Repairing Coordination Mismatches among Legacy Components
Panel Position Statement

Kurt C. Wallnau
Software Engineering Institute
Carnegie Mellon University
kcw@sei.cmu.edu

1 Increased Demand for Building Systems from Disparate Components

An increasingly commonplace reengineering problem involves the integration of legacy software components into new and often novel configurations. These scenarios are arising more frequently—in large measure because of the success of network computing concepts, and in particular the success of web-based Intranets. Consumers of information technology are no longer satisfied with “stovepipe” systems, but instead have been conditioned by the apparently effortless integration of systems via the web. Enterprises that have built generations of loosely (or uncoupled) information systems are now scrambling to integrate them into an organic whole.

In contrast to reengineering problems involving the restructuring of intact existing systems, the class of reengineering problem alluded to above involves the integration of components drawn from disparate systems. This introduces problems not found with intact systems—problems that are sometimes referred to as “architectural mismatch.” This term was coined by Dave Garlan to describe situations in which software components are resistant to integration because they exhibit one or more forms of incompatibility, or “mismatch.” Distributed object technologies (DOT) such as CORBA are useful because they address whole classes of mismatch (e.g., programming language, operating system). However, DOT is no panacea: components may exhibit a wide array of mismatches not directly addressed by DOT. What is needed is both a classification of mismatches and the development of repair strategies that can be used to ameliorate particular mismatches.

2 Classifying and Repairing Mismatches

One way of classifying mismatch is to consider three dimensions of assumptions: mechanism, function, and coordination. The mechanism dimension refers to assumptions derived from a component’s underlying hardware, operating system, programming language, and so forth. As noted above, DOT addresses may of problems arising from mechanism mismatch. The function dimension refers to assumptions about what services are required of a component, for example the degree of accuracy provided by a statistical modeling package. The coordination dimension refers to assumptions about how components interact with each other, for example whether service requests are synchronous, various quality of service attributes, security, and so forth.

At the SEI we have focused our attention on exploring repair strategies centered on the coordination dimension. Two different repair patterns have emerged: structural abstraction and adaptive abstraction. Structural abstraction refers to the encapsulation of components behind interfaces that express coordination rules rather than functionality. This results in object models that are quite different from the object models usually produced by OOA/OOD. Adaptive abstraction refers to a repair strategy that can be used to ensure non-interference of real-time and non-real-time components in a single system. This results in components that support multiple interfaces, each using different underlying mechanisms but supporting different quality of service.

3 Summary

Use of distributed object technology is only a starting point for integrating legacy systems—also necessary are techniques for diagnosing and repairing a wide variety of architectural mismatches. We believe that focusing on coordination mismatches is particularly fruitful since these raise the most difficult integration problems. In addition, coordination-oriented repair strategies, when combined with distributed object technology, may result in more universally applicable design patterns for integrating off-the-shelf components.