Journal of Web Engineering, Vol. 6, No.1 (2007) 002-018

SELECTING SERVICES FOR WEB APPLICATIONS: THE OPEN HYPERMEDIA CASE

NIKOS KAROUSOS, MANOLIS TZAGARAKIS, CHRIS DIOLIS, ATHANASIOS TSAKALIDIS

Research Academic Computer Technology Institute, Rion Greece

University of Patras, Rion Greece

{karousos, tzagara, diolis, tsak}@optionsnet.gr

Received April 23, 2005
Revised October 9, 2006

As long as the volume of the distributed information in the Internet increases, the need for useful and easy-to-use 3rd party services in Web Applications will be growing. Web developers adopt tactics for integrating external services into their applications, aiming to enrich both utility and efficiency with low cost. A variety of services derived from the area of Open Hypermedia Systems (OHS) can augment web functionality with valuable hypermedia features. Towards that, this paper proposes a framework for enabling the provision of hypermedia services to web developers in a service-oriented manner. It investigates and analyzes the requirements of developers for easily inserting hypermedia functionality into Web applications, thus facilitating rapid prototyping of web applications. A Service Discovery Mechanism for finding and using hypermedia services is defined, and solutions for increasing the usage of hypermedia systems by web developers are proposed.

Key words: Open Hypermedia Systems, hypermedia services, web applications, hypermedia service discovery mechanism, web development.
Communicated by: S Christodoulou

1 Introduction

The World Wide Web is a huge collection of information. Thus, the need for tools and methods that efficiently manage, collect and organize this huge amount of information is constantly growing. Issues like organizing, searching, comparing, commenting and projecting the information, have gained great importance in the past years. As long as the usage of the Web is rising, the need for effective information management systems proportionally increases. Web developers, who create applications which require information organization, authoring, versioning, annotation, sophisticated backtracking and information restructuring, to name few, can be helped by the usage of hypermedia services into their applications. The adoption of OHSs in the area of the World Wide Web will enrich business, science, web engineering, and personal applications with hypermedia functionality [11].

Open Hypermedia Systems consist of middleware components with structural awareness that provide hypermedia functionality to all applications on the client side. In particular, several components make available services relevant to issues such as structure, navigation, taxonomic and spatial organization, searching, annotation etc. Semantically typed anchors, links and nodes, bidirectional links, versioning of hypermedia constructs, landmarks, automatic link propagation, trails and guided tours are only some characteristics of the structural services. Navigation features include process enactment through link traversal, backtracking strategies, backjumping and history mechanisms. Via taxonomic organization, the information is stored in hierarchical and tree structures, and can be represented through different perspectives. Spatial structure provides privileges of

organizing information on space and automatic link creation, based on each object's position. Moreover, OHSs support advanced search mechanisms, based on both the hypermedia structure and the hypermedia content. Finally, annotation features include comments, bookmarks and reader-authored links with private workgroup and public access [30].

The aforementioned functionality is provided by a number of OHSs and can make the information and knowledge management procedure effective. Hicks in [18] approaches the contribution of OHSs based on the Engelbart's definition about 'B-class' and 'C-class' tools [14]. In particular, he argues that these systems can facilitate and provide services to other developers who build applications or tools.

An application of the above could be the following example: A web application is handling a company's data by using dynamic server pages and a relational database schema. A change of requirements demands the application to re-organize the customers' list in an alternative way than the classic relational one. In particular, the project must be able to depict customers over the country's map and relate each other according to their geographical position. In that case, the web application could use a spatial hypermedia service. Additionally, the application could include a hierarchical presentation of the customers, based on several grouping criteria by using a taxonomic hypermedia server. These kinds of inclusions allow developers to augment the web applications with extra features without the alteration of the existed web application backbone. In order to make that possible, OHSs should adopt common mechanisms to provide hypermedia services to the web.

Although web engineers have already paid great attention to the web application development procedure [12] and Service-Oriented-Architecture (SOA), the hypermedia community has not approached yet, neither the standardization of a methodology or the development of tools that can drive web developers to a quick and simple way to embody hypermedia functionality into web applications. Furthermore, Open Hypermedia Systems has not reached a satisfactory level of publicity, and the hypermedia service discovery issue still remains unaddressed. Consequently, the OHS's service provision process is still in an immature phase compared to the current status of the World Wide Web service provision.

This paper argues over the need for rethinking the design of OHSs and the provision of hypermedia functionality in a service-oriented and more easily usable fashion. It proposes a mechanism - in the context of SOA - for enabling web developers to use hypermedia services into their applications. It studies the design of both a hypermedia service discovery mechanism and a set of tools, towards a complete web developer support framework. Thus, the needs and requirements of the web developers concerning searching, selecting and using hypermedia services, are properly addressed. Finally, a prototype peer-to-peer hypermedia service discovery application built on top of Callimachus CB-OHS [27] is presented.

The paper is organized as follows. We initially focus on the service oriented development process, in contrast to traditional client server architecture. Then, we try to analyze what the needs of web developers are, regarding the use of services, and we take a closer look into the relationship between hypermedia systems and web development. In chapter 3, we propose tools for discovering and using hypermedia services, towards the enhancement of web application with hypermedia functionality. Chapter 4 presents a service discovery prototype that has been implemented in order to discover Callimachus based components. Finally, in chapter 5 we discuss the future work and we conclude.

2 Using Services: Analysis, Requirements and Observations

2.1 From Client-Server to Service Oriented Architectures

Traditionally, hypermedia systems have been built according to client server (or point-to-point) architectures that provided an adequate framework for bringing hypertext functionality to applications. However, the design and development of these hypermedia systems were based on assumptions that reflect the architecture upon which they were developed. Moving hypermedia systems to service oriented architecture, requires these assumptions to be re-examined and adjusted. This is needed because service oriented differs greatly from client server architectures. Table 1 summarizes the main differences between service oriented and client server architectures.

In service oriented architectures, binding to services (i.e. references to operations provided by services) are established dynamically and during runtime, which is completely incompatible with client server based hypermedia systems where such binding of clients to services happens very early in the development process (in particular during design or compile time). At run time, changing bindings is impossible.

Table 1. Main differences between client server and service oriented architectures when considering them as development environments.

Client Server Architectures	**Service Oriented Architectures**
Early binding (compile/development time)	Late binding (run time)
Domestic (evolve smoothly and planned)	Feral (evolve abrupt and uncontrolled)
Location dependent	Location independent/transparent
Single interface (protocol)	Set of interfaces (protocols)
Development oriented	Integration oriented
Tightly coupled	Loosely coupled
Monolithic	Composable
Stable	Unstable due to ad hoc nature

While client server architecture evolves in a controlled and disciplined fashion, service oriented evolves in a rather feral way [34]. This is mainly due to the autonomous nature of services that implies an autonomous evolution path as well. As a result, client-side bindings to hypermedia services can easily be invalidated. In addition, it is evident that, while client server architectures exhibit location dependence, thus forbidding changes in location information (e.g., in terms of host and port), service oriented architectures are location independent making conventional clients unable to operate in such an environment. With respect to the supported interfaces, in client/server systems only a small, bound number of interfaces are supported, whereas in service oriented systems an unbound number of interfaces exist. Thus, while in client server systems all software entities (e.g., client application) can be reactive when considering interfaces to hypermedia services, in service oriented architectures all software entities need to be proactive. Furthermore, client server systems are tightly coupled systems, meaning that design changes in the service design are followed by design changes on the client side. This is not the case in service oriented characterizing this architecture as loosely coupled. Finally,

while in client server systems the main task during development is to extend the client and the server respectively, in service oriented architectures the main task of a developer is to integrate services.

From the above discussion, it is evident that service oriented architectures represent an environment where all software entities need to exhibit flexibility, autonomy, and adaptability in order to function correctly and take advantage of the plethora of services that are present. Within such an agile environment, web developers require new tools and infrastructures to achieve rapid prototyping of systems as well as transition from prototype to production. Models aiming to supporting such tasks are already underway [25] and in the following sections we discuss issues that help web developers to deal with both the service discovery and usage aspects in service oriented hypermedia environments.

Challenges developing applications in SOA Environments

Within such active environment, current developing tools and foundations cannot cooperate easily. This is due to the fact, that their conceptual foundations are designed to address a passive computing environment. In hypermedia, contemporary development frameworks create applications that:

1. Cannot adapt to new server protocols (interfaces), since they presume only one particular and universally constant protocol.
2. Cannot adapt to location changes of structure servers, e.g., even when host or port information changes
3. Require redesign, recompilation and, in general, major effort when things change in their environment.

As a result, contemporary tools do not support the evolution of applications and thus do not allow their rapid development in SOA. The issue that concerns SOA is to permit hypermedia developers to rapidly create and deploy easily structure services into existing frameworks.

Service oriented architecture relies on the ability to identify services and their capabilities. In such architectures the ability to efficiently discovery services is crucial [1]. Therefore, a SOA depends on directories that describe the available services in their domain. Thus, service discovery capabilities, along with reconciling the design assumptions underlying hypermedia applications has been proven an important catalyst for addressing development issues in SOA.

This paper builds on this approach and presents how service discovery in Callimachus works and is deployed to shorten the development efforts of hypermedia applications. In our approach, service discovery is considered from a developers perspective, which differs greatly from existing approaches where service discovery is presented and analysed from an end-users viewpoint.

2.2 Developers' requirements

Web developers demand low cost services, which are discoverable and can be used or integrated into their applications. Thus, they need frameworks which provide easy-to-find, easy-to-understand and easy-to-use 3rd party services into his applications.

In this context, a methodology for easy discovering services is considered. A sound example of such a mechanism is the Universal Description, Discovery and Integration of Web Services – (UDDI) [29]. Through UDDI a developer can search into categories or query for a desired service. In many cases, the need of advanced discovery techniques arises, when meta-search and both functional and not - functional client requirements are served [13].

Assuming that a desired service is located, the user (developer) acquires information about the way the service operates and what is its application programming interface (API). This automatically generates the need for an understandable description of the service by both the human and the computer. Thus, meta-information that describes the service is needed to be carried out.

Another requirement demanded by the developer, is the existence of a driver that makes the usage of the service from the application, as simple as possible. An example of such tool is a demo client that is usually shipped together with the API or the source of a service, as well as the code generator (for example in RMI, CORBA etc.), that automatically creates the client/server communication.

Finally, some widely accepted issues like trust, security and reliability are raised as basic requirements towards the adoption and efficient operation of the application.

2.3 Open Hypermedia Systems and the Web

From the perspective of the hypermedia community, the World Wide Web and the 3rd party web applications have always been targets for service provision. The design of the Component Based Open Hypermedia Systems (CB-OHSs) and service oriented systems has facilitated those efforts [23]. Into this direction, the Multiple Open Services project [35] is a system not divided into structural servers, like CB-OHSs, but into services. The widely adopted concept of Structural Computing [22], and corresponding developed systems, like Themis [3] outline this trend. Nevertheless, despite interoperability improvement between OHSs, the unawareness of Web applications for the provided hypermedia services still remains a crucial aspect.

The universal popularity of the World Wide Web has encouraged the hypermedia researchers to make several integration efforts [2, 9]. Most of the well known hypermedia systems, like Microcosm [17], with the Distributed Link Service (DLS) [10], DHM [16], with DHM/WWW [15], and Chimera [2, 4] have presented a web integration solution. In most of the cases the integrations were ad-hoc solutions, without using any developer framework, or any developer support tool. The lack of a concise methodology makes them error-prone and difficult to maintain.

The Babylon Web Service [19] is another approach for providing hypermedia services into the Web. In this project, the taxonomic services of the Babylon hypermedia system are made available into the Web, through a Web Service [32]. Similar is the attempt of mapping SoFAR system to Web Services with respectful results [5].

2.4 Why hypermedia systems are not used efficiently in the web development?

Although principles, methodologies and models that originate from research in the area of OHSs are adopted in the field of informatics, commercial hypermedia systems and systems that operate with stability and scalability are not plenty enough. The hypermedia community mostly produces prototypes, or just light versions of system that are not predicate for commercial use. There is no doubt that hypermedia services are not as many as possible, and that building such system and services has to be the first step.

The hypermedia services do not have a global usage [24]. One of the main reasons for that, is their low publicity. The majority of web developers are unaware of the existence of the hypermedia systems as well as the required methodology for adding hypermedia services into their web applications. Additionally, the Open Hypermedia Systems are not presented adequately in the web to enlighten those who want to be informed about the developments in the hypermedia area. Unfortunately, the

most common way for someone to learn about OHSs is from conferences, publications and other developers' out-of-band. These facts lead us to the observation that most of the existing OHSs are not ready for a stable commercial usage and that most of these are still operating as prototype systems.

Another reason for the narrow usage of hypermedia systems in web developments is their high complexity. The provision of an autonomous OHS service is not an easy task. It is difficult for the hypermedia designer to produce a service with a simple API, and it is rather inconvenient for the web developer to exploit such a service. Structural operations and multiple perspectives over the same data, increase the complexity of hypermedia systems.

In many cases, a web developer who wants to use a specific hypermedia service is obliged to acquire and have access (and pay) for all the OHS usability [26]. This is due to the high connectivity between all components of the hypermedia system. Consequently, the cost and the complexity of the OHS service provision are multiplied. The goal is the user to be provided with simple, flexible, scalable and well-defined services.

As we mentioned previously, hypermedia extensions and web integrations are ad-hoc implementations, created by non-standardized methodologies. This implies that there is not a simple and standard way to use a hypermedia service. Each time a developer wants to integrate an OHS with the Web, he is forced to invent his own techniques. This issue is an emerging important problem in the area of CB-OHSs, where components are dynamically added or even modified, requiring from the developer to make new integration efforts every time. Finally, the developer support for searching and discovering services within OHSs still remains immature.

3 Towards Methodologies and Tools for Discovering and Using Hypermedia Services

In this section both a discovery mechanism and a set of tools, in order to enable the enhancement of Web applications with hypermedia functionality, are proposed. A necessary condition for the deployment of a discovery mechanism is that OHSs (and generally CB-OHS) are outfitted with a new feature. That is, the hypermedia server should have the ability to describe themselves. In that way, hypermedia systems are equipped with introspection capabilities.

3.1 Service Discovery Mechanism

Architecture. There are two main different architectural approaches for the design of a discovery mechanism: the centralized and the distributed.

In the centralized architecture, when a server starts operating, it registers to a hypermedia service registry server. When a potential user wants to find an appropriate hypermedia service, he performs a query to the service registry and waits until the service registry responds with a list of services, in which he may find the desired service.

In the distributed architecture, multiple server directories exist. In order to discover the information about appropriate servers from a distributed server directory system, each server registry interoperates with others by query passing. Moreover, a new approach of peer-to-peer (P2P) server discovery systems, based on a full P2P architecture, has increased the information distribution.

Which architecture should be chosen? Initially, it is worth mentioning that the service discovery mechanism depends on the characteristics of the registered (hypermedia) servers. Additionally, since no hypermedia service discovery mechanism has been deployed yet, a simple discovery mechanism

can cover the needs. Both the centralized and the distributed architecture can facilitate the requirement of the discovery framework. Thus, the use of a hybrid distributed model that initially operates as a centralized server and can be upgraded to a fully P2P distributed discovery system, seems satisfactory.

Introspection Capabilities of OHSs. The term "Introspection Capability" is defined as the awareness of the hypermedia system of all of its provided services and of the required information for the service invocation. That is, the system is aware of its API and the semantics of its operation. The system is able to answer questions like "Who are you? What services can you provide? And how can I have access to this specific operation?"

In order to equip with introspection capabilities the hypermedia servers, the existence of hypermedia service description specifications is very crucial. The aforementioned specifications can lead to the creation of a Hypermedia Service Description Language (HSDL) or to the extension of emerging standardized mark-up languages, such as the Web Service Description Language (WSDL) [33]. XML is the appropriate format for hypermedia service descriptions since the data representation is not only machine-readable and human understandable, but also machine understandable [7,21]. An interesting approach is the definition of the Hypermedia Resource Descriptor (HRD) [28]. By using the HRD, components of a CB-OHS can be self-described in an XML based language.

Hypermedia service description specs enable the complete description of hypermedia services, and provide adequate information to developers who want to select and use such services. A fundamental structure of these specs is presented below:

- **Service General Information.** It includes information about the service name, the service provider, the service scope and structural specific information such as the hypermedia domain and other.
- **Service Location and Access.** Information about the host and port (or other location information) and the required communication protocol(s) between client and hypermedia server.
- **Service Interface.** Analytical descriptions of the set of the available operations that the service supports. These operations are defined as functions that can be called remotely from a client and can return a result. Furthermore, the set of the necessary function parameters and their corresponding data types is defined.
- **Service Behavior.** This type of information refers to the behavior of the hypermedia server, after each request, and the dependences between the server operations.
- **Service Comments.** When developers read the service description while trying to understand the way the server operates, the service commentary is useful.

Finally, an important specification is the extensibility of the service description so as to be able to augment specifications from servers that come from new hypermedia domains. In addition, the description of the naming methodology that every OHS uses has to be also supported.

Hypermedia Service Registry Record. When a hypermedia service is ready to operate, the system administrator or the OHS on its own:

(a) fills a registry record that describes the service, and

(b) registers the service by passing a request with the registry record to the service registry.

A service registry record consists of a set of fields that are able to describe the service and can be queried from hypermedia or Web users who target to discover an appropriate service. A list of the most important required fields in the registry record is presented in Table 2.

Apart from the specification of the data that will be provided in the registry record, some metadata concerning the service authentication, the license policy etc. are important. These metadata can be

clustered into semantic chapters, e.g. metadata of the service for the technical issues, for the economic issues, for the usage or for semantic issues.

Table 2. List of the most important fields in the registry record.

Registry Fields
Owner Information
Owner Name, Address, URL, etc.
OHS Information
OHS Name, OHS Description
OHS Access Point
Domain Information
Domain Type
Service Information
Service Name, Service Description
Required Communication Protocol
Service Status
Service Location
Service Description Location
Registry Information
Registry Unique Name / ID
Register Date, Duration, Description, etc.

Hypermedia Service Registry Query Language. The communication between both clients and hypermedia registry is based on the registry's API. A critical requirement for this communication is the existence of a common and both human and computer understandable communication platform. Service registries with the same API and protocol may result into a common Hypermedia Service Registry Query Language by which, a Web developer will perform searching actions.

Hypermedia Service Discovery Usability. A hypermedia service discovery tool can help web developers to perform queries and navigate through visual environment. This tool will support service classification and service presentation through tree-views with multiple perspectives.

3.2 Discovering hypermedia services: a step-by-step example

A standard step-by-step procedure that a developer can follow when searching for an appropriate service (depicted in figure 1), is the following:

1. The developer (a) searches for a hypermedia service by filling in required fields and requesting the Hypermedia Registry Server (b), using a common communication Platform or a Visual tool.
2. The Hypermedia Registry Server (b) searches to its local query engine or to other Hypermedia Registry Servers (in case of distributed architecture), and locates appropriate hypermedia services.
3. A list of hypermedia services is returned to the Hypermedia Registry Server. The result is reformatted and then it is presented back to web developers as a response message.
4. The developer selects a service and obtains information about service characteristics.
5. The developer establishes communication with the service (c) and gets information about the way it works.

3.3 Additional Tools, Methodologies and Policies

Tools can provide additional benefits to the proposed mechanism. These tools concern both the service utility and usability aspects from the developer's point of view.

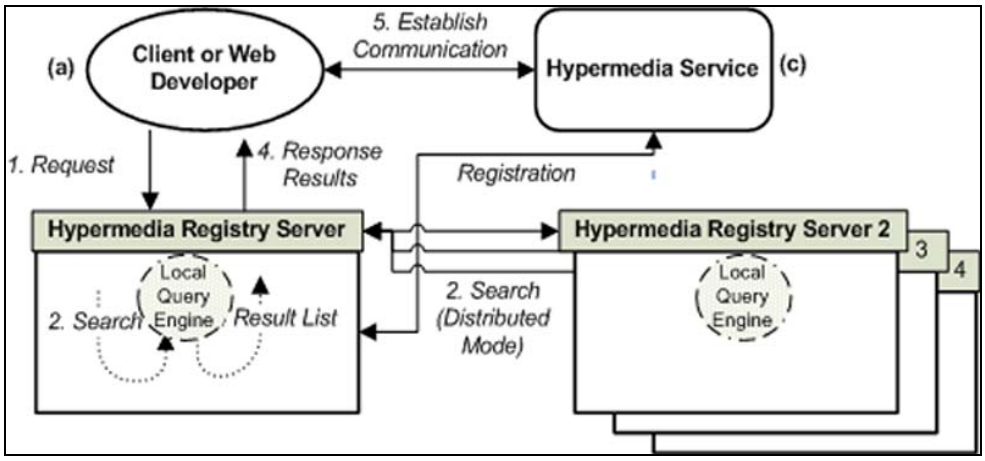

Fig. 1. Discovering hypermedia services.

Hypermedia Service Exploring Tools. Giving the opportunity to have access to OHS, explore for available services of a particular hypermedia system and navigate all hypermedia systems for available services, web developers will have a global view of the hypermedia entity. A Visual Hypermedia Exploring Tool will contribute to the comprehension of the entire hypermedia system.

Automatic Client Skeleton Creation. A significant aspect is the web developer's confidence about the communication protocol that has to be used, in order to communicate with specific hypermedia server and use hypermedia services. Thus, approaches that utilize widely accepted communication infrastructures such as SOAP [31] or the ability of dynamically locating and loading protocols – at run time – are issues to be considered. Such dynamic behaviour can already be witnessed in applications such as browsers and video players that automatically download plug-ins. As a result, when the discovery mechanism returns a list of matched records, for each selected service that uses a well-known communication protocol, the appropriate protocol access information will also be provided. Consequently, tools for the automatic generation of the client communication layer, which will serve a wide range of devices and platforms, are vital. Such tools are based on the ability to understand the provided service descriptions and to dynamically create client side communication libraries.

Service IDE, Examples and Tutorials. Such aids will help the understanding of the service and will dramatically reduce the total effort for the service development.

Fixed financial policy and stable service version for commercial use. By this way the service will be accompanied with guarantee for stable and continual operation in real time environments.

4 Case Study: A Service Discovery Prototype based on the Callimachus System

A service discovery prototype that discovers hypermedia services within the Callimachus CB-OHS, has been developed. The Callimachus service discovery procedure is based on the distributed architecture. Following the proposal for a service description language, the Callimachus discovery service is using Hypermedia Resource Descriptors (HRDs) in order to describe all the components within the system. In the future, the HRD could be upgraded, based on the principles of the presented specifications, aiming to describe in detail existing services from a variety of OHSs.

The reasons for using peer-to-peer architecture in this task are the following:

1. To avoid the possibility of system crash due to a simple operation failure on the only discovery server.
2. To avoid states of bottleneck, while all clients are simultaneously requesting the same server.
3. To avoid time delays in the new information distribution.
4. To support scalability by the addition of new nodes of discovery services.

4.1 An Overview of the Callimachus CB-OHS

The Callimachus CB-OHS attempts to provide the framework in which structure servers can provide abstractions and hypermedia services for existed hypermedia domains. Furthermore the framework supports the creation of structural servers for new domains. So, special attention has been given in the provision of suitable tools to facilitate such task. Within Callimachus these tools are part of a methodology i.e. a systematic, disciplined quantifiable approach in the construction of structure servers. One such tool is the structure template of Callimachus. The aim of structure templates is to maintain the specifications of the structure model of hypermedia domains. Structure servers operate guided by these structure templates to provide domain specific abstractions and constraints.

In the following, we briefly illustrate the various parts of the architecture:

- Structure server. It consists of a structure model (structure template), structure cache and a set of behaviours that are used to deliver domain, sub-domain and application specific abstractions and services to clients.
    1. Structure template. The formal specification of the abstractions that define a hypermedia domain, sub-domain or application.
    2. Structure cache. Provides an intermediate store for domain specific abstractions for structure servers.
    3. Services. Services are available to clients through the use of a specific API and protocol (e.g. openNode, followLink, etc.)
    4. Internal operations. They are used by the structure server internally for consistency reasons mainly (e.g. to affirm conditions and constraints, or to interpret abstractions in a suitable manner).
- Infrastructure. This includes the fundamental hypermedia functionalities that are available to all other entities. Functionalities, such as naming, service registry and persistent store, constitute an essential part of the infrastructure.
- Client. Any process that requests hypermedia functionality. Clients request hypermedia operations from one or more structure servers with the use of the appropriate APIs. Clients may be applications with native support for communication with structure server API, 3rd party applications that use a client API for such purpose, and, in general, applications making use of CSF (Client Side Foo) of any type.

4.2 Architecture

Service Discovery in the Callimachus Environment is taking place as a native procedure within its basic infrastructure. Figure 2 depicts the basic components of Callimachus. The Service Registry (henceforth SR) Component is responsible for service discovery and is located in the backbone of the system. When many instances of Callimachus System exist, the Service Registry acts as a member of a distributed service discovery network. Hence, the service discovery is fully supported in a local or in a global manner of operation.

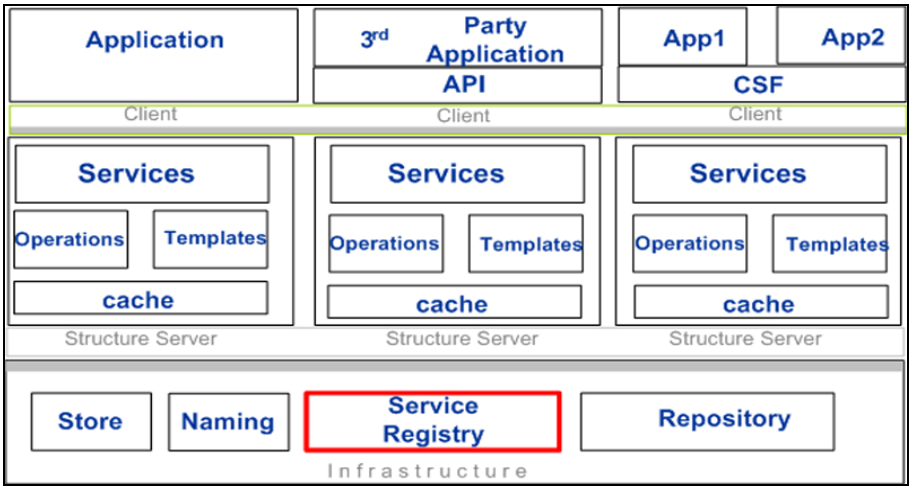

Fig. 2. The Callimachus Service Registry as a fundamental entity in the architecture of Callimachus System

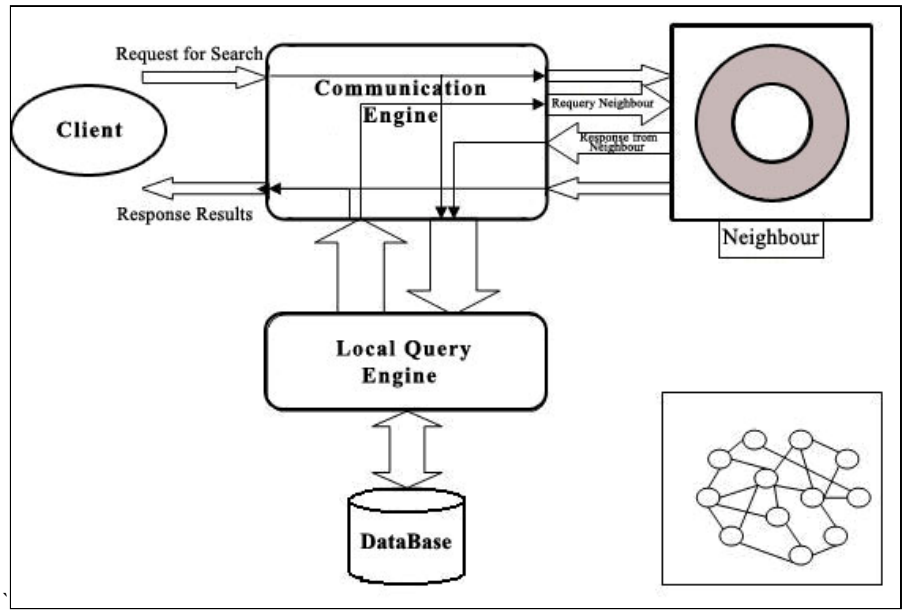

Fig. 3. The Callimachus Service Discovery: Analysis of the Callimachus Registry operation in a peer-to-peer mode.

The Service Registry operates as a distributed discovery service and is applied as a co-operative service [6]. A network of SRs is cooperating in order to serve the requests for discovery that come up from the systems' peers. In figure 3, a non structured peer-to-peer SR network is presented. Each SR is composed of two engines: the communication engine and the local query engine. That combination allows SR to:

1. Communicate and Cooperate: The communication engine provides the client with communication means and represents the SR into the peer-to-peer SR network. That engine is responsible for the following tasks:

- (Com Type1) It accepts queries for services from clients. It resolves them locally by using the local query engine or globally through re-querying the closest peer SRs, and it responds to the client by sending a merged list of answers.
- (Com Type2) It accepts queries for services from the closest peers, it resolves them locally by using the local query engine or globally through re-querying the closest peer SRs (if the life time of the query is still positive), and responds by sending a merged list of answers.

2. Local query: The local query engine accepts queries from the communication engine, performs the queries to the local database, where the SR operates, and finally, forwards the responses to the communication engine.

The peer-to-peer SR network is an unstructured peer-to-peer network, and it is based on the Gnutella [20] communication protocol. Following, we analyze some of the important components of the Callimachus discovery service.

Communication Engine. The SR's communication engine exchanges XML messages with both clients and other SRs. The difference between the two types of communication is a field named TTL that concerns the life duration of a query. This field is included into the message when it is being sent from SR to SR. Apparently, this field is not required in cases where the client communicates with an SR (Com Type1).

The communication engine operates as following: firstly, it initializes and conserves the peer-to-peer SRs' network, and secondly, it performs the distributed discovery. Both of the above actions are compatible with the Gnutella peer-to-peer network specifications.

Local Query Engine. When the Local Query Engine receives a query for a specific service from the communication engine, it analyzes it and prepares the corresponding relational queries to the database. The analysis is performed by an XML analyser. The XML is being transformed to a DOM tree after a structural validation test against the corresponding DTD. In the XML there is a Boolean field named "QTYPE" that determines the locality or the globosity of the service discovery. Then the local query machine connects and queries the relational local database. The search results are collected and sent back to the SR.

Network Iinitialization. Each SR keeps a list with the most recent SRs of the network. This list is called SR cache. Each time an SR is activated, it looks at the SR cache in order to find "k" active nodes, called neighbour (closest) nodes. By this way a new SR can be connected with the peer-to-peer network based on the local information without the need for a centralized registration. The first time that an SR is activated, the SR cache contains some SRs that are usually active. Each time an SR is connected to the network, it uses a ping – pong method (similar to Gnutella's) to find other active SRs and to update its cache.

Discovery Mechanism. Aiming to the discovery of a requested service by the client, the SR creates a query to the SR network. The SRs cooperate with each other so as to distribute the query. The distribution is based on the blind flooding mechanism and is being determined by its own TTL. When an SR receives a copy of the query, it decreases the TTL by one (if TTL>0) and forwards it to the closest SRs. At the same time, the query is being forwarded to the same SR. If the local search finds one or more matching results, then the SR prepares the response and sends it back to the parent SR. In this case, the response follows the reverse path of nodes. During the distribution of the query to the

SRs, it is possible for the path to make circles between same nodes. Thus, the SRs use a field ("QUERYUNIQUEID") that holds a unique query ID, and by this way, the execution of the same query in the same SR is avoided.

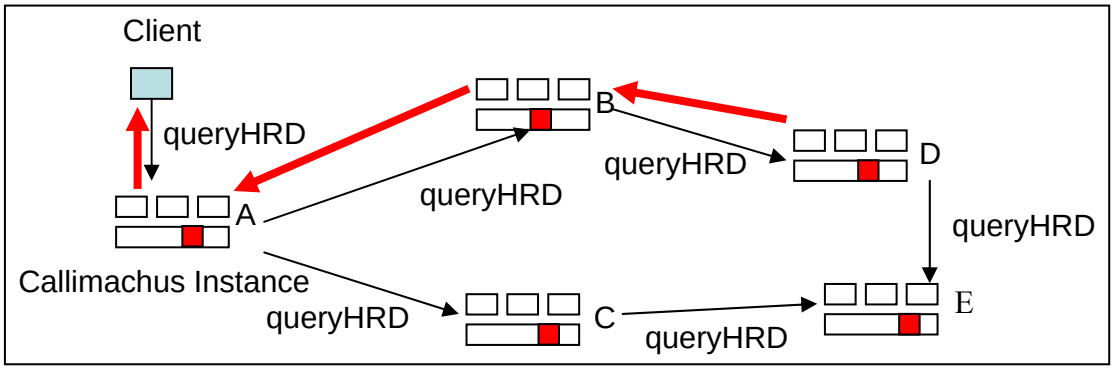

Fig.4 Service discovery in a peer-to-peer network of five (5) Callimachus' instances.

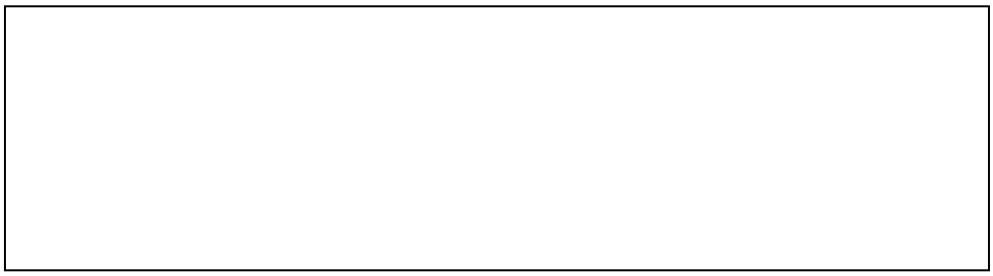

Fig. 5. Example of a Query to a SR

Selecting and using an appropriate hypermedia service. From the list of the matching services that comes back from the SR, the developer can select an appropriate service to work with. When a service is selected, the HRD of that service is being available to the developer. The HDR contains all the necessary information about using the service. In that case, the information piece that concerns service location, communication and description information is the starting point of the task of the service usage. A location of a detailed service operation description (such as URL of a WSDL file in a web service) is also available, in order to make the comprehension of the whole service API possible. Depending on the way each hypermedia server provides its services to the web, a communication establishment will take place between the web application and the hypermedia server. By using standard communication and interoperation technologies, like the web services, the above procedure can take place quickly and semi automatically.

4.3 Prototype Implementation

The implementation of the Callimachus discovery prototype includes the design of a relational database, the implementation of the service discovery component and the implementation of a service discovery tool for visual representation. The database stores the HRDs, while the service discovery server is responsible for both the registration of the HRDs in the database and the local and the global

search of them. The design of the database schema supports the addition of new fields and is aware of the XML format of the HRDs.

The service discovery server (which is using the HTTP communication protocol) has been implemented in Java programming language. This server performs the XML parsing and validation, the communication with database, and finally, participates in the peer-to-peer mechanism.

The service discovery tool helps developers by connecting to the service discovery system and searching for required services visually.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE COMPONENTHRD SYSTEM "replyhrd.dtd">
<RESPONSE>
        <SERVICES>
                <COMPONENTHRD name="Calimachus_9091">
                        <HMINFO>
                                <DOMAIN>Taxonomic</DOMAIN>
                                <PROTOCOL>SOAP</PROTOCOL>
                        </HMINFO>
                        <COMPONENTGESTALT>
                                <TYPE>Service</TYPE>
                                <FRAMEWORK>WebService</FRAMEWORK>
                        </COMPONENTGESTALT>
                        <FRAMEWORKDATA>
                                <HOST>diogenis.ceid.upatras.gr</HOST>
                                <PORT>4567</PORT>
                                <DESIGNDATA>
                                        <MODULEFILE>
                                                <MODULENAME>Babylon</MODULENAME>
                                                <MODULETYPE>stub</MODULETYPE>
                                                <MODULEVERSION>1.0.1</MODULEVERSION>
                                                <LOCATION>dir/modules</LOCATION>
                                                <NAMINGSERVICE>BABNS</NAMINGSERVICE>
                                                <ALLOW>authorizedusers</ALLOW>
                                        </MODULEFILE>
                                </DESIGNDATA>
                                <IMPLEMENTATIONDATA>
                                        <IMPLEMENTATIONFILE>
                                                <LIBRARYNAME>BabylonImpl</LIBRARYNAME>
                                                <LIBRARYTYPE>exe</LIBRARYTYPE>
                                                <LIBRARYVERSION>1.0.1</LIBRARYVERSION>
                                                <DIRECTORY>dir/implementations</DIRECTORY>
                                        </IMPLEMENTATIONFILE>
                                </IMPLEMENTATIONDATA>
                        </FRAMEWORKDATA>
                        <COMPONENTDATA>
                                <COMPONENTSTATUS>Available</COMPONENTSTATUS>
                                <DESCRIPTION>Taxonomic Reasoning. Allows youto create and work with taxonomies of URLs</DESCRIPTION>
                                <RESOURCES>
                                        <APIDEFINITION>
                                                <TYPE>wsdlfile</TYPE>
                                                <LOCATION>http://diogenis.ceid.upatras.gr/babylon/babylon.wsdl</LOCATION>
                                        </APIDEFINITION>
                                        <HELP>
                                                <TYPE>web page<TYPE>
                                                <LOCATION>http://diogenis.ceid.upatras.gr/babylon/index.php</LOCATION>
                                        </HELP>
                                </RESOURCES>
                                <QUALITY>
                                        <AVAILABILITY>99%</AVAILABILITY>
                                        <RESPONSIVENESS>1,2ms</RESPONSIVENESS>
                                        <USERWAITINGTIME>0,3 sec</USERWAITINGTIME>
                                        <MAXIMUMUSERNUMBER>100</MAXIMUMUSERNUMBER>
                                </QUALITY>
                        </COMPONENTDATA>
                </COMPONENTHRD>
        </SERVICES>
</RESPONSE>
```

Fig.6. Response of the Service Registry (SR): After the discovery procedure, the SR sends back the HRDs of the discovered services. In this paradigm the SR has discovered one service.

For the evaluation needs, three SR instances have been created. The operation of the whole discovery system has given successful results. As it can be seen in figure 7, the tool presents a list of matching services in the network of the Callimachus system instances.

By selecting the Babylon server, the tool presents detailed information about its provided services, and the web developer can follow the proposed links to the URL(s) with the full description of the Babylon server. The Babylon server can provide taxonomic services to developers and allow access to clients through the SOAP web service standard. Thus, developers can use the service in the same easy way they include web services into their applications.

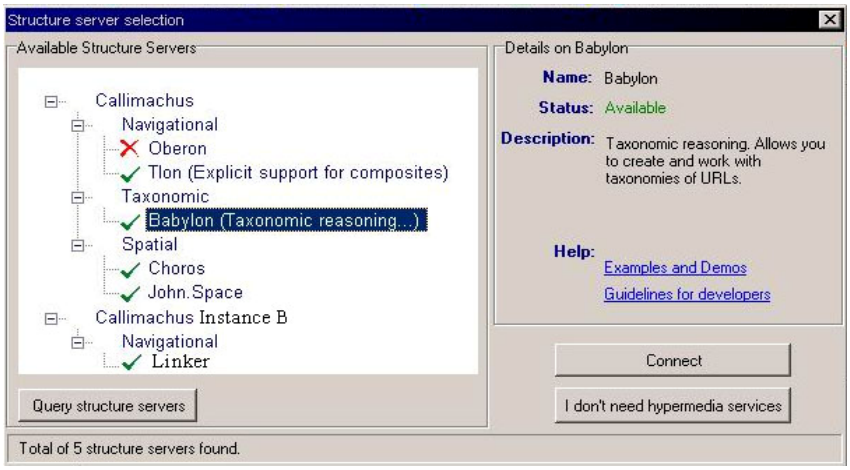

Fig. 7. The Callimachus Service Discovery Tool.

5 Observations and Future Work

The observations from the study of the above prototype concern both the hypermedia service discovery mechanism and the developer support, in the perspective of service usage or service integration with web applications.

The architecture of the presented prototype allows SR to be placed on distributed locations. This means that, apart from the Calimachus example, a P2P based hypermedia service discovery mechanism will be able to enlarge by encapsulating new distributed SR into its network, and to obtain resource descriptors for a variety of available hypermedia services. Furthermore, the unstructured nature of the P2P architecture, will allow new hypermedia systems to register their services in a decentralized manner without any central authority.

Secondly, an interesting thing is the native support of service description within service discovery. Instead of using predefine resource descriptors, SR is handling HRDs with an open set of describing fields. It is easy to augment HRD with new fields aiming to fulfill both the new services' specs and the searching requirements of the developers.

Moreover, the adoption of such architecture will satisfy the needs for publishing, locating and selecting hypermedia services that come from a variety of service-based hypermedia systems. Open Hypermedia Systems, Component based Hypermedia Systems and Service Oriented Hypermedia Systems will be able to corporate with a P2P service discovery network as service providers. Additionally, the next generation hypermedia systems: peer-to-peer hypermedia [8] could also interoperate with service discovery due to their common P2P network basic structure.

Despite the above potentialities, there are some interesting issues that have to be addressed in the future:

1. Large scale testing of service discovery prototype. In the Gnutella architecture it is hard to locate the desired information without flooding queries to most parts of the overlay network. Broadcasting on every query is not much scalable. More scalable search methods should be examined.
2. The Gnutella architecture is based on a pure peer-to-peer communication protocol. New protocols such as the sun's JXTA and the Microsoft's P2P SDK have improved both quality and efficiency.

Concluding, the current task shows us that in SOA, contemporary tools cannot fully cover the needs for rapid development and prototyping due to their static and no agile nature. Thus, development proves difficult and error prone, requiring great maintenance efforts. In this task, we have presented an infrastructure upon which new tools can be built. One of these tools is service discovery. It is the first step during integration and deployment of services, and an important factor for achieving short development cycles, rapid prototyping and smooth evolution.

6 Conclusions

The Open Hypermedia Community can enrich web applications with valuable hypermedia features. Towards that, hypermedia services have to become available to web developers. Aiming to the efficient provision of such services, the OHSs are required to move towards to SOA. In this context, the existence of a Developer Support Framework that helps developers to search and use hypermedia services is critical. Qualities like hypermedia service discovery and hypermedia service introspection are important issues included in that framework. Finally, rethinking the design of hypermedia systems from the developer's perspective can help the facilitation of the web development procedure and increase the OHS usage.

Acknowledgements

The authors would like to thank Ippokratis Pandis for his valuable contribution.

References

1. Agrawal, R., Bayardo Jr. R. J, Gruhl, D. and Papadimitriou, S. Vinci: A Service Oriented Architecture for Rapid Development of Web Applications. http://www10.org/cdrom/papers/506/
2. Anderson, K. M. (1997). Integrating Open Hypermedia Systems with the World Wide Web. Proceedings of 1997 ACM Hypertext.
3. Anderson, K. M., Sherba, S. A., Lepthien, W. V. (2003). Structure and behavior awareness in themis. Proceedings of 2003 ACM Hypertext, pp.138-147.
4. Anderson, K. M., Taylor, R. N., and Whitehead, E. J. (1994). Chimera: Hypertext for heterogeneous software environments. Proceedings of ACM ECHT '94, pp. 94-107.
5. Avila-Rosas, A., Moreau, L., Dialani, V., Miles, S., and Liu, X. (2002). Agents for the Grid: A comparison with Web Services (part II: Service Discovery). Proceedings of AAMAS'02.
6. Banaei-Kashani, F., Chen, C. and Shahabi, C. (2003). WSPDS: Web Services Peer-to-Peer Discovery Service. Intl. Symposium on Web Services and Applications, 2004.
7. Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H. F., and Secret, A. (1994). The World-Wide Web. In Communications of the ACM, 37(8), pp. 76-82.

8. Bouvin, N. O. (2002). Open Hypermedia in a peer-to-peer context. Proc. of 13th ACM Hypertext.
9. Bouvin, N. O. (1999). Unifying Strategies for Web Augmentation. Proc. of 10th ACM Hypertext.
10. Carr, L., et al. (1995). The Distributed Link Service: A Tool for Publishers, Authors and Readers. In Fourth International World Wide Web Conference: "The Web Revolution".
11. Chiu, C. M., Bieber, M., and Lu, Q. (2002). Towards Integrating Hypermedia on the Web. Proc. of 35th Annual Hawaii International Conference on System Sciences (IEEE 2002).
12. Christodoulou, S.P., Zafiris, P.A., and Papatheodorou, T.S. (2001). Web Engineering: The Developers' View and a Practitioner's Approach. Web Engineering, Software Engineering and Web Application Development, Lecture Notes in Computer Science, Volume 2016, pp.170-187.
13. Cotroneo, D., Di Flora, C., and Russo, S. (2003). An enhanced service oriented architecture for developing web-based applications. Journal of Web Engineering, Vol. 1, No. 2, 128-146.
14. Engelbart, D. (1998). ACM Hypertext Conference '98 OHS Workshop Keynote Address.
15. Gronbaek, K., Bouvin, O. N., Sloth, K. (1997). Designing Dexter-based Hypermedia Services for the World Wide Web. Proceedings of Hypertext '97, pp. 146-156.
16. Halasz, F., and Schwartz, M. (1994). The Dexter Hypertext Reference Model. Communications of ACM, 1994, 37 (2), pp. 30-39.
17. Hall, W., Davis, H., and Hutchings, G. (1996). Rethinking Hypermedia: Microcosm Approach.
18. Hicks, D. L. (MIS' 2002). In search of a user base: Where are the B's?. In Proc. of MetaInformatics Symposium, 2002.
19. Karousos, N., Pandis, I., Reich, S., and Tzagarakis, M. (2003). Offering Open Hypermedia Services to the WWW: A Step-by-Step Approach for the Developers. Proc. of WWW2003, pp. 482-489.
20. Lessig, L. (2001). Peer-to-Peer, Harnessing the Power of Disruptive Technologies, ed Oram.
21. Li, L., and Horrocks, I. (2003). A software framework for matchmaking based on semantic web technology. Proc. of WWW 2003, pp 331-339.
22. Nürnberg, P. J., Leggett, J. J. and Schneider E. R. (1997). As We Should Have Thought. In Proc. of 8th ACM Hypertext Ά97, pp. 96-101.
23. Nürnberg, P. J., Leggett, J. J., and Wiil, U. K. (1998). An agenda for open hypermedia research. In Proc. of 9th ACM Hypertext, pp. 198-206.
24. Nürnberg, P. J., and Schraefel, M. C. (2002). Relationships Among Structural Computing and Other Fields. JNCA Special Issue on Structural Computing, 2002.
25. Pandis, I., Karousos, N. and Tiropanis, T. (2005). Annotations: Semantically annotated hypermedia services. Proc. of 2005 ACM's Hypertext.
26. Shackelford, D. E., Smith J. B., Smith F. D. (1993). Architecture and Implementation of a Distributed Hypermedia Storage System. Proc. of 1993 ACM Hypertext, pp. 1-13.
27. Tzagarakis, M., Avramidis, D., Kyriakopoulou, M., Schraefel, M., Vaitis, M., and Christodoulakis, D. (2002). Structuring Primitives in the Callimachus Component-Based Open Hypermedia System. JNCA Special Issue on Structural Computing, 2002.
28. Tzagarakis,, M., Karousos, N., Christodoulakis, D. and Reich, S., (2000). Naming as a Fundamental concept of open hypermedia systems. Proc. of 2000 ACM's Hyperetext.
29. Universal Description, Discovery and Integration of Web Services (UDDI). http://www.uddi.org.
30. Vitali, F., Bieber, M. (1999). Hypermedia on the Web: what will it take? ACM Computing Surveys (CSUR)
31. W3C Simple Object Access Protocol (SOAP). http://www.w3.org/tr/SOAP.
32. W3C Web Services Architecture Domain. http://www.w3.org/2002/ws/.
33. W3C Web Services Description Language (WSDL). http://www.w3.org/tr/WSDL.
34. Walker, J. (2005). Authoring for comprehension: Feral hypertext: when hypertext literature escapes control. Proc. of 2005 ACM Hypertext.
35. Will, U. K., Hicks, L. D., and Nurnberg P. J. (2001). Multiple Open Services: A New Approach to Service Provision in Open Hypermedia Systems. Proc. of 2001 Hypertext, pp. 83-92.