	Contabilidad	Capital Humano
Configuracion		**BuscarMonedaPorId**	RegistrarCuentasControladas	NominasContabilizadas
		BuscarTasa	ObtenerCuentasHojas	
			ObtenerCuentasDadoEstructuraChecked	
			ObtenerCuentasPorId	
			ArbolCuentas	
			ExisteCOSinError	
Configuracion/Multimoneda	**SubsistemasCierreEntidad**			
	ContabilizarCancelacion		ObtenerCuentaFinanciera	
			ObtenerCuentasHojas	
			ArbolCuentas	
			ObtenerSaldoCuentaMoneda	
			EscribirFinCinicial	

Ilustración 40. Matriz de integración externa (Configuración).

Entradas/Salidas	Carga Inicial	Cierre	Comprobante Tipo/contrato	Comprobante Tipo/gestion
Cierre				
Comprobante Tipo/contrato				TipoDocumentoById
Comprobante Tipo/gestioncomprobante				
documentos/estadodoc		BuscarCierre(e)		
		AdicionarCierre(e)		
ejercicios				
fecha	EntidadConfigurada(e)	PeriodoActualSubsistema(e)		
multimoneda	EscribirFinCinicial(e)	SubsistemasCierreEntidad(e)	ContabilizarCancelacion(e)	
subsistemas		BuscarCierre(e)		
		AdicionarCierre(e)		

Ilustración 41. Matriz de integración interna (Configuración)Parte 1.

Entradas/Salidas	ejercicios	fecha	multimoneda	subsistemas
Cierre	UltimoPeriodo(externo)	ActualizarFechaSubsistema(e)		BuscarSubsistemasEstructura(desde el e
	PeriodoSiguiente(e,i)			BuscarSubsistemaPorId(desde el externo
	EjercicioSiguiente(e)			BuscarInstancia(desde el externo , i)
	EjercicioService(e)			
Comprobante Tipo/contrato			BuscarMonedaPorId(e)	BuscarSubsistemaPorId(e)
			BuscarTasa(e)	
Comprobante Tipo/gestioncomprobante				BuscarSubsistemasPorEntidad(e)
documentos/estadodoc	PrimerEjercicio(e)			
	PrimerPeriodo(e)			
ejercicios		FechaCinicial(e)		
fecha	ObtenerPeriodo(e)			BuscarInstancia(e)
	ObtenerEjercicios(e)			BuscarSubsistemasPorEntidad(e)
	ObtenerPeriodos(e)			
multimoneda				
subsistemas	PeriodoSiguiente			
	PrimerEjercicio(e)			
	PrimerPeriodo(e)			

Ilustración 42. Matriz de integración interna (Configuración) Parte 2.

Anexo 13. Matrices de integración del Subsistema Contabilidad.

Entradas/Salidas	Configuración	Configuración/Multimoneda	Estructura y C	Costo y Proceso
Contabilidad Financiera	SubsistemasPeriodo	BuscarMoneda	ComponenteArbol	obtenerCuentasGasto
	CerrarPeriodo	BuscarMonedaContable	listadoEstructurasT	descripcionCentro
	SubsistemasEjercicio	BuscarIdMonedaEntidad		elementoscuenta
	CerrarEjercicio(Cierre a Cierre	BuscarTasa		centrosHoja
	BuscarFormato			(Controlller)
	BuscarParteFormato			
	BuscarFormatoPorID			
	BuscarParteFormato			
	ObtenerPeriodos			
	ObtenerPeriodo			
	PeriodoActualSubsistema			
	EjercicioActualSubsistema			
	FechaContableSubsistema			
	ActFechContabSubsist			
	BuscarSubsistemaPorId			
	ObtenerObjPorSubsist			
	ObtenerSubsistemasCont			
	ObtenerOperacionesPorObj			
	ExisteCuenta			
	ObtenerEjercicio			
	CuentasContrPorObjeto			

Ilustración 43. Matriz de integración externa (Contabilidad).

Entradas/Salidas	comprobante_operaciones	configuraciones	nomenclador_cuentas
cierre	ObternerEstadoComprabantes	ObtenerEstructuraE	ObtenerCuentasHojasSL(e)
			ObtenerCuentasHojas(e)
			ObtenerCuentasDadoEstructura(e)
comprobante_operaciones		ObtenerEstructuraE(e)	ObtenerCuentasHojas(e)
		ObtenerEstructura	ObtenerCuentasPorId(e)
		ObtenerUM	ObtenerAperturasInmediatas
			ObtenerAperturasSuperiores
			ObtenerCuentaPorId
configuraciones			ExisteEstructuraEconomica
			CargarCuentaTipo
			ObtenerCuentasHojas
nomenclador_cuentas	cargarPaseCuenta	ObtenerNaturaleza	
		ObtenerContenidosEconomicos	
		ObtenerSubArbol	
		ObtenerNombreTabla	
		ObtenerIdentificadorTabla	
		ObtenerDescripcionTabla	
		ObtenerCentroRectorDe	
recuperaciones	InvitadoComprobante	ObtenerEstructura	ObtenerCuentaPorId
	CuentaSaldoPeriodo	BuscarEstructura	ObtenerHojas
	getIdCentroComun(ningun IOC)	ObtenerEstructuraE	ObtenerCuentasHojas
		ArbolEstructuraE	ObtenerCuentasPadres

Ilustración 44. Matriz de integración interna (Contabilidad).

Anexo 14. Matrices de integración del Subsistema Costos y Procesos.

Entradas/Salidas	Configuración	Configuración/Multimoneda	Estructura y C	Contabilidad
Costos y Procesos	ContabilizarAjusteCosto	BuscarMonedaContable	DameEstructurasInternas	ObtenerCuentasPorId
	ObtenerOperacionesPorId	BuscarIdMonedaEntidad	DameHijosInterna	ObtenerSaldoCuenta
	TipoDocumentoByAlias	BuscarTasaActual		ObtenerCuentasDadoEstructura
	ObtenerEjercicio			SaldoCCEL
	ObtenerPeriodo			ExisteDocPCA
	ContabilizarSecuencia			ModificarTipoCuentaDesdeCosto
	PeriodosAnteriores			
	FechaContableSubsistema			
	ObtenerUnidadesMedidaVolumen			
	CerrarPeriodo			
	SubsistemasPeriodo			
	CerrarEjercicio			
	BuscarFormato			
	BuscarFormatoPorID			
	BuscarParteFormato			

Ilustración 45. Matriz de integración externa (Costos y Procesos).

Entradas/Salidas	cierres_traspasos	nomenclador_configuracion
ajustes_costos		obtenercuentaspadrespatr
		obtenercuetasghijas
		obtenercentros
		obtenercentrosp
		obtenerelemnom
cierres_traspasos		obtenerelementosnomenclador
		obtenerelemnom
		obtenercuenta
		obtenercentros
		obtenerelementos
		obtenercuentaspatrimoniales
		mostrarcentrosorigen
nomenclador_configuracion	Obtenercentrosdestino	
	Obtenercentrosorigen	

Ilustración 46. Matriz de integración interna (Costos y Procesos).

Anexo 15. Matrices de integración del subsistema Logística. Incluye a los subsistemas Inventario y Facturación.

Entradas/Salidas	Configuración	Estructura y C	Contabilidad	Seguridad	Logistica
Inventario	OperacionesPorId	DameEstructurasInternas	ObtenerCuentaPorId	getUsersProfile(apertura)	obtenerUmedida
	TipoOperacionesDoc	DameEstructura		ObtenerEstructurasNoFormales	obtenerUmedida
	TipoDocumentoByAlias	BuscarGradosMilitar			
	BuscarParteFormato	DameAreasPorId			
	BuscarFormatos				
	TipoDocumentoPorCategoria				
	BuscarMonedaContable				
	BuscarFormatoPorID				
	BuscarMoneda				
	SoloMonedaCont				
	ExisteMonedaAlertnativa				
	ObtenerEspecialidadPorId				
	TipoDocumentoById				
Facturacion	SoloMonedaCont	BuscarGradosMilitar	ObtenerCuentaPorId	getUsersProfile	obtenerUmedida
	ObtenerClientes	DameEstructura	ObtenerCuentasDadoEstructura		obtenerUmedida
	BuscarMonedaPorId	DameEstructurasInternas			
	BuscarFormatoPorID	DameAreasPorId			
	BuscarParteFormato				
	ObtenerClientes				
	BuscarMoneda				
	BuscarFormatos				

Ilustración 47. Matriz de integración externa (Logística).

Anexo 16. Matrices de integración del subsistema Planificación.

Entradas/Salidas	Configuración	Estructura y C	Costo y Proceso	Logistica	Seguridad
Planificación	ObtenerEjercicio	DameEstructura	elementosHoja	obtenerUmedidaDadoId	dominoHastaUsuario
	ObtenerConceptos	MostrarCamposEstructura	centrosEstructura	obtenerProducto	nombreUsuario
	ObtenerContatoGeneral			ConvertirUM	
				obtenerUMedidad	
				obtenerProductoDadoIdEstructura	

Ilustración 48. Matriz de integración externa (Planificación) .

Entradas/Salidas	plantilla	procesador_matematico\editor_de_ecuaciones	realizacion_de_plan
configurar_plan	getIdModelos		
	getPlantillasbyidmodelo		
	getPlantilla		
construccion_planes	ConfEdicCelda	DameFormula	
	dameColumnasP		
	dameReader		
	dameIndicadores		
	getPlantillasbyidmodelo		
importar_exportar	ObtenerIdTipoModelo		DameEstadoPlan
	ObtenerIdPlantilla		

Ilustración 49. Matriz de integración interna (Planificación)parte 1.

indicadores	isUsed		IndicadorenUso
nomencladores	tipoModeloUsado		
plantilla		DameFormula	
procesador_matematico\editor_de_ecuacione	dameIndicadores		
	DameIdCelda		
	dameColumnasP		
	DameColumnasInd		
realizacion_de_plan	GetIdtipoModelo	DameResultado	
	dameColumnasP		
	dameReader		
	ConfEdicCelda		
	dameIndicadores		
	getColumna		

Ilustración 50. Matriz de integración interna (Planificación) parte 2.

Anexo 17. Modelado interno de los subsistemas seleccionados.

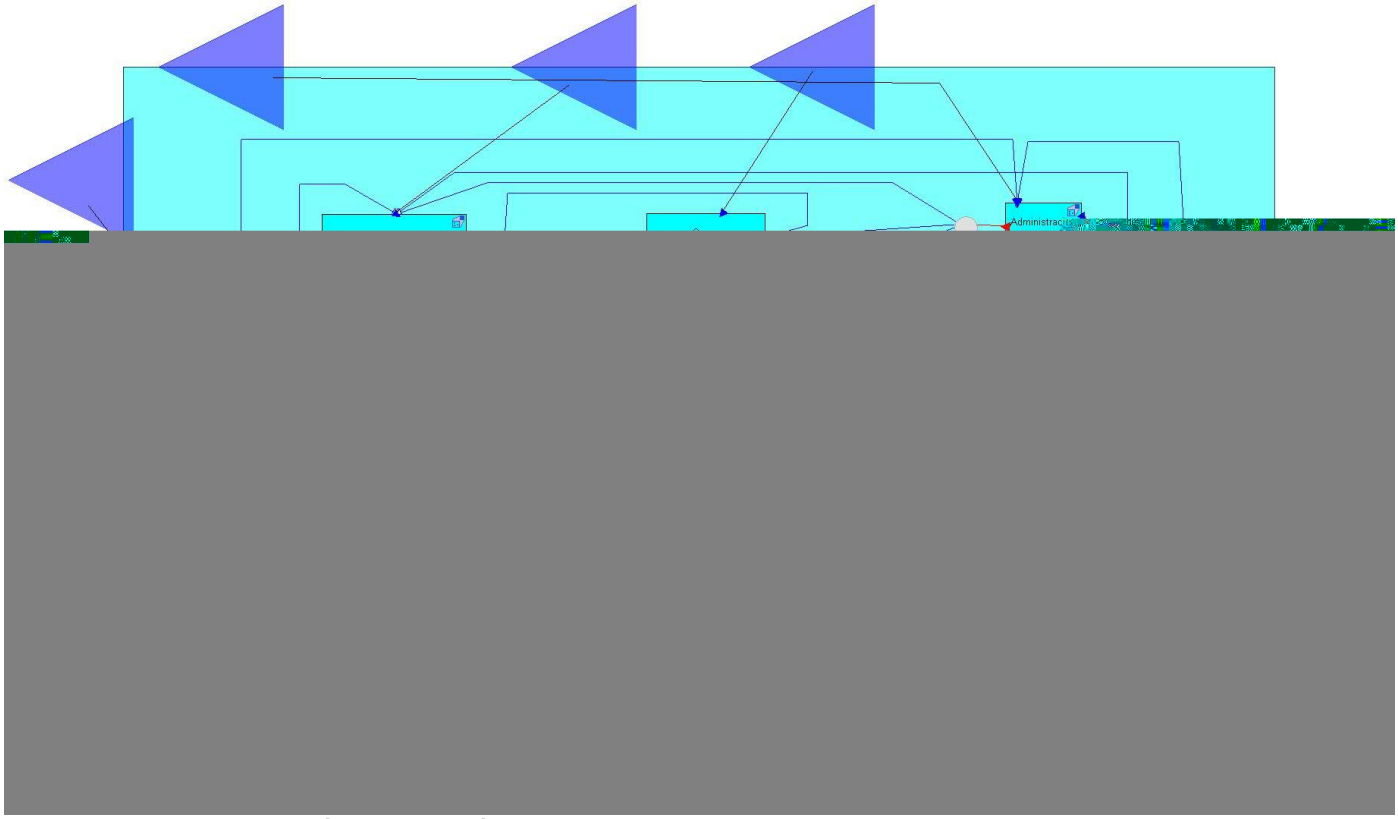


Ilustración 51 Relación interna general de los componentes (Capital Humano).

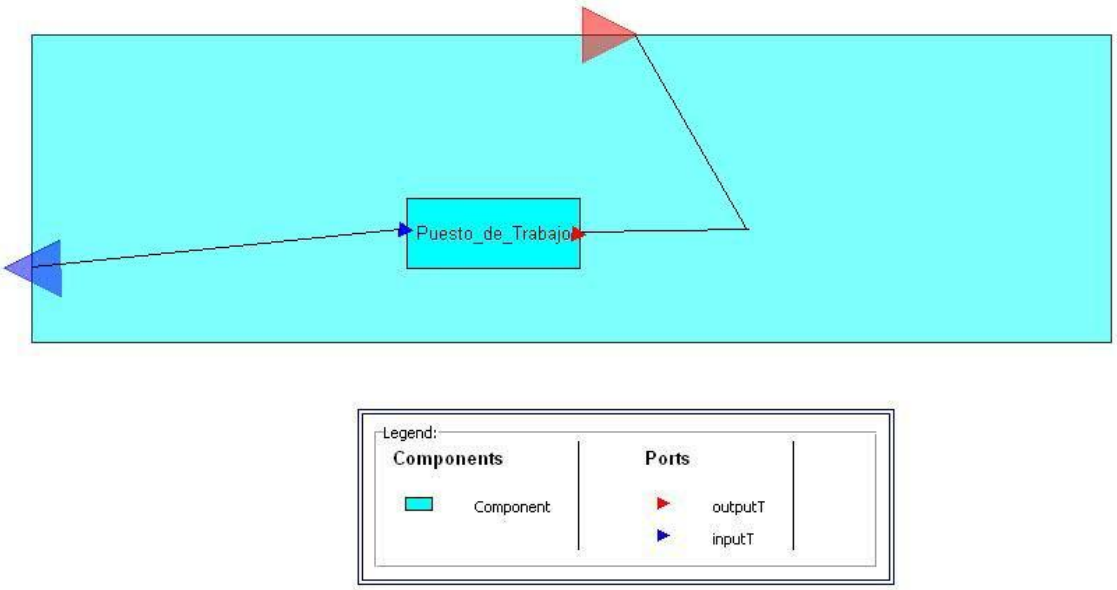


Ilustración 52 Representación de Remuneración del Trabajo(Capital Humano)

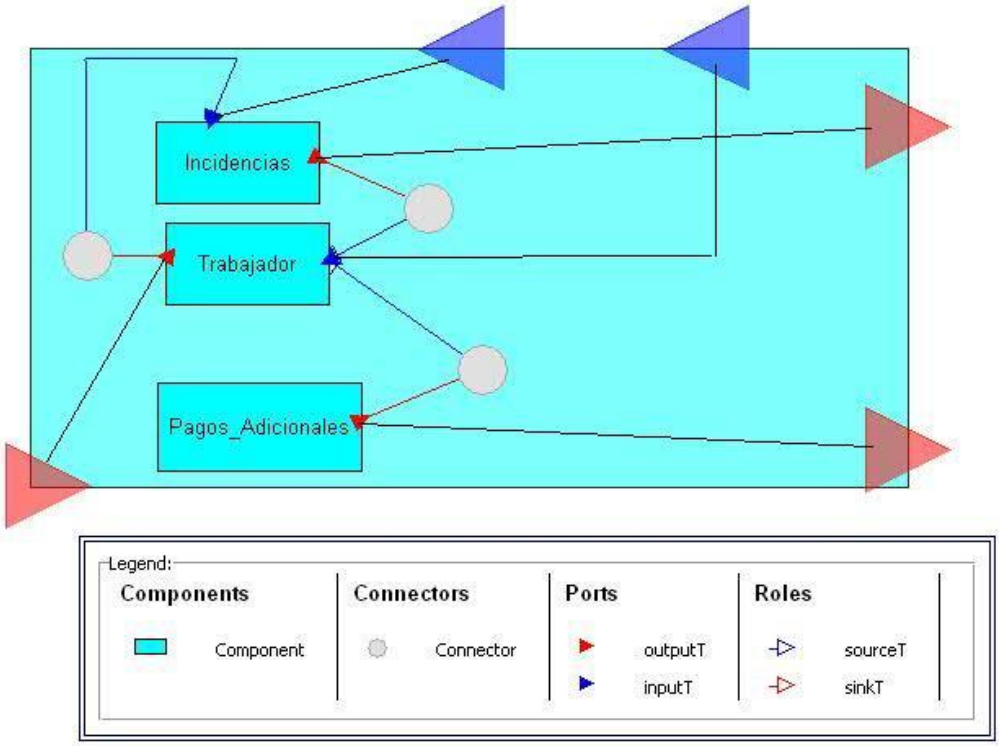


Ilustración 53 Representación interna de administración (Capital Humano).

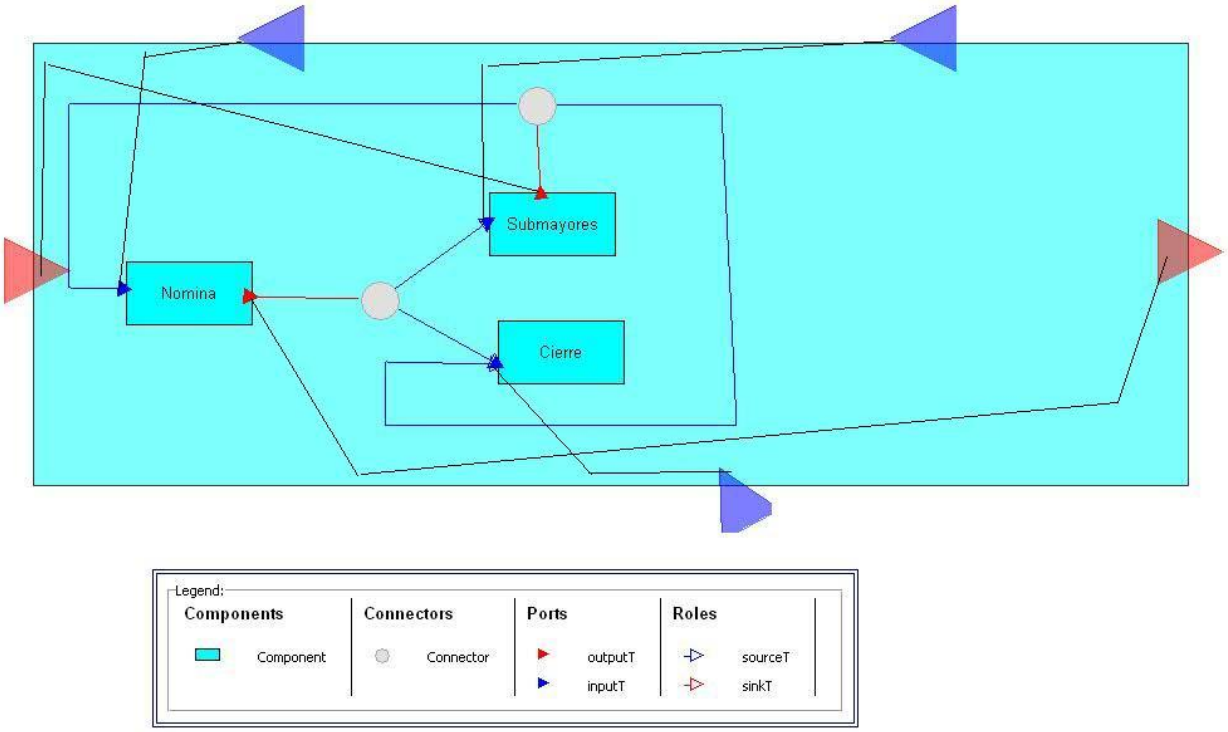


Ilustración 54 Representación interna de Remuneración y Nomina (Capital Humano).

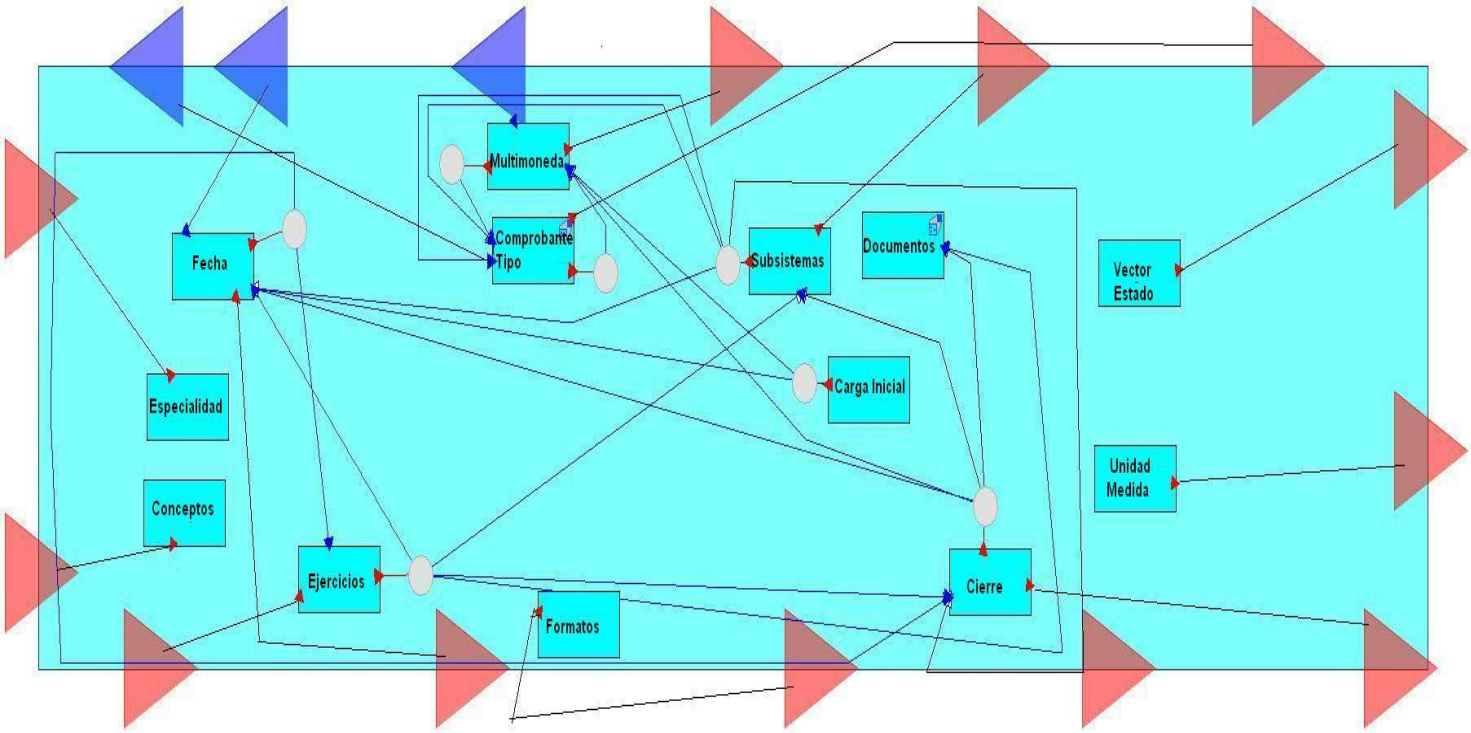


Ilustración 55 Representación interna general de los componentes (Configuración).

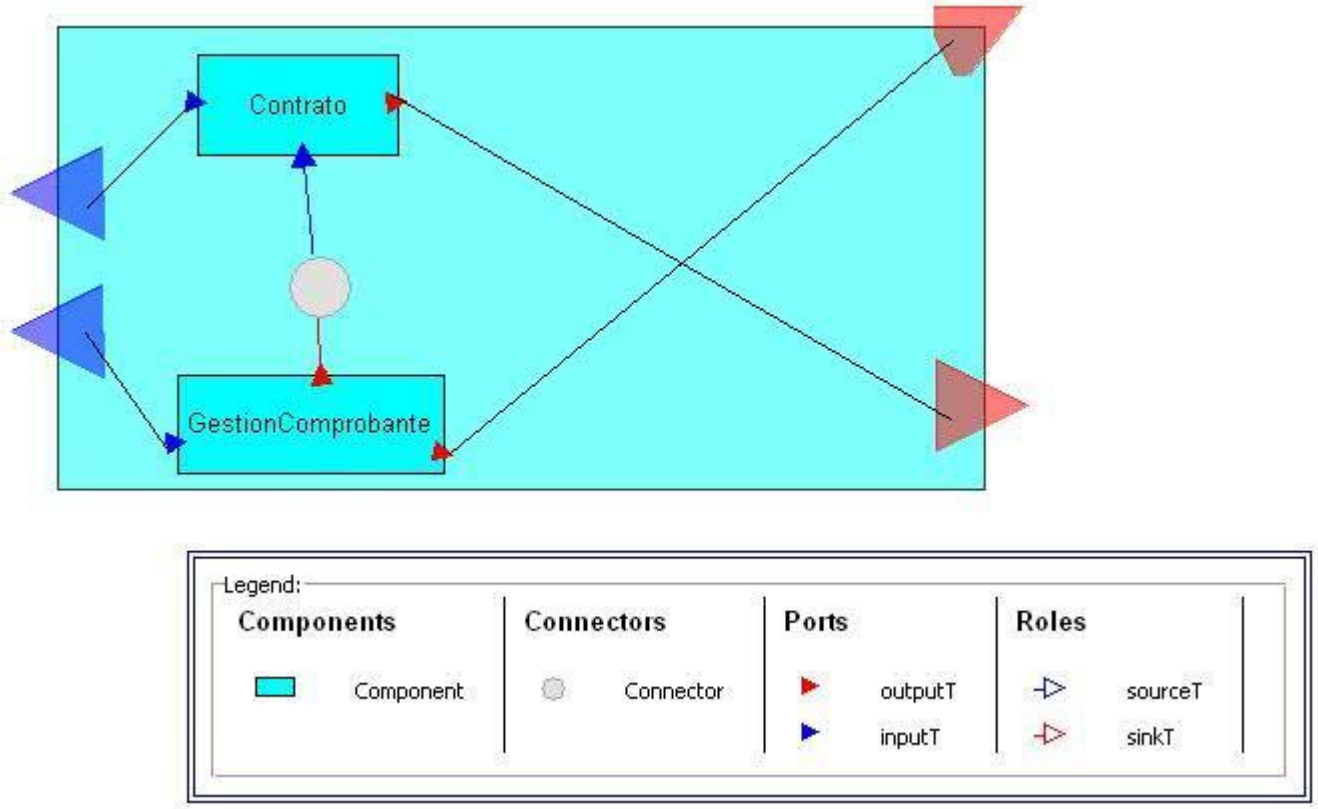


Ilustración 56 Representación interna de Comprobante Tipo (Configuración).

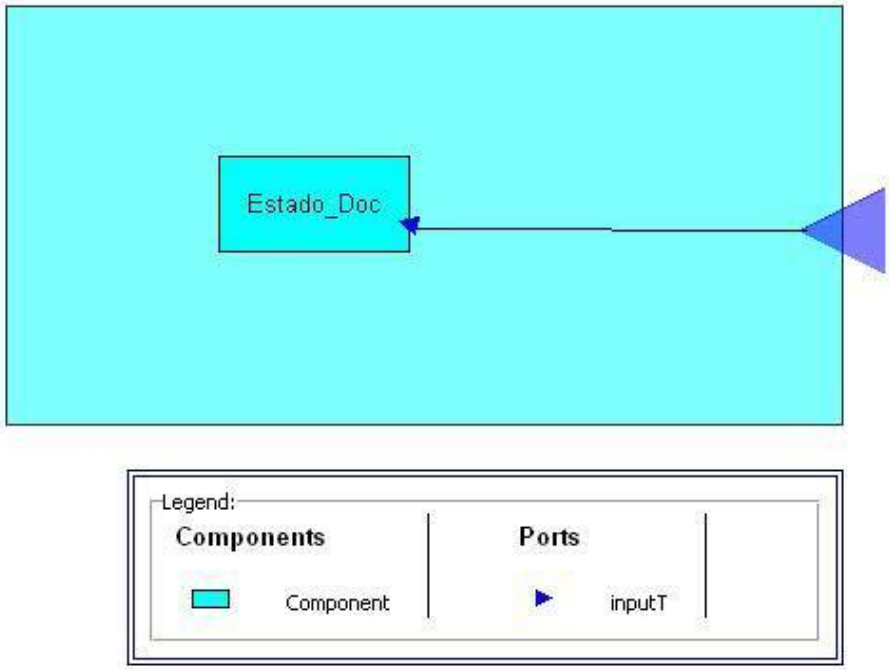


Ilustración 57 Representación interna de Documentos(Configuración).

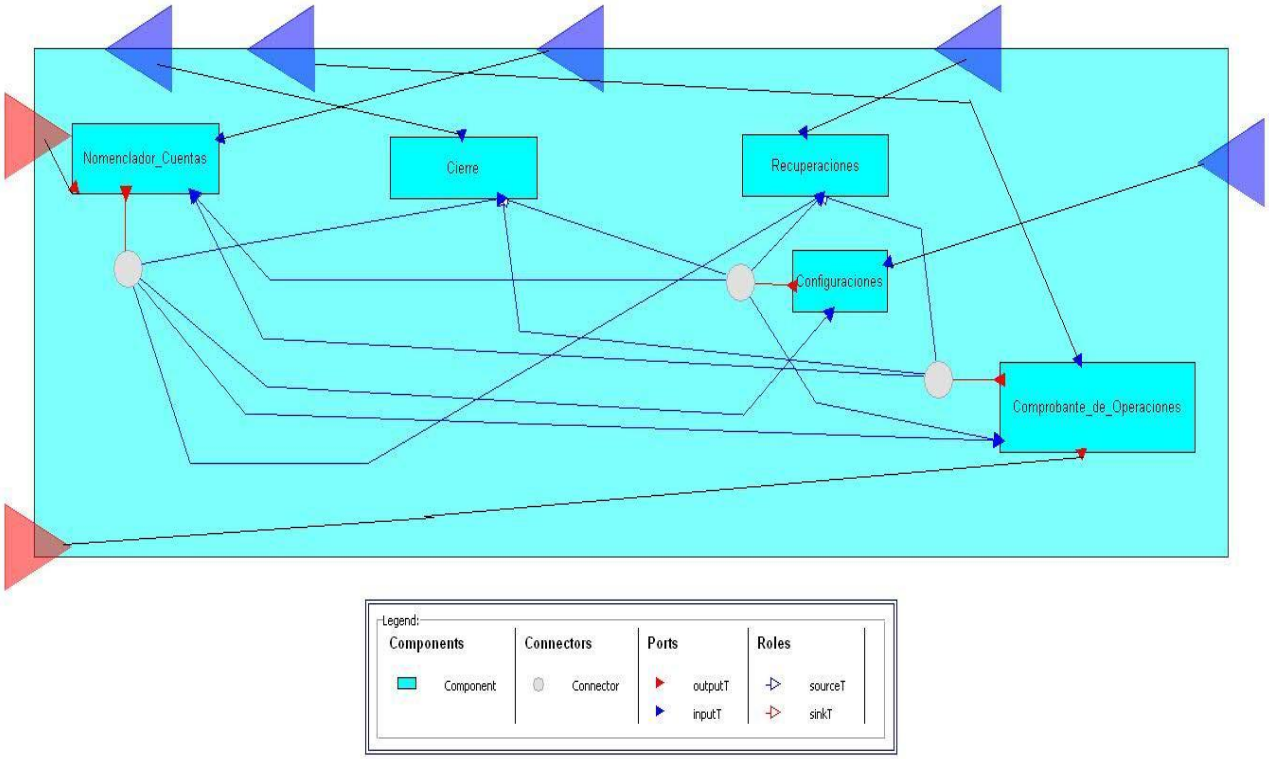


Ilustración 58 Representación interna general de los componentes (Contabilidad).

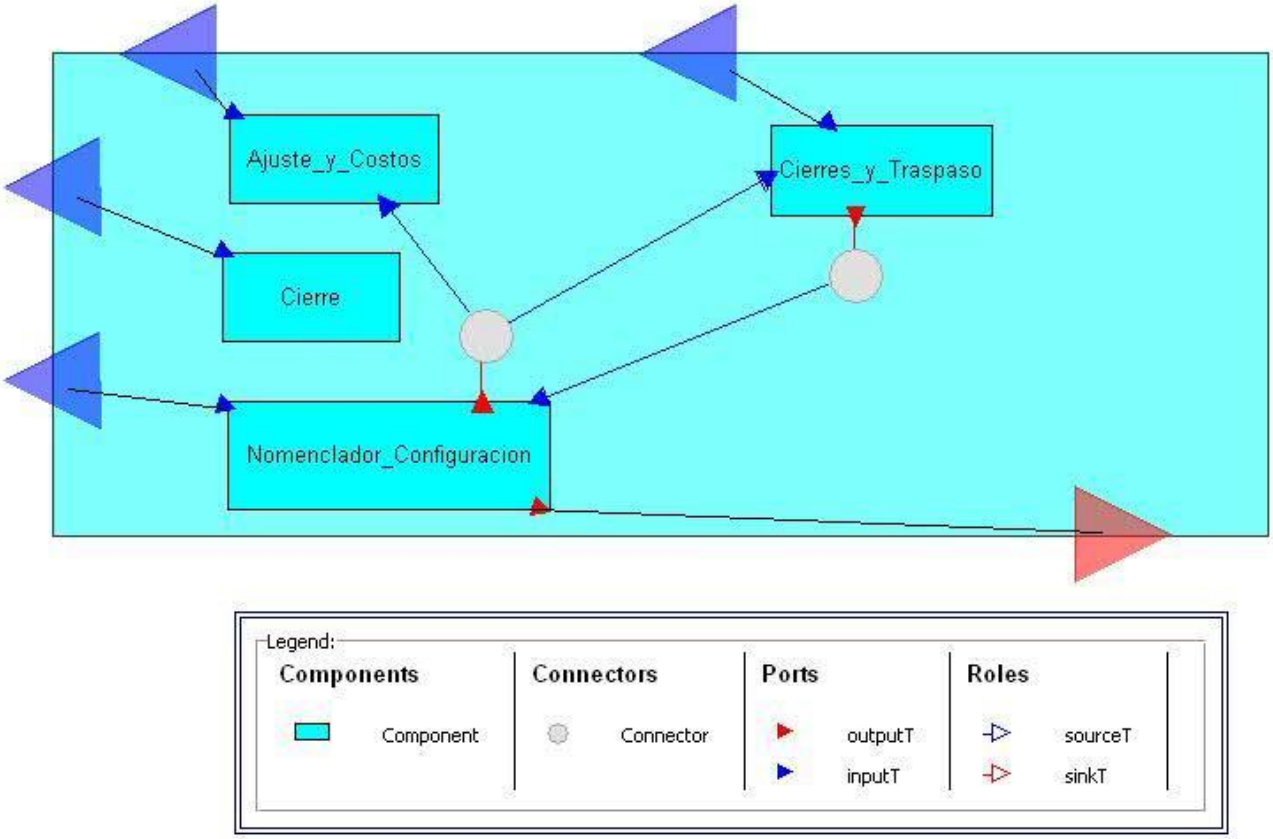


Ilustración 59 Representación interna general de los componentes (Costos y Procesos).

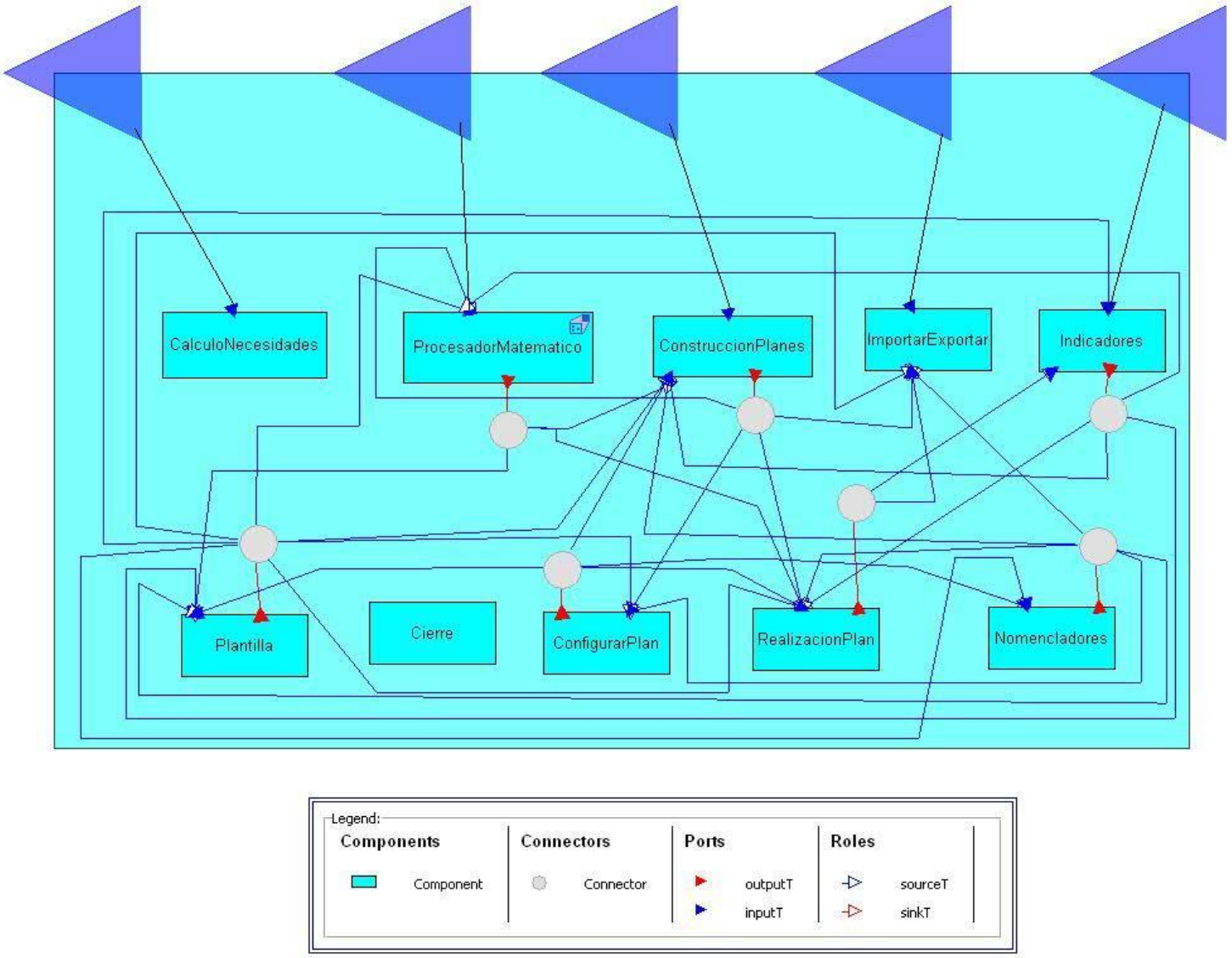


Ilustración 60 Representación interna general de los componentes (Planificación).

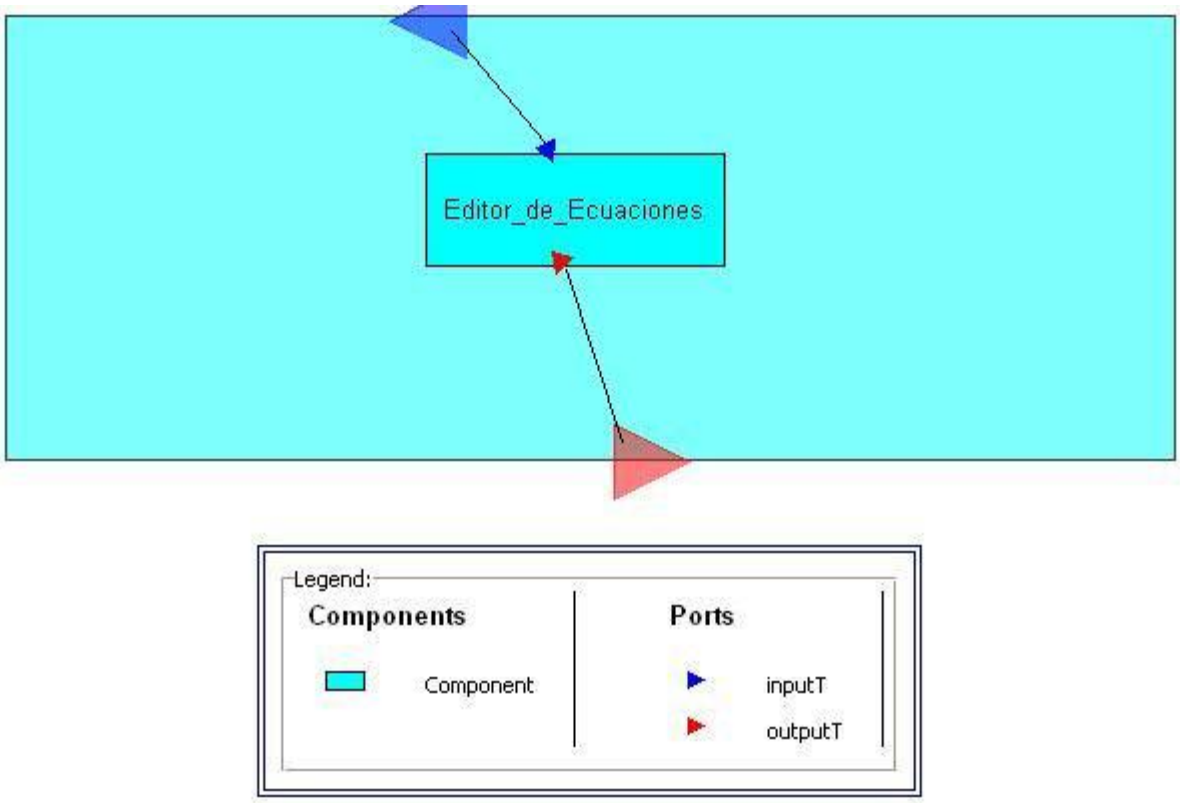


Ilustración 61 Procesador matemático (Planificación).

Anexos