Research Center for Integrational Software Engineering (RISE)

Uwe Assmann (ed.)
RISE, Institutionen för Datavetenskap, Linköpings Universitet
mailto:uweas@ida.liu.se tel: +46 13 28 66 78 fax: +46 13 28 44 99

15th November 2001

"Actually, we have seen not many paradigm shifts in computer science. Object-oriented technology was said to be one, but this is not true: it mainly consists of an improved style of imperative programming. However, in the late nineties, some new technologies have appeared that really introduce a paradigm, and these are aspect-oriented programming, multi-paradigm design, and generative programming."

Jim Coplien, Keynote Speech, 1st Symposium of Generative Software Engineering, Erfurt 1999.

RISE, the Research centre for Integrational Software Engineering, develops concepts, methods, technology, and process support for the integration of software systems, software parts, and software artefacts. At the same time, RISE integrates composition technology for XML based active documents. For a holistic approach, process, design, implementation, maintainance, and reengineering levels are considered in an integrated manner.

RISE pays special attention to the transfer of its results to industrial practice. Results should not be developed in isolation, but integrated into industrial partner's processes and products in an early stage.

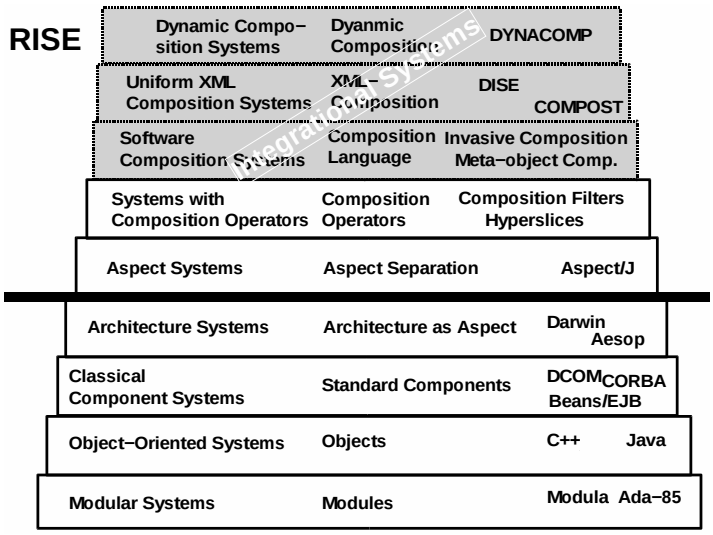

Figure 1: Integrational Systems: a new step in software constructing beyond the component-based methodology. Below the bar: black box components. Above the bar: component integration with grey box components. Grey layers: RISE work areas.

1 What is RISE?

Software is hard to build. Software systems become more and more complex, software and hardware must be integrated in embedded systems, e.g. mobile and wearable devices. In web-based applications, there is a strong trend to integrate software and documents with the uniform representation (XML), but engineering of web-based applications, web sites, and document bases is often a mess.

Additionally, software processes are difficult to organize. Often, a team is decomposed into requirements engineers, designers, implementors, and testers, and the contributions of these specialists cannot be maintained in an *integrated* way. Instead the specifications are kept separate. When they are evolved they often become inconsistent and out-of-date.

On the other hand, in software engineering, we are at the transition to a new age of software development, the age of *software integration*. Old-fashioned or well-known programming methods, such as modular programming, object-oriented development, or component-based development are superseded by powerful decomposition and integration mechanisms like aspect-oriented programming, view-based software development with composition operators, and composition-based development (Figure 1, levels 5-7). The decisive progress of these novel approaches is that they rely on *grey box component* models and merge components during composition. Currently, the scope of this technology is further broadened to both software and documents integrated through a uniform representation (XML), i.e., everything is uniformly represented as *active documents* or *docware* (PELAB's EASYCOMP EU project [Ass99], Figure 1 level 8). These new techniques of the integrational age will enable us to decompose problems and integrate software solutions in a way which is significantly more powerful than today's solutions.

The vision of RISE is that *integrational techniques* will form a new era in software and active document engineering, the era of *integrational software engineering (ISE)*. RISE develops *integrational* methods and techniques to simplify software and docware construction. RISE's main focus are decomposition and integration mechanisms, i.e., methods that split the complex software construction problem into manageable parts, and integrate partial solutions to the final solution. Since this is done uniformly for XML, the uniform representation for both documents and software, this integrates also the processing of active documents.

1.1 The Mission of RISE

RISE is aimed at this new dawning age, developing methods and tools for integrational software engineering (ISE). Its mission is to go one step beyond the classical methods, and to investigate strategically into the new integration techniques, conquering a leading edge for Sweden in the software engineering of the 2010s. RISE's research will immediately benefit to the following strategic industrial application areas, attempting to transfer its results to industrial practice early:

RISE, the Research Centre for Integrational Software Engineering, develops concepts, methods, technology, and process support for the integration of software systems, software artefacts, and XML based active documents, in the areas of

- *software composition techniques*,
- *uniform XML composition techniques*,
- and *dynamic composition techniques*.

These techniques subsume aspect-oriented programming, view-based programming, architectural description languages as well as techniques for XML-based active documents. For a holistic approach, process, requirements, design, implementation, maintainance, and reengineering levels are considered in an integrated manner.

RISE will initiate projects, in particular to transfer the ISE technology to industry. RISE's research will immediately benefit to the following strategic industrial application areas, attempting to transfer its results to industrial practice early:

- Increasingly complex and capable network based software systems. Tomorrows increased bandwidth enables increased opportunity and demand for integrated mobile services, peer-to-peer computing, etc. More powerful techniques such as composition coupled with improved software processes are needed. Our industry contacts, e.g. Ericsson, IFS, Ida Infront, etc. confirm that integrational techniques will play a major role here.
- Integration of software and documents in web-based applications. Many large software applications become increasingly web-oriented, e.g. business software from companies such as IFS and Intentia, or mobile applications.
- The Semantic Web, i.e., the second generation web. Form processing and content checking will be integrated with tomorrow's web pages which will be XML based. Such semantics can e.g. be expressed in the DAML rule-based language closely related to the RML language developed here.
- Integrated evolution of requirements and testing aspects in future projects, ensuring better product quality and maintenance.
- Integrated application provisioning of distributed mobile applications. This is the ability for a system to handle a request for an application, finding or generating a suitable version of the requested application, and provide it to the requestor.
- Integrated application provisioning is a key component in future support systems for mobile computing. For example, recently several mobile phone vendors announced Java enabled phones, providing functionality not only for browser based applications in mobile handsets, but also true dynamic applications, downloaded and executed in the handset.

2 The New Dream: Flexible Software Integration

In the second half of the 90s, a new dream starts to pervade software engineering: integration-based software engineering. Researchers realize that for software, it is not sufficient to plug components together by modular composition [ABV92] [HO93a] [NM95]. Instead, there should be more flexible composition mechanisms which allow to integrate components in a more powerful way than with black-box composition. A very important aspect of this trend is to seperate concerns of software into several dimensions, called *aspects* [KLM+97] [EFB01] or *hyperslices* [TOHS99]. Such a composition technique should be much more powerful than black-box composition: it should be able to *merge* and *integrate* components during composition (Figure 2).

This new dream directly follows up the component dream and extends it beyond the classi-

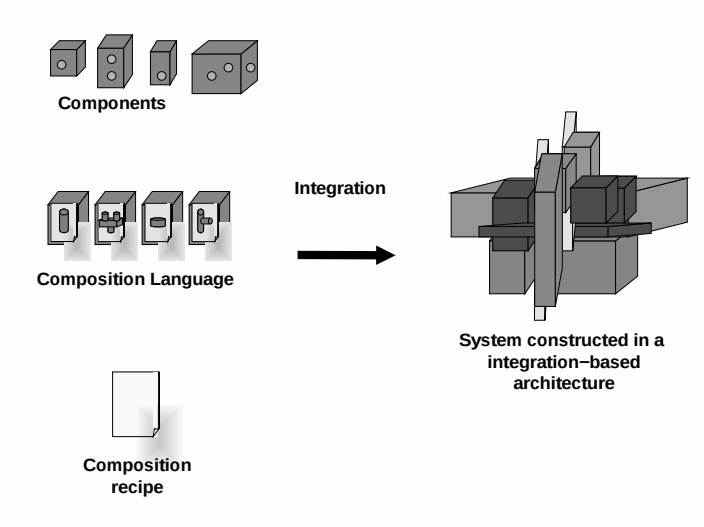

Figure 2: Integrational system construction goes a step further, integrating system parts tightly.

cal composition mechanisms which are used so far and known also from other disciplines, e.g., mechanical engineering. In these disciplines, an engineer works mainly with divide-and-conquer strategy based on black-box composition of components of the shelf. Faced with a problem, he/she decomposes the problem is decomposed recursively into smaller problems, as long as there is no direct solution possible. When the problem units are so small enough the engineer looks up solution components in catalogues that provide the desired features. Those solution components are plugged together in a black-box manner, to build up the final product (*black-box composition*). Here, integrational composition of software goes one decisive step further. In problem decomposition, concerns that cross-cut each other, whose parts are intertwined, and whose slices are dependent on each other, must be separated in very complex ways. In consequence, during composition solution components are *woven together* so that their boundaries vanish, their parts (Figure 2). Hence, integrational software engineering relies on decomposing a problem by powerful separation of concerns and on composing a solution by integrational methods that are more flexible than black-box integration.

Other disciplines have already adopted this way of engineering as a standard. For instance, in the architecture of buildings, no architect would dare to present the plan for a building in an integrated way. Instead, all plans are separated into the aspects for building structure, water, electricity, gas, and information networks (Figure 3). This simplifies the description of the building and yields plans that can be understood also by non-experts. Starting from the aspect plans for each aspect, the building is build in a well-defined integration process. During this process, the architect often detects failures in his plains and corrects them on-the-fly.

RISE devotes itself to this new paradigm of *integrational* software engineering.

2.1 Entering the Age of Integrational Software Engineering

As outlined above, software engineering research has evolved over time several approaches to component-based systems. Recently, also the first approaches to integrational software platforms have appeared, and Figure 1 summarizes them. From bottom to top, the following techniques and systems can be distinguished. First, 4 classes of component-based systems can be identified:

Modular systems. These systems connect modules by linkers.

Object-oriented systems. These systems allow for dynamic dispatch of objects, i.e., for dynamic connection of runtime modules.

Classical component systems, such as CORBA [Sie98]. They add standardization and default

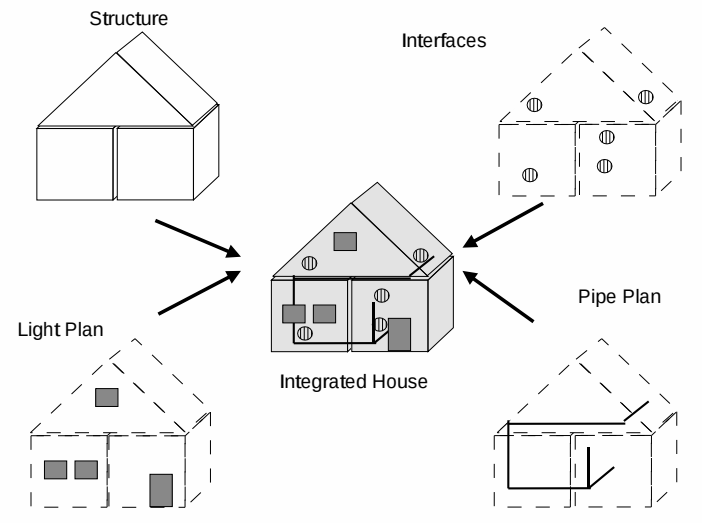

Figure 3: Aspects in buildings are always separated in plans.

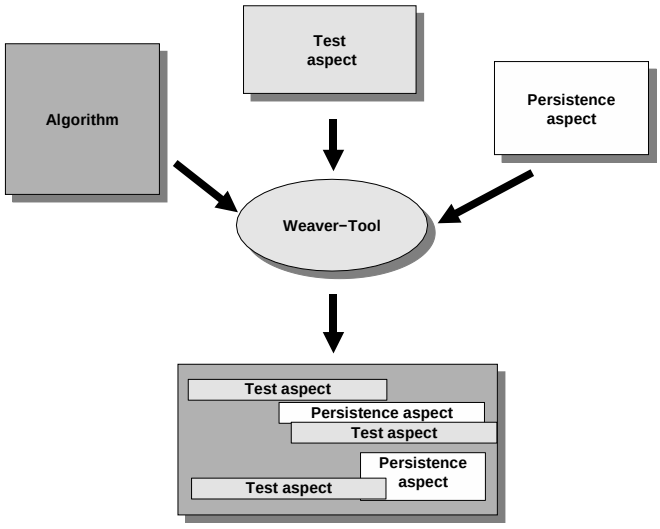

Figure 4: Integration of Software Aspects in an Aspect-Oriented Programming System (*Weaving*).

services to the former two classes.

Architecture systems [GS93] [MDK92] [GAO95] These systems allow for connectors, explicit components for communication. These can be varied independently of the application-specific components which increases reuse.

All these systems have in common that software components are composed in the style of LEGO: components are not integrated, but composed in a simple black-box way. The next 4 classes go one step further since they are able to merge components. They form specific instances of *integrational software engineering* methods.

Aspect-oriented systems [KLM+97] [EFB01] In those systems, aspects of software are woven together by weaver tools (Figure 4).

Systems with composition operators These approaches define explicit composition operators which merge, combine and integrate software components [HO93b] [CHOT99]. They go beyond aspect-oriented programming since they break down the weaver into composition operations, making composition more explicit.

Software Composition Systems These approaches define composition languages, in which composition programs can be described [Aßm98] [AL99] [AGB00] [AGL+02]. Based on a composition assembler with integration operators as the systems before, they additionally provide the possibility to employ a turing-complete language to describe composition. In 2002, a book will be published on *Invasive Software Composition*, one of the only existing approaches, a specific instance for Java composition of Java components [Aßm02]. The technology is demonstrated by the COMPOST composition system, currently under development at PELAB and University of Karlsruhe, Germany [ALN00]. COMPOST is open source software, and a commercial version is marketed by XPTools AB, Linköping [A.B01].

Uniform XML Composition Systems These systems will be able to work uniformly on XML documents, both data and software. In particular, complex documents, such as web sites or workflow descriptions, will be managed. A first version of such a system is under development at PELAB [Con00b] [Con00a]. XML-COMPOST will extend the refactoring and invasive composition techniques of COMPOST to XML. XML-COMPOST will allow uniform evolution of XML based docware, and will be the first tool of its kind world wide.

Dynamic Composition Systems These systems will be able to execute composition programs at run time of a system. Their architecture can be reconfigured during the system run time, allowing dynamic applications in pervasive computing and dynamic application provitioning (DAP). To this end, they will have to regenerate and recompile architectural code, execute weavers at run-time, and reload code dynamically.

The classes 5-9 are the beginning of a new era of *integrational systems*, which provide more powerful techniques for component integration, such as constraint-based integration, rule-based integration, and other mechanisms. However, this era has just begun, and more complex methods to integrate software parts will be developed in the next 30 years.

It is the goal of RISE to develop methods, techniques and tools for integrational software systems. Its seed project, RISE-I, investigates into *software composition systems*, *uniform XML composition systems*, *dynamic composition systems*, and applies this technology to *integrational requirements and test evolution* in an integrated development environment.

References

[A.B01] XPTools A.B. eXtreme Programming tools home page. http://www.xptools.com, Linköping, Sweden, October 2001.

[ABV92] Mehmet Aksit, Lodewijk Bergmans, and Sinan Vural. An object-oriented language-database integration model: The composition-filters approach. In Ole Lehrmann Madsen, editor, *Proceedings of the 6th European Conference on Object-Oriented Programming (ECOOP)*, volume 615 of *Lecture Notes in Computer Science*, pages 372–395, Berlin, Heidelberg, New York, Tokyo, June 1992. Springer-Verlag.

[AGB00] Uwe Aßmann, Thomas Genßler, and Holger Bär. Meta-programming Grey-box Connectors. In R. Mitchell, editor, *TOOLS Europe 2000*, St. Malo, France, June 2000. IEEE Press.

[AGL+02] Uwe Aßmann, Thomas Genßler, Andreas Ludwig, Dirk Heuzeroth, and Rainer Neumann. Refactoring and beyond. *Software - Tools and Techniques*, 2002. submitted to special edition on object-oriented refactoring tools.

[AL99] Uwe Aßmann and Andreas Ludwig. Introducing Connections into Classes with Static Metaprogramming. In Paolo Ciancarini and Alexander Wolf, editors, *3rd Int. Conf. on Coordination*, volume 1594 of *Lecture Notes in Computer Science*, Heidelberg, April 1999. Springer.

[ALN00] Uwe Aßmann, Andreas Ludwig, and Rainer Neumann. COMPOST home page. http://i44w3.info.uni-karlsruhe.de/~compost, March 2000.

[Aßm98] Uwe Aßmann. Meta-programming Composers In Second-Generation Component Systems. In J. Bishop and N. Horspool, editors, *Systems Implementation 2000 - Working Conference IFIP WG 2.4*, Berlin, February 1998. Chapman and Hall.

[Aßm02] Uwe Aßmann. *Invasive Software Composition.* Submitted habilitation, Universität Karlsruhe, 2002.

[CHOT99] Siobhán Clarke, William Harrison, Harold Ossher, and Peri Tarr. Subject-oriented design: Towards improved alignment of requirements, design and code. In *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 325–339, 1999.

[Con00a] The EASYCOMP Consortium. Easy composition in future generation component systems (easycomp). 5th Framework Project, Future and Emerging Technologies (FET), European Commission, http://www.easycomp.org, June 2000 - May 2003, June 2000.

[Con00b] The EASYCOMP Consortium. EASYCOMP home page. http://www.easycomp.org, August 2000.

[EFB01] Tzilla Elrad, Robert E. Filman, and Atef Bader. Aspect-oriented programming. *Communications of the ACM*, 44(10), October 2001.

[GAO95] David Garlan, Robert Allen, and John Ockerbloom. Architectural mismatch: why reuse is so hard. *IEEE Software*, 12(6):17–26, November 1995.

[GS93] David Garlan and Mary Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, volume 1, pages 1–40, Singapore, 1993. World Scientific Publishing Company.

[HO93a] William Harrison and Harold Ossher. Subject-oriented programming (A critique of pure objects). In Andreas Paepcke, editor, *OOPSLA 1993 Conference Proceedings*, volume 28 of *ACM SIGPLAN Notices*, pages 411–428. ACM Press, October 1993.

[HO93b] William Harrison and Harold Ossher. Subject-oriented programming (A critique of pure objects). In Andreas Paepcke, editor, *OOPSLA 1993 Conference Proceedings*, volume 28 of *ACM SIGPLAN Notices*, pages 411–428. ACM Press, October 1993.

[KLM+97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopez, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *ECOOP 97*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer-Verlag, 1997.

[MDK92] Jeff Magee, Naranker Dulay, and Jeffrey Kramer. Structuring parallel and distributed programs. In *Proceedings of the International Workshop on Configurable Distributed Systems*, London, March 1992.

[NM95] Oscar Nierstrasz and Theo Dirk Meijler. Research directions in software composition. *ACM Computing Surveys*, 27(2):262–264, June 1995.

[Sie98] Jon Siegel. OMG overview: CORBA and the OMA in enterprise computing. *Communications of the ACM*, 41(10):37–43, October 1998.

[TOHS99] Peri Tarr, Harold Ossher, William Harrison, and Stanley M. Sutton, Jr. N degrees of separation: Multi-dimensional separation of concerns. In *Proceedings of ICSE'99*, pages 107–119, Los Angeles CA, USA, 1999.