Existing Approaches to Software Integration – and a Challenge for the Future

Rikard Land, Ivica Crnkovic

Mälardalen University, Department of Computer Science and Engineering
PO Box 883, SE-721 23 Västerås, Sweden
+46 21 10 70 35

{rikard.land, ivica.crnkovic}@mdh.se, http://www.idt.mdh.se/{~rld, ~icc}

Abstract

This paper analyzes three fields of practice and research from a software integration perspective: component-based software, standard interfaces and open systems, and Enterprise Application Integration (EAI). The circumstances under which each is applicable are presented, as well as the expected benefits and drawbacks seen from the integrator's perspective. The paper concludes with describing an integration context encountered in practice that challenges the established approaches, and outlines future research from there.

Keywords
Component-Based Software, Enterprise Application Integration, Open Systems, Software Integration, Software Merge, Software Standards.

1. Introduction

The IEEE Standard Glossary of Software Engineering Terminology [34] defines *integration* as "the process of combining software components, hardware components, or both, into an overall system". This paper analyzes approaches to integrating existing software, ranging from relatively small components to large complex systems, as described by literature in different fields.

1.1 Motivation

The motivation for this work is an industrial integration project we have studied earlier, where after a company merger the new company wished to integrate or merge their products by taking the best of each [43]. We had difficulties finding literature on related work on integration in this particular context. This paper is the result of a broad search of approaches to software integration. Our starting point was the qualitative question:

> In what contexts does software reuse and integration occur, and how does this apply to our case?

Additionally, we were interested in the expected benefits and possible drawbacks with existing approaches, as reported by the literature. Thus, at the outset of the literature review, we tried not to let our expectations color our findings, but to as large extent as possible let the literature speak for itself. We do not want to provide a classification framework, but merely report a snapshot of what "software integration" may mean today.

Numerous other surveys of software integration have been published previously [26,31,38,50,71,86,98], but each is done from a particular point of view and the field is large.

1.2 Methodology

Major databases such as IEEE Xplore and ACM Digital Library were searched with keywords like "integration", "interoperability", "reuse", and "merge". We limited ourselves to what we considered most relevant for our purposes and consciously excluded integration as one of the activities during new development when newly developed components are made to work together. Integration of hardware was also excluded. Even so, the amount of remaining texts on integration is enormous. Newer publications were given precedence over older, with the motivation that we wanted to mirror the newest research. Also, by searching in article databases, textbooks are found only indirectly (via references in the papers found). There is therefore a risk that older, seminal references, especially in the form of textbooks are missing.

Surveys and classifications are always subjective to the mind and purposes of the researcher. When organizing the material into the present paper, it appeared to us that most of the literature was presented as belonging to either of three fields of research and practice, in which integration is one, but not the only important challenge: (i) component-based software, (ii) standard interfaces and open systems, and (iii) Enterprise Application Integration (EAI). There were also some texts on integration at low level (e.g. language interoperability), on interfaces and architecture in a fairly general sense, and on information integration. These texts will also be presented briefly.

Some of the characterizations made are not exclusive to a particular field, and there are clearly overlaps between the fields as well. Aware of these limitations, we believe this classification suits the purpose of this paper: to survey existing approaches to software integration in different fields and relate the case study to them.

In section 2, we discuss the basic concepts of interface, architecture, and information in the context of integration. Component-based software is presented in section 3, standard interfaces and open systems in section 4, and Enterprise Application Integration (EAI) in section 5. In section 0, the case study is discussed in the light of these approaches.

2. Basic Concepts

Some general concepts important of integration are here briefly discussed.

2.1 The Interface

Interoperability is the ability for two or more software components to communicate and cooperate with one another despite differences in language, interface, and execution platform [98,99]. To be able to do this, components need to have the same understanding of their *interface*, i.e. the “shared boundary across which information is passed” [34]. The idea of a mutually understood interface is present at all levels of integration, from function signatures to protocols for transactions, and the present paper treats the term “interface” in this wide sense.

2.2 Architecture

When two software systems or components are to be integrated, there is a risk that their understanding of the shared interface is incorrect. For example, there is a problem if two components each assume they control the overall execution of the system and will call other components upon demand. This “architectural mismatch” as it has been called [25] will result in system malfunction, or no possibility to integrate at all. Although a general integration approach cannot mandate any specific architecture (in the sense “structure of components” [6]), one of the main goals of integration approaches is to make components’ assumptions about interfaces match the actual situation.

Designing for future reuse, maintenance, evolution, and integration is difficult since the new contexts and requirements are unknown. Some design patterns [24] and architectural patterns [14,80] address reuse, maintenance and evolution, but there is to the authors’ knowledge only little research on design patterns facilitating integration [40,58,104].

2.3 Information

To be able to integrate systems, the systems’ views of the data – i.e. their *data models*, *taxonomies*, or *ontologies* [27] must be integrated, an undertaking not so trivial [33,43,67,73,87]. Geographic Information Systems (GIS) [13,57] is one significant example of a domain where ontology integration has attracted attention [17,66,84,96].

3. Component-Based Software

The integration context of component-based software is when there are pre-existing software components with clearly defined interfaces available for integration [5,61,82,90,97]. A Commercial-Off-The-Shelf (COTS) component is a commercially available, already existing and available component; an Off-The-Shelf (OTS) component is a non-commercial ditto. Some claim that that a component typically presents 90% of the desired functionality [69], and the developing organization then has to decide whether the additional 10% can justify a much higher cost and delayed release date. The market for commercial software components has increased during the nineties [97] but currently seem to decrease.

To make components interact, there are component technologies such as COM [7], CORBA [81,85], J2EE [63,75], and .NET [92]. These and more middleware technologies have been evaluated for interoperability [105]. Interface Definition Languages (IDLs) are a central part of a component technology, and integration at the function call level is relatively straightforward. But IDLs can today only achieve syntactic interoperability [39,100] which is not enough to make two components interact as desired [98]. To ensure true interoperability between systems or components, the semantics must be specified as well [31,98]. To arrive at integration at a higher level, XML [29,102] has become a popular encoding language which may be a common denominator of systems and used for integration [3,15,22].

Even when a system is completely developed in-house, a component-based approach may be chosen. A product line approach [16] means that there is a strategy for internal development of components to be reused in different products. Both the products and the architecture of the product line itself must be evolved and the situation is not too different from component integration [37,88,89,93,94].

Expected Benefits. By using already developed and tested components, the desired benefit is that the system can be built rapidly and cheaply (compared to developing everything from scratch), and that the system will be of high quality.

Drawbacks. There are some limitations with this approach. The functionality desired but not implemented may be crucial. If a component is updated, the system may cease to function (most often due to semantic differences). Requirements for keeping control of compatibility call for new types of configuration management [44,46]. Using a component in a long-lived system creates a dependency on a third party regarding maintenance, updates, error corrections, etc. Also, it may be very costly to exchange one component to one that is similar. There is not yet a standardized way of certifying component quality and behavior although there is research on how it could be achieved [20,32,45]. And as said, semantic interoperability is not completely solved.

4. Standard Interfaces and Open Systems

The common understanding of an “open” system is that it should e.g. be portable, scalable, and (which is important for this paper) interoperable through means of a standard interface. Meyers and Oberndorf argue that although these are desirable properties, they are difficult to demonstrate in general [61,62,82]. It is impossible to demonstrate interoperability in isolation, without specifying something concrete a component should interoperate with. Their definition of an open system is therefore: a set of components with interface specifications fully defined, available to the public, maintained according to group consensus, and in which the implementations of components are conformant to the specification. Also, anyone may produce (and profit from) implementations of that specification. Major organizations for software standards are ANSI [4], IEEE [35], and ISO [36].

In computer networks and tele-communications, open systems with standard interfaces (in the form of protocols) are prevalent [28,64]. One of the major driving forces is there interoperability between vendors. Other fields in similar contexts, where systems from different vendors need to interoperate and exchange information are Geographic Information Systems (GIS) [17,49,66,91], and hypermedia [2,3,33] to mention a few. Application domains with an identified need to create their own standard interfaces for interoperability include – just to illustrate the applicability of the approach – public libraries [70], mathematical computations [48], and photo archives [42]. Interoperability through standardized interfaces is also a concern of software agents [101]. Although autonomous, agents need to communicate and exchange data, and to enable interoperability between agents developed with different technologies this needs to be done in a uniform manner [49].

From an integration point of view, the importance of standards applies not only to interfaces but domain-specific architectures as well. There may be standard reference architectures [61,83], or vendor-specific architectures (which are implementations as much as specifications), like ABB's Industrial IT architecture [9].

As said, the desirable properties of open systems cannot be shown in general. Conformance testing is carried out to show conformance to a standard, while interoperability testing means testing whether two products (said to adhere to the same standard) actually work together as intended [41]. Conformance to a standard is in practice not enough to ensure interoperability between two implementations [12,53,59].

Expected Benefits. There appears to be two major reasons for building systems based on standard interfaces. First, building open systems is suitable when an integrator wants to avoid being dependent on a single vendor [23,76,76]. Second, when there is no single integrator, the only possibility to make different components and systems interoperate is to ensure they conform to a standardized interface [70].

Drawbacks. To have a practical impact, standards need implementations. A drawback (from the interoperability point of view) with standards is the commercial marketplace itself with the option for implementers to adhere to standards or not – the choice depends on commercial forces. Another drawback is that reaching consensus often takes a long time, and both vendors and acquirers may need to act quickly in order to produce products and integration solutions on time [54]. This may lead to a number of similar but incompatible de facto-standards. Also, a vendor strong enough may provide an implementation violating the standard and force its competitors to follow.

5. Enterprise Application Integration (EAI)

Information systems are systems with the primary purpose to store and manage data [47]. *Enterprise Resource Planning* (ERP) [8,65] systems such as SAP R/3 [78], *Product Data Management* (PDM) and *Software Configuration Management* (SCM) systems [19] are typical examples of systems, used to plan and manage an enterprise' assets and resources. As enterprises need to streamline their processes to be competitive there is a need for integrating different information systems [30,47] to make information consistent and easily accessible. The typical solution is "loose" integration, where the system components operate independently of each other and continue to store data in their own repository. Building unique interfaces between each pair of systems that needs to communicate is not cost efficient [23]. *Enterprise Application Integration* (EAI) [21,38,55,56,77] is the name of structured integration of information systems within an organization [30,50,58,87]. EAI includes building wrappers, adapters, and using standardized middleware to connect and integrate the systems.

Many information systems are long-lived. They have to be adapted to ever-changing requirements, and thus evolve, often over decades [51,52,72]. Evolving and integrating these systems may be crucial for an organization to become more efficient and competitive, but due to problems like lacking or outdated documentation, few people having overview over the whole system, design erosion [72,95], and different technologies from different eras being mixed, maintenance, evolution, and integration is a major challenge. Still, there is often no practical option to start over from scratch, since these systems represent enormous efforts invested in requirements engineering, designing, implementation, testing, debugging, tuning etc., and choosing among two bad things, organizations usually stays with the existing systems.

EAI is a broad term and may include activities such as data mining and reverse engineering [1] and content integration [87] (to understand the existing data and systems), migration [11] (to get rid of the most problematic technologies and solutions), using a common messaging middleware [10,55,56,60,77,105] (of which there are many commercial solutions), and encapsulating and wrapping legacy systems in a component-based approach [79]. The market for application integration and middleware (AIM) is large ($6.3 billion worldwide in 2003) and is expected to continue growing [18].

As of the business case leading to EAI efforts, EAI addresses the context of in-house integration of systems an organization *uses* rather than *produces*. The usage may be in terms of in-house usage of ERP systems [21,50] or electronic business [103], such as business to business, B2B [56]. Also, EAI is the choice when it is not a practical option to modify the existing systems – source code or documentation may not be available (physically or due to legal restrictions), or they may too large and complex.

There is a correlation between the structure of an organization and that of its software [30], hence the notion of "enterprise architectures". The integration may occur at different levels, ranging from data and application to the more difficult levels: business processes and humans [74]. The "Zachman Framework for Enterprise Architecture" is a framework within which a whole enterprise is modeled in two dimensions: the first describing its data, its people, its functions, its network, and more, and the other dimension specifying views of different detail [106,107]. Another, similar, enterprise information systems framework is "The Open Group Architectural Framework" (TOGAF) [68].

Expected Benefits. The benefit of integrating information systems is to have information synchronized and more accessible. The reason for choosing the EAI approach is that it provides ways of achieving this that are structured and more cost-efficient than integrating systems pair-wise with unique solutions in an ad-hoc manner.

Drawbacks. EAI requires a high degree of commitment, coordination, and upfront investments [50]. EAI may break down when integration occurs between enterprises, when data is operational rather than historical, and more unstructured data need to be integrated [87]. And the integration problem continues: as systems being integrated use different (not fully compatible) commercial technologies, the need arises to integrate the integration technologies [26].

Table 1: Summary of the integration approaches.

	Context	**Expected benefits**	**Possible drawbacks**
Components	Parts of a system's functionality already available in external, general components, or A product-line approach with internally pre-developed components.	Faster and cheaper development process. High system quality.	Not all desired functionality available. Risk of components being of low quality. Risk of strong vendor dependency.
Open Standards	Vendor independence desired, or No single integrator.	Smooth integration of components. Possibility to switch to another provider.	No standard applicable at all. No standard yet in place, leading to vendor-specific variants of the expected standard. Conformance testing not always enough in practice.
EAI	Existing information systems, practically impossible to rewrite or replace systems.	Information consistent and easily accessible. EAI more cost-efficient than building pair-wise inte-gration solutions.	Expensive, requires long-term commitment. Does not address integration between enterprises. Integration problems remain at a higher level.

6. Discussion

Table 1 summarizes the contexts, expected benefits, and drawbacks (from the point of view of the integrator) for each of the presented approaches separately. As hinted at in the introduction, these fields may overlap in practice (there may e.g. be EAI solutions using commercial components with standard interfaces, if several of the contexts of the table apply to a given situation).

6.1 Case Study – Challenging Existing Approaches

We have participated in an industrial integration project that does not seem to fit in any of the presented approaches. Here, we present the case very briefly, focusing on what is relevant for the present paper; please refer to [43] for details. A company merger led to a wish to integrate three of the software tools of the previous two organizations, products that overlapped functionally. These systems were completely owned and controlled by the integrating organization.

The desired effects of integrating the systems were several. There was a desire to have only one product to market as well as to use internally, which should contain the best from each of the existing systems. Also, there was a hope that an integrated system would be less costly to maintain and evolve than the existing separate systems.

Considering the contexts and expected benefits in Table 1, none of the existing integration approaches seems entirely suitable. The situation reminds of a component-based approach in that the future system would be built by existing parts. However, although the systems were modularized, only one was componentized in the sense “supported by a component technology”. If the systems in the case study were first componentized, there would be duplicate components with similar functionality and the situation would remind of an EAI context, although in a smaller scale. As all software were developed and used internally, the case reminds somewhat of a product line approach. However, there would only be one system; there was no payoff in creating general and reusable components. Using standard interfaces would be necessary only if the system was to be interoperable with other systems from other vendors, which was not the case. The context was not totally unlike EAI, since there was a possibility to not modify source code, although it was available, and instead wrap the systems. EAI was not explicitly considered mainly because the approach was not known. Even if it was, it seems unlikely it would be chosen since it would not achieve all of the integration goals: such a loose integration would give some benefits to the users such as data consistency, but not a homogeneous user interface. The integrated system would arguably be more difficult to maintain than some other type of integration since even more program code and technologies would be added. Also, it appears that EAI would require too much in terms of commitment and investments compared to what would be gained.

Some of the techniques of EAI and component-based software were considered though, most notably the idea of wrapping some existing parts of one system and treat them as Java components in the other.

6.2 Solutions Discussed

Instead of suggesting solutions adhering completely to any of the existing approaches, the following integration solutions were the main topics of analysis and discussion in the case study:

- **Data level integration.** This solution would mean keeping the applications separate but consolidating their data models and put all data in a common database. This would keep all data accessible and consistent, i.e. improve usability, but not necessarily make maintenance easier. The integration would also be fairly costly, since consolidating the data models (ontologies) would result in ripple effects throughout the program logic of the systems. This reminds both of a component-based approach and of EAI, but the database structure and some source code would be modified to minimize duplication of data and functionality.

- **Code level integration.** This would mean integrating the systems "component by component" and merge the "best" parts (with most functionality and highest quality) of the source code of the existing systems. This would give the users the most homogeneous system as well as be the easiest to maintain. The drawback is the commitment and resources required to integrate systems on time, which imposes a major risk. This alternative utilizes the fact that it is possible to modify the source code to the fullest extent, something the surveyed approaches do not.
- **Extending one system.** There was also the alternative of basing all future development on one of the systems. The resulting system would arguably have a homogeneous user interface and be relatively simple to maintain. One drawback is that functionality is rewritten, implying a high cost that is difficult to motivate. Also, the organization must be committed to a long term strategy for how to discontinue and retire the other systems. This became the final decision (made after the previous publication of the case study [43]).

Although these proposed solutions are not entirely new, research is needed to explore and describe the benefits and drawbacks of these (and more) in the context of the case study, i.e. when there is a wish to merge existing software systems into one product.

7. Summary and Conclusions

This paper analyzed three large fields of practice and research from a software integration perspective: component-based software, standards and open systems, and Enterprise Application Integration (EAI). An industrial integration project was presented where none of the existing integration approaches were considered suitable. Total ownership over the systems to be integrated gives more possibilities than the existing approaches takes into account, and other solutions may be more suitable for the new needs.

Challenges for the future include finding more cases in a similar context and investigating what integration solutions are considered and chosen, and during which circumstances.

7.1 Acknowledgements

Many thanks to Imad Eldin Ali Abugessaisa, Igor Cavrak, Goran Mustapic, Johan Schubert, and the anonymous reviewers for their valuable comments on this paper.

8. References

[1] Aiken P. H., *Data Reverse Engineering : Slaying the Legacy Dragon,* ISBN 0-07-000748-9, McGraw Hill, 1996.

[2] Anderson K. M., Och C., King R., and Osborne R. M., "Integrating Infrastructure: Enabling Large-Scale Client Integration", In *Proceedings of eleventh ACM Conference on Hypertext and Hypermedia,* pp. 57-66, ACM Press, 2000.

[3] Anderson K. M. and Sherba S. A., "Using XML to support Information Integration", In *Proceedings of International Workshop on XML Technologies and Software Engineering (XSE),* IEEE, 2001.

[4] ANSI, ANSI, *American National Standards Institute,* http://www.ansi.org, 2004.

[5] Bachman F., Bass L., Buhman S., Comella-Dorda S., Long F., Seacord R. C., and Wallnau K. C., *Volume II: Technical Concepts of Component-Based Software Engineering,* report CMU/SEI-2000-TR-008, Software Engineering Institute, Carnegie Mellon University, 2000.

[6] Bass L., Clements P., and Kazman R., *Software Architecture in Practice* (2nd edition), ISBN 0-321-15495-9, Addison-Wesley, 2003.

[7] Box D., *Essential COM,* ISBN 0-201-63446-5, Addison-Wesley, 1998.

[8] Brady J., Monk E., and Wagner B., *Concepts in Enterprise Resource Planning,* ISBN 0619015934, Course Technology, 2001.

[9] Bratthall L. G., van der Geest R., Hofmann H., Jellum E., Korendo Z., Martinez R., Orkisz M., Zeidler C., and Andersson J. S., "Integrating Hundred's of Products through One Architecture: the Industrial IT architecture", In *Proceedings of the 24th International Conference on Software Engineering,* pp. 604-614, ACM, 2002.

[10] Britton C. and Bye P., *IT Architectures and Middleware: Strategies for Building Large, Integrated Systems* (2nd edition), ISBN 0321246942, Pearson Education, 2004.

[11] Brodie M. L. and Stonebraker M., *Migrating Legacy Systems: Gateways, Interfaces & the Incremental Approach,* Morgan Kaufmann Series in Data Management Systems, ISBN 1558603301, Morgan Kaufmann, 1995.

[12] Bub T. and Schwinn J., "VERBMOBIL: The Evolution of a Complex Large Speech-to-Speech Translation System", In *Proceedings of Fourth International Conference on Spoken Language (ICSLP),* pp. 2371-2374, IEEE, 1996.

[13] Burrough P. A. and McDonnell R., *Principles of Geographical Information Systems* (2nd edition), ISBN 0198233655, Oxford University Press, 1998.

[14] Bushmann F., Meunier R., Rohnert H., Sommerlad P., and Stal M., *Pattern-Oriented Software Architecture - A System of Patterns*, ISBN 0-471-95869-7, John Wiley & Sons, 1996.

[15] Chester T. M., “Cross-Platform Integration with XML and SOAP”, In *IT Professional*, volume 3, issue 5, pp. 26-34, 2001.

[16] Clements P. and Northrop L., *Software Product Lines: Practices and Patterns*, ISBN 0-201-70332-7, Addison-Wesley, 2001.

[17] Clément G., Larouche C., Gouin D., Morin P., and Kucera H., “OGDI: Toward Interoperability among Geospatial Databases”, In *ACM SIGMOD Record*, volume 26, issue 3, pp. 108-, 1997.

[18] Correia J. M. and Biscotti F., *Forecast: AIM Software, Worldwide, 2003-2008*, Gartner, 2004.

[19] Crnkovic I., Asklund U., and Persson-Dahlqvist A., *Implementing and Integrating Product Data Management and Software Configuration Management*, ISBN 1-58053-498-8, Artech House, 2003.

[20] Crnkovic I. and Larsson M., *Building Reliable Component-Based Software Systems*, ISBN 1-58053-327-2, Artech House, 2002.

[21] Cummins F. A., *Enterprise Integration: An Architecture for Enterprise Application and Systems Integration*, ISBN 0471400106, John Wiley & Sons, 2002.

[22] Decker S., Melnik S., van Harmelen F., Fensel D., Klein M., Broekstra J., Erdmann M., and Horrocks I., “The Semantic Web: The Roles of XML and RDF”, In *IEEE Internet Computing*, volume 4, issue 5, pp. 63-74, 2000.

[23] Emmerich W., Ellmer E., and Fieglein H., “TIGRA - An Architectural Style for Enterprise Application Integration”, In *Proceedings of 23rd International Conference on Software Engineering*, pp. 567-576, IEEE, 2001.

[24] Gamma E., Helm R., Johnson R., and Vlissidies J., *Design Patterns - Elements of Reusable Object-Oriented Software*, ISBN 0-201-63361-2, Addison-Wesley, 1995.

[25] Garlan D., Allen R., and Ockerbloom J., “Architectural Mismatch: Why Reuse is so Hard”, In *IEEE Software*, volume 12, issue 6, pp. 17-26, 1995.

[26] Gorton I., Thurman D., and Thomson J., “Next Generation Application Integration Challenges and New Approaches”, In *Proceedings of 27th Annual International Computer Software and Applications Conference (COMPSAC)*, pp. 585-590, IEEE, 2003.

[27] Guarino N., *Formal Ontology in Information Systems*, ISBN 9051993994, IOS Press, 1998.

[28] Halsall F., *Data Communications, Computer Networks, and Open Systems* (4th edition), ISBN 020142293X, Addison-Wesley, 1996.

[29] Harold E. R. and Means W. S., *XML in a Nutshell* (2nd edition), ISBN 0596002920, O'Reilly, 2004.

[30] Hasselbring W., “Information System Integration”, In *Communications of the ACM*, volume 43, issue 6, pp. 33-38, 2000.

[31] Heiler S., “Semantic Interoperability”, In *ACM Computing Surveys*, volume 27, issue 2, pp. 271-273, 1995.

[32] Hissam S. A., Moreno G. A., Stafford J., and Wallnau K. C., *Packaging Predictable Assembly with Prediction-Enabled Component Technology*, report Technical report CMU/SEI-2001-TR-024 ESC-TR-2001-024, 2001.

[33] Hunter J., “Enhancing the Semantic Interoperability of Multimedia Through a Core Ontology”, In *IEEE Transactions on Circuits & Systems for Video Technology*, volume 13, issue 1, pp. 49-59, 2003.

[34] IEEE, *IEEE Standard Glossary of Software Engineering Terminology*, report IEEE Std 610.12-1990, IEEE, 1990.

[35] IEEE, IEEE, *IEEE Standards Association Home Page*, http://standards.ieee.org/, 2004.

[36] ISO, ISO, *ISO - International Organization for Standardization*, http://www.iso.org, 2004.

[37] Johansson E. and Höst M., “Tracking Degradation in Software Product Lines through Measurement of Design Rule Violations”, In *Proceedings of 14th International Conference in Software Engineering and Knowledge Engineering (SEKE)*, ACM, 2002.

[38] Johnson P., *Enterprise Software System Integration - An Architectural Perspective*, Ph.D. Thesis, Industrial Information and Control Systems, Royal Institute of Technology, 2002.

[39] Kaplan A., Ridgway J., and Wileden J. C., “Why IDLs are Not Ideal”, In *Proceedings of 9th International Workshop on Software Specification and Design*, ACM, 1998.

[40] Keshav R. and Gamble R., “Towards a Taxonomy of Architecture Integration Strategies”, In *Proceedings of third International Workshop on Software Architecture*, pp. 89-92, ACM, 1998.

[41] Kindrick J. D., Sauter J. A., and Matthews R. S., “Improving conformance and interoperability testing”, In *StandardView*, volume 4, issue 1, 1996.

[42] Kramer R. and Sesink L., “Framework for Photographic Archives Interoperability”, In *Proceedings of The 3rd Conference on Standardization and Innovation in Information Technology*, pp. 135-140, IEEE, 2003.

[43] Land R., *An Architectural Approach to Software Evolution and Integration*, Licentiate Thesis, Department of Computer Science and Engineering, Mälardalen University, 2003.

[44] Larsson M., *Applying Configuration Management Techniques to Component-Based Systems*, Licentiate Thesis, Dissertation 2000-007, Department of Information Technology Uppsala University., 2000.

[45] Larsson M., *Predicting Quality Attributes in Component-based Software Systems*, Ph.D. Thesis, Mälardalen University, 2004.

[46] Larsson M. and Crnkovic I., “New Challenges for Configuration Management”, In *Proceedings of 9th*

Symposium on System Configuration Management, Lecture Notes in Computer Science, nr 1675, Springer Verlag, 1999.

[47] Laudon K. C. and Laudon J. P., *Management Information Systems* (8th edition), ISBN 0131014986, Pearson Education, 20030.

[48] Le H. and Howlett C., "Client-Server Communication Standards for Mathematical Computation", In *Proceedings of International Conference on Symbolic and Algebraic Computation*, pp. 299-306, ACM Press, 1999.

[49] Leclercq E., Benslimane D., and Yétongnon K., "ISIS: A Semantic Mediation Model and an Agent Based Architecture for GIS Interoperability", In *Proceedings of International Symposium Database Engineering and Applications (IDEAS)*, pp. 87-91, IEEE, 1999.

[50] Lee J., Siau K., and Hong S., "Enterprise Integration with ERP and EAI", In *Communications of the ACM*, volume 46, issue 2, pp. 54-60, 2003.

[51] Lehman M. M. and Ramil J. F., "Rules and Tools for Software Evolution Planning and Management", In *Annals of Software Engineering*, volume 11, issue 1, pp. 15-44, 2001.

[52] Lehman M. M. and Ramil J. F., "Software Evolution and Software Evolution Processes", In *Annals of Software Engineering*, volume 14, issue 1-4, pp. 275-309, 2002.

[53] Li M., Puder A., and Schieferdecker I., "A Test Framework for CORBA Interoperability", In *Proceedings of Fifth IEEE International Enterprise Distributed Object Computing Conference*, pp. 152-161, IEEE, 2001.

[54] Libicki M., "Second-Best Practices for Interoperability", In *StandardView*, volume 4, issue 1, pp. 32-35, 1996.

[55] Linthicum D. S., *Enterprise Application Integration*, Addison-Wesley Information Technology Series, ISBN 0201615835, Addison-Wesley, 1999.

[56] Linthicum D. S., *B2B Application Integration: e-Business-Enable Your Enterprise*, ISBN 0201709368, Addison-Wesley, 2003.

[57] Longley P. A., Goodchild M. F., Maguire D. J., and Rhind D. W., *Geographic Information Systems and Science*, ISBN 0471892750, John Wiley & Sons, 2001.

[58] Losavio F., Ortega D., and Perez M., "Modeling EAI", In *Proceedings of 12th International Conference of the Chilean Computer Science Society*, pp. 195-203, IEEE, 2002.

[59] Mazen M. and Dibuz S., "Pragmatic method for interoperability test suite derivation", In *Proceedings of 24th Euromicro Conference*, pp. 838-844, IEEE, 1998.

[60] Medvidovic N., "On the Role of Middleware in Architecture-Based Software Development", In *Proceedings of the 14th international conference on Software Engineering and Knowledge Engineering (SEKE)*, pp. 299-306, ACM Press, 2002.

[61] Meyers C. and Oberndorf P., *Managing Software Acquisition: Open Systems and COTS Products*, ISBN 0201704544, Addison-Wesley, 2001.

[62] Meyers C. and Oberndorf T., *Open Systems: The Promises and the Pitfalls*, ISBN 0-201-70454-4, Addison-Wesley, 1997.

[63] Monson-Haefel R., *Enterprise JavaBeans* (4th edition), ISBN 059600530X, O'Reilly & Associates, 2004.

[64] Newton H., *Newton's Telecom Dictionary: Covering Telecommunications, Networking, Information Technology, Computing and the Internet* (20th edition), ISBN 1578203090, CMP Books, 2004.

[65] O'Leary D. E., *Enterprise Resource Planning Systems: Systems, Life Cycle, Electronic Commerce, and Risk* (1st edition), ISBN 0521791529, Cambridge University Press, 2000.

[66] OGC, OGC, *Open GIS Consortium, Inc.*, http://www.opengis.org/, 2004.

[67] Omelayenko B., "Integration of Product Ontologies for B2B Marketplaces: A Preview", In *ACM SIGecom Exchanges*, volume 2, issue 1, pp. 19-25, 2000.

[68] OMG, *The Open Group Architectural Framework*, URL: http://www.opengroup.org/architecture/togaf8-doc/arch/, 2003.

[69] Oreizy P., "Decentralized Software Evolution", In *Proceedings of International Conference on the Principles of Software Evolution (IWPSE 1)*, pp. 20-21, 1998.

[70] Paepcke A., Chang C.-C. K., Winograd T., and García-Molina H., "Interoperability for digital libraries worldwide", In *Communications of the ACM*, volume 41, issue 4, pp. 33-43, 1998.

[71] Palsberg J., "Software Evolution and Integration", In *ACM Computing Surveys*, volume 28, issue 4es, 1996, http://www.acm.org/pubs/citations/journals/surveys/1996-28-4es/a200-palsberg/.

[72] Parnas D. L., "Software Aging", In *Proceedings of The 16th International Conference on Software Engineering*, pp. 279-287, IEEE Press, 1994.

[73] Pinto H. S. and Martins J. P., "A Methodology for Ontology Integration", In *Proceedings of International Conference on Knowledge Capture*, pp. 131-138, ACM, 2001.

[74] Pollock J. T., "The Big Issue: Interoperability vs. Integration", In *eAI Journal*, volume October, 2001, http://www.eaijournal.com/.

[75] Roman E., *Mastering Enterprise JavaBeans and the Java 2 Platform, Enterprise Edition*, ISBN 0-471-33229-1, Wiley, 1999.

[76] Royster C., "DoD Strategy on Open Systems and Interoperability", In *StandardView*, volume 4, issue 2, pp. 104-106, 1996.

[77] Ruh W. A., Maginnis F. X., and Brown W. J., *Enterprise Application Integration*, A Wiley Tech Brief, ISBN 0471376418, John Wiley & Sons, 2000.

[78] SAP, www.sap.com, *SAP R/3*, www.sap.com, 2003.

[79] Sauer L. D., Clay R. L., and Armstrong R., "Meta-component architecture for software interoperability", In *Proceedings of International Conference on Software Methods and Tools (SMT)*, pp. 75-84, IEEE, 2000.

[80] Schmidt D., Stal M., Rohnert H., and Buschmann F., *Pattern-Oriented Software Architecture - Patterns for Concurrent and Networked Objects*, Wiley Series in Software Design Patterns, ISBN 0-471-60695-2, John Wiley & Sons Ltd., 2000.

[81] SEI Software Technology Roadmap, *Common Object Request Broker Architecture*, URL: http://www.sei.cmu.edu/str/descriptions/corba_body.html, 9-6-2004.

[82] SEI Software Technology Roadmap, *COTS and Open Systems--An Overview*, URL: http://www.sei.cmu.edu/str/descriptions/cots.html, 9-6-2004.

[83] SEI Software Technology Roadmap, *Reference Models, Architectures, Implementations--An Overview*, URL: http://www.sei.cmu.edu/str/descriptions/refmodels_body.html, 9-6-2004.

[84] Shanzhen Y., Lizhu Z., Chunxiao X., Qilun L., and Yong Z., "Semantic and interoperable WebGIS", In *Proceedings of the Second International Conference on Web Information Systems Engineering*, pp. 42-47, IEEE, 2001.

[85] Siegel J., *CORBA 3 Fundamentals and Programming* (2nd edition), ISBN 0471295183, John Wiley & Sons, 2000.

[86] Stanford University, *Stanford Annotated Interoperability Bibliography*, URL: www-diglib.stanford.edu/diglib/pub/interopbib.html, 8-4-2004.

[87] Stonebraker M. and Hellerstein J. M., "Content Integration for E-Business", In *ACM SIGMOD Record*, volume 30, issue 2, pp. 552-560, 2001.

[88] Svahnberg M. and Bosch J., "Characterizing Evolution in Product Line Architectures", In *Proceedings of 3rd annual IASTED International Conference on Software Engineering and Applications*, pp. 92-97, IASTED/Acta Press, 1999.

[89] Svahnberg M. and Bosch J., "Issues Concerning Variability in Software Product Lines", In *Proceedings of Software Architectures for Product Families: 7th International Workshop on Database Programming Languages, DBPL'99, Revised Papers (Lecture Notes in Computer Science 1951)*, Springer Verlag, 2000.

[90] Szyperski C., *Component Software - Beyond Object-Oriented Programming*, ISBN 0-201-17888-5, Addison-Wesley, 1998.

[91] Teng W., Pollack N., Serafino G., Chiu L., and Sweatman P., "GIS and Data Interoperability at the NASA Goddard DAAC", In *Proceedings of International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 1953-1955, IEEE, 2001.

[92] Thai T. and Lam H., *.NET Framework Essentials* (2nd edition), O'Reilly Programming Series, ISBN 0596003021, O'Reilly & Associates, 2002.

[93] van der Hoek A., Heimbigner D., and Wolf A. L., *Versioned Software Architecture*, 1998.

[94] van der Hoek A., Heimbigner D., and Wolf A. L., *Capturing Architectural Configurability: Variants, Options, and Evolution*, report Technical Report CU-CS-895-99, 1999.

[95] van Gurp J. and Bosch J., "Design Erosion: Problems & Causes", In *Journal of Systems & Software*, volume 61, issue 2, pp. 105-119, 2002.

[96] Visser U., Stuckenschmidt H., Schuster G., and Vögele T., "Ontologies for Geographic Information Processing", In *Computers & Geosciences*, volume 28, issue 1, pp. 103-117, 2002.

[97] Wallnau K. C., Hissam S. A., and Seacord R. C., *Building Systems from Commercial Components*, ISBN 0-201-70064-6, Addison-Wesley, 2001.

[98] Wegner P., "Interoperability", In *ACM Computing Surveys*, volume 28, issue 1, 1996.

[99] Wileden J. C. and Kaplan A., "Software Interoperability: Principles and Practice", In *Proceedings of 21st International Conference on Software Engineering*, pp. 675-676, ACM, 1999.

[100] Wileden J. C., Wolf A. L., Rosenblatt W. R., and Tarr P. L., "Specification Level Interoperability", In *Proceedings of 12th International Conference on Software Engineering (ICSE)*, pp. 74-85, ACM, 1990.

[101] Wooldridge M., *Introduction to MultiAgent Systems*, ISBN 047149691X, John Wiley & Sons, 2002.

[102] XML, *XML.org*, URL: http://www.xml.org/, 2004.

[103] Yang J. and Papazoglou M. P., "Interoperation Support for Electronic Business", In *Communications of the ACM* , volume 43, issue 6, pp. 39-47, 2000.

[104] Yau S. S. and Dong N., "Integration in component-based software development using design patterns", In *Proceedings of The 24th Annual International Computer Software and Applications Conference (COMPSAC)*, pp. 369-374, IEEE, 2000.

[105] Young P., Chaki N., Berzins V., and Luqi, "Evaluation of Middleware Architectures in Achieving System Interoperability", In *Proceedings of 14th IEEE International Workshop on Rapid Systems Prototyping*, pp. 108-116, IEEE, 2003.

[106] Zachman J. A., "A Framework for Information Systems Architecture", In *IBM Systems Journal*, volume 26, issue 3, 1987.

[107] ZIFA, *Zachman Framework for Enterprise Architecture*, URL: http://www.zifa.com/, 2003.