3rd Australasian Workshop on Software and System Architectures, Sydney, Australia, 19-20 Nov, 2000

Directions in Engineering Non-Functional Requirement Compliant Middleware Applications

John Grundy* and Anna Liu**

*Department of Computer Science, University of Auckland
Private Bag 92019, Auckland, New Zealand
Phone: +64-9-3737-599 ext 8761
john-g@cs.auckland.ac.nz

**Software Architectures and Component Technologies Group,
CSIRO Mathematical and Information Sciences,
Locked Bag 17, North Ryde, NSW 1670, Sydney, Australia
Phone: +61 (2) 9325 3237
Anna.Liu@cmis.csiro.au

Abstract

Large Enterprise Distributed Systems (EDSs) are proliferating with many organisations undertaking major software developments in order to meet increased Information Technology usage demands. Many of these stem from the move to providing E-commerce and E-business solutions, the need to integrate information systems from many sources, and the needs of virtual organisations. Key challenges in engineering such systems include the large range non-functional constraints the systems impose. These range from scalability, performance, fault-tolerence and security constraints, to management issues such as the need for developers to organise very complex systems development artefacts. This position paper proposes an approach for better-factoring non-functional requirements into middleware development, and proposals for extending an architecture modelling and analysis tool to support this technique.

Introduction

Many organisations are undertaking large Enterprise Distributed System integration and development projects. This is due to a variety of factors, including the need to provide sophisticated on-line customer services and business-to-business E-commerce services, and the need to better-integrate and extend the various Information Systems organisations require [1, 2]. Interesting new driving factors include the growth of virtual organisations, requiring sophisticated multi-organisation systems integration and co-ordination, and the growth of tele-working, requiring organisation Information Systems facilities be available from remote sites.

EDS applications are comprised of a variety of software systems, or components, ranging from legacy systems to newly-engineering components leveraging the latest design and implementation technologies. Middleware is used to integrate many architectural components. Middleware includes basic technologies to facilitate distributed inter-component communication, such as sockets, RMI, CORBA and DCOM [11, 14]. Various data and control exchange formats exist that are passed by middleware, including EDI protocols, XML and custom protocols [8, 15]. Distributed transaction services ensure multi-system data integrity, examples including MTS, OTS and Enterprise Java Beans transaction managers [14]. Various tools facilitate building systems using such middleware, including BEA Systems's WebLogic and IBM's Websphere. Several EDS integration solutions exist, including Virtuoso, eXcelon and BusinessWare. These various tools and technologies provide a range of support for software developers building or integrating EDS applications.

Many advances have been made in software architecture design for large EDS applications. These include the identification of various common patterns, or styles, in architectures, including various multi-tier solutions [2, 4]. Several architecture design and reasoning methods and languages have been produced, including C2, Wright, Rapide and Darwin

[3, 9, 10]. Many tools have been developed for software architecture design, including basic support such as UML deployment and component diagrams [12, 13], to sophisticated architecture-focused support, such as Clockworks and SoftArch [6, 7]. Middleware components of large EDS software architectures are usually the critical places where the satisfaction (or failure to satisfy) overall non-functional system requirements occurs. The middleware being the plumbing code, are parts of systems where they most affect distributed performance, security, integrity and quality of service characteristics.

Despite the wealth of research and practice applied to EDS middleware and architecture development, many systems are produced that fail to meet key non-functional requirements. One of the most notable being the Atlanta Games web site, where the server simply could not handle the volume of requests coming from the internet community. Often these problems manifest themselves in the facilities provided directly or indirectly by the middleware components of the systems. The root of many of these problems are usually inadequate architectural design decisions, inappropriate deployment of middleware components and tools, or a poor mix of architecture, design and middleware technologies.

We review the common characteristics of complex middleware-based systems, particularly those focused on providing enterprise distributed computing facilities. We propose an approach to engineering such systems that incorporates careful identification of middleware-related non-functional constraints during system architecting, design, implementation, and testing. We present proposals for enhancing a software architecture modelling and analysis tool with information for guiding developers in making better architectural and design decisions, aimed at ensuring a system will meet non-functional constraints on its middleware performance. We briefly summarise conclusions and indicate directions for future research.

Enterprise Distributed Systems

The target of our work is to develop tools and methodologies that will assist initially in the production of specific classes of distributed applications. The initial application domain is transaction/database oriented e-commerce systems built around components and using commodity technologies. The goal is to build tools that will substantially assist users with the tasks involved in building, deploying and maintaining this type of commercially important distributed application.

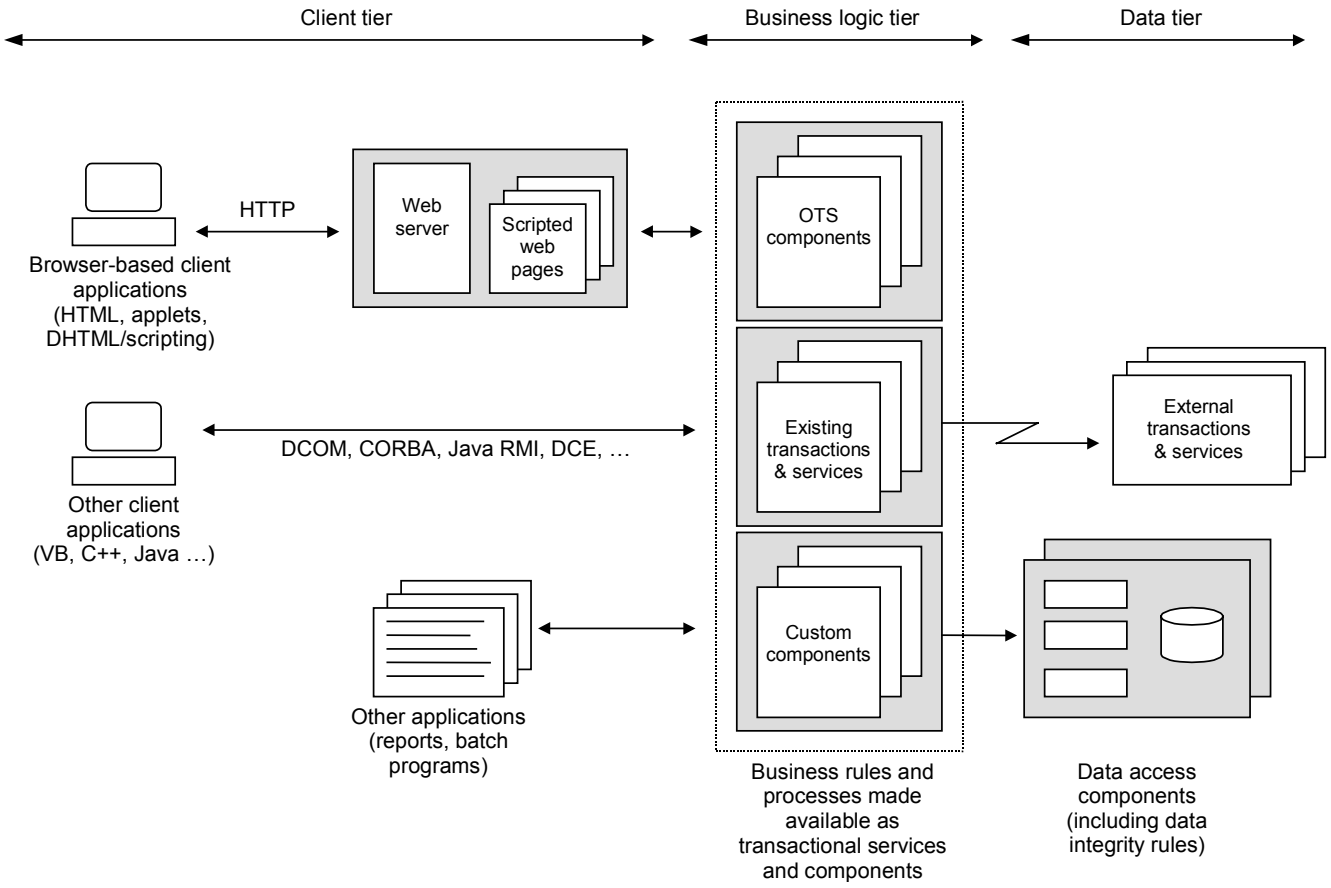

Figure 1. Logical view of 3-tier application.

Error! Reference source not found. shows a logical view of such target EDS architecture. The client tier is concerned solely with presentation and user interfaces, making calls on the middle tier to invoke business processes such as taking orders or accepting payments. The middle tier holds the core business processes and rules for the organisation. These are implemented as a set of components that export services for use by the client layer. The data tier holds the persistent data that represents the state of the organisation, probably kept in one or more databases. Ideally, this layer is responsible for ensuring the validity of its stored data through the use of integrity rules, such as defined value domains and declared foreign keys. These rules may be stored as part of the database definition or may be implemented as code within stored procedures or data access components.

The primary focus of our work is on constructing such a three-tier component-based transactional distributed applications. Users will be able to define the overall architecture of their application as a set of components, linkages and attributes. Some of these components may be purpose-built for the application under development, while others may come from collections of off-the-shelf components or already exist as parts of other applications. Client layers and other external client applications are included in the overall application architectural model, primarily to support change management and performance analysis.

Challenges in Engineering Middleware Solutions

When developing such systems that leverage middleware facilities and or commodity technology, a number of challenges and risks present affecting the ability of a system to meet overall non-functional requirements:

- *Component integration*. Many EDS applications are comprised of disparate software components. These typically include legacy organisational Information Systems, typically newer web-based customer user interfaces and staff

intranets, third party databases, application servers and desktop applications, and components using leading edge technologies for sophisticated data capture or output.

- *Transaction throughput*. As outlined in the previous section, many EDS applications have multiple tiers that must be traversed (i.e. whose software components must communicate) in order to service transaction processing requests. The growth in the number of components in each tier, particularly exacerbated by E-commerce and Business-to-Business integration needs, can greatly increase throughput demands, often well beyond those some components were originally designed to handle.
- *Performance and Quality of Service*. The transaction processing speed of a system is a common measure of performance with constraints that need to be achieved under varying system operating conditions. Other important factors include exception handling ability of various components, potential downtime of components, and time to recover after failure.
- *Security and Integrity*. Different EDS components support security using mixes of authentication, access rights and encryption. Middleware technologies typically provide a mix and many systems adopt custom solutions. Distributed transactions, while slowing system performance using 2-phase commits or optimistic locking and conflict resolution strategies, are nevertheless required to ensure distributed data integrity.

Many of these conflict in terms of the non-functional constraints they attempt to meet. Transactions and security models achieve integrity and information security and privacy needs, at the expense of costly performance delays. Large numbers of client connections and server component replication assist meeting quality of service and performance measures, at the introduction of extra complexity and potentially adverse security, integrity and throughput affects.

Engineering Non-Funtional Requirement Compliant Middleware Solutions

To combat these challenges in engineering middleware solutions, we present the DeBOT methodology.The DeBOT approach is a practical and novel approach that ensures high performance and scalable distributed systems are constructed. The DeBOT approach promotes an iterative and progressive prototyping method, interleaved with the activities of gradually scaled up performance tests, followed by the activities of bottleneck elimination. The end result is a system architecture that exhibits high throughput, fast response time and good scalability.

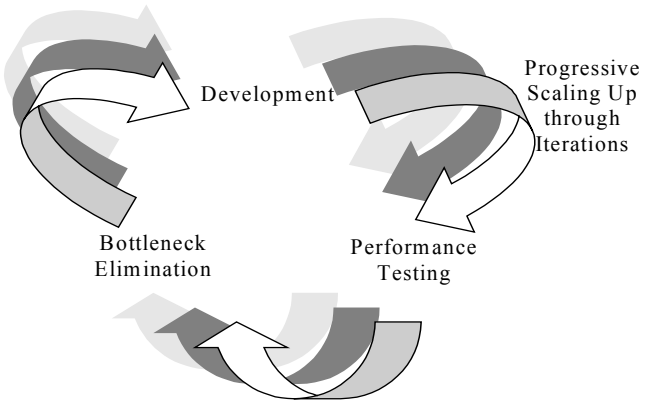

Figure 2. The DeBOT Approach to EDS development.

The largest risks of the system construction are reduced early on by building skeleton code that enables a single transaction to be initiated at client side, processed through the business logic implemented by the application server, through to the resource manager or database. This quickly validates the correctness of the integration of various system components. Early performance tests are also run at this stage to give indicative figures of transaction times. This process is iterated by increasing the number of transaction implementations, coupled with scaling up the performance tests through an increased number of simulated clients.

Modelling, Analysis and Implementation Support

To support this approach we are investigating the extension of a software architecture modelling, analysis and design/code generation tool. This aims at providing a "middleware knowledge base" incorporated into the tool providing software architects and other developers:

- Architectural abstractions characterising middleware facilities, integrated with other architectural and design notations
- An experience base, containing "best practice" combinations of middleware elements and connectors, including non-functional characteristics of these elements. This is drawn upon to statically compare proposed or reverse-engineered architectures against.
- Performance measurement statistics for individual and small aggregates of middleware elements. These are drawn upon to interpolate rough expected performance benchmarks for candidate architectures.
- Tool integration facilities, allowing EDS architecture models to be imported and exported to middleware-oriented development tools, leveraging the architecture modelling tool's problem domain knowledge to assist developers in constructing appropriate implementations and test suites.

The tool we are extending is SoftArch, which provides multiple views of complex architectures from varying levels of abstraction and architecture refinement [7]. We envisage SoftArch allowing developers to rapidly prototype EDS middleware-oriented applications drawing on both best-practice experience and various prototype performance measurements. We also hope it will prove very useful when reverse engineering legacy components and integrating these with other architecture component characterisations.

In SoftArch, an OOA specification (codified functional and non-functional requirements) is imported, typically from a CASE tool. Architects then build high-level architecture(s) for the system that will satisfy these specifications. These high-level models captures the essence of the organisation of the system's software components. It includes information about the non-functional properties of parts of the system, and links architectural components to parts of the OOA specification they are derived from. Architects then refine this high-level model to add more detail, making various architectural design decisions and trade-offs, and ensure the refined architectural models meet constraints imposed by the high-level model. Eventually architects develop OOD-level classes which will be used to realise the architecture, and export these to CASE tools, programming environments and/or middleware-oriented development environments for further refinement and implementation. Alternatively, existing software component models can be imported into SoftArch, typically as low-level design information reverse-engineered by a CASE tool. Abstractions of the low-level components are built up by developers to form multiple levels of refinements of architectures.

Figure 3 shows an example of SoftArch being used to model the architecture of an e-commerce application (a 3-tier collaborative travel itinerary planner). The travel planner system is made up of a set of client applications/applets (shown in view (1) at the top). These communicate via the internet to a set of servers, in this example comprising a chat server, itinerary data manager and RDBMS. View (2) shows a more detailed view of the itinerary management part of this system. This includes the itinerary editor client and its connection to the itinerary management server, a client map visualisation, and a map visualisation agent, which updates the map to show a travel path when the itinerary editor client is updated by the user. Architecture components can be refined by creating a subview containing their refinements, by enclosing their refinements (like for "server apps" in view (1)), or using explicit refinement links. OOA and D-level classes and services can also be modelling in SoftArch, and refined to/from appropriate architecture abstractions. View (3) shows an analysis agent reporting dialogue. A collection of user-controllable analysis agents monitor the state of the architecture model under development. They report inconsistencies, problems or suggested improvements to the user non-obtrusively via this dialogue, are run on-demand by the developer, or act as "constraints" that validate modelling operations as they are performed. SoftArch OOA level abstractions can be sourced from a CASE tool, and OOD-level classes exported to a CASE tool or programming environment (by generating class stubs).

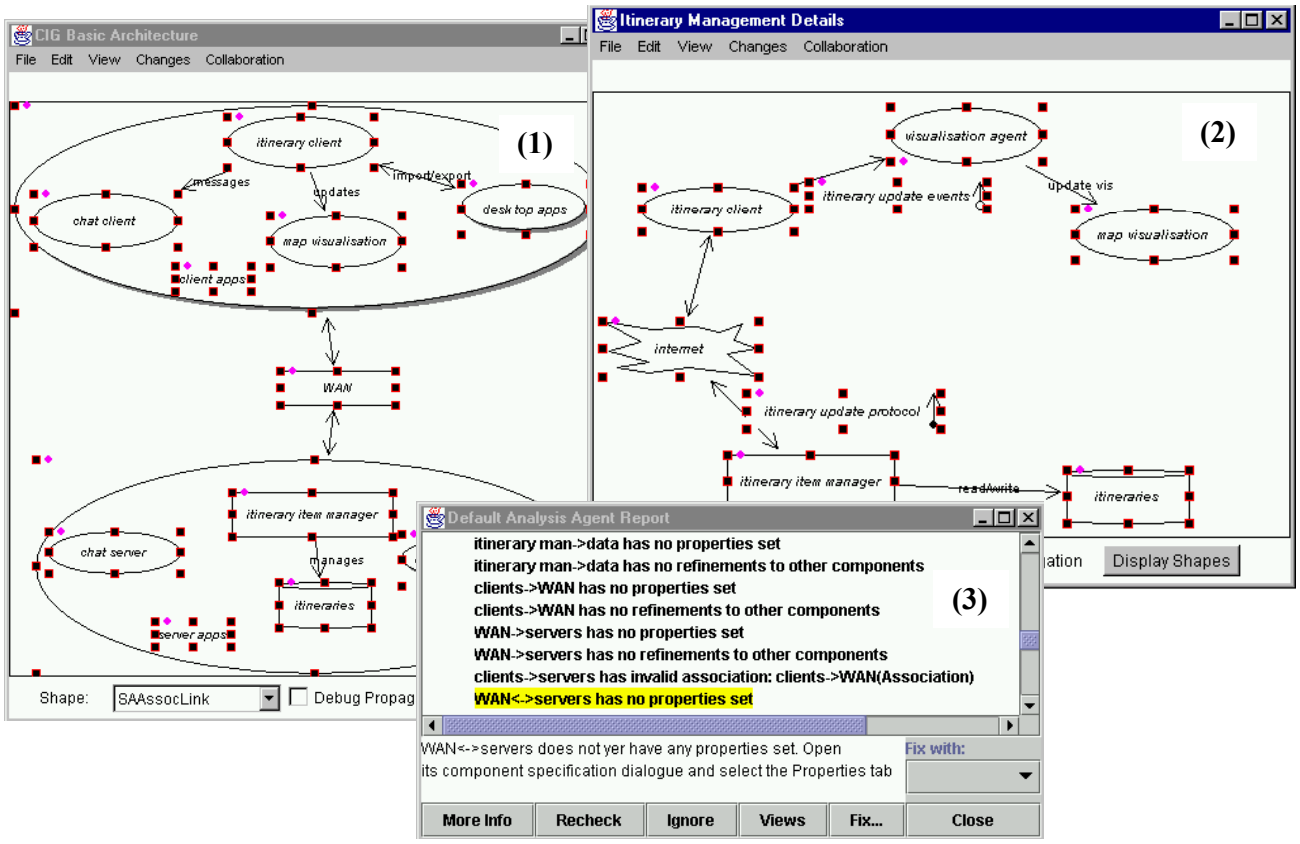

Figure 3. Examples of architecture modelling and analysis in SoftArch.

The path we see the research taking is to use the existing capabilities of SoftArch, initially work on the overall architectural definition and then start annotating non-functional requirements and properties as needed to support refinement. For example, preliminary performance results obtained from initial skeleton code can be used to define resource consumption and run-time behaviour. Using the analysis facility provided by SoftArch, the iterative refinement process through performance analysis can be supported. Further, as the testing is scaled out, performance analysis results from stress testing can be used to feed into the architectural model in SoftArch to assist with identifying potential architectural bottlenecks, which can then be eliminated through further design refinement.

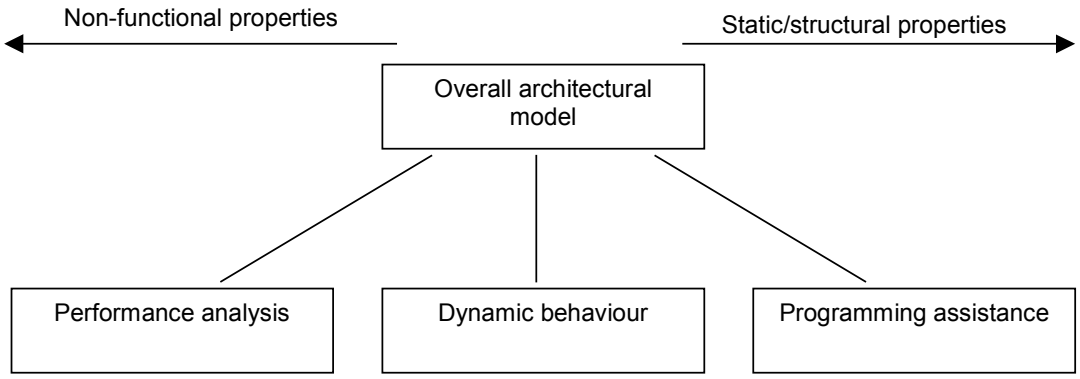

Figure 4. Architectural model refinement.

To enable this we are designing extensions to SoftArch's architecture meta-modelling facility to allow developers to more precisely describe non-functional requirements associated with architecture components and component aggregates. These descriptions typically include information generic to different kinds of architectural components e.g. an http server has a broad range of performance, quality of service, security etc. constraints. Particular instantiations of architectural components will have associated with them more precise non-functional constraints e.g. the Appache web server can serve X number of clients simultaneously, Y number of requests (of some average size) per second etc. Common combinations of architectural components, augmented with actual DeBOT prototype performance measurements, will be managed to allow much more precise estimates of non-functional compliance to be deduced in SoftArch. SoftArch critics using such information will advise developers on likely performance of architecture components under design or review. Feedback from actual DeBOT performance analysis experiments will be continually fed back into SoftArch to enrich the knowledge base it uses to assist developers in defining and refining EDS architecture solutions.

Conclusions

Building complex, distributed enterprise applications is a very challenging task. While middleware technologies and architectural styles and patterns assist developers, actually meeting highly constrained system requirements, particularly non-functional constraints, is still very difficult in general. The DeBOT approach uses iterative specification, prototyping, performance testing and bottleneck removal to successively refine middleware-based application designs and implementations. SoftArch provides high-level architectural modelling and analysis support we are extending to better capture and reason with non-functional architectural properties. A key to this work is supporting architects and middleware-based systems implementers model, analyse and learn from developments, and to optimise EDS applications through appropriate reuse of middleware technologies and architectural styles.

References

1. Alonso G, Fiedler U, Hagen C, Lazcano A, Schuldt H, Weiler N. WISE: business to business e-commerce. Proceedings Ninth International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises. RIDE-VE'99. IEEE Comput. Soc. 1999, pp.132-9. Los Alamitos, CA, USA.
2. Aleksy M, Schader M, Tapper C. Interoperability and interchangeability of middleware components in a three-tier CORBA-environment-state of the art. Proceedings Third International Enterprise Distributed Object Computing. Conference (Cat. No.99EX366). IEEE. 1999, pp.204-13. Piscataway, NJ, USA.
3. Allen, R. and Garlan, D. A formal basis for architectural connection, *ACM Transactions on Software Engineering and Methodology*, July 1997.
4. Bass, L., Clements, P. and Kazman, R. Software Architecture in Practice, Addison-Wesley, 1998.
5. Gorton, I. Et al. Enterprise Middleware Evaluation Reports, Advanced Distributed Systems and technologies group, CSIRO, http://www.cmis.csiro.au/adsat/publications.htm.
6. Graham, T.C.N., Morton, C.A. and Urnes, T. ClockWorks: Visual Programming of Component-Based Software Architectures. *Journal of Visual Languages and Computing*, Academic Press, pp. 175-196, July 1996.
7. Grundy, J. Software Architecture Modelling, Analysis and Implementation with SoftArch, In Proceedings of the 34th Hawaii International Conference on System Sciences, Jan 3-6, Maui, Hawaii, IEEE CS Press.
8. HL7 Organisation, HL7 Standard Interchange Format, www.hl7.org
9. Luckham, D.C., Augustin, L.M., Kenney, J.J., Veera, J., Bryan, D. and Mann, W. Specification and analysis of system architecture using Rapide, IEEE Transactions on Software Engineering, vol. 21, no. 4, April 1995, 336-355.
10. Magee, J., Dulay, N., Eisenbach, S. and Kramer, J. Specifying distributed software architectures. In *Proceedings of the 5th European Software Engineering Conference*, pages 137--153, Barcelona, Spain, September 1995.
11. Mowbray, T.J., Ruh, W.A. Inside Corba: Distributed Object Standards and Applications, Addison-Wesley, 1997.
12. Quatrani, T. Visual Modeling With Rational Rose and Uml, Addison-Wesley, 1998.
13. Robbins, J. Hilbert, D.M. and Redmiles, D.F. Extending design environments to software architecture design, *Automated Software Engineering*, vol. 5, No. 3, July 1998, 261-390.
14. Sessions, R. COM and DCOM: Microsoft's vision for distributed objects, John Wiley & Sons 1998.
15. WC3 Consortium, XML: eXtensible Markup Language format, www.xml.org.