

An Engineering Perspective on Structural Computing: Developing Component-Based Open Hypermedia Systems

Michail Vaitis¹, Manolis Tzagarakis^{2,3}, George Gkotsis³

¹ Department of Geography, University of the Aegean, Greece
University Hill, GR-811 00 Mytilene, Greece

² Research-Academic Computer Technology Institute, Greece
Riga Ferraïou 61, GR-262 21 Patras, Greece

³ Department of Computer Engineering and Informatics, University of Patras, Greece
GR-265 00 Patras, Greece
vaitis@aegean.gr, tzagara@cti.gr, gkotsis@ceid.upatras.gr

Abstract. The emergence of Component-Based Open Hypermedia Systems aims at the releasing of Hypermedia and Web applications from the monocracy of link as an information structuring primitive. Instead, an open set of structure services, each one providing structure abstractions relevant to a specific domain, are offered to an open set of client applications. Nonetheless, the lack of an engineering framework guiding the development process of CB-OHS has a part in their limited exploitation. In this paper, we analyze the characteristics of CB-OHS from an engineering approach, and we propose a framework and a number of tools, supporting all phases of their development process.

1. Introduction

Many researches during the past decade have pointed out a certain kind of inadequacy in both Hypermedia and Web applications, concerning the information structuring abstractions they provide. This “Structure Crisis” mainly originates from the nature and implementation of the notion of link. In the Web, links are limited in functionality (they constitute starting-points for unidirectional jumps), embedded into the HTML file (so characterized as “second-class” entities). In Hypermedia Systems, although links and anchors are first class entities, they are employed for incarnating all information structuring situations. Unfortunately, any closed set of abstractions cannot be guaranteed to be useful in a practical sense for all possible applications [13]. These attempts raised convenience and efficiency problems, besides lack of standards and interoperability capabilities.

A significant amount of research and development efforts aiming to overcome the above issues has resulted in the releasing of the structure abstractions from both data and core systems’ functionality. Instead, structure abstractions have been promoted to first class entities, being provided to third party client applications on demand, through specific software components (called *structure servers*) that form the middleware part of a tri-tier architecture (figure 1).

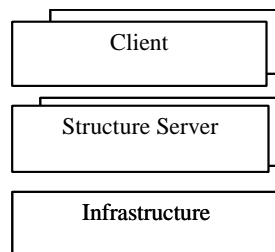


Fig. 1. Component -Based Open Hypermedia Systems architecture

Component-Based Open Hypermedia Systems (CB-OHS) deliver an open set of structural abstractions to an open set of client applications, while all fundamental functionalities (like persistent storage, concurrency and event notification control, versioning and naming) are provided by the infrastructure backend.

The emergence of CB-OHS as the realization of structure emancipation, has further promote research in both the analysis of the different ways people use to structure information (called *hypermedia domains*), and the development of systems and tools that assist these structural activities (belonging to the “B-level” or “C-level” of work¹). The new field of *structural computing* (SC), asserting the “primacy of structure over data” [13], is aiming to shape the theoretical and practical foundations upon which structure services will eventually become ubiquitous to all computing environments.

Although CB-OHS and structural computing are among the first forerunners of nowadays trend for service-oriented computing [17], there is little acceptance of their potential role in hypermedia and web applications development efforts [14]. This constitutes another “crisis” that should be overcome. We argue that one of the reasons is the lack of a predefined software engineering framework for CB-OHS, coupled by the appropriate tools to support it. This results in ad-hoc development methodologies, producing systems that lack certain essential characteristics. The development of a structure server is a complicate task to be repeated from scratch every time a new structure abstraction has to be supported [26]. In this paper we analyze structure servers from an engineering point of view and propose a development framework, involving all aspects of their life cycle. We anticipate the strengthening of the research and development work in the field of structural computing and the drawing of attention of researchers of relative fields.

The rest of the paper is organized as follows. In section 2 we present the field of structural computing and describe the functionality and internal organization of related systems. In Section 3 we propose an engineering framework aiming to steer all the development processes of CB-OHS, while in section 4 we concentrate on the appropriate tools for supporting the framework. Section 5 comments a number of existing structural computing environments, and finally section 6 concludes the paper and presents future research and development directions.

2. Structural Computing

Hypermedia has been used to support a wide variety of user tasks. These tasks range from Bush’s association of information to more elaborate activities, such as hyperfiction authoring and reading, information analysis and classification. Each of the tasks exemplifies how the human mind perceives structure in different problem domains. The identification of new problem domains is the main concern of *hypermedia domain research*. On the contrary, *hypermedia system research* is focused on designing and building the computational foundations to support people working with structure, concentrating especially on issues regarding openness. The Open Hypermedia movement [16] originated from such an approach. Yet, the conceptual foundations of Open Hypermedia – its underlying structures and behaviours – have all focused on supporting one task: information navigation. As it has been shown, the abstractions provided by systems supporting information navigation cannot address issues in new domains (e.g. spatial, taxonomic) in a convenient and efficient way. These domains require structural abstractions markedly different from those used to support navigational hypermedia, manifesting, thus, a gap between hypermedia domain and system research. The need for delivering the tailored support required by different domains gave birth to Component-Based Open Hypermedia Systems (CB-OHS).

CB-OHS are the incarnation of a new approach in solving data organization problems, called *structural computing*. Research in structural computing focuses on all aspects of information structuring problems. Without being biased towards supporting only navigation among data items, structural computing attempts to provide a framework where a number of different hypermedia domains can co-exist. By providing such a unifying framework, it aims at narrowing the gap between hypermedia domain and system research. Structural computing focuses on providing general structure-oriented models and services that are able to be adapted to domain specific abstractions easily and efficiently. A ‘hypermedia domain’ is defined by a coherent set of abstractions that solve a particular data organization problem. CB-OHS are able to support well known domains, such as the navigational, argumentation, spatial, taxonomic and configuration management, as well as other ‘exotic’ domains, such as workflow, hyperfiction and linguistics.

¹ According to Douglas Engelbart [5], there are three types of work that can be performed in an organization. The A-level is the work of the organization itself. The B-level is work that develops tools to improve the ability of people performing A-level work, while C-level is work that develops tools to augment the ability of people performing B-level work.

The provision of dedicated structure services for each domain, results in a more convenient and efficient utilization of its abstractions, improving in turn the performance, quality and cost-effectiveness of client applications. This fact is acknowledged as a great benefit of CB-OHS, since contemporary systems struggle with the close coupling of structure models to their infrastructure.

Structural computing systems may be viewed as part of multiple open service systems, i.e. systems supporting arbitrary middleware services that can be divided into infrastructure and application services [25]. Structural computing attempts to change the way the invisible, but important, infrastructure of contemporary Open Hypermedia Systems work, in providing open structure-based services in heterogeneous environments. Thus, structure-based services are considered as components in the context of service-oriented computing (SOC), where services reuse and composition constitute a fundamental activity for application development [3, 17]. Therefore, structural computing focuses on the developer's side, aiming to provide tools and services for assisting structure servers and client systems' development.

In figure 2, the conceptual internal architecture of a CB-OHS is presented. The various entities of both middleware and infrastructure layers are described below, along with the appropriate protocols and interfaces.

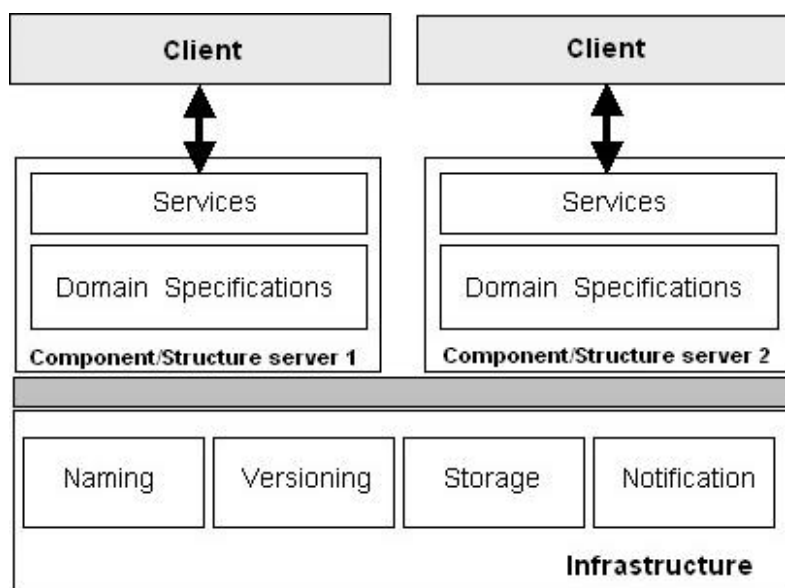


Fig. 2. Conceptual internal architecture of CB-OHS

- *Component/Structure server*: Reifies the domain specific abstractions, providing the domain specific services to clients. They are semi-autonomous components, since they rely on the infrastructure services for common functionality. They establish a well-defined interface for communication with client applications.
- *Structure specifications*: Comprise specifications about the structure abstractions of the domain in terms of both structure entities (patterns) and behavior semantics. Behavior models the computational aspects of the domain and can be divided into *internal operations* used mainly for consistency reasons (e.g. to affirm conditions and constraints or to interpret abstractions in a specific manner), and *external operations* invoked by clients.
- *Services*: Implement the domain specific external operations of the domain. They are available to clients through the use of a specific interface (e.g. `openNode`, `traverseLink`).
- *Infrastructure*: Includes the fundamental functionality that is available to all structure servers. Persistent storage, naming, event notification control and versioning constitute essential common services. A well-defined protocol is provided for the communication among structure servers and infrastructure services.
- *Storage*: Provides persistent storage services for structures as well as for domain specifications. The storage protocol manipulates primitive (domain neutral) structure entities. It is the responsibility of

the structure server to transform (or cast) them to the domain specific structure abstractions. Domain specifications are managed separately in a *repository*, in order to support reusability and extensibility of structure patterns among structure servers.

- *Client application*: Any third-party program that requests structure functionality, from one or more structure servers. To utilize structure services, clients may be either custom-build applications, or extensions to existing applications. In the later case, either direct extensions are made, or wrapper programs are separately developed.

The presented internal architecture of CB-OHS implies the necessity of a methodology for structure patterns and behavior specification. Although such a methodology is still an open research issue in the structural computing community, the resultant benefits have already been underlined:

- Better understanding of the domain. So far, domain foundations are hard coded into systems and services, and are informally described.
- The domain specifications could be the framework of a structure server. It may be possible to automatically configure a structure server by setting or modifying structure specifications.
- Exploitation of common structures among different domains, thus enhancing reusability and interoperability.
- Narrowing the gap between hypermedia domain and system research, by providing a common framework to express structural abstractions.

As stated above, the transition from early monolithic hypermedia systems to Open Hypermedia Systems and recently to CB-OHS was directed by the vision to provide open structure-oriented functionalities to every concerning application, in a convenient and efficient way. The layered architecture of CB-OHS aims to improve the work of both application engineers and structure server developers, enabling them to utilize high-level abstractions offered by the underneath layer. Nonetheless, the lack of an engineering framework guiding the development process counteracts most of the anticipations of structural computing.

3. Structural Engineering

Hypermedia and web applications are differentiated from conventional software products in a number of characteristics, including navigability, provision of search mechanisms, appropriate content organization, aesthetic and cognitive aspects. As pointed out in [11], hypermedia applications “*uses associative relationships among information contained within multiple media data for the purpose of facilitating access to, and manipulation of, the information encapsulated by the data*”, while in [4] a Web Hypermedia Application is defined as “*the structuring of an information space in concepts of nodes (chunks of information), links (relations among nodes), anchors, access structures and the delivery of this structure over the Web*”. The above definitions imply a number of specific activities during hypermedia development, such as content acquisition and structuring, navigational and aesthetics design, and multimedia synchronization. The fields of Hypermedia and Web engineering have emerged, aiming to provide a systematic (scientific and practical), disciplined, quantifiable approach to the development, operation and maintenance of hypermedia (or web) applications [11, 7].

The development process of a hypermedia application includes a *design phase*, where issues like application architecture, content scope, structure, depth, granularity, presentation metaphor, viewpoints and access mechanisms are considered [11]. A number of design models have been proposed in the literature to assist this particular phase (e.g. HDM [6], RMM [8] and OOHDM [20]). What is common in all the aforementioned applications, development processes and design models, is the implied support of the navigational domain, resulting in the utilization of constructs like the node, the anchor and the link. Since structural computing perceives the navigational domain as just an instance (even the most significant) of an open set of structure services, hypermedia and web applications has the potential to exemplify customized structural abstractions according to their needs. Consequently, some modifications should be carried out in their development process.

We introduce *structural engineering* as the framework referring to a systematic and disciplined approach to the development, operation and maintenance of applications and infrastructures that solve structure-oriented problems. We argue that the design phase of applications should follow or comprise a *structure assessment phase*. The purpose of this phase is to analyze the structure abstractions of the hypermedia application and identify the structure services that have to be used. During the implementation phase the de-

veloper should locate the appropriate structure servers and exploit their protocols. In case the application designer or developer cannot identify or locate a structure service, there is an opportunity for the establishment of a new hypermedia domain.

For the definition of a structural engineering framework, the special characteristics of the structure service components should be identified and analyzed. This is the purpose of the next subsection.

3.1 Characteristics of structure servers

So far, [2] is the only work that has addressed a number of engineering requirements for structure servers. We extend this list in order to provide a more complete illustration on the subject.

- *Structural completeness*: The structure abstractions supported by the structure server should completely solve the structure-oriented problems of the domain.
- *Size*: Structure servers are considered small to medium software projects. The main task is the development of the middleware component, as the infrastructure is already functional.
- *Performance*: Although depending on domain behavior semantics, the response time of clients' requests to structure servers should be short.
- *Distribution and Heterogeneity*: Structure servers should operate in a distributed environment composed of different hardware and software platforms.
- *Specifications evolution*: The decision for the construction of a new structure server should be taken only when the application needs could not be satisfied by existing services. This prerequisites a deep study of the application domain, so there is only a small possibility for the specifications to be changed during the development of the structure server. Nevertheless, the usage of CASE tools during the implementation project minimizes the cost of unexpected requirements changes.
- *Reusability and Extensibility*: Structure services at a fine granularity level constitute building blocks that have a great potential to be extended or reused during the development of other, more complicated ones.
- *Life time*: The duration of a structure server is long, presuming that the decision for its development is carefully determined. As the software implementation technologies continually evolve, structure servers may need to migrate to different platforms from time to time, while providing a constant interface to third party applications.
- *Robustness, Scalability and Availability*: These properties are essential since structure servers may be used continually by a huge number of applications.
- *Introspection capabilities*: Structure serves should be able to communicate their behavior to other applications (i.e., their services interface and descriptions, location, and access control details).
- *Interoperability*: Structure servers that provide functionality for the same hypermedia domain should be able to interoperate. In addition, a useful requirement is the existence of supporting mechanisms for the transformation of structures between different hypermedia domains. In this way, structures may be shared among structure servers.

3.2 Life cycle of structure servers

Although the size of the structure servers is usually small, there is a need for a disciplined development methodology, due to the demanding characteristics that should be met, as presented in the previous subsection. Based on the conventional software process phases of Specifications, Development (itemized in analysis, design, coding, and integration), Testing, and Evolution/Maintenance [18, 19], we propose the following life cycle for the structure servers (figure 3), while in the following paragraphs we describe each one of its phases.

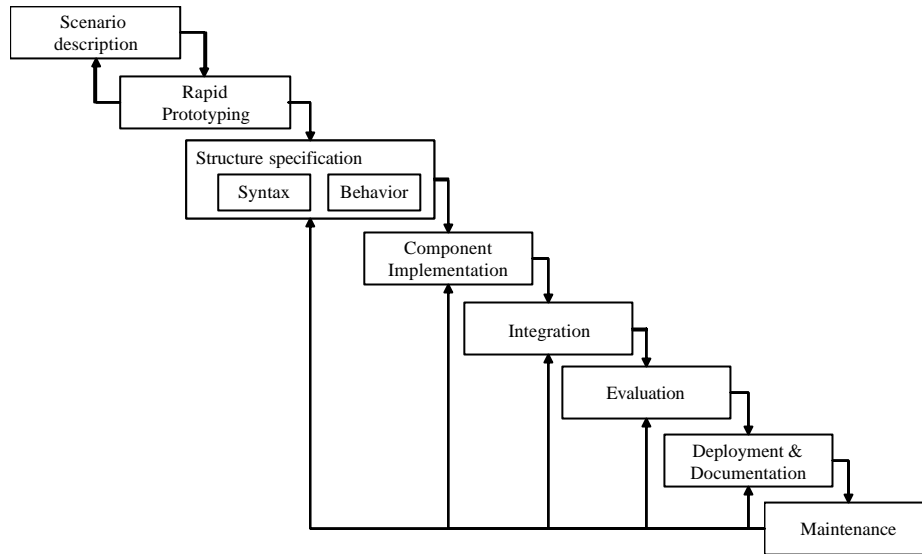


Fig. 3. CB-OHS life cycle

Scenario description

We incorporate the scenario-based specification for open hypermedia systems [16], providing some essential modifications. All functionality proposed to be part of a given structure server should be justified through one or more scenarios of its use. That is, when a proposal that some given structure abstraction should be implemented is arisen, a scenario based on actual or foreseen use should be mapped out. This policy facilitates discussions among hypermedia application designers and developers, as to better specify the desired structure functionality and avoid “reinventing the wheel”. The description of a scenario could include the following paragraphs:

- Goals (name of each goal, plus a possibly lengthy description of it),
- Characters (the different kind of users of the service, plus a possibly lengthy description for each one),
- Data (some examples of data items that may be associated together with the structure abstraction),
- Requirements for third party applications (requests),
- Structure configuration (description and constraints among the structural elements),
- Behavior description (operations and propagation of them, synchronization among elements),
- Infrastructure requirements (storage, naming etc.).

Rapid Prototyping

The output of this phase is a prototype structure server that simulates the intended functionality, while not being fully operational in real. In this way, the evaluation of the scenario is possible and the developer of the client application has the opportunity to test the effectiveness and correctness of the desired services. Accrued ambiguities or misunderstandings are clarified and modifications to the scenario paragraphs are revealed. This testing and backtracking cycle eventually leads to a complete and correct scenario specification.

Structure syntax specification

In this phase, the *structure model* of the domain is specified. The purpose of the structure model is to make concrete the desired structure abstractions and distinguish them from the data abstractions where they usually reside in traditional software applications. In this way, structure is elevated to a first class entity, enabling users, designers and developers to discuss and reason about it. Such a model should contain the basic structural elements, their properties and the connection constraints among them, or to data items. In addition, a thorough analysis of already developed components should be carried out, in order to detect relevant elements that maybe reused. Methodologies that have been used for the specification of structure models

include UML (in the Construct [27] and Themis [1] structural computing environments), XML (in the Callimachus CB-OHS [21]) or proprietary formalizations (like FOHM [12]).

Structure behavior specification

Behavior embodies the computational aspects of a domain and is tightly coupled with the structure model of it. There are two categories of behavior, depending on the calling entities: Services, that are available to clients through the server protocol, and internal operations, that are used by the structure server internally (for example, for consistency checking), or when communicating with the infrastructure. In spite of the word “computing” in its name, the field of structural computing has no significant research results to present regarding behavior. In most cases, behavior is considered as an “add-on” that is developed on top of the structure model. Recently, structure and behavior are considered by many researchers as different views of the same “whole” [23, 15], introducing the possibility of specifying structure and behavior within the same phase or even in one step (for example, using UML in Construct [27]).

Component implementation

Based on the outcomes of the previous phases, during component implementation the structure server is being reified. Activities that should be carried out are the implementation of the structure model and internal operations, the realization of the protocol (interface) for communication with the clients, the exploitation of the infrastructure services, and the eventual reuse, extension or customization of already existing components. A corner-stone task is the implementation of functions that cast the neutral structural objects stored by the infrastructure, to the specific structure elements concerning the application domain. In [21] two internal layers are identified in a structure server: The Abstraction Factory Layer (AFL) which is responsible for reifying un-typed structural objects to domain specific abstractions, and the Abstraction Utilization Layer (AUL) where the domain specific abstractions — once created — may be used by clients requesting structuring functionality.

Integration

Integration is a dispensable task in current structure servers since they are developed as embedded services in the development environment and they are tightly-coupled with it. As we envision the independence of structure services from their development environments, such a phase is essential. Activities to be carried out include integration within web servers, binding of port numbers, arrangement of authoritative and security issues.

Evaluation

The evaluation activities comprise mainly the ensuring of properties such as functionality, performance, compatibility, reliability, and usability.

Deployment and Documentation

The deployment activity turns the structure server in operational mode, so clients maybe use its services. A certain prerequisite for that is clients’ ability to discover and locate the required service. The documentation activity includes both the registration of the structure server to the dedicated directory services, and the configuration of its introspection capabilities. In [9] a Hypermedia Service Description Language is proposed, providing a wide range of information that requesting clients can exploit, such as host and port numbers, interfaces definitions, comments, etc.

Maintenance

As mentioned before, if a careful analysis of the structure abstractions is carried out, the functional part of a structure server is almost unlikely to evolve. Maintenance is mainly engaged in *corrective*, *adaptive* and *perfective* tasks, targeting to correct, complete and efficient structure services.

4. Tools and services

Tools that facilitate the entire development process of structure servers are still missing. In the following, we list a number of tools that seem promising in deploying the rich services of CB-OHS and furthermore strengthen the structural computing community. These tools are classified into two categories: *Theoretical armoury* tools, aiming at supporting problem analysis within the structural computing framework, and *development tools*, attempting to assist developers when working with CB-OHS.

4.1 Theoretical tools and armoury

As already mentioned, CB-OHSs are an incarnation of structural computing, which suggests a specific approach to problems dealing with organization of data. Being a technology as well as a philosophy and school of thought, structural computing requires new ways that will help: (a) Examine its own foundations, and (b) Analyze real-world organizational problems in a proper manner. These tools are needed primarily by analysts and to a lesser degree by developers. Theoretical tools are required on two important research fronts, analyzed below.

1. *Structural completeness* of models and systems: Since structural computing (and thus CB-OHSs) asserts that it provides a framework able to cover any need for working with structure, processes are required that will examine to what degree models may cover or solve structural problems. The coverage of these models determines their *structural completeness*.

2. Methodologies for *structural analysis* and *decidability*: CB-OHS support domains in a very abstract way. This means that the abstractions provided by components solve a *family of problems*. However, real life organizational problems, do not have an abstract view, but are very concrete. Currently there are no systematic approaches to reduce a real life organizational problem to a hypermedia domain; or more concisely, given an organizational problem, how can we determine to which hypermedia domain it belongs? Consider for example hierarchical security models that support users and groups, which in turn may consist of other groups. Is such a hierarchical security model a special case of the taxonomic domain? And if not, why? Does that mean that this problem introduces a new domain? Currently, structural computing cannot answer these questions in a systematic manner. In order to do so, methods for structural problem analysis need to be established, methods that can compare models and determine their differences, as well as methods that are able to decide whether or not a particular organizational problem belongs to a hypermedia domain or not. An organizational problem is said to be *decidable* within a structural computing environment if there exists an effective and systematic procedure (i.e. comprised of finite steps) that solves the problem within the structural computing framework. All those theoretic procedures determine the property of *structural decidability* of a system.

4.2 Development tools

Besides the theoretical tools needed by analysts when working within a structural computing environment, actual development tools are needed by developers that will help them taking full advantage of the provided services of CB-OHS. They deal with every aspect of CB-OHS, involving model and system issues.

1. *Tools for structure model definition and configuration*. The development of components would be substantially easier if a structure specification formalism would be available to them. Such formalism should be open to extensions, model-neutral and provide a common ground for cooperation. Although initial attempts of such formalisms exist (e.g. [1], [27], [21]), they do not cover all domain specific aspects and have not been excessively deployed in order to see their shortcomings.

2. *Tools for structure behaviour specification*. While tools for structure model configuration are aiming primarily to syntactical aspects of structure, tools for structure behaviour specification are aiming to dynamic and computational aspects. Based on the structure model definition, tools for behaviour specification

would allow controlling the life-span of structural abstractions, how structural abstractions interact and how they react to messages.

3. *Tools for discovering components in CB-OHS.* The plethora of potential components provided by CB-OHS raises not only issues with respect to components usage [10] but also issues relevant to locating components and interrogate their status and availability. While naming can solve the problem of locating structure servers [22], it can do so only if the names of structure servers are known. When names of structure servers are unknown, locating structure servers in contemporary CB-OHS is rather impossible. The latter characterizes the situation of developers that get in touch with CB-OHS for the first time. In such cases, they do not know what kind of structure servers exists, and more important, how they may deploy them. The act of attempting to locate available structure servers without prior knowledge of their existence or their name is referred to as *discovering of structure servers/components*. These tools can take the form of browsers that examine the available structure servers within a CB-OHS and report on their properties or of APIs that allow the integration of discovery mechanisms into third party applications. Irrespective of the form these tools may take, special protocols are needed to facilitate the discovery of structure servers. By the term “properties of a structure server”, we denote all these characteristics that distinguish one structure server from the other. These include: the name of the domain a structure server is servicing, the protocol the structure server is using to receive requests and in general to communicate, its availability, etc. Maintaining these characteristics within a special repository within the infrastructure and selecting an appropriate representation mechanism, would also ease the development of CB-OHS clients (as well as the integration of third party applications), since client-side code could be generated automatically. For example, using WebServices and in particular WSDL to represent the provided services, would allow the automatic generation of client side protocol stubs. Going with these thoughts even further, this approach would also allow the runtime binding of protocol classes into clients, as it is the case on the World Wide Web.

5. Related work

The questions of how to support development tasks in structural computing have already been the concern of contemporary CB-OHSs. In this section a number of existing structural computing environments are briefly presented emphasizing the supported development tools.

The *Themis* structural computing environment [1] consists of a framework interface, a generic structure server and two extension subsystems for the definition of structure templates and structure transformations, respectively. Application developers can extend the generic structure server by defining templates for the needed domain-specific structures. The template subsystem provides also instantiation operations for structures, which can be manipulated through the framework interface. The primary structure abstraction is the *Element* abstract class, associated with an open set of attribute-value pairs. The value of an attribute may be another element instant. Element instances belong to either of two subclasses, *Atom* and *Collection*, where the second subclass aims to group together other elements. These simple conceptual constructs can be combined to support a variety of domain-specific structures, both tree-based and non-hierarchical. The transformations subsystem provides supporting operations that automatically transform structure instances from one template to another. This functionality is delivered through custom-developed plug-ins that are loaded on demand into the transformation subsystem. Themis utilization in real application development projects has resulted in reducing the amount of code required, along with raising the level of abstraction such that it is easier to understand and maintained.

The *Construct* structural computing environment [26, 27] is designed to support the development and hosting of an open set of structure and infrastructure services. The development process consists of using a UML tool to specify the classes (in terms of both state and behaviour) that make up a new hypermedia service. The derived diagram is automatically transformed to an IDL specification, which in turn delivered to the *Construct Service Compiler* (CSC). CSC produces a set of files, including an XML DTD, a skeleton service, and a set of common service behaviours. The skeleton service consists of a set of classes specifying a set of methods having their bodies (semantic parts) empty. The developer has to fill the missing code and load the service to the Construct environment. Generated services are available not only to third-party clients, but also to the development environment itself. The current set of services includes navigational,

metadata, taxonomic, spatial, cooperation, and data mining services, while some of them are also provided on the web [24].

The *Callimachus* CB-OHS [21] provides the framework in which various hypermedia domains can co-exist. Motivated by the complexity of designing and implementing structure servers, it supports a methodology and tools to facilitate these tasks. A formal XML-based syntax for the definition of domain-specific structure abstractions and their interrelationships is offered, setting up *structure templates*. *Structure types* are the building blocks for templates, while each structure type is composed of a unique id, a name, and an arbitrary number of *properties* and *endsets*. Instantiation of structure types produces *structure objects*, comprising the basic first-class entities manipulated by the system. Properties take one or more values of a specific data type, while endsets are placeholders for structure objects' ids. A set of core operations has been implemented, enabling the creation, modification, deletion, loading, saving and querying of structure objects. The methodology of developing structure server within Callimachus is divided into two phases: the *structural design phase* and the *behavioural design phase*. During the former phase, the developer defines the structure template, while during the later he/she employs the set of core operations in order to build the computational aspects of the domain.

As it is evident, contemporary CB-OHS have already attempted to approach and solve development issues within their systems. Nevertheless, the approaches of current CB-OHS focus only on specific tasks - e.g. formal structure specification, building structure servers - without providing an integrated and complete development environment.

6. Conclusions – Future Work

In this paper we have attempted to approach some aspects of CB-OHS from an engineering point of view, since despite their advantages, they are not widely accepted by the development community. We believe that this is partly due to the lack of an engineering framework and a set of development tools, which makes the use of CB-OHS cumbersome and difficult. Without proper development tools, developers hesitate to incorporate CB-OHS into their daily tasks. We argue that development issues should be explicitly addressed within CB-OHS and reflected by the infrastructure.

Based on well established software engineering practices and on the experience gained from previous works, we have presented a development process centralized to emphasize critical characteristics of CB-OHS, rather than exploiting specific technologies. Towards establishing proper development tools, we have identified some areas where such tools are mostly needed, including the theoretical as well as the practical aspects of structural computing. With regard to the theoretical aspects, new tools to analyze problems and evaluate the conceptual structure models are required, based on the notion of structural completeness and decidability. Regarding practical aspects, environments are needed that are able to manipulate domain specifications, including both structure syntax and behavior. Moreover, given the polymorphic nature of CB-OHS with respect to supported domains, contemporary approaches in locating structure servers seem inappropriate. New methods of discovering appropriate structure servers are necessary, directed to provide a sound foundation to enable Peer-to-Peer hypermedia.

Our current and future research plans are mainly focused on the design and implementation of the aforementioned development tools. Especially, we are working on the subject of behavior specification and the establishment of a framework for the propagation of operations through structure entities [23]. We believe that formal methods for the definition of structure semantics are a prerequisite to address the structural completeness and decidability issues.

References

1. Anderson, K. M., Sherba, S. A., Lepthien, W. V., Structural Templates and Transformations: The Themis Structural Computing Environment, *Journal of Network and Computer Applications*, 26(1), January 2003, pp. 47–71.

2. Anderson, K. M., *Software Engineering Requirements for Structural Computing*, in Proceedings of the 1st Int'l Workshop on Structural Computing (SC1, Darmstadt, Germany), Technical Report AUE-CS-99-04, Aalborg University Esbjerg, Computer Science Department, Denmark, 1999, pp. 22–26.
3. Beringer, D., Melloul, L., Wiederhold, G., *A Reuse and Composition Protocol for Services*, in Proceedings of Symposium on Software Reusability (SSR'99, Los Angeles, California, USA), 1999, pp. 54–61.
4. Christodoulou, S., Zafiris, P., and Papatheodorou, T. S.: *Web Engineering: The Developers' View and a Practitioner's Approach*, Web Engineering, Software Engineering and Web Application Development, Springer-Verlag LNCS 2016, 2001, p.170-187.
5. Engelbart, D., *Keynote speech*, 4th Int'l Workshop on Open Hypermedia Systems (OHS4, Pittsburg, PA, USA), 1998.
6. Garzotto, F., Paolini, P., Schwabe, D., *HDM— A Model-Based Approach to Hypertext Application Design*, *ACM Transactions on Information Systems*, 11(1), 1993, pp. 1–26.
7. Ginige, A., Murugesan, S., *Web Engineering: An Introduction*, *IEEE MultiMedia*, 8(1), Jan.–Mar. 2001, pp. 14–18.
8. Isakowitz, T., Stohr, E. A., Balasubramanian, P., *RMM: A Methodology for Structured Hypermedia Design*, *Communications of the ACM*, 38(8), 1995, pp. 34–44.
9. Karousos, N., Pandis, I., *Developer Support in Open Hypermedia Systems: Towards a Hypermedia Service Discovery Mechanism*, in Proceedings of the 2nd Int'l Metainformatics Symposium (MIS'03, Graz, Austria, September 2003), Springer-Verlag LNCS 2994, 2004, pp. 89–99.
10. Karousos, N., Pandis, I., Reich, S., Tzagarakis, M., *Offering Open Hypermedia Services to the WWW: A Step-by-Step Approach for Developers*, in Proceedings of 12th Int'l World Wide Web Conference (WWW 2003, Budapest, Hungary), 2003, pp. 482–489.
11. Lowe, D., Hall, W., *Hypermedia and the Web: An Engineering Approach*, Wiley, 1999.
12. Millard, D. E., Moreau, L., Davis, H. C., Reich, S., *FOHM: A Fundamental Open Hypertext Model for Investigating Interoperability between Hypertext Domains*, in Proceedings of 11th ACM Int'l Conference on Hypertext and Hypermedia (Hypertext '00, San Antonio, Texas, USA), 2000, pp. 93–102.
13. Nürnberg, P. J., Leggett, J. J., *A Vision for Open Hypermedia Systems*, *Journal of Digital Information (JoDI)*, Special issue on Open Hypermedia Systems, 1(2), November 1997.
14. Nürnberg, P. J., Schraefel, M. C., *Relationships among Structural Computing and Other Fields*, *Journal of Network and Computer Applications*, 26(1), January 2003, pp. 11–26.
15. Nürnberg, P., Wiil, U. K., Hicks, D. L., *A Grand Unified Theory for Structural Computing*, in Proceedings of the 2nd Int'l Metainformatics Symposium (MIS'03, Graz, Austria, September 2003), Springer-Verlag LNCS 2994, 2004, pp. 1–16.
16. Open Hypermedia Systems Working Group (OHSWG), <http://www.csdl.tamu.edu/ohs/>
17. Papazoglou, M. P., Georgakopoulos, D. (eds.), *Service-oriented Computing*, *Communications of the ACM*, 46(10), October 2003.
18. Pfleeger, S. L., *Software Engineering: Theory and Practice*, Prentice Hall, 2001.
19. Pressman, R. S., *Software Engineering - A Practitioner's Approach*, McGraw – Hill, Fourth Edition, 1997.
20. Schwabe, D., Rossi, G., Barbosa, S. D. J., *Systematic Hypermedia Application Design with OOHDM*, in Proceedings of 7th ACM Int'l Conference on Hypertext (Hypertext '96, Bethesda, Maryland, USA), 1996, pp. 116–128.
21. Tzagarakis, M., Avramidis, D., Kyriakopoulou, M., Schraefel, M., Vaitis, M., Christodoulakis, D., *Structuring Primitives in the Callimachus Component-Based Open Hypermedia System*, *Journal of Network and Computer Applications*, 26(1), January 2003, pp. 139–162.
22. Tzagarakis, M., Karousos, N., Christodoulakis, D. and Reich, S.: *Naming as a fundamental concept of open hypermedia systems*, in Proceedings of 11th ACM Int'l Conference on Hypertext and Hypermedia (Hypertext '00, San Antonio, Texas, USA), 2000, p.103-112.
23. Vaitis, M., Tzagarakis, M., Grivas, K., and Chrysochoos, E.: *Some Notes on Behavior in Structural Computing*, in Proceedings of the 2nd Int'l Metainformatics Symposium (MIS'03, Graz, Austria, September 2003), Springer-Verlag LNCS 2994, 2004, pp. 143–149.
24. Wiil, U. K., Hicks, D. L., *Providing Structural Computing Services on the World Wide Web*, in Proceedings of the 3rd Int'l Workshop on Structural Computing (SC3, Aarhus, Denmark, August 2001), Springer Verlag LNCS 2266, 2002, pp. 160–171.
25. Wiil, U. K., *Multiple Open Services in a Structural Computing Environment*, in Proceedings of the 1st Int'l Workshop on Structural Computing (SC1, Darmstadt, Germany), Technical Report AUE-CS-99-04, Aalborg University Esbjerg, Computer Science Department, Denmark, 1999, pp. 34–39.
26. Wiil, U. K., Nürnberg, P. J., Hicks, D. L., Reich, S., *A Development Environment for Building Component-Based Open Hypermedia Systems*, in Proceedings of 11th ACM Int'l Conference on Hypertext and Hypermedia (Hypertext '00, San Antonio, Texas, USA), 2000, pp. 266–267.
27. Wiil, U. K., *Using the Construct Development Environment to Generate a File-Based Hypermedia Storage Service*, in Proceedings of the 2nd Int'l Workshop on Structural Computing (SC2, San Antonio, Texas, USA), Springer Verlag LNCS 1903, 2000, pp. 147–159.