Panel

Component-Based Software Engineering (CBSE)

Organizers: W. (Voytek) Kozaczynski and Jim Q. Ning

Center for Strategic Technology Research (CSTaR)
Andersen Consulting Technology Park
3773 Willow Road, Northbrook, IL 60062-6212
{voytek, jning}@cstar.ac.com

This panel will focus on Component-Based Software Engineering (CBSE), an approach to software development based on extensive use of pre-existing, standard (or customizable) components. CBSE has received considerable attention recently, since it promises to bring to software development the engineering discipline that has long been accepted by more mature engineering domains.

Central to CBSE is the notion of a component, which in our definition has the following characteristics:

- It is a large-grain system building block. Its granularity is larger than functional procedure or object classes. It corresponds closer to the concept of an application or executable.
- It embodies some meaningful functionality. It carries out a unique subset of a system's functional behavior.
- It is a unit of distribution and configuration in that a component can be independently manufactured, packaged, and distributed for reuse with little consideration of other related components.
- It can be a subsystem itself in that it may consist of other components (subcomponents).

CBSE is based on the assumption that the future software industry will be supported by a component market, as illustrated in the figure below.

In this market there will be the role of component brokers. Brokers will set standards for commodity components and will support their inventory, pricing, and distribution. An example of a potential future component broker would be the Object Management Group (OMG). Another role in this market will be that of component producers. It will be played most likely by companies like Microsoft which manufacture and maintain commodity components. Component producers will also accept orders directly from customers to manufacture custom, or more domain-specific components. Finally, there will be the role of component integrators. Integrators add value to existing components by assembling them in unique ways to solve particular business problems. Andersen Consulting may be an example of an integrator.

This panel will invite leading researchers and visionaries from industry, government, and academia to brainstorm on the following subjects related to CBSE:

- **Market vision.** The panel discussion will center around the question: what a component-based software market look like in the future? Would such a market really emerge? Is it already emerging? What are the key forces in this market? What roles will different organizations in industry, government, and universities play in such a market?

- **Definitions.** What do we really mean when we talk about some of the key concepts to CBSE such as components, architectures, component assembly/integration, etc.? How do these concepts relate to similar concepts in other areas (e.g., frameworks, patterns, domain-specific software architectures, Internet/Java).

- **Enabling technologies.** CBSE demands new technologies and processes. What are those technologies? What roles will integration middleware layers such as CORBA and OLE play? What experience do we have in researching and developing these technologies?

The following panelists will offer their views on the above issues.
Dr. Ted Biggerstaff is Research Program Manager at Microsoft Research for the Intentional Programming Project, an environment for developing reusable componentry. As Director of Design Information Research at Microelectronics and Computer Technology Corporation, he led the development of a design-recovery and maintenance system for legacy code. While manager of Software Reuse Research at ITT, Biggerstaff organized the first workshop on software reuse (1983). He has written two books and many papers on reuse, design recovery, reengineering, and software tools.

Biggerstaff received a PhD in computer science from University of Washington. He is a member of the Washington Technology Center Advisory Committee on Software, IEEE Software magazine's Industrial Advisory Board, the IEEE Computer Society, and the ACM. His address is Microsoft Research, One Microsoft Way, MS 9S/1032, Redmond, WA 98052-6399.

Position Statement: Componentry - Two Markets and One Paradigm Shift

It is said that "Prediction is difficult, especially with respect to the future." Indeed, few soothsayers get the sooth right very far into the future. Nevertheless, soothsaying is fun. So, with the disclaimer that we are sure to be wrong to some degree, let's try to predict the future of the componentry market.

I believe that there will actually be two different kinds of componentry markets separated by a paradigm shift. The first market, which will last perhaps five or more years, will just be a continuation and perfection of the componentry market of today. The main players are likely to be OLE, CORBA, and Java based components and they are likely to spend this phase of the market "exchanging DNA." That is to say, driven by market forces, the competing technologies will be incorporating the good ideas and key features of each other. By the end of this phase of the market, I believe that the competing technologies will be conceptually if not actually indistinguishable and mostly interoperable. Much of this change will happen rapidly because it will be driven by the market changes engendered by the Internet and the World Wide Web. More and more web pages will incorporate components. Further, in very short order, virtually all applications will be net-enabled and all applications will just view the net as a transparent extension of their local data storage. Thus, operating on the net will evolve rapidly from mostly passive browsing using special browsers to active application processing using applications on net-enabled OSs. Components will be at the center of much of this evolutionary change. But I predict that a funny thing will happen on the way to Nirvana. In a few years, the developers will begin to have trouble scaling "concrete" components (i.e., components that are written in conventional programming languages such as C++ or Java). They will run into what I have called the "vertical/horizontal scaling dilemma" and this will hinder progress. In short, the vertical/horizontal scaling dilemma is the difficulty of simultaneously scaling componentry in both feature variation (i.e., horizontally) and component sizes (i.e., vertically). This dilemma is inherent to today's mainstream programming languages. Every choice allowed by today's mainstream programming languages produces bad consequences -- i.e., exponentially exploding libraries or bad execution time performance or marginally usable components (i.e., components that hue to obsolete standards and do not address the current technology needs). (See also T. Biggerstaff, "The Library Scaling Problem and the Limits of Concrete Component Reuse," 3rd Int'l Conf. on Software Reuse, 1994, Rio de Janeiro, Brazil, IEEE CS Press.)

I believe that the vertical/horizontal scaling dilemma will produce a paradigm shift toward libraries of "factors" where factors are pure abstractions and pure features that can be composed in combinatorially many ways. A "pure" abstraction or feature is one that makes no commitment to any implementation and is not directly mappable into conventional computer language based constructs. If one adds a little generation technology, then these compositions of factors can be turned into customized conventional components (e.g., COM/OLE components) that contain exactly the right set of abstractions and set of features, mixed in exactly the right way. Further, the custom components will execute as fast as hand coded components. For example, a factored version of the Booch library would contain fewer than 50 seed factors from which many times the 500+ Booch components could be generated. Current research has proved the feasibility of such factored libraries. (Vivek Singhal, personal communication).

This scenario is a plausible theory for how the events will unfold but by no means exact. Many elements of the actual scenario will be slightly different. For example, conventional componentry may be able to stretch its useful life beyond five to seven years. As evidence, note that the VBX componentry market is rapidly growing and shows little sign of topping out. Nevertheless, in the long run, I believe that constraints imposed by the vertical/horizontal scaling dilemma will force a shift to factored libraries even though the predicted timing may be off. And I believe that factored libraries will realize many of the unfulfilled promises of reuse. And lead to the next problem and the next paradigm shift.
Dr. Barry Boehm is TRW Professor of Software Engineering, Computer Science Department, University of Southern California. He received his B.A. degree from Harvard in 1957, and his M.S and Ph.D. degrees from UCLA in 1961 and 1964, all in Mathematics. Between 1989 and 1992, he served within the U.S. Department of Defense (DoD) as Director of the DARPA Information Science and Technology Office and the Software and Intelligent Systems Technology Office. as Director of the DDR&E Software and Computer Technology Office, and as Director of two major DoD software initiatives: the DoD Software Technology Plan and the DDR&E Software Action Plan. He worked at TRW from 1973 to 1989, culminating as Chief Scientist of the Defense Systems Group, and at the Rand Corporation from 1959 to 1973, culminating as Head of the Information Sciences Department. He is currently Director of the USC Center for Software Engineering.

His current research interests include software process modeling, software requirements engineering, software architectures, software metrics and cost models, software engineering environments, and knowledge-based software engineering. His contributions to the field include the Constructive Cost Model (COCOMO), the Spiral Model of the software process, and two advanced software engineering environments: the TRW Software Productivity System and Quantum Leap Environment.

He has served on the editorial boards of several scientific journals, including the IEEE Transactions on Software Engineering, IEEE Computer, IEEE Software, ACM Computing Reviews, and Information and Software Technology. He has served as Chair of the AIAA Technical Committee on Computer Systems, Chair of the IEEE Technical Committee on Software Engineering, and as a member of the Governing Board of the IEEE Computer Society. His honors and awards include Guest Lecturer of the USSR Academy of Sciences (1970), the AIAA Information Systems Award (1979), the J.D. Warnier Prize for Excellence in Information Sciences (1984), the ISPA Freiman Award for Parametric Analysis (1988), and the NSIA Grace Murray Hopper Award (1989). He is an AIAA Fellow and an IEEE Fellow.

Dr. W. (Voytek) Kozaczynski is the Director of the Software Engineering Laboratory at the Andersen Consulting's Center for Strategic Technology Research which is an applied research organization working on multiple aspects of software development processes and tools. He is responsible for the research directions and coordination of research initiatives and projects of the lab. He is also actively involved in projects as a Senior Researcher.

His current research interests include reuse of domain-specific architectures and components, compositional approaches to system development, and distributed systems design. He is also interested in application of object-oriented software development approaches and techniques. Previously he has worked on automatic software analysis and understanding, software re-engineering, application of AI to software development, and database systems.

He has a MS degree in Computer Science and a Ph.D. in Organization and Management from the Technical University of Wroclaw, Poland. Prior to joining Andersen Consulting he was an Associate Professor at the Department of Information and Decision Sciences, University of Illinois at Chicago. He has published over 20 journal and conference papers and is a member of the Industrial Advisory Board of the IEEE Software Magazine.

Position Statement

Most of the discussions on how to make future software solutions composable from reusable software parts has been conducted in what I call technology space. Most frequently addressed subjects include software architectures, domain modeling, component specifications, means to interconnect components, means to communicate component semantics, etc. Much fewer discussion occurs in what I call business space. This is the space of different market forces that will provide incentives for companies to produce, sell and use standard components.

I would like to show that future developers, integrators, reseller and users of standard components will face very interesting and not at all trivial choices. In the heart of these choices lay system evolution and component maintenance. System evolution is inevitable both in functional as well as technical dimensions. Suppose that a happy system owner and user decides to introduce a change only to realize that that change is severely constrained by a “standard” component that has been used in the system. Who owns the component? Who can change it? Can an attempt not to change the component...
result in progressive degradation of initially clean system architecture?

I will illustrate the extent of the problems with an example of one component producer and integrator that assembles systems for many clients (component users). Using this example I will try to show a few different scenarios of how the roles and relationships of the component developers and users can be structured in the future. An interesting, and potentially controversial, part of the argument is that none of these scenarios scales up well beyond the single producer/integrator situation.

Dr. Jim Q. Ning is a Senior Researcher at the Center for Strategic Technology Research (CSTaR), Andersen Consulting. He is currently the Principal Investigator of the Component-Based Software Engineering (CBSE) research project, which explores advanced technologies enabling architecture-driven component integration.

Dr. Ning received his Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign. He has published many journal articles and conference papers in the areas of software understanding, program transformation, concept recognition, re-engineering, reuse, and architectures. He has served on Program Committees of many conferences and workshops and is on the Editorial Board of Automated Software Engineering, An International Journal.

Position Statement:

CBSE Enabling Technologies

Predicting what will happen in the future is hard. But predicting what will NOT happen is usually less difficult. We don’t know whether a component market will eventually emerge in the software industry. We cannot tell whether component-based development will ever become a reality. But one thing for sure: until a strong technology base is in place, the component-based market idea will not fly.

At Andersen Consulting’s Center for Strategic Technology Research (CSTaR), we are experimenting innovative technologies that can potentially assist the work of different stakeholders (component producers, brokers, and integrators) in the development of component-based software. Some of the technologies are summarized below.

Architecture Specification Language (ASL)

ASL can be promoted by brokers to set component standard and used by integrators to communicate component production contracts with producers and capture system architectural designs. It is a collection of three sublanguages:

- **Interface Specification Language (ISL).** CORBA IDL is gaining wide acceptance and is emerging as an industry standard for component interface specification. ISL is a pure extension of IDL: any valid IDL specifications are also valid ISL specifications. But ISL has a richer set of semantic constructs supporting descriptions of pre/postconditions of operations, and invariants and protocols of interfaces (a protocol of an interface describes the order of invocation of operations in that interface).

- **Glue Specification Language (GSL).** A key element in GSL is that of components. We distinguish two types of components: primitive components and composite components. A primitive or concrete component consists of a set of provided and required interfaces. Provided interfaces represent classes encapsulated in a component that provide object services to other components. Required interfaces represent external classes and describe object services that the component, as a client, expects from other components. A composite component is equivalent to the concept of a subsystem: it is an abstraction mechanism encapsulating a set of primitive and/or composite components. GSL supports gluing: interconnections between provided and required interfaces can be specified in composite components.

- **Configuration Specification Language (CSL).** An important concept in ASL is the separation of component interfaces from their implementations: it is possible to associate multiple implementation choices to a single component. CSL attaches implementations (code) to components and describes platform-specific attributes of the implementations. Component configuration specifications are primarily used by a system packager (to be discussed later) to generate system executables.

The figure on the next page summarizes key concepts in ASL. The idea that a component supports multiple interfaces is not new; Microsoft OLE and its Component Object Model (COM) has the same concept. What is unique in ASL is its ability to also specify interface semantics (via pre/postconditions, invariants, and protocols), required interfaces of components, and component configuration.
Architecture Design Environment (ADE)

ADE can be used by component integrators to visually construct ASL specifications using graphical notations. Architecture design decisions specified in ASL are verified and analyzed before they are committed into implementation. ADE supports compatibility analysis to determine whether a required interface can be plugged with a provided interface. ADE also helps generate adaptor components to bridge differences between incompatible components.

Reusable Asset Management System (RAMS)

RAMS assists brokers to manage reusable components. Components are organized based on a faceted-classification scheme along several dimensions, or facets, such as application domain, technical architecture, functional architecture, implementation. Sub-facets, or terms, further discriminate the components. Through a graphical user interface, a librarian is able to conveniently classify and retrieve reusable components. RAMS also supports the browsing, retrieval, and submission of components from remote locations on the Internet with a web browser front-end.

Packager

System packaging is a component integration technology and concerns the automated derivation of compilation procedures ("makefiles") from ASL specifications. The packager being developed by our project assumes a CORBA-based product to provide component instantiation and distributed communication support. Specifically, the packaging process involves the extraction of CORBA IDL code from ISL and GSL code, the generation of testing code from pre/postconditions, invariants, and protocols, the invocation of an IDL compiler to generate C++ header files, the generation of makefiles based on composition requirements and implementation constraints specified in GSL and CSL code, and the execution of the makefiles to finally create system executables.

The selection of a CORBA product does not restrict the generality of our approach; the design of the packager is independent of the underlying execution architecture. Additional packaging tools can be incrementally developed to support the generation of code compatible with other distributed execution architectures (e.g., Microsoft OLE).

Dr. Richard Mark Soley is Vice President and Technical Director of the Object Management Group, Inc. (OMG). In this position, Dr. Soley oversees the product direction of the leading industry consortium detailing and publicizing standards in distributed systems using object-oriented development and execution environments.

Dr. Soley also chairs the OMG Technical Committee, which is responsible for producing standards documents, adopting OMG standard technology, and proposal of new technology to the OMG membership and the OMG Board of Directors. Previously, Dr. Soley was a cofounder and former Chairman/CEO of A. I. Architects, Inc., maker of the 386 HummingBoard and other PC and workstation hardware and software. Prior to that, he consulted for various technology companies and venture firms on matters pertaining to software investment opportunities. Dr. Soley has also consulted for IBM, Motorola, PictureTel, Texas Instruments, Gold Hill Computer and others.

A native of Baltimore, Maryland, U.S.A., Dr. Soley holds the SB, SM and Ph.D. degrees in Computer Science and Engineering from MIT.

Position Statement:

Enabling a Component Software Market

Two major factors inhibit the rapid growth of the emergent component software market:

1. widespread, standardized component definition and integration support
2. widespread acceptance of software component purchasing models

My position on this panel is, of course, in the area of point (1) above. The Object Management Group (OMG) has, for seven years, been developing software interface specifications to allow integration of software systems (and therefore leverage reuse) in heterogeneous, distributed settings. The Common Object Request Broker Architecture (CORBA) which underlies the OMG's Object Management Architecture enables applications to:
• advertise services in a distributed system, using a
  implementation language independent Interface
  Specification Language (IDL, which is an object-
  oriented specification language);
• discover available services advertised in that same
  IDL;
• make use of application/component services,
  regardless of host platform, operating system,
  implementation language, network topology, etc.

While there are other specifications for such systems
integration, none are available on such a wide range of
environments today, offering object-oriented integration
services in a distributed, heterogeneous, multi-language
environment. This is the key to legacy systems
integration as well, and the first part of component
software availability. Until component integration is a
straightforward activity, there will never be a component
software market.

What hasn't been worked out yet is how people -- and
enterprises -- will pay for these components in the future.
Components of "applications" (we say "objects") are
likely to be much smaller than today's memory and disk
hogs, and are likely to offer much lighter functionality.
Standardized interfaces to these objects will also enable a
"pick and choose" (some say "best of breed") software
integration strategy.

These forces will combine with the continuing drop in
software pricing to create a crisis in software purchasing.
It is one thing to say "you'll pay less for your software
because you only buy (and integrate) what you need."
That implies a pricing and purchasing model in which
the point of sale is the users' desktops (rather than a retail
store). Well and good, but can you then tell the
company's CFO how much he'll be spending on software
next year? And who will pay for the new appliance in
your desktop which monitors software usage and arranges
payment? (I don't know about you, but getting me to pay
extra for a service that makes me pay more money will be
difficult).

I contend that the integration problem has solutions
staring us in the face; but we need to think hard about the
pricing problem.