# PASTEUR: Un Sistema para Indexación y Búsqueda de Componentes de Software

PASTEUR: An Indexing and Localization System for Software Components

Alexandra Pomares Quimbaya
*Universidad de los Andes,*
*Universidad Javeriana, Colombia*
a.pomares267@uniandes.edu.co

Javier Mauricio Morales Chavarro
*Universidad de los Andes,*
*Zemoga S.A., Colombia*
jm.morales120@uniandes.edu.co

## Resumen

*Este artículo presenta PASTEUR, un sistema de indexación y localización de componentes de software a gran escala que promueve el comercio de componentes en ambientes abiertos, dinámicos y heterogéneos. Su diseño e implementación está apoyado en la tecnología de los sistemas P2P DHT de los cuales toma los servicios de búsqueda y almacenamiento distribuido, adicionando servicios de datos distribuidos, particularmente para el manejo de metadatos y consultas avanzadas a partir de ellos. La propuesta de PASTEUR como capa DDS está propuesta para ser usada en el ámbito de identificación y selección de componentes pero puede ser utilizada en cualquier ámbito en donde se requiera compartir recursos. PASTEUR fue desarrollado usando una implementación de código abierto de Pastry, sus funcionalidades fueron probadas y evaluadas en un ambiente controlado.*[1]

## Abstract

*This article presents PASTEUR, a system of indexing and localization of software components on large scale that promotes the commerce of components, in open, dynamic and heterogeneous environment. Its design and implementation are supported in the technology of systems P2P DHT from which it takes the distributed lookup and distributed storage services, adding services of distributed data, particularly for the handling of metadata and queries from them. The PASTEUR proposal as a DDS is proposed to be used in the scope of identification and selection of components, but can be used in any scope where it is required to share resources. PASTEUR was developed using an open source implementation of Pastry, their functionalities were proven and evaluated in a controlled environment.*

## 1 Introducción

El desarrollo basado en componentes surgió de la necesidad de reducir el tiempo y el costo de construcción del software, gracias a la facilidad de reutilización de partes de software ya construidas [1]. Este nuevo enfoque promete mejorar la calidad, productividad, desempeño, confiabilidad e interoperabilidad, así como también reducir el tiempo de desarrollo, documentación y mantenimiento [3].

Para soportar la evolución de este nuevo paradigma se han creado diferentes tecnologías de apoyo, la mayoría enfocadas a la construcción del software basado en componentes como las infraestructuras basadas en el modelo COM (Component Object Model), la arquitectura CORBA (Common Object Request Broker Architecture), la arquitectura J2EE (Java 2 Enterprise Edition) y la arquitectura .Net. Sin embargo, este nuevo paradigma no sólo involucra la creación e incorporación de los componentes, sino también la identificación de los componentes candidatos, la selección de los componentes adecuados y su posterior integración a los componentes ya existentes [14] todo enmarcado dentro de un proceso formal de desarrollo de software.

Buscando mejorar la actividad de identificación de componentes candidatos, este proyecto presenta un middleware de indexación y localización de componentes de software en ambientes altamente distribuidos llamado PASTEUR, que facilita la

[1] *Este proyecto se inscribe en el marco del proyecto Ecos-Colciencias "Desarrollo de una infraestructura de GRID para cálculo y datos - código C06M02*

identificación y ubicación de los componentes adecuados para una organización que describe sus necesidades a través de metadata. El objetivo es viabilizar el comercio de componentes empleando mecanismos de búsqueda que exploten las posibilidades que brindan los ambientes abiertos de internet convirtiéndolo en un proceso eficiente y con bajo costo.

## 2 Marco Teórico

Los sistemas peer to peer (p2p) son sistemas distribuidos con características particulares de "auto-organización, comunicación simétrica y control distribuido" [2], diseñados para soportar un alto volumen de nodos autónomos, dinámicos y altamente distribuidos. En los últimos años, han tenido un gran auge en ambientes abiertos como el de internet, especialmente para implementar sistemas de intercambio y búsqueda de archivos como Gnutella[18], eDonkey[19] y Napster[20]. Así como también, en aplicaciones de comunicación y colaboración, computación distribuida, servicios de soporte a internet y administración de sistemas de bases de datos [4].

Los sistemas p2p pueden clasificarse de acuerdo al nivel de centralización o de acuerdo a la estructura de la red [4]. En el primer caso, pueden existir arquitecturas puramente descentralizadas donde todos los nodos actúan como clientes o como servidores de forma dinámica; arquitecturas parcialmente centralizadas en donde algunos nodos asumen roles más importantes y son asignados dinámicamente y arquitecturas descentralizadas híbridas en donde hay servidores centrales sobre los cuales se realizan las operaciones de localización, pero posteriormente la comunicación es realizada directamente entre los nodos. La segunda clasificación de acuerdo a la estructura de la red, define dos tipos de sistemas, los p2p no estructurados en donde la ubicación de recursos es independiente de la topología y los p2p estructurados en donde la topología es controlada y los recursos se ubican en nodos específicos, generalmente a partir de una función de hash que posteriormente facilita su localización. La primera estructura no garantiza respuestas comprehensivas[2], pues por lo general su esquema de localización se hace a partir de inundación en donde se definen tiempos máximos de espera que no necesariamente corresponden al tiempo necesario para ubicar todos los recursos asociados. La segunda, garantiza comprehensividad gracias a su manera de indexar usando funciones hash3 que permite conocer con exactitud el nodo donde fue ubicado un recurso al aplicar la misma función de hash durante la localización.

El middleware de indexación y localización de Componentes PASTEUR está construido sobre un sistema p2p estructurado, puramente descentralizado, basado en funciones de hash, más conocidos como sistemas p2p DHT (por sus siglas en inglés de Distributed Hash Table) cuyas implementaciones más conocidas son Pastry [6], Chord [7], CAN [5] y Tapestry [8]. Para garantizar la persistencia, la búsqueda y la disponibilidad de los componentes de software, la implementación de PASTEUR se hizo empleando los servicios de búsqueda de Pastry y los servicios de almacenamiento distribuido de Past [6] un sistema construido sobre Pastry para proveer servicios de inserción, borrado y búsqueda de archivos en redes a gran escala.

## 3 Descripción de PASTEUR

### 3.1 Caracterización de un Componente de Software en PASTEUR

Para establecer un mecanismo de indexación, es necesario definir la manera de caracterizar un componente de software, que habilite su posterior localización de acuerdo a las necesidades de una organización. Esta caracterización se basó en la taxonomía de componentes propuesta por González & Torres [3] y la clasificación basada en facetas propuesta por Prieto & Díaz [21], el resultado es la taxonomía plasmada en la Figura 1 Taxonomía de Componentes.

Todo componente de software que se incorpore a PASTEUR debe describirse usando esta taxonomía para facilitar su localización. Sin embargo, para garantizar la flexibilidad y evolución de PASTEUR, es posible definir o eliminar atributos sobre los cuales se realizará la indexación.

### 3.2 Funcionalidad de PASTEUR

Las organizaciones interesadas en ofrecer o solicitar componentes de software en un ambiente abierto como

[2] Una respuesta comprehensiva es aquella que asegura la localización de todos los recursos asociados a una consulta en el momento de su ejecución

[3] Una función de hash asigna un identificador

Internet empleando servicios eficientes y confiables de publicación, búsqueda y entrega serán los usuarios de PASTEUR que ofrece las funcionalidades descritas a continuación.

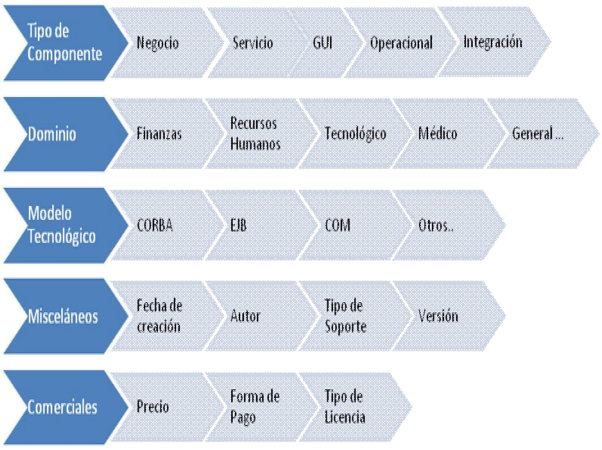


**Figura 1 Taxonomía de Componentes**

**3.2.1 Inserción de componentes.** Permite la inserción de un componente de software con su taxonomía asociada.

**3.2.2 Consultar componentes.** Permite solicitar los componentes coincidentes a partir de una consulta determinada. PASTEUR permite expresar las consultas usando los operadores igual (=), mayor (>), menor (<), mayor o igual (>=) y menor o igual (<=).

**3.2.3 Contar componentes.** Permite realizar el conteo del número de componentes agrupados por uno de sus atributos y que opcionalmente cumplen con una condición especificada; dicha condición se expresa con los operadores presentados en el punto anterior.

**3.2.4 Máximo/mínimo valor de atributo.** Permite consultar el valor máximo o mínimo que toma un atributo dentro de los componentes incluidos en el sistema. Al igual que con la operación de conteo, se permite especificar una condición de filtrado.

**3.2.5 Requerimientos no funcionales.** Así mismo, PASTEUR provee los requerimientos no funcionales: Autonomía de los nodos y tolerancia a fallas, escalabilidad, Balanceo de Carga, Flexibilidad para aumentar o reducir el número de atributos que hacen parte de la descripción de un componente de software.

# 4 Solución Propuesta

El cumplimiento de las funcionalidades y los requerimientos no funcionales descritos en la sección anterior fueron materializados por PASTEUR a través de los siguientes procesos:

## 4.1 Proceso de Indexación

La Figura 2 Proceso de Indexaciónrepresenta el modelo de indexación de PASTEUR. En este caso, una organización desea dejar disponible un componente de software C1. Para ello entrega el componente de software y su taxonomía asociada, como lo indica la funcionalidad InsertarComponente. PASTEUR asigna un identificador al componente (101) y lo inserta en el nodo al cual le corresponde almacenar este identificador (de acuerdo a los servicios proveídos por el DHT estas inserciones se realizan de forma balanceada sobre todos los nodos de la red).

Posteriormente, en los pasos 3 y 4 inserta el identificador del componente (101) asociado a las tuplas [atributo=valor] que fueron definidos en la taxonomía. La asignación de los nodos que almacenan el índice de componentes asociados a una tupla se hace de forma dinámica a partir de la función de hash sobre la cadena atributo=valor.

Finalmente, en los pasos 5 y 6, por cada uno de los atributos de la taxonomía, se mantiene un índice de todos los posibles valores que puede llegar a tomar. Al igual que el índice de objetos, este índice de valores es asignado de forma dinámica al nodo al cual le corresponda la función de hash de la cadena atributo.

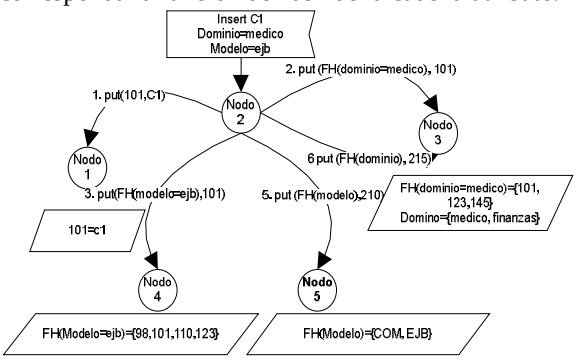


**Figura 2 Proceso de Indexación**

## 4.2 Proceso de Localización

Para facilitar y hacer más flexible la localización de los componentes adecuados para una organización, PASTEUR provee consultas que pueden involucrar múltiples términos y diferentes operaciones como igualdad y rangos, y los operadores lógicos count, min y max. Así mismo, los resultados pueden ser

agrupados por un atributo específico de la taxonomía. A continuación se hace la descripción de ellas.

Consulta con términos de igualdad: El usuario ingresa la consulta con la sintaxis de la funcionalidad ConsultarComponente. Como se observa en Figura 3 Consulta Términos de Igualdaden el paso 1, PASTEUR a partir de la función de hash de cada uno de los términos, consulta de forma paralela los nodos responsables de dichos términos. En el paso 2 cada nodo retorna los identificadores de los componentes que corresponden a este término. Con esta información el nodo coordinador de la consulta (Nodo 2) realiza la intersección y solicita los componentes coincidentes como se observa en el paso 3.

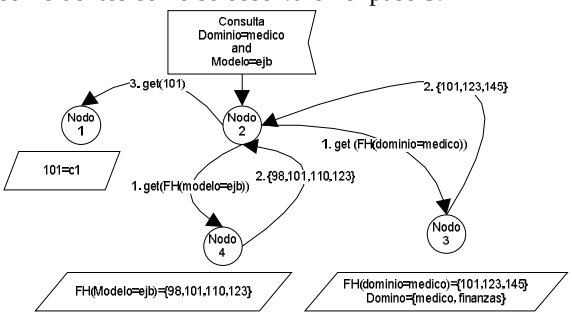


**Figura 3 Consulta Términos de Igualdad**

Consultas con términos de Rangos: Como se observa en la Figura 4, un usuario desea consultar todos los componentes que tienen un atributo entre un rango de valores y que cumplen con una condición de igualdad siguiendo la sintaxis descrita en la funcionalidad ConsultarComponente. Para hacerlo, se vale del índice del atributo presente en el término de desigualdad de la consulta. En el paso 1 y 2, a partir de la función de hash del atributo, obtiene todos los valores del dominio de este atributo. Teniendo estos valores traduce la desigualdad a las igualdades (nótese que únicamente crea los términos de igualdad estrictamente necesarios) y procede a ejecutar la consulta como una de múltiples términos de igualdad como se observa en los pasos 3 y 4. Al recibir las respuestas, el nodo coordinador ejecuta una función de unión, para los términos de igualdad originados de términos de desigualdad, y los intersecta con los términos de igualdad originales. Finalmente, en el paso 5 obtiene todos los id's coincidentes con las condiciones de la consulta.

Teniendo en cuenta que una consulta de rango puede dar origen a una gran cantidad de consultas de igualdad, PASTEUR contiene una versión optimizada de consultas de rangos, basada en un umbral que define el número máximo de consultas de igualdad que un nodo puede llegar a ejecutar. Este nuevo algoritmo llamado el algoritmo del Buen Vecino evalúa si la traducción a términos de igualdad excede un umbral máximo, si es así, la consulta es entregada

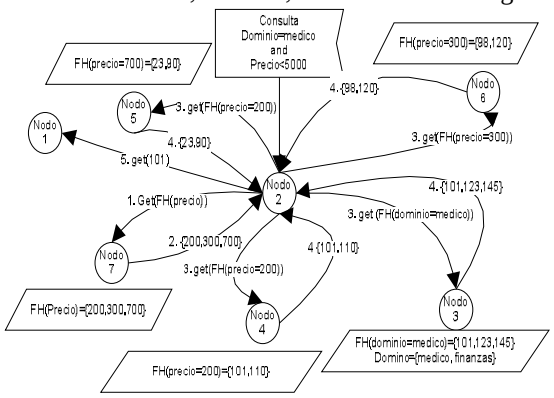


**Figura 4 Consultas de Rangos**

a un grupo de vecinos del nodo coordinador para que sea quienes ejecuten un subRango de la consulta y retornen al nodo coordinador los componentes coincidentes con este subRango. El funcionamiento de esta optimización puede observarse en la Figura 4 Consultas de Rangos

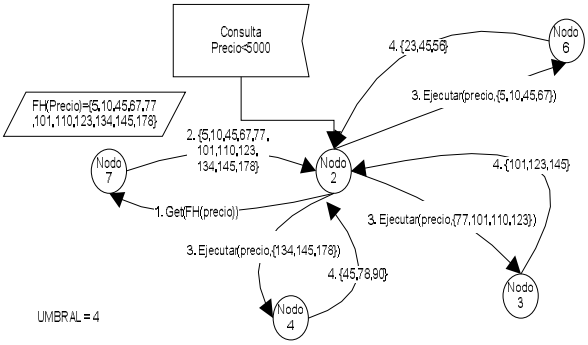


**Figure 5 Consulta de Rango Usando la Estrategia del Buen Vecino**

Consulta de operador Count: Como se ilustra en la Figure 6 Consulta Operador Counten el paso 1 y 2, el nodo coordinador realiza la consulta de igualdad (o desigualdad) normalmente. Posteriormente en el paso 3 y 4 procede a consultar el dominio del término agrupador, a partir de la respuesta obtenida genera un conjunto de consultas de igualdad que ejecuta en los pasos 5 y 6 que le permiten obtener todos los componentes correspondientes. Finalmente, realiza la intersección entre los componentes obtenidos en el paso 2 y los obtenidos en el paso 6, realiza el conteo y

entrega la respuesta al usuario. Los operadores Min y Max, se ejecutan de forma similar al Count.

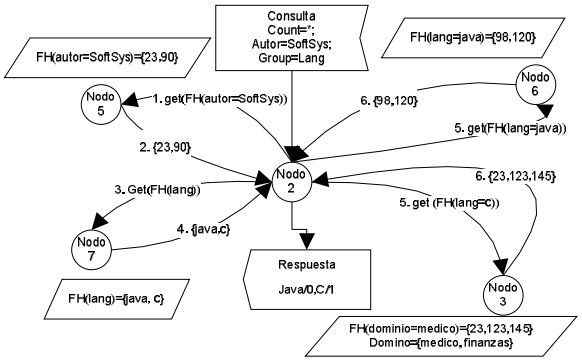


**Figure 6 Consulta Operador Count**

## 5 Evaluación de PASTEUR

Para evaluar el diseño de los procesos de PASTEUR, se construyó un primer prototipo en Java 1.5 usando los servicios de búsqueda y de almacenamiento distribuido de FreePastry [13]. PASTEUR implementa la funcionalidad de la capa de servicios distribuidos de datos. Para garantizar la propiedad de portabilidad se aisló la lógica ofrecida por PASTEUR de la ofrecida por FreePastry. Sin embargo, para efectos de optimización fue necesario modificar el envío de mensajes de la capa DLS (Distributed Lookup Services) como se pudo observar en la optimización de las consultas de igualdad y la ejecución del operador Count.

El objetivo de las pruebas es presentar el comportamiento del sistema frente al aumento del número de objetos que maneja. Se realizaron pruebas con diferentes tipos de consultas manteniendo constante el número de nodos participantes. El número de nodos usado para las pruebas fue 100.

Tomando como base la taxonomía establecida para categorizar los objetos de PASTEUR, se procedió a generar los objetos a insertar de manera que los valores de meta-datos tomaran una distribución uniforme.

Para la primera prueba se generaron 60 objetos junto con su meta-data, mientras que para la segunda se aumentó en un orden de magnitud el número de objetos a insertar, pasando a 600. Esta meta-data cumplía con los parámetros presentados en la tabla anterior.

En vista de que el sistema soporta diferentes estilos de consultas, para cada una de las pruebas se ejecutaron las siguientes consultas:

1. Tipo de componente igual a operacional (Selectividad de 20%)
2. Soporte igual a Si (Selectividad de 50%)
3. Versión mayor a 4 (Selectividad de 20%)
4. Versión menor o igual a 5 (Selectividad 100%)
5. Versión mayor a 4 y tipo de componente igual a operacional (Selectividad de 4%)

A continuación se presentan los resultados obtenidos tras la ejecución de las pruebas en la Tabla 1 Pruebas

**Tabla 1 Pruebas**

| ms | Prueba 1 | Prueba 2 | Prueba 3 | Prueba 4 | Prueba 5 |
|---|---|---|---|---|---|
| N[4] | tipo= operacional | soporte=true | version >4 | version<=5 | version>4;tipo=operacional |
| 60 | 167.46 | 159.24 | 126.98 | 137.38 | 166.6 |
| 175 | 163.06 | 271.98 | 113.16 | 162.82 | 151.22 |
| 330 | 187.68 | 392.56 | 152.82 | 271.18 | 186.76 |
| 465 | 182.82 | 438.84 | 164.86 | 261.54 | 202.7 |
| 600 | 232.8 | 440.82 | 180.28 | 233.92 | 212.56 |

Todas las gráficas que se presentan a continuación presentan en el eje X el número de objetos y en el Y el tiempo de respuesta de la consulta dado en milisegundos.

En general el aumento en el tiempo de respuesta con respecto al número de objetos que maneja el sistema se acerca a un comportamiento lineal. Sin embargo, en las pruebas 2, 3 y 5, se evidencia un comportamiento similar que tiende a disminuir el aumento en el tiempo de respuesta conforme se aumenta el número de objetos.

En la figura 12 se aprecia que el comportamiento de

[4] Número de nodos

las consultas de las pruebas 1, 3 y 5 son similares. Esto se debe a que el nivel de selectividad de las consultas está relacionado. Podemos observar que la consulta de la prueba 5 está compuesta por las consultas de las pruebas 1 y 3 lo cual trae como consecuencia que el comportamiento de la prueba 5 se pueda derivar del comportamiento de las pruebas que le dan origen.

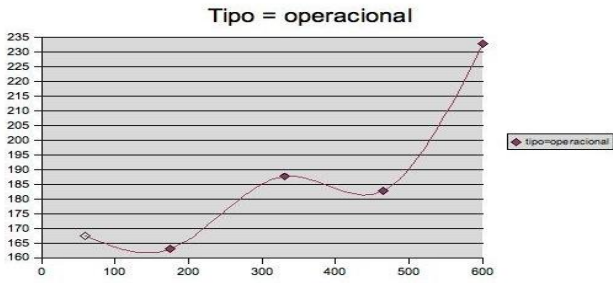


**Figura 7 Prueba 1**

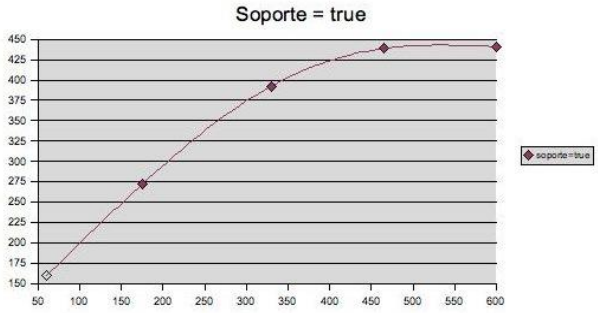


**Figura 8 Prueba 2**

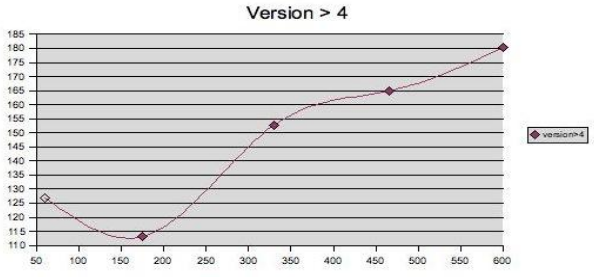


**Figura 9 Prueba 3**

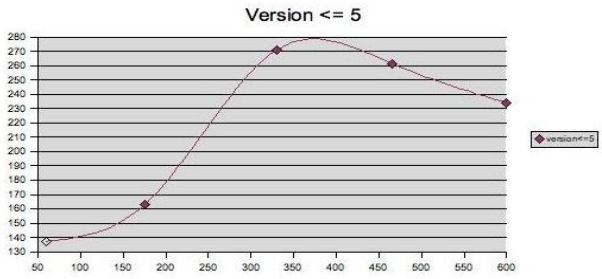


**Figura 10 Prueba 4**

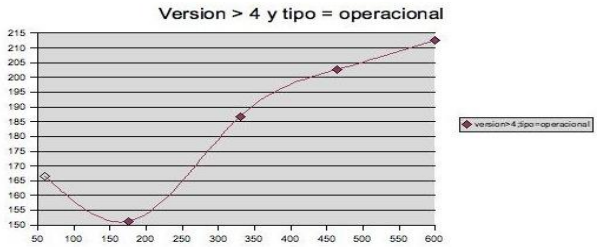


**Figura 11 Prueba 5**

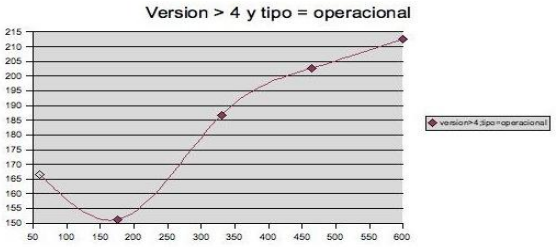


**Figura 12 Comparativo**

En el caso de las pruebas 1 y 4, el último punto de ambas series presenta un comportamiento irregular en comparación a los valores anteriores de la serie. Este comportamiento puede explicarse por factores estado de los nodos y evidencia la necesidad de ejecutar pruebas adicionales para validar los resultados obtenidos.

Pese a los problemas que se mencionan, la diferencia en los tiempos promedios se mantiene en rangos razonables de acuerdo al número de objetos que maneja el sistema que permiten que mantenga características de escalabilidad aceptables.

## 6 Trabajos Relacionados

PASTEUR es un middleware para la indexación y localización de componentes de software que fue implementado como la capa DDS (Distributed Data

Service) de un sistema DHT. Por su doble naturaleza, para su diseño fue necesario evaluar proyectos de indexación y localización de componentes de software y sistemas que proveen las funcionalidades de la capa DDS de un sistema DHT.

Con respecto a los sistemas de indexación y localización de componentes, se han construido diferentes sistemas para proveer estos servicios. Algunos, como el propuesto por Henninger [11] en donde los repositorios de componentes evolucionan a partir del uso en términos de consultas que se han realizado, o la propuesta de Isakowitz & Kauffman [12] que explora la manera de realizar consultas más flexibles guiadas por mecanismos de navegación sobre el repositorio, que permitan obtener mejores resultados, presentan un enfoque centralizado que no es viable cuando se quiere incursionar de forma masiva en mercados abiertos como Internet, por todos los problemas que puede traer un repositorio centralizado como el riesgo de un solo punto de falla, la accesibilidad limitada, la poca escalabilidad, el riesgo de ataques maliciosos al repositorio central, etc.

Por otro lado, otros trabajos como Agora [10] y MoreCOTS [14] están dirigidos a ambientes distribuidos, pero desde el punto de vista de la obtención automática de componentes empleando motores de búsqueda web para posteriormente ser indexados de forma central por el atributo tipo de componente. Estas propuestas presentan una buena alternativa para obtención automática de componentes, sin embargo además de tener los problemas de ser un repositorio centralizado, sólo permite realizar consultas restringidas y en el caso de Agora restringidas a un solo atributo de los componentes.

Con respecto a los sistemas que proveen la funcionalidad DDS, se hizo el análisis específicamente de los servicios de consultas. En el sistema PIER [16] por ejemplo, se ofrece un motor de consultas masivamente distribuido que combina la escalabilidad de internet con aproximaciones de bases de datos. Por estar enfocado a modelos de datos relacionales, no era una opción viable para soportar las funcionalidades de indexación y localización de componentes de software. Adicionalmente, no garantiza la durabilidad de los datos. Otro de los sistemas evaluados fue PINS [15], que es un middleware que proporciona servicios para compartir datos a través de la administración de metadatos y de las consultas de localización declarativas. Esta propuesta proporcionó varios de los principios que fueron empleados en el diseño de PASTEUR, agregando nuevos servicios para ejecución de operadores lógicos adicionales como Count, Min y Max y la optimización de consultas de desigualdad. Por último, fue analizado el sistema PHT (Prefix Hash Trees) [17] para realizar consultas de desigualdad a partir de árboles binarios que facilitan las operaciones de desigualdad y de Min y Max. Sin embargo, al estar enfocado únicamente en atributos numéricos no fue incorporado a la solución.

## 7 Trabajo Futuro

En la implementación actual de PASTEUR, las consultas que involucran más de un término están siendo ejecutadas de forma paralela desde un nodo coordinador (o en la versión optimizada desde los vecinos del nodo coordinador) que consulta el nodo responsable de cada uno de los términos, sin tener en cuenta su grado de selectividad[5]. Esto puede generar que consultas que contengan términos altamente selectivos sean realizadas de forma ineficiente al traer todos los objetos independientemente de que cumplan otras condiciones de la consulta. Para solucionar esta problemática se propone la recopilación y uso de estadísticas asociadas al dominio de la metadata. Estas estadísticas mantendrán el registro del número de objetos que han sido insertados al sistema con un valor determinado para un atributo de metadata.

Por otra parte, teniendo en cuenta que a nivel global el establecimiento de un lenguaje controlado es poco viable, es necesario incorporar mecanismos que permitan la ejecución de consultas más relajadas que reconozcan la existencia de resultados adicionales a los obtenidos a partir de la coincidencia exacta de términos. Esto permitirá optimizar el recall[6] del proceso de localización. Para la implementación de esta mejora se incorporarán nuevos operadores como el Contains y Like para atributos de cadena y se incorporarán ontologías como la tecnología para definir la semántica de los datos y permita construir consultas derivadas a partir de una consulta inicial.

## 8 Conclusiones

Los sistemas que apoyan la identificación y selección de los componentes adecuados para ser integrados a un proyecto de software se han enfocado en la extracción automática de componentes para

[5] Un atributo es más selectivo que otro si es aplicable a un menor número de objetos asociados

[6] Recall es la propiedad que mide que pocos elementos relevantes sean descartados

dejarlos disponibles en un repositorio central. Este enfoque tiene debilidades, principalmente en lo que respecta a la escalabilidad y eficiencia en la ejecución de consultas, que dieron lugar a la creación de PASTEUR, un sistema de indexación y localización de componentes de software a gran escala que promueve el comercio de componentes de software en ambientes abiertos, dinámicos y heterogéneos. PASTEUR, fue construido apoyado en la tecnología de los sistemas P2P DHT pues proporciona los medios para realizar indexación y búsquedas sin requerir una entidad central que administre y controle el comercio de componentes.

La propuesta de PASTEUR como capa DDS de un sistema P2P DHT, demuestra una buena alternativa al permitir la ejecución de consultas de desigualdad optimizadas a través de la técnica del Buen Vecino y la incorporación de operadores lógicos Count, Min y Max sobre atributos numéricos y cadenas de caracteres que dan flexibilidad en la ejecución de consultas.

## 9 Referencias


[1] E.S. Almeida, A. Alvaro, V. C. Garcia, J.C.C.P. Mascena, J. C. C. P., V. A. A. Burégio; L.M. Nascimento, D. Lucrédio and S. R. L. Meira, "C.R.U.I.S.E: Component Reuse in Software Engineering", C.E.S.A.R e-book, 2007.

[2] J. RISSON and T. MOORS, "Survey of Research Towards rbust peer to peer networks: search methods, Technical Report", UNSW-EE-p2p-1-1, University of New South Wales, Sydney, Australia, 2004.

[3] R. González and M. Torres, "Critical Issues in Component-Based Development" in Proceedings of The 3rd International Conference on Computing, Communications and Control Technologies (CCCT '05), July 24-27, 2005 - Austin, Texas, USA. 2005.

[4] S. Androutsellis-Theotokis and D. Spinellis, "A Survey of Peer-to-Peer Content Distribution Technologies", ACM Computing Surveys, Vol. 36(4), 2004.

[5] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network. In Proc.", ACM SIGCOMM, 2001.

[6] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Proceedings of Middleware", 2001.

[7] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications. In Proc.", ACM SIGCOMM, 2001.

[8] B.Y. Zhao, J. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report", UCB/CSD-01-1141, University of California, Berkeley, 2001.

[9] A. Rowstron and P. Druschel, "Storage Management and Caching in PAST, a LargeScale, Persistent Peer-to-Peer Storage Utility." In Proc. 18th Symp. Operating System Principles, Banff, Canada, 2001.

[10] R. Seacord and S. Hissam and K. Wallnau, "AGORA: A Search Engine for Software Components", Software Engineering Institute, Carnagie Mellon University, USA, 1998.

[11] S. Henninger, "An Evolutionary Approach to Constructing Effective Software Reuse Repositories", ACM Transactions on Software, Engineering and Methodology, Vol. 06, No. 02, 1997, pp. 111-140. 1997.

[12] T. Isakowitz, R. J. Kauffman, "Supporting Search for Reusable Software Objects. IEEE Transactions on Software Engineering", Vol. 22, No. 06, 1996.

[13] Rice University, Houston, USA & Max Plank Institute for Software Systems, Saarbrücken, Germany. FreePastry disponible en línea en http://freepastry.rice.edu/FreePastry/, 2007.

[14] N. Yanes, et al. " MoReCOTS: a Specialized Search Engine for COTS Components on the Web". International Conference on COTS-Based Software Systems, IEEE Society, 2006.

[15] M. Villamil, C. Roncancio and C. Labbé, "PinS: Peer-to-Peer Interrogation and Indexing System", Proc.of the 8th International Database Engineering and Applications Symposium (IDEAS 04) Coimbra, 2004 .

[16] R. Huebsch, B. N. Chun, J. M. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. R. Yumerefendi (2005). "The architecture of PIER: an Internet-scale query processor". In CIDR, pages 28--43, 2005.

[17] S. Ramabhadran, J. M. Hellerstein, S. Ratnasamy, and S. Shenker. "Prefix hash tree: An indexing data structure over distributed hash tables", http://citeseer.ist.psu.edu/ramabhadran04prefix.html, 2004.

[18] Project web page: http://www.gnutella.com/

[19] Project web page: http://www.edonkey2000.com/

[20] Project web page: http://free.napster.com

[21] Prieto-Díaz, R. "Implementing Faceted Classification fro Software Reuse". Communications of the AC<. Vol 34 (5), 1991