HYPER-LINKED SOFTWARE ARCHITECTURES FOR CONCURRENT ENGINEERING[1]

Juan C. Dueñas
ETSI Telecomunicación
Universidad Politécnica de Madrid,
Ciudad Universitaria, sn, E-28045 Madrid, Spain
E-mail: jcduenas@dit.upm.es

Manfred Hauswirth
Distributed Systems Group
Technical University of Vienna
Argentinierstr. 8, Vienna, Austria
E-mail: m.hauswirth@infosys.tuwien.ac.at

KEYWORDS

Software architectures, hyper-linked information systems, concurrent engineering, World Wide Web.

ABSTRACT

In this paper, the idea of considering software architecture (SA) as a tool for the gradual introduction of Concurrent Engineering (CE) issues in the industrial area is explored. SAs are placed among the first steps in the development of a large software system. Using them following the principles of CE implies some requirements on the supporting infrastructure, that can be met by hyper-linked information systems. The quickly growing World Wide Web (WWW) system fits all the requirements expressed, and makes a realistic choice for a gradual introduction of SA in production environments. It also serves as an integration framework for tools dealing with SA reasoning. For these reasons, it can be used to provide services oriented towards helping designers in the discovery, communication, and evaluation of software designs in an incremental way.

SOFTWARE ARCHITECTURES

Software architecture (SA) has been recently described as "a communication technology for the development of software systems, instead of a software engineering technology" (Brown 1996). We can find almost as many SA definitions as researchers are, but almost all of them coincide to include in the definition both models of the system being built, as well as constraints or guidelines for design (Shaw and Garlan 1996). This coincidence reflects a common understanding about the final users of SAs: the people developing the system. To be more specific, groups of people working on the developing system. The main uses of the SA can further clarify its actual objective as a communication technology.

SAs are used for **teaching** newcomers the basics of systems being built. In this case, the main advantage of using a SA is that it gives an abstract overview of the system, helping developers to find their working context (this is especially needed when “programming in the large”). It is also worth noting that SA has been proposed by some authors as the foundation of Software Engineering teaching, in the form of paradigmatic solutions for well-known problems, using architectural styles (Buschmann et al. 1996).

In the initial phases of development, SA helps to **understand** the system requirements and guide the discussion on features and resources required. In fact, the initial design decisions are proven to be crucial for the future system quality, and they must be taken with the agreement of different stakeholders (perhaps with different backgrounds). Then, SA acts as a "common ground" to all of them, facilitating the discussion in terms easily understood by all of the group.

The **assessment** (on objective or subjective basis) of the SA with respect to quality requirements (mainly concentrated on non-functional ones) is an important use, often done by a group of experts working on the same model. For this matter, we propose to link all the information regarding the objective or subjective decision making process to the SA models.

SA can be used as a **reengineering** vehicle for the maintenance and improvement of the current systems. It is a recognized fact that some of the main efforts towards SA research are driven by large companies with long life span systems that would need to be improved, and the very first step towards that is obtaining the "conceptual integrity" (Brooks 1996) of these very large systems. It would also be of primary importance to obtain the design trajectory for the decisions taken on the system during its life cycle. In this aspect, the usage of reengineered models to discover structural or behavioral patterns (Mendoza and Kramer 1997) could lead to obtain the “hidden” architecture. Unfortunately, there is no evidence towards a possibility of solving this automatically at this moment, so the only choice is human inspection (Eixelsberger et al. 1997). Again, the documentation nature of SA appears.

The model portion of the SA is valid in the development process to assign work loads or to define subsystems to be developed by different work groups. But the guidelines included in the SA are even more important for the **controllable and repeatable design** and implementation of the system. Initially, these guidelines can be handled as a "Frequently Asked Questions" list, or as a company’s style book. Our proposal is to give access to this information in a context-sensitive way, and offering several views depending on its intended use.

The SA evolves in time, acting as a **design trace** on the development of systems of related domains. Initially, the SA composed by the Architecture Description Language (ADL) models (Carnegie Mellon University 1997b) and the guidelines should be enriched adding information about previous systems developed. Then, the SA should be

[1] This work was partially funded by ASIA (Austrian Spanish Integrated Action) “Open Distributed Processing for Remote Collaboration.” and by the ARES (Architectural Reasoning for Embedded Software) project. ARES is supported by the European Commission under ESPRIT framework IV contract no. 20477 and is pursued by Nokia RC Finland, ABB Norway, Philips RC Holland, Imperial College, Universidad Politécnica de Madrid, and Technical University of Vienna.

modified accordingly to the incremental evolution of the systems; and, as soon as major improvements are added to the development process, the set of guidelines must change to accomplish them. The SA can be searched for previous design decisions or to reuse parts of the architecture.

HYPER-LINKED DOCUMENTATION SYSTEMS

A hyper-linked documentation system (Conklin 1987) is such that the contents are not arranged in an sequential way but they are organized in several chunks of information and links between them, so that the overall organization is determined by these links. No restrictions are imposed on the information content. Thus a hyper-linked documentation system (HLDS) breaks up the linear documentation paradigm of conventional paper or electronical documentation. Though linearity is still possible this is replaced by a navigation paradigm that allows placing related information "at hand", i.e. close to referrals. E.g. hyper-links can be added (attributed) to flow text, (regions of) pictures, etc., to facilitate references, i.e. related information, to other chunks of information that can be "visited" by simple mouse clicks or keyboard commands (Fig. 1). Hyper-links add a high degree of flexibility to a documentation by allowing numerous tracks for traversing the documentation depending on the reader. From the SA perspective it would mean that each role has different path through the hyper-linked documents, depending on his/her interests.

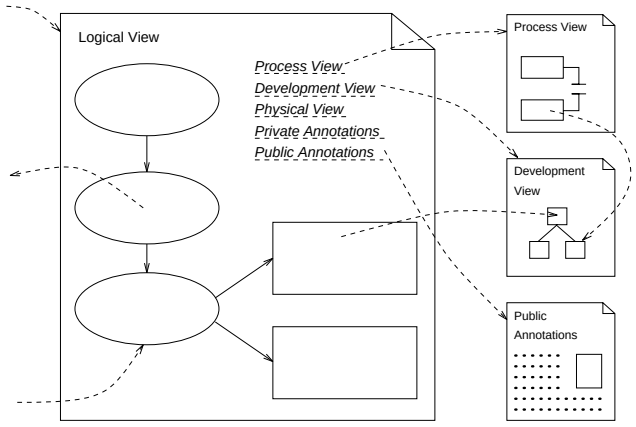

Figure 1: Hyper-linked documentation

Moreover hyper-linked documentation can gain an even higher degree of versatility by separating base information and navigation information, i.e. documents and navigation links are generated dynamically, e.g. from a database, dependent on user's configuration and/or navigation history. Thus the delivered document is both dynamic in its contents and navigation infrastructure included therein. Again this flexibility allows that, starting from the same basic chunks of information several sets or "views" can be obtained.

In its application to SA representation hyper-linked information systems are especially suitable because of several reasons.

Information model. In SAs, information comes from two sources: models and guidelines. Models usually contain the components, the connectors and the information required to define their interactions, sometimes in a graphical way. The syntax of the models depends on the architectural description language (ADL) used. On the other hand, guidelines for design decisions encapsulate the "know-how" about the SA and design. Since this last source of information is more general, there is no easy way to represent it, other than natural language. This kind of information is sometimes structured in lists of questions and answers (the "Frequently Asked Questions").

Both pieces of information must be linked. It is then possible to know the guidelines applied in the development of each piece of the architecture, or–given a certain module–to obtain which possibilities exist for further improvement, based upon the application of the guidelines. This is the main missing point when traditional approaches are used in the documentation of SAs: the lack of hyper-links makes it difficult to navigate among the several useful chunks of information that build up the SA.

Additional sources of information, as records of design decisions, rationales, design trajectories, or timing information, can be included in the SA description by the same means. Even more, the existence of hyper-links allows to define several views on the information stored, depending on its purposes, the role of the viewer, etc.

CE support. The creation and usage of SAs are group activities in which different partners co-operate to reach balanced solutions that face both the quality and functional requirements. At the very beginning, there is only one group working towards the first model and set of guidelines; after that, and once the abstract SA has been created, the refinements to each of the modules and connections in the SA can be done in parallel by different work groups operating on subsets of the whole model. HLDSs can support this, since there is not a single block of information over which all the work is done, but the information is distributed in small chunks that can be modified and updated in parallel. This can be done in a distributed way among the participants.

As always when different groups are working on models, not all the parts of models advance or mature at the same speed. Thus, the degrees of abstraction vary in the whole model, and heterogeneity appears. In the case of SAs there are other additional sources for heterogeneity: the different nature of ADL models and guidelines, and the possibility of using the SA for different purposes implies that several heterogeneous views of the system must be offered. Regarding this, it is important to note that the basic mechanisms of hyper-linked systems are able to handle heterogeneity, provided that models or information can be enriched with the links. Information nodes can hold information with very different nature.

As a subsequent requirement, the SA support must be available throughout all the life cycle, from its inception to the maintenance phase. This situation is dealt with in the same way like heterogeneity, ensuring that the same underlying supporting system is able to be used for any kind of model, regardless of its maturity.

When tool support is provided such that the links between information nodes can be distributed geographically, these nodes can be physically close to their information sources,

but the distribution is hidden. In fact, what appears is a virtual workspace shared by all the SA creators and users, regardless of their locations. This fact is very convenient if different stakeholders or different experts will participate in the creation, assessment or usage of the SA, perhaps from different business units in the company.

FRAMEWORK FOR A CONCURRENT ENGINEERING SYSTEM

The World Wide Web (WWW) framework (Berners-Lee et al. 1994) is a well established hypertext system that can be used to implement a HLDS for CE support. It does not impose constraints on the contents it holds and is inherently distributed, and can easily be adapted for SA representation and management. Figure 2 shows the overall structure of a HLDS for SAs. We want to point out that this design does not necessarily need WWW or a WWW client as depicted in Figure 2. It can be used with any system that offers similar hypertext facilities like WWW. In this framework the WWW client only represents the visualization aspects of the system. Nevertheless, we will stick to WWW in the following since it is the most widespread used and understood system nowadays that offers the highest degree of support, on nearly all software platforms.

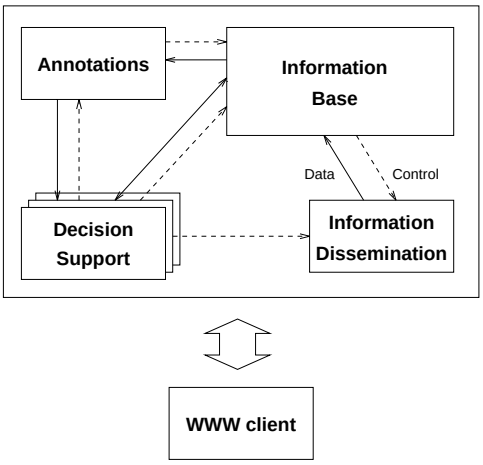

Figure 2: Basic system structure

The following table gives an overview of the characteristics of the subsystems:

	Annotation	Information Base	Decision Support	Information Dissemination
Functionality	locally focussed links / information	presentation, navigation	integrate new information and annotated information into information base	report changes and keep developers and architects up-to-date
Data	informal PostIt data	committed design information	dependent on decision technique	changes to the information base
Format	free	HTML, pictures, VRML, multi-media	dependent on decision technique	e-mail, HTML, Java objects
Tools	Java	databases, file system	Quality Function Deployment	e-mail tools, Castanet, Pointcast, etc.

Information Base

The information base subsystem (IBS) is a central part of the system whose functionality is :

- to generate the data related to the SA, retrieving it from a development database, extracting from metric extractors, etc.,
- to store, handle, and provide navigation information, in order to allow the users to handle the information efficiently,
- and to pre-process the information in a consistent format that hides the heterogeneity of internal data.

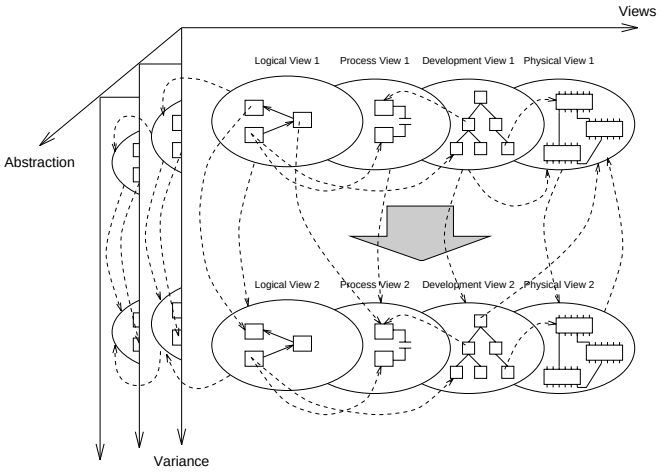

Figure 3: Logical Information Base Design

With respect to the SA data maintained under the control of the IBS, it can be seen from different axis (see Fig. 3):

1. The architectural view (Kruchten 1995), there are several views to see the high-level design of the system. Depending on the special feature we are interested in: the logical view provides the functional design, containing the main abstractions necessary to fulfill the functional requirements ; the development view focuses on the organization of the actual software modules and it is thus closely related to the way the organization is structured; the process view focuses on the runtime behavior of the system, and the physical view shows the mapping of software onto hardware.
2. The abstraction axis. Each one of the mentioned architectural views is formed by hierarchical nodes and arcs, able to be decomposed as the development evolves. Thus, the first views give only a hint of the main elements in the system. As it evolves, more refinements are added, giving more details about nodes and arcs. These refinements of the views define the abstraction axis.
3. The variance axis. SAs are blueprints for families of products. There is no point in producing a SA, unless several related products are done, and they share large parts. The variance axis shows how the system once developed evolves to fit several products, the differences between them, and the changes required to adapt to several customers or variances.

Several exploration paths can be followed within this structure, depending on the intended use and stake-holders using the architectural information; e.g.:

- exploring one architectural view in the abstraction axis will give the designer the complete and detailed description of how the system is built from that view, allowing the assessment of certain quality attributes,
- comparing the same views at the same abstraction level but with different variance, it is possible to get an insight of more changed parts in the system and foresee future directions of changes.

There is no agreement in the software engineering community to represent architectural data yet. The proposed hyper-linked systems are open enough to handle any kind of data. At the moment, several experiences have been reported in the literature about using hyper-linked systems to represent SAs (Carnegie Mellon University 1997a), there are additional standardization efforts (Electronic Industries Association 1995) on the common usage of engineering data, but they have not reached widespread use in the software engineering community.

Annotations

A common situation when applying CE and having distributed teams involved is the need for fast and easy communication in an informal way. This can be done via telephone, fax, email, newsgroups, etc., but this is disjoint from the SA system itself and requires the users to "switch context." A better, more integrated approach is the use of annotations, that are attached to information of the IBS for limited (private, group) or unlimited use. The concept of annotations can be compared to the use of PostIt™ notes in real life: an engineer can attach a formal or informal annotation to (part of) a hyper-linked document to state comments, questions, remarks, etc. to the document and define access rights to it.

Thus the document is enriched in an "unofficial" way. Later if the same user or others retrieve the annotated document the annotations will show up again depending on the access rights.

Regardless of the concrete realization the advantage of annotations are: persistency of annotated information, informal and focused communication, user-defined strategies for changing the status of annotations, i.e. how and if annotations are invalidated or converted into IBS information. Also, in connection with navigation information, annotations can serve as a means for easier understandable questions on the contents of the IBS.

Decision Support

The decision making process appears when there are several choices to meet an architectural requirement : in producing the software architectural models, when these models have to be updated or when a decision on new products in the family must be taken. For this purpose, all the available information on the product (current architecture) and the process (the design steps followed to achieve that product) must be made available. As a result of the decision making process new information is inserted into the information base.

Decision making is an area where automatisation is hard to achieve. The most promising approach seems to be the Computer Supported Co-operative Work (CSCW) techniques, e.g. (Coleman and Khanna 1995), to support the decision making process, because eventually, group of humans will take the decisions. This is specially clear in the SA area, where the plain usage of metrics on models has proved to be ineffective.

Two examples of common decision making techniques are:

- The design space technique (Lane 1990), based on Quality Function Deployment (QFD). When a SA is to be modified, several hypothesis are created and all the stake-holders consider the requirements, the technical possibilities, costs and risks and formulate their interests on several matrices. Giving weight to each considered factor, it is then possible to achieve and agreement on the technical features that the system must include. The decision support subsystem provides allows the users to create such matrixes, and to communicate and collect all the data from the decision group.
- The design decision trees technique (Ran and Kuusela 1996): for a lower level in the design, when the current design decisions can be codified, and the order in which they must be applied is fixed, a decisions tree can be built and explored. Each decision contains information about when and how to apply, the way it changes the SA, who can apply it, et cetera. All this information is stored in the information base, and thus navigation through the hyper-linked documentation corresponds to the fact of applying rules; each link traversed is a request for changing the SA model.

Information Dissemination

This is a key topic in the area of CE: how to keep participants up-to-date with project status. Part of the solution has already been sketched in the section on annotations. But a more general solution is needed here. Techniques ranging from simple e-mail notifications to sophisticated distribution and CSCW methods can be employed here. Regardless of the methods used, the important issue again is the aspect of integration into the SA system itself. The use of open technologies greatly eases integration problems.

Besides this the Information Dissemination Subsystem (IDS) must be flexible enough to implement user-defined dissemination processes. After the SA system's user has defined a process for how to create documents aimed for distribution and/or insertion into the IBS, the IDS offer support for distribution of these documents. Functionalities the IDS must offer include: definition of dissemination priorities, consistency constraints, subscription facilities ("info-line"), automatic and user-initiated notification, automatic creation of meaningful notification messages, etc.

The implementation of these functionalities can already be based on highly-functional base technologies. Their

usability, however, still strongly depends on the skills and experience of the implementor.

SOFTWARE ARCHITECTURE AND HYPER-LINKED DOCUMENTATION SYSTEMS

In this section we discuss issues how HLDSs may be used to support development, representation, and evolution of SAs. The current situation in industry is that a SA is often developed once a product or set of products are successful and their life span is foreseen to last for several years. The major activities in developing and evolving a SA are:

1. In case a SA is to be developed for an existing system, reengineering of the current product or set of products, starting with all the available information (design diagrams, code, design rationale, documentation, interviews with architects, etc.) has to be performed. Regarding the framework, this first step corresponds to the creation of the information base. Although this process involves several roles and people, it is usually performed by a small group of "re-engineers" (Gall et al. 1996).
2. Making decisions on the system. Once the architecture is well defined, the validation process starts (although it can be seen as an iterative cycle). Then the possibility to refine or fix the architecture with respect to the current systems exists ; additionally new components of the product family can be developed, forcing the SA to be adapted. For all these activities, software architects make decisions : comparisons between several candidate architectures, comparisons between several options to refine a given architecture, etc. The decision making subsystem helps the developers to carry out the decision.
3. Updating and disseminating changes. Given changes or updates to the architecture must be noticed and understood by all the roles involved in the architecture life cycle, specially by their users. With respect to the visibility of changes, there are two steps supported in the framework:
    - unofficial changes or annotations. While in the decision making process, architects propose changes that must be assessed. Since not proven suitable yet, they are at the draft level. Annotations are the way to support this step. These annotations have reduced visibility (even personal), and allow the group of architects to follow a certain discussion.
    - official (messaging service). When the changes are actually performed because the assessment was positive, a new architecture appears. The messaging service is able to tell the architecture users those regions updated in the architecture.
4. Groupware. It has been mentioned that SA is not plainly the models used, but also the rationale or the rules that guide the design. It is really important to ensure that communication channels are ready to carry the architects all the questions, advises and doubts from users. Effective application of SA must rely on groupware technologies, as those mentioned in this section, and support the interaction between architects and users, perhaps by the use of multimedia communications.

WWW TOOLS

As stated above, the development process for models and systems usually is not performed in a sequential, linear way, but it often shows high degrees of distribution and parallelism which frequently change during the evolution of a system. This may either be due to changing needs at different stages of the maturation process of a system or be caused by new design decisions or even by partial redesigns. Thus a tool for documenting and supporting SA has to be flexible in adopting to such changing demands. The main facilities that WWW offers to SA creators and users are:

Openness

WWW is an open, highly dynamic domain. Due to its public acceptance and support many tools can be integrated. On the other hand many software companies already offer WWW support for their products since this has become a major advantage in competition.

Availability

Other interesting characteristics of WWW are its availability on all platforms and its low price, which make it suitable for a gradual introduction of SA issues in development environments. Besides the WWW base technologies lots of software for enhancing WWW coverages in respects like active components, e.g. Java (Arnold and Gosling 1996), JavaScript (Netscape Communications Corporation 1997b), ActiveX (Cluts 1997), and systems built upon them exist and can be exploited. As a by-product this infrastructure can be reused in other projects since WWW is a very portable system to base on.

Generation of Information

As stated above contents provided on the WWW can easily be kept up-to-date by providing dynamically-generated documents. This can be accomplished by gateways to existing systems, e.g. databases, versioning systems, etc., that hyper-text and hyper-link the information contained therein. Also other existing functionalities like evaluation tools, consistency checkers, code generators, decision making support, etc. can be integrated into the WWW in this way.

Thus WWW is used as an integration platform for tools. This approach is used frequently with WWW applications based on common database systems. The generation of documents and connection to WWW is usually based on the Common Gateway Interface (CGI) standard (McCool 1994), advanced technologies like the Java Database Connectivity (JDBC) standard (Sun Microsystems Inc. 1997), or proprietary interfaces supplied by vendors, e.g. Oracle WebServer (Oracle Corporation 1997). These techniques are well understood and used by many Internet sites. Besides

this other emerging WWW technologies like “plug-ins” (Netscape Communications Corporation 1997a) and Java programs can be applied, to enhance the integration process. Refinement of designs is mirrored on the WWW by enriching existing coverages with new links and nodes. So also intermediate design results (trajectories) can be communicated easily among the project partners. Since information gets available early in the SA definition process (pre-ADL) more parties can participate in the exploration of models, discuss intermediate results (annotations, integrated talk service or newsgroups). Annotations can be integrated with existing techniques by using additional hyper-links, annotation windows popping up, etc. Such active components can be implemented with new technologies like Java, JavaScript, or ActiveX, and be based on a rich set of existing code, e.g. Gamelan (EarthWeb Inc. 1997). Annotations and suggestions might also be checked against models.
Traditional decision making tools, to define further project steps, could also be integrated easily.

Presentation of Information

For presentation, WWW clients are able to handle the basic types of information (text, HTML, pictures). Additional extensions (viewers or plug-ins) are required to visualize complex formats (VRML, MM). A more integrated solution that became possible with the advent of dynamic loaders is the possibility of extending the functionality of basic clients on-line, depending on the kind of information received.
Since the central concept of WWW are "hyper-links", i.e. the relations between "objects", which can be defined freely, WWW based information systems offer a high degree of flexibility. E.g., part of a system can be seen as an entity-relationship-diagram by defining relations accordingly and later (or even in parallel by providing different views of the system) be restructured by defining new (rules for) links.
This notion of flexibility is also interesting in other respects. One of the main advantages here is the possibility for all participants in a project to use their favourite tools. For interaction purposes it is of course necessary to define an interchange format or (generic) interfaces that can be used for automatic generation of WWW content. Among the possibilities for presenting designs, it is important to note the possibilities of using complex visualization tools attached to WWW.
Role-defined views of a SA can be defined by using dynamically-generated documents whose hyper-link structure is not defined statically but created according to the needs of users (model-view-controller pattern applied to SA representation on the WWW). A rich set of tools to support presentation on the WWW exists. Besides proprietary software supplied by tool vendors, free software tools for dynamically generating pictures or creating graphs from symbolic descriptions, e.g. (Reina 1997) are available. To enforce consistent layout, both in terms of information and navigation facilities, prepocessing tools like HTML++ (Barta 1995), FrontPage (Microsoft Corp. 1997), etc. can be employed.

Navigation

Flexibility as mentioned before also allows to investigate navigation strategies in order to offer the simplest, configurable set of navigation facilities to support users in finding the information they are looking for. E.g., facilitate navigation between architecture and source code by hyper-linking these two views or providing hypertexted source code in a language sensitive way. Tools for uncoupling information and navigation properties by the use of symbolic link information are available, e.g. HTML++ (Barta 1995).
One navigation strategy especially suited to support the teaching aspect of SA are "guided tours", where inherent knowledge of both the SA and project participants can be made explicit and thus help newcomers to quickly get an overview.
Additionally to a passive view model, active documents can be helpful: users may define preferences, interests and requirements and hand them to the system. Based on this information active documents may perform actions like notification on change, triggering of programs, etc. Here triggering mechanisms of databases may be exploited. Besides standard information dissemination techniques like (MIME enhanced) e-mail and newsgroups, newly available systems for actively disseminating information can be facilitated here. Castanet (Lemay 1997), PointCast (PointCast Inc. 1997) and other tools offer some kind of “broadcast” mechanism where users can subscribe to information channels and get regular information updates automatically. Also CSCW tools like BSCW (Bentley et al. 1997) can be used here and integrated into the SA system framework.
A special aspect of the navigation issue are search engines which support users in discovering information of interest. Many sophisticated search engines, e.g. Harvest (Bowman et al. 1994) for full-text or attributed queries, are available for the WWW which both aid the user in finding information and on the other hand provide facilities for automatically indexing information (and distributing it). Thus it is easy to provide powerful capabilities like distributed full-text search by using a combination of WWW and search engines.

DISCUSSION AND FUTURE WORK

The work proposed is focused towards the development of large systems, i.e. “programming in the large” (Boehm and Scherlis 1992), whose complexity grows exponentially with project size, due to both technical and social factors. In this article we have studied a framework able to support the development and documentation of software architectures as guidelines for the design of such systems, and the use of concurrent engineering and information sharing techniques to incorporate knowledge from different domains.
For that purpose, the use of hyper-linked documentation systems as a support system for the concurrent development of software architectures is discussed. A framework for the organization of tools and the usage of WWW technology is proposed. Amongst the advantages that can be cited for this framework, the main ones are :

- that it allows the provision of new services for the development of complex systems obtaining extended functionality as the integration of existing products,
- this technology is available at low cost; it can solve the "architectural mismatch" that hinders the adaptation of required products in an engineering environment,
- the proposed framework can be incrementally built and incorporated to the development; it is possible to reuse previous investments and experience of companies in the development of SA models and documentation, since its openness fosters the creation of small tools that incorporate previous knowledge to the information base,
- it is a simple and easy way to distribute information and communicate, offering simple and consistent user interfaces for the functionalities described in this article.

Our research will proceed in the framework of several common european projects devoted to the development of software architectures and to the distributed collaboration using WWW technology. In the near future our research will concentrate on the following areas :

- the application of the principles described to other application domains apart from software development,
- the use of knowledge derived from other engineering disciplines that have a longer tradition in the use of CE,
- and the enhancement of the WWW support to CE, especially in the less mature fields as the decision making process and techniques. A set of theoretically well understood techniques exists. Since most of them are intelligible and easy to use implementations suitable for the WWW should appear soon. This especially hold true because powerful implementation platforms including GUI builders already are available; e.g., Java, Java's Abstract Windowing Toolkit, Bongo (Goodman 1997).

REFERENCES

Arnold, K. and J.Gosling. 1996. *The Java Programming Language*. Addison-Wesley, Reading, MA.

Barta, R. 1995. "What the Heck is HTML++ ? Salvation for the Souls of Webmasters." Technical Report TUV-1841-95-06. Distributed Systems Group, Technical University of Vienna, Vienna, Austria. [http://www.infosys.tuwien.ac.at/reports/repository/TUV-1841-95-06.ps]

Bentley, R.; W.Appelt; U.Busbach; E.Hinrichs; D.Kerr; S.Sikkel; J.Trevor; and G.Woetzel. 1997. "Basic Support for Cooperative Work on the World Wide Web". In *Proceedings of the International Journal of Human-Computer Studies: Special issue on Innovative Applications of the World Wide Web*, Academic Press, to appear.

Berners-Lee, T.; R.Cailliau; A.Loutonen; H.F.Nielsen; and A.Secret. 1994. "The World-Wide Web." *Communications of the ACM* 37, no. 8 (Aug.): 76-82.

Boehm B. and W.Scherlis. 1992. "Megaprogramming". In *Proceedings of Software Technology Conference DARPA*. ARPA.

Bowman M; P.B.Danzig; D.R.Hardy; U.Manber; M.F.Schwartz; and D.P.Wessels. 1994. "Harvest: A Scalable, Customizable Discovery and Access System." Technical Report CU-CS-732-94. Department of Computer Science, University of Colorado, Boulder, (Aug.).

Brooks, F. 1996. *The Mythical Man-Month.* Addison-Wesley, Reading, MA.

Brown, A. (editor). 1996. *Component-Based Software Engineering.* IEEE Computer Society Press, Los Alamitos, CA.

Buschmann, F.; R. Meunier; H.Rohnert; P.Sommerlad; and M.Stal. 1996. *Pattern-Oriented Software Architecture - A System of Patterns.* John Wiley and Sons, NY.

Carnegie Mellon University. 1997. *ABLE - Architecture Based Languages and Environments*. [http://www.cs.cmu.edu/afs/cs.cmu.edu/project/able/www/able.html]

Carnegie Mellon University, Software Engineering Institute. 1997. *Architecture Description Languages*. [http://www.sei.cmu.edu/~architecture/adl.html]

Cluts, N.W. 1997. *The Microsoft Active Platform*. Microsoft Corporation. [http://www.microsoft.com/workshop/prog/aplatfrm/platform-f.htm]

Coleman, D. and R.Khanna. 1995. *Groupware : Technology and Applications*. Prentice-Hall, Englewood Cliffs, N.J.

Conklin, J. 1987. "A Survey of Hypertext." Technical report MCC STP-356-86.

EarthWeb Inc. 1997. *Gamelan - the official directory for Java*. [http://www.gamelan.com/]

Eixelsberger, W; L.Warholm; R.Klösch; and H.Gall; "Software Architecture Recovery of Embedded Software." In *Proceedings of the International Conference on Software Engineering 1997* (Boston, MA, May 17-23, 1997), to appear.

Electronic Industries Association 1995. *CASE Data Interchange Format*. [http://www.cdif.org/]

Gall, H; M.Jazayeri; R.Klösch; W.Lugmayr; and G.Trausmuth. 1996. "Architecture Recovery in ARES." In *Proceedings of the Second International Software Architecture Workshop and International Workshop on Multiple Perspectives in Software Development 1996*. SIGSOFT 96, ACM.

Goodman, D. 1997. *Official Marimba Guide to Bongo*. sams.net.

Kruchten, P. 1995. "The 4+1 View Model of Architecture." *IEEE Software 12*, no.6 (Nov.): 42-51.

Lane, T.G. 1990. "Studying Software Architecture through Design Spaces and Rules." Technical report CMU/SEI-90-TR-18. Carnegie Mellon University.

Lemay, L. 1997. *Official Marimba Guide to Castanet*. sams.net.

McCool, R. 1994. *The Common Gateway Interface*. [http://hoohoo.ncsa.uiuc.edu/docs/cgi/overview.html].

Mendoza, N. and J.Kramer. 1997. "Requirements for an Effective Architecture Recovery Framework." In *Proceedings of the Second International Software Architecture Workshop and International Workshop on Multiple Perspectives in Software Development 1996*. SIGSOFT 96, ACM.

Microsoft Corp. 1997. *FrontPage*. [http://www.microsoft.com/frontpage/]

Netscape Communications Corporation. 1997. *Inline Plug-Ins*. [http://home.netscape.com/comprod/products/navigator/version_2.0/plugins/index.html]

Netscape Communications Corporation. 1997. *JavaScript Guide*. [http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/index.html]

Oracle Corporation. 1997. *Oracle WebServer*. [http://www.oracle.com/products/websystem/webserver/index.html]

PointCast Inc. 1997. *What is the PointCast Network?* [http://www.pointcast.com/whatis.html]

Ran, A. and J.Kuusela. 1996. "Design Decision Trees." In *Proceedings of the Eight International Workshop on Software Specification and Design*. Schloss Velen Germany. IEEE Computer Society Press, 1996.

Reina, M. 1997. *The Grapher Package*. Distributed Systems Group, Technical University of Vienna, Vienna, Austria. [http://www.infosys.tuwien.ac.at/~reina/vstl/grapher/grapher.html]

Shaw, M. and D.Garlan. 1996. *Software Architecture - Perspectives on an Emerging Discipline.* Prentice-Hall, Upper Saddle River, N.J.

Sun Microsystems Inc. 1997. *The JDBC(tm) Database Access API*. [http://splash.javasoft.com/jdbc/]

BIOGRAPHY

Juan C. Dueñas received his Ph.D. degree on Telecommunications Engineering with Special Award from the Universidad Politécnica de Madrid (Spain) in 1994, and now works there as an assistant professor. His thesis received also the special Award from the Spanish Telecommunication Engineers Official College. He has participated in several european projects on the application of concurrent engineering to software architectures and development.

Manfred Hauswirth is a Ph.D. student at the Distributed Systems Group at the Technical University of Vienna, Austria. He received his M.Sc. in Computer Science in 1994 from Technical University of Vienna. His current research focus is on WWW, Java, and information acquistition by subscription. His research interests include distributed information systems, distributed information management, resource discovery, and programming languages.