A Taxonomy of Compositional Adaptation

Technical Report MSU-CSE-04-17

May 2004
(Updated July 2004)

Philip. K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, and Betty H. C. Cheng

Software Engineering and Network Systems Laboratory
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan 48824
{mckinley,sadjadis,kasten,cheng}@cse.msu.edu

Abstract

Driven by the emergence of pervasive computing and the increasing need for self-managed systems, many approaches have been proposed for building software that can dynamically adapt to its environment. These adaptations involve not only changes in program flow, but also run-time recomposition of the software itself. We discuss the supporting technologies that enable dynamic recomposition and classify different approaches according to how, when and where recomposition occurs. We also highlight key challenges that need to be addressed to realize the full potential of run-time adaptable software. This survey is intended to be a living document, updated periodically to summarize and classify new contributions to the field. The document is maintained under the RAPIDware project web site, specifically, at **http://www.cse.msu.edu/rapidware/survey**.

Keywords: adaptive software, compositional adaptation, middleware, survey, taxonomy, pervasive computing, autonomic computing, computational reflection, separation of concerns, component-based design, aspect-oriented programming, object-oriented programming.

1 Introduction

Interest in adaptive computing systems has grown dramatically in the past few years, and the research community has responded. A variety of techniques now enable software to adapt dynamically to its current use or environmental conditions. Even the structure of the software itself can be changed during execution to fix errors, improve performance, enhance fault tolerance, or harden security in response to attack. Rooted in the evolutionary and adaptive processes used by natural systems, our understanding of computational adaptation is improving, as we learn and explore new methods for designing and implementing adaptive software.

We view the growing interest in adaptive systems as a response to two ongoing revolutions in the computing field: the emergence of ubiquitous computing and the growing demand for autonomic computing. These areas share several requirements for adaptive behavior, but differ in their overall objectives and strategies for realizing dynamic adaptation.

Ubiquitous computing [1] focuses on removing traditional boundaries for how, when, and where humans and computers interact. To do so, computer systems must adapt to both the surrounding physical environment and the virtual environment of computing platforms and communication networks. In particular, conditions at the "wireless edge" of the Internet are often highly dynamic, requiring that software in mobile devices balance several conflicting and possibly cross-cutting concerns, including quality-of-service on wireless connections, changing security policies, and energy consumption. Research contributions in this area are many and varied, for example [2–20].

Autonomic computing [21] refers to self-managed systems that require only high-level human guidance. Clearly, autonomic computing will play an important role in ubiquitous computing. However, autonomic computing also refers to the ability of systems to manage and protect their own resources. This capability is especially important to systems such as financial networks, transportation systems, water and power systems that must continue to operate correctly during exceptional situations. Such systems require run-time adaptation in order to survive hardware component failures, network outages, and security attacks. Researchers are addressing a variety of issues in software adaptation for autonomic computing, for example [22–29].

Our group is actively participating in the study of adaptive software through our work on two research projects: *RAPIDware* (www.cse.msu.edu/rapidware) addresses adaptive software for protecting critical infrastructures, and *Meridian* (www.cse.msu.edu/meridian) addresses automated software engineering for mobile computing applications. Given the increasing pace of research in adaptive software, we believe the time has arrived to step back and review the reasons for this growth, compare and contrast proposed solutions, and identify key areas that require further study. This document is intended to provide a vehicle to enhance the understanding of this research area and help guide future studies.

This paper is organized as follows. Section 2 introduces compositional adaptation. Section 3 describes key enabling technologies for compositional adaptation. Section 4 provides a taxonomy of compositional adaptation. Section 5 discusses key challenges required to be addressed in future. Section 6 concludes this paper draws conclusions. In the appendix, we summarize numerous projects related to compositional adaptation and classify them according to the taxonomy.

2 Compositional Adaptation

Two general approaches have been used to realize dynamic adaptation in software: parameter and compositional adaptation. *Parameter adaptation* involves the modification of variables that determine program behavior. As described by Hiltunen and Schlichting [30], a well-known example of parameter adaptation is the way that the Internet's TCP protocol adjusts its behavior by changing values that control window management and retransmission in response to apparent network congestion [31, 32]. Recently, parameter adaptation has been used in many *context-aware* systems [11, 13, 33–37], in which software execution is directly affected by the external environment. A weakness of parameter adaptation is that it cannot adopt algorithms or components left unimplemented during the original design and construction of an application. That is, parameters can be tuned or an application can be directed to use a different *existing* strategy, but strategies implemented after the construction of the application cannot be adopted.

In contrast, *compositional adaptation* results in the exchange of algorithmic or structural parts of the system with ones that improve a program's fit to its current environment [30, 38–44]. In comparison to parameter adaptation, compositional adaptation enables an application to adopt new algorithms for addressing concerns unforeseen during original design and construction. The flexibility of compositional adaptation enables more than simple tuning of program variables or strategy selection. Dynamic recomposition is needed when resource limitations (for example, memory in small devices) restrict the number of responding components that can be deployed simultaneously, or when adding new behavior to deployed systems to accommodate unanticipated conditions or requirements (for example, detection of and response to a new

security attack).

The history of compositional adaptation dates back to the EDVAC in the 1940's [45]. In the design of this computer, both program instructions and data were stored in mutable memory, thereby enabling instructions to act on both data and other instructions. This "self-modifying code" was used for dynamic optimizations in programs and offered a work-around solution to limited memory space. Nonetheless, such programming was soon considered bad practice due to issues such as program inconsistency, difficulty in debugging, and vulnerabilities to viruses. However, dynamic recomposition has continued to be used and even well-accepted in certain contexts. Examples include run-time upgrade of highly-available systems such as telecommunication switches [46], hot-swap bug fixing (or on-the-fly code fixing) [47], protecting against software piracy [48, 49], and as a means to *avoid* infection or manipulation by attackers [50].

Over the years, several programming languages have directly supported dynamic recomposition. In the 1970's, Smalltalk introduced meta classes as first-class objects, in which information about classes are stored [51]. 3-LISP, the first language that treats reflection as an important part of the language, was developed in 1982 by Smith [52]. In 1984, Maes developed 3-KRS and demonstrated feasibility of a reflective object-oriented language [53]. In 1991, Kiczales *et. al.* [54] developed CLOS Meta Object Protocols. In 1995, Sun formally announces the Java language [55] and later introduced structural reflection capabilities in Java. Since then, several projects have extended Java with behavioral reflection [44, 56–62].

In the 1990's, interest in compositional adaptation gained considerably, due in part to those who envisioned a world of ubiquitous computing [1], and to those who recognized the corresponding need for self managing systems. Numerous workshops and conferences focused on adaptive software and dynamic recomposition [22, 63–71], and funding programs for this research were created [72–75].

This survey examines and classifies these new approaches to compositional adaptation and attempts to explain the technological reasons behind this rapidly growing area of study. We focus primarily on dynamic recomposition of user-level software, namely, middleware and applications. We emphasize, however, that extensive research has been conducted in several related and often overlapping areas, including design of software to facilitate its evolution [76–91]; software-architecture techniques for supporting dynamic adaptation [67, 92–94]; systems that distill data streams and distribute processing tasks to support resource-limited devices [4, 5, 37, 43, 95, 96]; adaptable and extensible operating systems [97–103]; and dynamic recomposition within the network infrastructure itself [104–107]. Surveys of contributions in these areas would be very useful to the research community.

3 Enabling Technologies

At the core of all approaches to compositional adaptation is a level of indirection that enables interception and redirecton of interactions among program entities. The realization of self-configuring systems has been aided by the confluence of three key technologies: separation of concerns, computational reflection, and component-based design (Figure 1). Together, they provide programmers with the tools to construct self-adaptive systems in a systematic and principled (as opposed to ad hoc) manner [108]. In addition, the widespread use of middleware in distributed computing has served as a catalyst for research in compositional adaptation, since middleware is a natural place to locate many types of adaptive behavior. In this section, we discuss each of these technologies and describe how they support compositional adaptation.

3.1 Separation of Concerns

Separation of concerns [109, 110] enables the separate development of the functional behavior and the cross-cutting concerns (e.g., quality of service, fault tolerance, security) of an application. The functional behavior of an application is sometimes referred to as its *business logic*. This separation simplifies development and

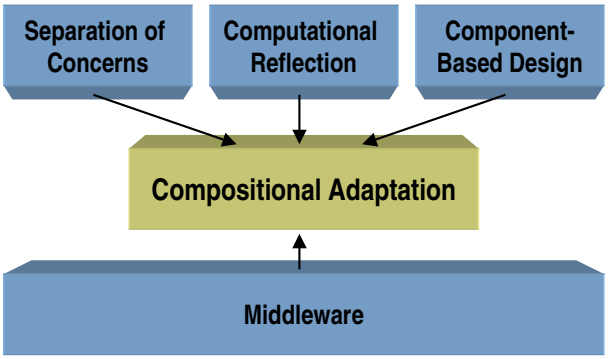


Figure 1: Main technologies supporting compositional adaptation.

maintenance, while promoting software reuse. Separation of concerns has become an important principle in software engineering [111], and many development techniques apply it to some degree. Examples include domain-specific languages, generative programming, generic programming, constraint languages, feature-oriented development, and aspect-oriented programming [78].

Presently, the most widely used approach appears to be aspect-oriented programming (AOP) [112–114]. Kiczales et al. [112] recognized that complex programs include various *crosscutting concerns* (properties or areas of interest such as QoS, energy consumption, fault tolerance, and security). While object-oriented programming abstracts out commonalities among classes in an inheritance tree, crosscutting concerns are scattered among different classes, complicating the development and maintenance of applications. As depicted in Figure 2, AOP enables separation of crosscutting concerns during development of the software. Specifically, the code implementing crosscutting concerns of the system, called *aspects*, are developed separately from other parts of the system. In AOP, locations in the program where aspect code can be woven, called *pointcuts*, are typically identified during development. Later, for example during compilation, an *aspect weaver* can be used to weave different aspects of the program together to form a program with new behavior. AOP proponents argue that disentangling crosscutting concerns leads to simpler development, maintenance, and evolution of software [110, 112]. Examples of AOP approaches include AspectJ [115], Hyper/J [116], DemeterJ (DJ) [117], JAC [59], Kava [118], and Composition Filters [56].

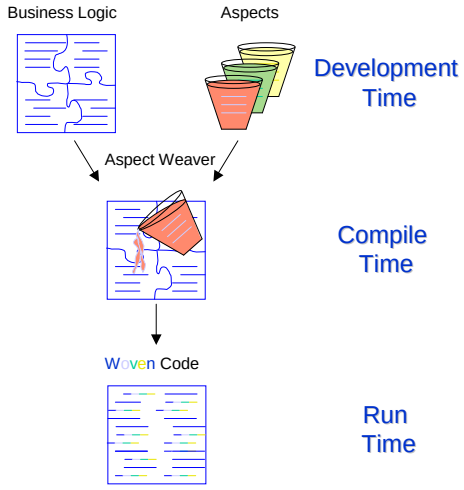


Figure 2: Conceptual representation of aspect-weaving. (Adapted from [112].)

AOP supports dynamic recomposition in three major ways. First, most adaptations are relative to some crosscutting concern, such as quality-of-service or fault tolerance. AOP enables the code associated with

these aspects to be written and managed independently of the application code as well as other parts of the system, such as traditional middleware platforms. Such separation is needed in order to dynamically replace one instantiation of a particular concern with another. Second, although compile-time aspect weaving produces a tangled executable that cannot easily be reconfigured, delaying the weaving process until run-time provides a systematic way to realize dynamic recomposition [119–122]. Finally, if adaptability itself is considered as a "generic" aspect [123, 124], then run-time weaving can be used to enhance the program with adaptive behavior not necessarily anticipated during the original development. (for example, to tolerate newly discovered faults or to detect and respond to new security attacks). This kind of upgrade is especially important in situations where the application is required to run continuously and cannot be easily taken off-line for upgrade.

3.2 Computational Reflection

Computational reflection refers to the ability of a program to reason about, and possibly alter, its own behavior [52, 53]. Reflection enables a system to reveal (selected) details of its implementation without compromising portability [54]. In other words, reflection exposes a system implementation at a level of abstraction that hides unnecessary details, but still enables changes to the system behavior. Reflection comprises two activities. *Introspection* enables an application to observe its own behavior and *intercession* enables a system or application to act on these observations and modify its own behavior [95, 125, 126]. In a self-auditing distributed application, for example, introspection would enable software "sensors" to observe and report usage patterns for various components. Intercession would enable new types of sensors, as well as components that implement corrective action, to be inserted at run-time.

As depicted in Figure 3, a reflective system (represented as *base-level* objects) and its self representation (represented as *meta-level* objects) are *causally connected*, meaning that modifications to either one will be reflected in the other [127]. A *meta-object protocol (MOP)* [54] is an interface that enables "systematic" (as opposed to ad hoc) introspection and intercession of the base-level objects. Moreover, MOPs can be categorized as enabling either structural or behavioral reflection. *Structural reflection* addresses issues related to class hierarchy, object interconnection, and data types. As an example, an object can be examined to determine what methods are available for invocation. Conversely, *behavioral reflection* focuses on the computational semantics of the application. For instance, a distributed application can use behavioral reflection to select and load a communication protocol well-suited to current network conditions.

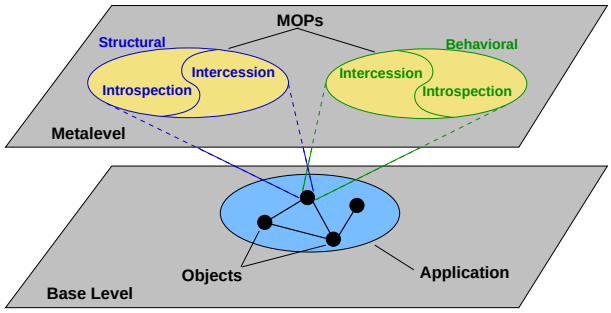


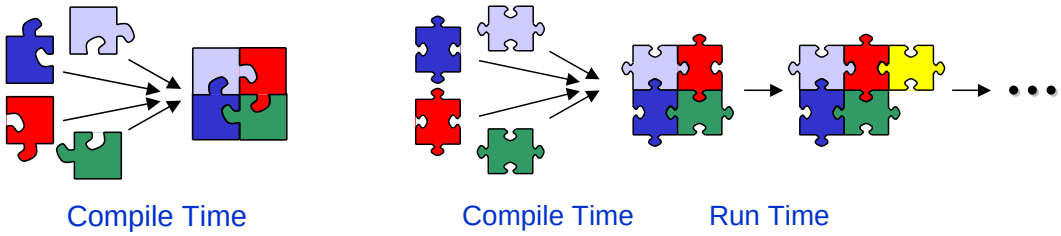
Figure 3: Metalevel understanding collected into metaobject protocols (MOPs).

An adaptive system uses reflection to observe and "reason" about changes to its structure and behavior. Reflection may be provided by a programming language [44, 51, 53, 54, 56–60, 62, 124, 128–137] or may be implemented directly in a middleware platform [61, 108, 118, 126, 138–152]. We note that in general, reflective programming languages are concerned with the representation of a programming language (*e.g.*, constructs, implementation, and interpretation of a programming language) while reflective middleware approaches are concerned with the representation of middleware services (*e.g.*, host-infrastructure, distribution,

common, and domain-specific services). Combined with aspect-oriented programming, reflection enables a MOP to contain logic that weaves crosscutting concerns into an application at run time [120, 153, 154].

3.3 Component-Based Design

The third major contributing factor to dynamic recomposition of software arises from a level of maturity attained in object-oriented design and, more recently, in component-based design. Well-defined interface specifications enable service clients and providers to be developed independently, and potentially by different parties, using the interface as a contract. Figure 4(a) depicts a *static* recomposition approach, in which four components are combined at compile time to produce an application. Of particular importance to dynamic recomposition is binding time. *Late*, or dynamic, binding supports coupling of compatible service clients and providers through well-defined interfaces at run time. As shown in Figure 4(b), new components can be bound to an application at run time. In addition, object-oriented languages use *indirect* interfaces, primarily as a means to support inheritance and polymorphism [155]. Effectively, method calls are redirected to the appropriate method implementation. This level of indirection, when coupled with dynamic class loading and late binding, helps to support compositional adaptation.



(a) Static recomposition.

(b) Dynamic recomposition.

Figure 4: Component-based design enables static and dynamic recomposition.

Another important factor is the independent deployment and versioning of software elements introduced in component-based design [155]. *Software components* are units that can be independently developed, deployed, and composed by third parties [156]. Popular component-based platforms include COM/DCOM [157, 158], .NET [159], CCM [160], and EJB [161].

Component-based design supports two types of composition. In static composition, a developer can combine several components at compile time to produce an application. In dynamic composition, the developer can add, remove, or reconfigure components within an application at runtime. To provide dynamic recomposition, a component-based framework must support late binding, which enables coupling of compatible components at runtime through well-defined interfaces used as contracts. In addition, to provide consistency with other applications, a component-based framework must support coexistence of multiple versions of components.

By enabling the assembly of off-the-shelf components from different vendors, component-based design promotes software reuse. Moreover, mechanisms for maintaining a program's component structure after the initial deployment, when combined with late binding, facilitate compositional adaptation [29].

3.4 Middleware

Much of the recent research in adaptive software focuses on adaptive *middleware*. Middleware is commonly defined as a layer of services separating applications from operating systems and network protocols. Schmidt [3] decomposed middleware into four layers, depicted in Figure A. *Host-infrastructure middleware* resides

atop the operating system and provides a high-level API that hides the heterogeneity of hardware devices, operating systems, and to some extent network protocols. *Distribution middleware* provides a high-level programming abstraction, such as remote objects, enabling developers to write distributed applications in a way similar to stand-alone programs. CORBA, DCOM, and Java RMI all fit in this layer. *Common-middleware services* include fault tolerance, security, persistence, and transactions involving entities such as remote objects. Finally, *domain-specific middleware* is tailored to match a particular class of applications.

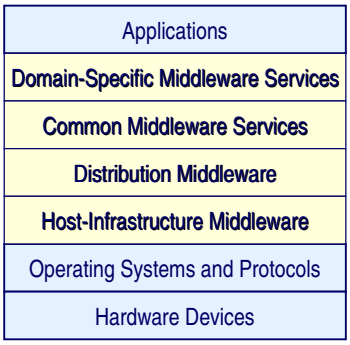


Figure 5: Schmidt's middleware layers.

Since the traditional role of middleware is to hide resource distribution and platform heterogeneity from the business logic of applications, it is a logical place to put adaptive behavior related to various crosscutting concerns such as quality-of-service, energy management, fault tolerance, and security policy.

Most adaptive middleware approaches are based on an object-oriented programming paradigm and derive from popular object-oriented middleware platforms such as CORBA, Java RMI, and DCOM/.NET. Many techniques work by intercepting and modifying messages. Figure 6 shows the flow of a CORBA request/reply sequence in a simplified CORBA client/server application. This application comprises two autonomous programs hosted on two computers connected by a network. Let us assume that the client has a valid CORBA reference to the CORBA object realized by the servant. The client's request to the servant is first received by the stub, which represents the CORBA object at the client side. The stub marshals the request and sends it to the client ORB. The client ORB sends the request to the server ORB, where it is delivered to the servant by way of a skeleton, which unmarshals the request. The servant replies to the request, by way of the server ORB and skeleton. The reply will be received by the client ORB and be dispatched to the client.

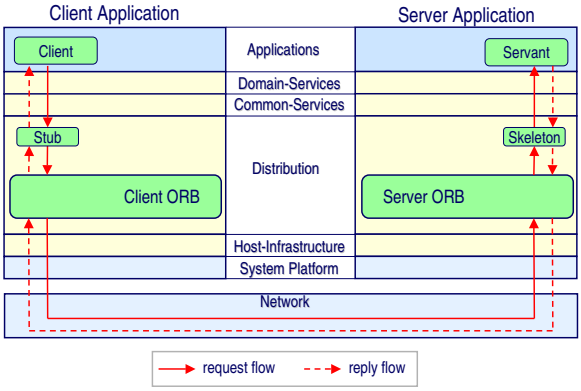


Figure 6: CORBA call sequence.

Whether a given system uses message interception or one of several other approaches, discussed later, middleware effectively provides a level of indirection and transparency that can be exploited to implement

adaptation. Indeed, the majority of approaches to compositional adaptation discussed in this paper are implemented in middleware.

3.5 Other Factors

Although we consider the four technologies described above to provide the primary ingredients needed for dynamic recomposition, many other factors have contributed to the growth in this area. For example, *software design patterns* [162, 163] provide a way to reuse best software designs practiced successfully for several years. The goal of software design patterns is to create a common vocabulary for communicating insight and experience about recurring problems and their known solutions. Schmidt and colleagues [163] have identified a relatively concise set of patterns that enables developing adaptive middleware. For example, the virtual component pattern [164], used in TAO [165] and ZEN [152], enables adapting a distributed application to the memory constraints of embedded devices by providing a small middleware footprint including only a minimum core and a set of "virtual" components, whose code can be dynamically loaded on demand. Numerous adaptive middleware projects [126, 145, 146, 150, 152, 165–170] benefit from the use of adaptive design patterns. Other technologies that have been applied to the design of adaptive software include: generative programming [78], adaptive programming [171], intentional programming [172], open implementation [173], subject-oriented programming [174, 175], composition filters [56], and mobile agents [176–179]. An excellent discussion of most of these methods can be found in [78].

4 A Taxonomy of Compositional Adaptation

Researchers and developers have proposed a wide variety of methods for supporting compositional adaptation. Table 1 lists a number of research projects and commercial software packages that support some form of compositional adaptation. The list is *by no means* exhaustive – many other groups, both in academia and industry, are working on various aspects of this problem. Rather, the list includes representative projects. In this section, we describe a taxonomy that distinguishes approaches by *how*, *when*, and *where* software composition takes place. We use the examples in Table 1 to help explain important distinctions. In the appendix, we provide additional details on the taxonomy and use it to compare and contrast these and other projects.

4.1 How to Compose

The first dimension of our taxonomy is *how* composition is implemented, that is, what specific mechanisms are used to enable compositional adaptation. Table 2 lists several key techniques that have been used, along with examples. Aksit and Choukair [38] provide an excellent discussion of such methods.

All of the techniques in Table 2 create a level of indirection in the interactions between program entities. Some techniques use specific software design patterns to realize this indirection, whereas others use AOP, reflection or both. The two middleware techniques both modify interaction between the application and middleware services, but they differ in the following way: middleware interception is not visible to the application, whereas integrated middleware provides adaptive services invoked explicitly by the application.

We use the term *composer* to refer to the entity that uses these techniques to adapt an application. The composer might be a human – a software developer or an administrator interacting with a running program through a graphical user interface – or a piece of software – an aspect weaver, a component loader, a runtime system, or a metaobject. Indeed, autonomic computing promises that increasingly composers will be software components.

When and where the composer modifies the program determines the transparency of the recomposition. Transparency refers to whether an application or system is aware of the "infrastructure" needed for recomposition. For example, a middleware approach to adaptation is transparent with respect to the application

Table 1: Representative projects and systems involving compositional adaptation.

Projects	Affiliation	References	Web Site
Language-Based Projects:			
Open Java	Tokyo Institute of Technology	[57]	http://www.csg.is.titech.ac.jp/openjava/
FRIENDS	LAAS-CNRS, France	[147]	http://www.laas.fr/ fabre/Friends.htm
PCL	University of Illinois	[132]	http://www-sal.cs.uiuc.edu/ vadve/adaptive.html
Adaptive Java	Michigan State University	[44]	http://www.cse.msu.edu/rapidware/
AspectJ	Xerox PARC	[115]	http://aspectj.org/
Composition Filters	Universiteit Twente, The Netherlands	[56]	http://trese.cs.utwente.nl/composition_filters/
RNTL ARCAD	EMN/INRIA, France	[124, 133]	http://www.emn.fr/x-info/obasco/
TRAP/J	Michigan State University	[62]	http://www.cse.msu.edu/rapidware/
JOIE	Duke University	[180]	http://www.cs.duke.edu/ari/joie/
Kava	University of Newcastle, UK	[118]	http://www.cs.ncl.ac.uk/research/
R-Java	U. Federal de São Carlos, Brazil	[58]	http://www.dc.ufscar.br/
Middleware-Based Projects:			
Domain-Specific Services:			
Boeing Bold Stroke	Boeing Corporation	[181]	http://www.cs.rmit.edu.au/conf/doa/2001/sharp.html
Common Services:			
OGS	Swiss Federal Inst. of Tech.	[182]	http://www.eurecom.fr/ felber/
QuO	BBN Technologies	[168]	http://quo.bbn.com/
FTS	Technion, Israel	[183]	http://www.cs.technion.ac.il/ roy/
IRL	University of Rome, Italy	[184, 185]	http://www.dis.uniroma1.it/ irl/
TAO Load Balancing	DOC Group (Schmidt, et al.)	[169]	http://www.cs.wustl.edu/ schmidt/TAO.html
ACT	Michigan State University	[186, 187]	http://www.cse.msu.edu/rapidware/
Distribution:			
TAO/ZEN/CIAO	DOC Group (Schmidt, et al.)	[152, 165, 167]	http://www.cs.wustl.edu/ schmidt/
DynamicTAO & UIC	University of Illinois	[126, 145]	http://choices.cs.uiuc.edu/2k/
OpenORB & OpenCOM	Lancaster University, UK	[108, 144]	http://www.comp.lancs.ac.uk/computing/research/mpg/
FlexiNet	APM Ltd, UK	[149]	http://www.ansa.co.uk/
Squirrel/Infopipes	Universität Berlin, Germany	[188, 189]	http://wwwagss.informatik.uni-kl.de/Projekte/Squirrel/
AspectIX	Friedrich-Alexander Univ., Germany	[190]	http://www.aspectix.org/
OpenCorba	École des Mines de Nantes	[142]	http://www.emn.fr/x-info/obasco/
Electra/Horus/Isis	Cornell University	[191–193]	http://www.cs.cornell.edu/Info/Projects/HORUS/main.html
Orbix/Isis	IONA Technologies	[194, 195]	http://www.iona.com/
Orbix/E	IONA Technologies	[194, 195]	http://www.iona.com/
Host-Infrastructure:			
ACE	DOC Group (Schmidt, et al.)	[166]	http://www.cs.wustl.edu/ schmidt/ACE.html
Ensemble	Cornell University	[42]	http://www.cs.cornell.edu/Info/Projects/Ensemble/
Metasockets	Michigan State University	[186, 187]	http://www.cse.msu.edu/rapidware/
Eternal/Totem	U. California, Santa Barbara	[196, 197]	http://www.eternal-systems.com/
Rocks/Racks	University of Wisconsin	[198]	http://www.cs.wisc.edu/ zandy/rocks/
PROSE	Swiss Federal Inst. of Tech.	[140]	http://prose.ethz.ch/Wiki.jsp
Iguana/J	Trinity College, Dublin	[61]	http://www.dsg.cs.tcd.ie/ coyote/coyote-documents.html
Guaranà	UNICAMP, Brazil	[141]	http://www.ic.unicamp.br/ oliva/guarana/
Personal/Embedded Java	Sun Microsystems	[199]	http://java.sun.com/products/embeddedjava/
Cross-Layer Projects:			
DEOS	Georgia Institute of Technology	[17, 200]	http://www.cc.gatech.edu/systems/projects/DEOS/
GRACE	University of Illinois	[201]	http://rsim.cs.uiuc.edu/grace/
Graybox	University of Wisconsin	[202]	http://www.cs.wisc.edu/graybox/

source code if the application does not need to be modified to take advantage of the adaptive features. Different degrees of transparency (with respect to application source, virtual machine, middleware source, and so on) determine both the portability of a proposed solution and how easily new adaptive behavior can be added to existing programs.

Table 2: Compositional adaptation techniques.

Technique	Description	Examples
Function pointers	Application execution path is dynamically redirected through modification of function pointers.	Vtables in COM, delegates and events in .NET, callback functions in CORBA, ACE
Wrapper pattern	Objects are subclassed or encapsulated by other objects (wrappers), enabling the wrapper to control method execution.	PCL, Adaptive Java, ARCAD, TRAP/J, R-Java, QuO, ACE, MetaSockets
Proxy pattern	Surrogates (proxies) are used in place of objects, enabling the surrogate to redirect method calls to different object implementations.	OGS, FTS, IRL, TAO-LB, ACT, Squirrel/Infopipes, AspectIX, ACE, Racks, Guaranà
Strategy pattern	Each algorithm implementation is encapsulated, enabling transparent replacement of one implementation with another.	PCL, TAO, ZEN, CIAO, DynamicTAO, UIC, ACE
Virtual component pattern	Component placeholders (virtual components) are inserted into the object graph and replaced as needed during program execution.	TAO, ZEN, CIAO
Meta-Object Protocol	Mechanisms supporting intercession and introspection enable modification of program behavior.	Open Java, FRIENDS, PCL, Adaptive Java, AspectJ, Composition Filters, ARCAD, TRAP/J, JOIE, Kava, R-Java, DynamicTAO, UIC, Open ORB/COM, FlexiNET, OpenCORBA, MetaSockets, PROSE, Iguana/J, Guaranà
Aspect weaving	Code fragments (aspects) that implement a cross-cutting concern are woven into an application dynamically.	AspectJ, Composition Filters, ARCAD, TRAP/J, AspectIX, PROSE
Middleware interception	Method calls and responses passing through a middleware layer are intercepted and redirected.	FTS, IRL, TAO-LB, ACT, Eternal, Rocks, Racks, PROSE, Iguana/J, Guaranà
Integerated middleware	An application makes explicit calls to adaptive services provided by a middleware layer.	OGS, QuO, TAO, ZEN, CIAO, DynamicTAO, UIC, Open ORB/COM, FlexiNET, Squirrel, AspectIX, OpenCorba, Electra, Orbix/Isis, Orbix/E, ACE, Ensemble, MetaSockets, Personal Java, Embedded Java

4.2 When to Compose

Second, we differentiate approaches according to *when* the adaptive behavior is composed with the business logic. Generally speaking, later composition time supports more powerful adaptation methods, but also complicates the problem of ensuring consistency in the adapted program. For example, when composition take place at development, compile or load time, dynamism is limited, but it is easier to ensure that the adaptation will not produce anomalous behavior. On the other hand, while run-time composition is very powerful, it greatly complicates the task of assuring the safety and correctness of the program.

Figure 7 illustrates the use of composition time as the classification metric for adaptive applications. On the vertical axis are application types that implement either *static* or *dynamic composition*. Static composition refers to composition methods that take place at development, compile or load time, whereas dynamic composition includes methods that can be applied at run time.

Static Composition. If a program is composed at development time, then any adaptability is *hardwired* into the program – the adaptive behavior cannot be changed without recoding. Alternatively, a limited form of adaptation can be implemented at compile time or link time by changing how an application is composed. Different compositions can address the requirements of a different environment, such as a new computing

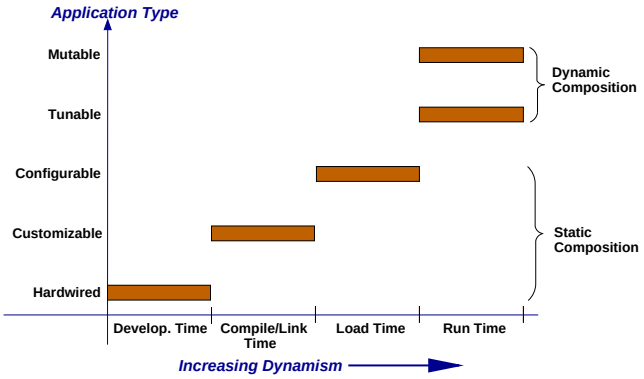


Figure 7: Classification for software composition using the time of composition or recomposition as a classification metric.

platform or network type. For example, aspect-oriented programming languages such as AspectJ [115] enable weaving of aspects into programs during compilation. Aspects might implement an environment-specific security or fault-tolerance policy. Such *customizable* applications require only recompilation or relinking to be fitted to a new environment.

Load-time composition allows the final decision on which algorithmic units to use in the current environment to be delayed until the component is loaded by a running application. For instance, the Java Virtual Machine (JVM) loads classes when they are first used by a Java application. Although load-time composition is considered as a type of static composition, it enjoys increased dynamism over the other static methods. When the application requests the loading of a new component, decision logic might select from a list of components with different capabilities or implementations, choosing the one that most closely matches current needs. As such, an application can be *configured* based on when or where a program is used. For instance, if an application is started on a handheld computer rather than a desktop, a minimal display component might be loaded to guarantee proper presentation. Other load-time approaches work by dynamically modifying the class itself as it is loaded. For example, Kava [118] provides a class loader that rewrites the bytecode of classes as they are loaded to provide run-time monitor and debugging capabilities.

Dynamic Composition. The most flexible approaches are those that implement run-time composition: algorithmic and structural units can be replaced or extended during execution without halting and restarting the program. We differentiate two types of approaches according to whether or not the business logic of the application can be modified.

Tunable software prohibits modification of functional code. Tunable software enables fine-tuning in response to changing environmental concerns, such as dynamic conditions encountered in mobile and pervasive computing environments. An example is the fragment object model used in AspectIX [190], which enables the distribution behavior of a CORBA-compliant application to be tuned at run time.

In contrast, *mutable* software allows even the imperative function of the program to be changed, enabling a running program to be dynamically recomposed into one that is functionally different. For example, in the OpenORB middleware platform [108] all objects (in the middleware and application code) have reflective interfaces, so at run time virtually any change can be made to any object (change the interface, implementation, etc). While very powerful, in most cases this flexibility needs to be constrained to ensure the integrity of the system across adaptations.

4.3 Where to Compose

The final dimension in which we compare approaches to compositional adaptation is according to *where* in the system the adaptive code is inserted. The possibilities include one of the middleware layers or the

application code itself (see Figure 5). Another approach is to use an extensible operating system [97–102, 203–206] or to extend a commodity operating system to support dynamic recomposition [207–218]. As our focus is on application-level adaptation, we do not discuss these approaches further. However, we note that in several *cross-layer* adaptive frameworks (e.g., Odyssey [219], DEOS [200], GRACE [201], and Graybox [202]) middleware cooperates with the operating system to provide adaptation.

Projects involving compositional adaptation at the host-infrastructure middleware layer generally fall in one of two groups. The first includes those approaches that construct a layer of adaptable communication services, for example [42, 166, 196, 198]. An early example that used object-orientation is ACE [166], which uses a service configurator pattern and C++ dynamic binding to support dynamic composition. The second approach is to extend a virtual machine with facilities to intercept and redirect interactions in the functional code. Given the popularity of Java, many of these projects extend the JVM; example projects include Iguana/J [61], Meta Java [139], Guaraná [141], PROSE [140], R-Java [58], and JDrums [220]. For example, R-Java [58] enables the class of an object to be changed (to a subclass) at run-time by adding a new instruction to the Java interpreter. In general, approaches in this category are very flexible with respect to dynamic reconfiguration, in that new code can be introduced at run time. However, while they provide transparency to the application, they are not transparent to the JVM, reducing their portability.

Introducing adaptive behavior in the distribution or common services middleware layers enables both portability across platforms and transparency with respect to the application program. A large number of adaptive software projects fall in one of these categories, and those listed in Table 1 comprise only a small representative sample. Examples at the distribution layer include TAO [165], DynamicTAO [126], CIAO [167], ZEN [152], ORBacus [195], Squirrel [188], UIC [145], LegORB [146], OpenORB [108], Quarterware [221], mChaRM [148], Electra [191], and FlexiNet [149] Examples at the common services layer include QuO [168], AQuA [170], IRL [184], TAO load balancing [169], and ACT [186]. These approaches typically involve intercepting messages associated with remote method invocations and processing them in a manner that accounts for current internal or external conditions. Messages might be redirected, modified, or even deleted. In addition to providing transparency to the functional code, some approaches (e.g., IRL [184], Eternal [196], and ACT [186, 187]) even provide transparency to the distribution middleware code, that is, they can be integrated into the program without modification to either the application *or* the underlying middleware platform.

Middleware approaches are an effective means to support adaptability, but they are applicable only to programs that are written against a specific middleware platform. The most general approach to supporting compositional adaptation is to provide this capability directly in the application program itself. Two main techniques are used. The first is to program all or part of the application code using a language that supports recomposition inherently, such as CLOS [54] or Python [128]. In addition, a number of programming language extensions have been introduced recently to facilitate the development of adaptive code. Many extend Java (e.g., Open Java [57], R-Java [58], Handi-Wrap [134], Adaptive Java [44]) to include new keywords and constructs to enhance the expressiveness of the adaptive code. However, this approach does not enable transparent composition of adaptive behavior with existing applications.

The second technique is to weave the adaptive code into the functional code at either compile or load time. To enable the application to be extended at run time, a two-phase approach can be used [123, 124]. In the first phase, interception hooks are prepared at compile time using static weaving of aspects. In the second phase, intercepted operations are forwarded to adaptive code using reflection. This approach is used in TRAP/J [62], DADO [222], Reflex [133], and the OBASCO method [124].

5 Key Challenges

Despite many advances in the mechanisms needed to support compositional adaptation, we contend that the full potential of dynamically recomposable software systems can be realized only with fundamental advances on four other fronts.

Assurance. The design of recomposable software must be supported by a programming paradigm that lends itself to *automated* checking of both functional and nonfunctional properties of the system. First, in order to help ensure the correctness of the adapted system, all components to be composed into the system must be certified for correctness with respect to specifications. This assessment can be obtained in two ways: (a) selecting components that have already been verified and validated offline using traditional techniques, such as testing, inspection, and model checking; and (b) generating code automatically from specifications [62, 223]. In addition to functional requirements, certification can also include non-functional requirements, such as security and performance. Second, during the adaptation process, techniques are needed to ensure that the system continues to execute in an acceptable, or *safe* manner. Our group and others are currently investigating this problem using a variety of methods, including dependency analysis [39, 206, 224–227] and explicit management of state information [71]. Third, high-level contracts [228] and invariants can be used to define policies for adaptation, feature interaction, and the correctness of the system before, during, and after adaptation. The contracts and invariants, including those for synchronization, behavioral, and QoS, can focus on individual layers or address cross layers [201, 229–236].

Security. Whereas assurance deals primarily with system integrity, security deals with protecting the system from malicious entities. An important issue is how to prevent the adaptation mechanisms from being exploited by a would-be attacker. In addition to verifying the sources of inserted components, the core of an adaptive software system must be extremely well protected from attackers through the usual set of security tools (e.g strong encryption to ensure the confidentiality and authenticity of messages related to adaptation). However, an attacker who is able to penetrate weaker perimeter defenses may have adequate power to perform the traffic analysis necessary to observe adaptive actions, which may themselves be part of system security. An interesting research topic is how to hide the management of adaptation from would-be intruders and prevent them from impeding or corrupting the adaptive process [237, 238]. A complete approach to this problem not only must ensure the integrity of data used in decision-making, but must also conceal adaptive actions from would-be intruders.

Interoperability. Designing distributed systems that can adapt to their environments requires not only adaptation of individual components, but coordinated adaptation across system layers and across platforms [145, 200–202, 239]. Different components are likely to have been developed by different parties, and the developer must be able to integrate separately-designed adaptive mechanisms such that they cooperate to meet the needs of the application. The problem is that many of the adaptive software solutions proposed for different layers have been developed independently, and even solutions within the same layer are often not compatible. Tools and methods are needed to enable developers to (1) integrate the operation of adaptive components across layers of a single system and among different systems; and (2) support adaptability in legacy systems and interoperability among different adaptive frameworks. Topics already under study include application integration using web services [240] and adaptive behavior in sensor networks [241–244].

Decision Making. In confronting a dynamic physical world, *decision making* in adaptive systems must modify software composition to better fit the current environment while preventing damage or loss of service. Decision makers must monitor both their physical and virtual environments using software and hardware sensors. Moreover, pervasive computing environments may require that software learn about and adapt to user behavior. Some existing decision makers use rule-based approaches [245], while others are supported by theoretical models, including those based on control theory [246, 247], resource optimization [248], and those inspired by biological processes, such as the human nervous system [249] and emergent behavior in

species that form colonies [250]. Several researchers have explored artificial intelligence-based techniques for managing dynamic system reconfiguration [251, 252]. While these methods have been effective in certain domains, environmental dynamics and software complexity have defied their general application. More extensive research in this area is needed. Future systems will need to accommodate high-dimensional sensory data [253], and continue to learn from new experience and accommodate new adaptations as they become available [254–256]. Moreover, these decision makers may need to tolerate the removal or addition of sensors gracefully, continuing to guide application recomposition in the face of new or incomplete information.

6 Conclusions

We envision that the use of compositional adaptation will continue to increase, as programmers become more familiar with adaptive software technologies, the boundary between cyberspace and the physical world continues to diminish, and Society increasingly expects computer systems to manage themselves. There is a potential downside, however. While many of the mechanisms are available now, and therefore *will* be used, the necessary supporting development environments are, for the most part, an area of ongoing research. Compositional adaptation is very powerful, but without appropriate tools to automate the generation and verification of code, its use can negatively impact, rather than improve, the integrity and security of systems. The challenge to the computer science community is to build a foundation of development technologies and tools, grounded in rigorous software engineering, that will enable compositional adaptation to raise the next generation of computing to new levels of flexibility, autonomy, and maintainability, without sacrificing assurance and security.

Further Information

As noted earlier, our group is actively participating in this area of research through two projects: RAPIDware (www.cse.msu.edu/rapidware) addresses adaptive software for protecting critical infrastructures, and Meridian (www.cse.msu.edu/meridian) addresses automated software engineering for mobile computing. Among other artifacts, these projects produced Adaptive Java, Trap/J and ACT, mentioned earlier. Source code is available for downloading via the RAPIDware URL.

Acknowledgements

We would like to express our gratitude to the many individuals who have contributed to this emerging area of study. Discussions with researchers associated with many of the research projects listed in Table 1, have greatly improved our understanding of this area. We would also like to thank all the faculty and students in the SENS Laboratory at Michigan State University for their contributions to RAPIDware, Meridian and related projects. This work was supported in part by the U.S. Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, and in part by National Science Foundation grants CCR-9912407, EIA-0000433, EIA-0130724, ITR-0313142, CDA-9700732, and CCR-9901017.

References

[1] M. Weiser, "Ubiquitous computing," *IEEE Computer*, vol. 26, pp. 71–72, October 1993.

[2] D. C. Schmidt, D. F. Box, , and T. Suda, "ADAPTIVE: A Dynamically Assembled Protocol Transformation, Integration, and eValuation Environment," *Journal of Concurrency: Practice and Experience*, pp. 269–286, June 1993.

[3] T. Fitzpatrick, G. Blair, G. Coulson, N. Davies, and P. Robin, "A software architecture for adaptive distributed multimedia applications," *IEE Proceedings - Software*, vol. 145, no. 5, pp. 163–171, 1998.
[4] B. D. Noble and M. Satyanarayanan, "Experience with adaptive mobile applications in Odyssey," *Mobile Networks and Applications*, vol. 4, pp. 245–254, 1999.
[5] S. McCanne, E. Brewer, R. Katz, L. Rowe, E. Amir, Y. Chawathe, A. Coopersmith, K. Mayer-Patel, S. Raman, A. Schuett, D. Simpson, A. Swan, T. Tung, D. Wu, and B. Smith, "Toward a common infrastructure for multimedia-networking middleware," in *Proc. 7th Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '97), St. Louis, Missouri*, May 1997.
[6] M. Roussopoulos, P. Maniatis, E. Swierk, K. Lai, G. Appenzeller, and M. Baker, "Person-level routing in the mobile people architecture," in *Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems*, (Boulder, Colorado), October 1999.
[7] M. A. Hiltunen and R. D. Schlichting, "The Cactus approach to building configurable middleware services," in *Proceedings of the Workshop on Dependable System Middleware and Group Communication (DSMGC 2000)*, (Nuremberg, Germany), October 2000.
[8] M. Modahl, I. Bagrak, M. Wolenetz, P. Hutto, and U. Ramachandran, "Mediabroker: An architecture for pervasive computing," in *Proceedings of IEEE PerCom 2004*, 2004.
[9] The Planet Blue project at IBM. http://www.research.ibm.com/compsci/planetblue.html.
[10] The Oxygen project at MIT. http://oxygen.lcs.mit.edu/.
[11] J. P. Sousa and D. Garlan, "Aura: an architectural framework for user mobility in ubiquitous computing environments," in *Proceedings of the third Working IEEE/IFIP Conference on Software Architecture*, pp. 29–43, 2002.
[12] F. Kon, C. Hess, M. Roman, R. Campbell, and M. Mickunas, "A flexible, interoperable framework for active spaces," in *Proceedings of OOPSLA Workshop on Pervasive Computing*, (Minneapolis), October 2000.
[13] A. K. Dey and G. D. Abowd, "The context toolkit: Aiding the development of context-aware applications," in *Proceedings of the 22nd International Conference on Software Engineering (ICSE): Workshop on Software Engineering for Wearable and Pervasive Computing*, (Limerick, Ireland), June 2000.
[14] A. Fox, S. D. Gribble, Y. Chawathe, and E. A. Brewer, "Adapting to network and client variation using active proxies: Lessons and perspectives," *IEEE Personal Communications*, August 1998.
[15] W. G. Griswold, R. Boyer, S. W. Brown, and T. M. Truong, "A component architecture for an extensible, highly integrated context-aware computing infrastructure," in *Proceedings of the International Conference on Software Engineering*, May 2003.
[16] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. S. Milojicic, "Adaptive offloading inference for delivering applications in pervasive computing environments," in *Proceedings of the First IEEE Conference on Pervasive Computing and Communications PerCom 2003*, pp. 107–114, 2003.
[17] C. Poellabauer, K. Schwan, S. Agarwala, A. Gavrilovska, G. Eisenhauer, S. Pande, C. Pu, and M. Wolf, "Service morphing: Integrated system- and application-level service adaptation in autonomic systems," in *Proceedings of Autonomic Computing Workshop Fifth Annual International Workshop on Active Middleware Services (AMS'03)*, 2003.
[18] S.-W. Cheng, D. Garlan, B. R. Schmerl, J. P. Sousa, B. Spitznagel, P. Steenkiste, and N. Hu, "Software architecture-based adaptation for pervasive systems," in *Proceedings of the International Conference on Architecture of Computing Systems*, pp. 67–82, Springer-Verlag, 2002.
[19] S. S. Yau and F. Karim, "An energy-efficient object discovery protocol for context-sensitive middleware for ubiquitous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 11, pp. 1074–1085, 2003.
[20] N. Medvidovic, M. Mikic-Rakic, N. Mehta, and S. Malek, "Software architectural support for handheld computing," *IEEE Computer, Special Issue on Handheld Computing*, vol. 36, pp. 66–73, September 2003.
[21] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, pp. 41–50, January 2003.
[22] *Proceedings of the International Conference on Autonomic Computing (ICAC-04)*, (New York), May 2004.
[23] S.-W. Cheng, A.-C. Huang, D. Garlan, B. R. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," in *Proceedings of the International Conference on Autonomic Computing (ICAC-04)*, (New York), pp. 276–277, May 2004.
[24] A. Bohra, I. Neamtiu, P. Gallard, F. Sultan, and L. Iftode, "Remote repair of operating system state using backdoors," in *Proceedings of the 2004 International Conference on Autonomic Computing*, (New York), May 2004.

[25] A. Brown and D. A. Patterson, "Embracing failure: A case for recovery-oriented computing (ROC)," in *Proceedings of the 2001 High Performance Transaction Processing Symposium*, (Asilomar, California), October 2001.
[26] M. Mikic-Rakic and N. Medvidovic, "Support for disconnected operation via architectural self-reconfiguration," in *Proceedings of the International Conference on Autonomic Computing (ICAC-04)*, (New York, NY), May 2004.
[27] T. Boyd and P. Dasgupta, "Preemptive module replacement using the virtualizaing operating system realizing multi-dimensional software adaptation," in *Proceedings of the ACM Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN)*, held in conjunction with the *16th Annual ACM International Conference on Supercomputing*, (New York City, NY), June 2002.
[28] G. Kaiser, P. Gross, G. Kc, J. Parekh, and G. Valetto, "An approach to autonomizing legacy systems," in *Proceedings of the ACM Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN)*, held in conjunction with the *16th Annual ACM International Conference on Supercomputing*, (New York City, NY), June 2002.
[29] H. Liu and M. Parashar, "Component-based programming model for autonomic applications," in *Proceedings of the International Conference on Autonomic Computing (ICAC-04)*, (New York, NY), May 2004.
[30] M. A. Hiltunen and R. D. Schlichting, "Adaptive distributed and fault-tolerant systems," *International Journal of Computer Systems Science and Engineering*, vol. 11, pp. 125–133, September 1996.
[31] Information Sciences Institute University of Southern California, "RFC 793: Transmission control protocol." `http://www.faqs.org/rfcs/rfc793.html`, September 1981.
[32] J. F. Kurose and K. W.Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*. Boston, Massachusetts: Addison Wesley, 2001.
[33] S. Fickas, G. Kortuem, and Z. Segall, "Software organization for dynamic and adaptable wearable systems," in *Proceedings First International Symposium on Wearable Computers (ISWC'97)*, (Cambridge, Massachusetts), October 1997.
[34] N. Minar, M. Gray, O. Roup, R. Krikorian, and P. Maes, "Hive: Distributed agents for networking things," in *Proceedings of ASA/MA'99, the First International Symposium on Agent Systems and Applications and Third International Symposium on Mobile Agents*, 1999.
[35] R. W. DeVaul and A. Pentland, "The Ektara architecture: The right framework for context-aware wearable and ubiquitous computing applications." The Media Laboratory, Massachusetts Institute of Technology, unpublished, 2000.
[36] G. Kortuem, J. Schneider, D. Preuitt, T. G. C. Thompson, S. Fickas, and Z. Segall, "When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad-hoc networks," in *Proceedings of the 2001 International Conference on Peer-to-Peer Computing (P2P2001)*, (Linköpings, Sweden), August 2001.
[37] J. Flinn, E. de Lara, M. Satyanarayanan, D. S. Wallach, and W. Zwaenepoel, "Reducing the energy usage of office applications," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, (Heidelberg, Germany), pp. 252–272, November 2001.
[38] M. Aksit and Z. Choukair, "Dynamic, adaptive and reconfigurable systems overview and prospective vision," in *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03)*, (Providence, Rhode Island), May 2003.
[39] N. Venkatasubramanian, "Safe composability of middleware services," *Communications of the ACM*, vol. 45, June 2002.
[40] M. Aksit, L. Bergmans, and S. Vural, "An object-oriented language-database integration model: The composition-filters approach," in *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'92)*, (Utrecht, Netherlands), pp. 372–395, June 1992.
[41] W.-K. Chen, M. A. Hiltunen, and R. D. Schlichting, "Constructing adaptive software in distributed systems," in *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, (Mesa, Arizona), pp. 635–643, April 2001.
[42] R. van Renesse, K. P. Birman, M. Hayden, A. Vaysburd, and D. Karr, "Building adaptive systems using Ensemble," *Software Practice and Experience*, vol. 28, p. 963979, August 1998.
[43] P. K. McKinley, U. I. Padmanabhan, N. Ancha, and S. M. Sadjadi, "Composable proxy services to support collaboration on the mobile internet," *IEEE Transactions on Computers (Special Issue on Wireless Internet)*, pp. 713–726, June 2003.
[44] E. P. Kasten, P. K. McKinley, S. M. Sadjadi, and R. E. K. Stirewalt, "Separating introspection and intercession in metamorphic distributed systems," in *Proceedings of the IEEE Workshop on Aspect-Oriented Programming for Distributed Computing (with ICDCS'02)*, (Vienna, Austria), pp. 465–472, July 2002.

[45] W. Aspray, "John von Neumann's contributions to computing and computer science," *Annals of the History of Computing*, vol. 11, pp. 189–195, Fall 1989.
[46] G. Haugk, F. M. Lax, R. D. Royer, and J. R. Williams, "The 5ESS switching system: Maintenance capabilities," *AT&T Technical Journal*, vol. 64, pp. 1385–1416, July-August 1985.
[47] P. Leszek, "Use the Eclipse platform for debugging your software projects," tech. rep., IBM, May 2003. http://www-106.ibm.com/developerworks/opensource/library/os-ecbug/.
[48] M. Mönnig, "Self-modifying code with Delphi." http://www.undu.com/Articles/990212d.html.
[49] J. Misra, "Strategies to combat software piracy," tech. rep., Department of Computer Science, University of Texas at Austin, 2000.
[50] F. Hohl, "Time limited blackbox security: Protecting mobile agents from malicious hosts," in *Mobile Agents and Security*, pp. 92–113, Springer-Verlag, 1998.
[51] A. Goldberg and D. Robson, *Smalltalk-80*. Addison-Wesley, 1989.
[52] B. C. Smith, *Reflection and Semantics in a Procedural Language*. PhD thesis, Massachusetts Institute of Technology, Jan 1982.
[53] P. Maes, "Concepts and experiments in computational reflection," in *Proceedings of the ACM Conference on Object-Oriented Languages (OOPSLA)*, pp. 147–155, ACM Press, December 1987.
[54] G. Kiczales, J. des Rivières, and D. G. Bobrow, *The Art of Metaobject Protocols*. MIT Press, 1991.
[55] "Java history." http://www.ils.unc.edu/blaze/java/javahist.html.
[56] L. Bergmans and M. Aksit, "Composing crosscutting concerns using composition filters," *Communications of ACM*, pp. 51–57, October 2001.
[57] M. Tatsubori, S. Chiba, K. Itano, and M.-O. Killijian, "OpenJava: A class-based macro system for Java," in *Proceedings of OORaSE*, pp. 117–133, 1999.
[58] J. de Oliveira Guimarães, "Reflection for statically typed languages," in *Proceedings of 12th European Conference on Object-Oriented Programming (ECOOP'98)*, pp. 440–461, 1998.
[59] R. Pawlak, L. Seinturier, L. Duchien, and G. Florin., "JAC: A flexible and efficient solution for aspect-oriented programming in Java.," in *Proceedings of Reflection 2001, LNCS 2192*, pp. 1–24, September 2001.
[60] Z. Wu, "Reflective Java and a reflective compnent-based transaction architecture," in *Proceedings of Workshop on Reflective Programming in C++ and Java*, 1998.
[61] B. Redmond and V. Cahill, "Supporting unanticipated dynamic adaptation of application behaviour," in *Proceedings of the 16th European Conference on Object-Oriented Programming*, (Malaga, Spain), Springer-Verlag, June 2002. volume 2374 of Lecture Notes in Computer Science.
[62] S. M. Sadjadi, P. K. McKinley, R. E. K. Stirewalt, and B. H. Cheng, "TRAP: Transparent reflective aspect programming," Tech. Rep. MSU-CSE-03-31, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, November 2003.
[63] *Proceedings of the Workshop on Adaptable and Adaptive Software (held in conjunction with OOPSLA'95)*, (Austin, Texas), October 1995.
[64] *Proceedings of the ECOOP'97 Workshop on Reflective Real-Time Object-Oriented Programming and Systems*, (Jyvaskyla, Finland), June 1997.
[65] *Proceedings of the OOPSLA'98 Workshop on Reflective Programming in C++ and Java*, (Vancouver, Canada), October 1998.
[66] *Proceedings of the Middleware'2000 Workshop on Reflective Middleware*, (New York), April 2000.
[67] *Proceedings of the ACM SIGSOFT Workshop on Self-Healing Systems (WOSS02)*, (Charleston, South Carolina), November 2002.
[68] *Proceedings of the Workshop on Self-Healing, Adaptive, and self-MANaged Systems (SHAMAN) (held in conjunction with the 16th Annual ACM International Conference on Supercomputing)*, (New York), June 2002.
[69] *Proceedings of the 5th Annual International Workshop on Active Middleware Services (AMS 2003)*, (Seattle, Washington), June 2003.
[70] *Proceedings of the First Workshop on the Design of Self-Managing Systems (in conjunction with DSN-2003)*, (San Francisco), June 2003.
[71] *Proceedings of Fourth International Workshop on Distributed Auto-Adaptive and Reconfigurable Systems (in conjunction with ICDCS 2004)*, (Hachioji, Japan), March 2004.
[72] Program on Self Adaptive Software, sponsored by DARPA, point of contact Robert Laddaga, 1998. BAA: http://msrc.wvu.edu/nsf_epscor/cluster_research/arpa_baa98_1.html.
[73] Program on Dynamic Assembly for Systems Adaptability, Dependability, and Assurance (DASADA), sponsored by DARPA and AFRL. http://www.rl.af.mil/tech/programs/dasada/.

[74] Program on Critical Infrastructure Protection and High Confidence, Adaptable Software, sponsored by U.S. DoD. http://alpha.ddm.uci.edu/zotmail/archive/2000/20000623103.html.
[75] Program on Next Generation Software, sponsored by NSF, point of contact Frederica Darema. RFP: http://www.nsf.gov/pubs/2001/nsf01147/nsf01147.htm.
[76] D. Batory and S. O'Malley, "The design and implementation of hierarchical software systems with reusable components," *ACM Transactions on Software Engineering and Methodology*, vol. 1, pp. 355–398, October 1992.
[77] D. Batory and B. J. Geraci, "Composition validation and subjectivity in GenVoca generators," *IEEE Transactions on Software Engineering (special issue on Software Reuse)*, pp. 62–87, 1997.
[78] K. Czarnecki and U. Eisenecker, *Generative programming*. Addison Wesley, 2000.
[79] J. Lamping, G. Kiczales, L. H. R. Jr., and E. Ruf, "An architecture for an open compiler," in *Proceedings of the IMSA'92 Workshop on Reflection and Meta-level Architectures*, 1992.
[80] S. Chiba, "A metaobject protocol for C++," in *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pp. 285–299, October 1995.
[81] Y. Ishikawa, A. Hori, M. Sato, M. Matsuda, J. Nolte, H. Tezuka, H. Konaka, M. Maeda, and K. Kubota, "Design and implementation of metalevel architecture in C++ – MPC++ approach," in *Proceedings of the Reflection'96*, pp. 141–154, 1996.
[82] R. Arnold, "Software restructuring," *Proceedings IEEE*, pp. 607–617, April 1989.
[83] P. L. Bergstein, "Object-preserving class transformations," in *Conference proceedings on Object-oriented programming systems, languages, and applications*, pp. 299–313, ACM Press, 1991.
[84] R. Arnold, *Software Reengineering*. IEEE Computer Society Press, 1993.
[85] P. L. Bergstein, "Maintenance of object-oriented systems during structural schema evolution," *TAPOS Journal*, vol. 3, no. 3, pp. 185–212, 1997.
[86] P. Bottoni, F. Parisi-Presicce, and G. Taentzer, "Coordinated distributed diagram transformation for software evolution," in *Electronic Notes in Theoretical Computer Science* (R. Heckel, T. Mens, and M. Wermelinger, eds.), vol. 72, Elsevier, 2003.
[87] E. Casais, "An incremental class reorganization approach," in *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'92)* (O. Lehrmann Madsen, ed.), pp. 114–132, Berlin, Heidelberg: Springer, 1992.
[88] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: principles, techniques, and tools*. Addison-Wesley Longman Publishing Co., Inc., 1986.
[89] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Publication Corporation, 1999.
[90] I. D. Baxter and M. Mehlich, "Reverse engineering is reverse forward engineering," *Science of Computer Programming*, vol. 36, no. 2-3, pp. 131–147, 2000.
[91] R. Heckel, T. Mens, and M. Wermelinger, "Software evolution through transformations," in *Electronic Notes in Theoretical Computer Science* (R. Heckel, T. Mens, and M. Wermelinger, eds.), vol. 72, Elsevier, 2003.
[92] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent Software*, vol. 14, pp. 54–62, May-June 1999.
[93] J. Aldrich, V. Sazawal, C. Chambers, and D. Notkin, "Architecture-centric programming for adaptive systems," in *Proceedings of the ACM SIGSOFT Workshop on Self-Healing Systems (WOSS02)*, November 2002.
[94] J. Aldrich, V. Sazawal, C. Chambers, and D. Notkin, "Language support for connector abstractions," in *Proceedings of the European Conference on Object-Oriented Programming (ECOOP03)*, July 2003.
[95] L. Capra, W. Emmerich, and C. Mascolo, "Reflective middleware solutions for context-aware applications," in *Proceedings of Reflection 2001, Lecture Notes in Computer Science*, (Kyoto, Japan), Springer Verlag., 2001.
[96] R. H. Katz, E. A. Brewer, et al., "The Bay Area Research Wireless Access Network (BARWAN)," in *Proceedings Spring COMPCON Conference*, 1996.
[97] Y. Saito and B. Bershad, "System call support in an extensible operating system," *Software Practice and Experience*, vol. 1, pp. 1–10, January 1999.
[98] L. van Doorn, *The Design and Application of an Extensible Operating System*. PhD thesis, Vrije Universiteit, Amsterdam, The Netherlands, 2001.
[99] B. Ford, K. V. Maren, J. Lepreau, S. Clawson, B. Robinson, and J. Turner, "The Flux OS Toolkit: Reusable Components for OS Implementation," in *Proceedings of Sixth Workshop on Hot Topics in Operating Systems*, pp. 14–19, May 1997.

[100] A. Senart, O. Charra, and J.-B. Stefani, "Developing dynamically reconfigurable operating system kernels with the THINK component architecture," in *Proceedings of the workshop on Engineering Context-aware Object-Oriented Systems and Environments, in association with OOPSLA 2002*, (Seattle, Washington), November 2002.

[101] M. Seltzer and C. Small, "Self-monitoring and Self-adapting Operating Systems," in *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems*, 1997. http://www.eecs.harvard.edu/ vino/vino/.

[102] D. R. Engler, M. F. Kaashoek, and J. O'Toole Jr., "Exokernel: an operating system architecture for application-level resource management," in *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP '95)*, (Copper Mountain Resort, Colorado), pp. 251–266, December 1995. http://www.pdos.lcs.mit.edu/exo.html.

[103] C. A. N. Soules, J. Appavoo, K. Hui, R. W. Wisniewski, D. D. Silva, G. R. Ganger, O. Krieger, M. Stumm, M. Auslander, M. Ostrowski, B. Rosenburg, and J. Xenidis, "System support for online reconfiguration," in *Proc. of the Usenix Technical Conference*, 2003.

[104] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, pp. 80–86, January 1997.

[105] D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith, "The SwitchWare active network architecture," *IEEE Network, Special Issue on Active and Controllable Networks*, vol. 12, no. 3, pp. 29–36, 1998.

[106] O. Angin, A. T. Campbell, M. E. Kounavis, and R.R.-F.M. Liao, "The Mobiware toolkit: Programmable support for adaptive mobile networking," *IEEE Personal Communications Magazine, Special Issue on Adapting to Network and Client Variability*, August 1998.

[107] A. T. Campbell and M. E. Kounavis, "Toward reflective network architectures," in *Middleware'2000 Workshop on Reflective Middleware (RM2000)*, (New York), April 2000.

[108] G. S. Blair, G. Coulson, P. Robin, and M. Papathomas, "An architecture for next generation middleware," in *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, (The Lake District, England), September 1998.

[109] D. M. Hoffman and D. M. Weiss, *Software fundamentals: collected papers by David L. Parnas*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[110] P. Tarr and H. Ossher, eds., *Workshop on Advanced Separation of Concerns in Software Engineering at ICSE 2001 (W17)*, May 2001.

[111] K. J. Lieberherr, *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns*. PWS Publishing Company, Boston, 1996. ISBN 0-534-94602-X.

[112] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Videira Lopes, J. M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Springer-Verlag LNCS 1241, June 1997.

[113] R. J. Walker, E. L. A. Baniassad, and G. C. Murphy, "An initial assessment of aspect-oriented programming," in *International Conference on Software Engineering*, pp. 120–130, 1999.

[114] *Communications of the ACM, Special Issue on Aspect-Oriented Programming*, vol. 44, October 2001.

[115] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An overview of AspectJ," *Lecture Notes in Computer Science*, vol. 2072, pp. 327–355, 2001.

[116] H. Ossher and P. Tarr, "Using multidimensional separation of concerns to (re)shape evolving software," *Communications of the ACM*, vol. 44, no. 10, pp. 43–50, 2001.

[117] K. Lieberherr, D. Orleans, and J. Ovlinger, "Aspect-oriented programming with adaptive methods," *Communications of the ACM*, vol. 44, no. 10, pp. 39–41, 2001.

[118] I. Welch and R. J. Stroud, "Kava - A Reflective Java Based on Bytecode Rewriting," in *Reflection and Software Engineering* (W. Cazzola, R. J. Stroud, and F. Tisato, eds.), Lecture Notes in Computer Science 1826, pp. 157–169, Heidelberg, Germany: Springer-Verlag, June 2000.

[119] E. Truyen, B. N. Jörgensen, W. Joosen, and P. Verbaeten, "Aspects for run-time component integration," in *Proceedings of the ECOOP 2000 Workshop on Aspects and Dimensions of Concerns*, (Sophia Antipolis and Cannes, France), 2000.

[120] F. Akkai, A. Bader, and T. Elrad, "Dynamic weaving for building reconfigurable software systems," in *Proceedings of OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems*, (Tampa Bay, Florida), October 2001.

[121] D. Wagelaar, "Towards a context-driven development framework for ambient intelligence," in *Proceedings of the Fourth IEEE International Workshop on Distributed Auto-adaptive and Reconfigurable Systems (with ICDCS'04)*, (Tokyo, Japan), March 2004.

[122] R. Hirschfeld and K. Kawamura, "Dynamic service adaptation," in *Proceedings of the Fourth IEEE International Workshop on Distributed Auto-adaptive and Reconfigurable Systems (with ICDCS'04)*, (Tokyo, Japan), March 2004.
[123] Z. Yang, B. H. Cheng, R. E. K. Stirewalt, J. Sowell, S. M. Sadjadi, and P. K. McKinley, "An aspect-oriented approach to dynamic adaptation," in *Proceedings of the ACM SIGSOFT Workshop On Self-healing Software (WOSS'02)*, November 2002.
[124] P. C. David, T. Ledoux, and N. M. N. Bouraqadi-Saadani, "Two-step weaving with reflection using AspectJ," in *OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems*, (Tampa), October 2001.
[125] F. Costa, H. Duran, N. Parlavantzas, K. Saikoski, G. Blair, and G. Coulson, "The role of reflective middleware in supporting the engineering of dynamic applications"," *Reflection and Software Engineering*, pp. 79–98, 2000.
[126] F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, L. C. Magalhães, and R. H. Campbell, "Monitoring , security, and dynamic configuration with the dynamicTAO reflective ORB," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2000)*, (New York), April 2000.
[127] G. Blair, G. Coulson, and N. Davies, "Adaptive middleware for mobile multimedia applications," in *Proceedings of the Eighth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pp. 259–273, 1997.
[128] "Python publications." http://www.rmi.net/ lutz/pybooks.html.
[129] J. McAffer, "Meta-level architectue support for distributed obejcts," in *Proceedings of Reflection'96*, (San Francisco, California), pp. 39–62, 1996.
[130] H. Masuhara, S. Matsuoka, T. Watanabe, and A. Yonezawa, "Object-oriented concurrent reflective languages can be implemented efficiently," in *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)* (A. Paepcke, ed.), vol. 27, (New York, NY), pp. 127–144, ACM Press, 1992.
[131] H. Okamura, Y. Ishikawa, and M. Tokoro, "AL-1/D: A distributed programming system with multi-model reflection framework," in *Proceedings of the Workshop on New Models for Software Architecture*, Nov. 1992.
[132] V. Adve, V. V. Lam, and B. Ensink, "Language and compiler support for adaptive distributed applications," in *Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001)*, (Snowbird, Utah), June 2001.
[133] É. Tanter, J. Noyè, D. Caromel, and P. Cointe, "Partial behavioral reflection: Spatial and temporal selection of reification," in *Proceedings of the 18th ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications (OOPSLA 2003)* (R. Crocker and G. L. Steele, Jr., eds.), (Anaheim, California), pp. 27–46, ACM Press, October 2003.
[134] J. Baker and W. Hsieh, "Runtime aspect weaving through metaprogramming," in *Proceedings of the First International Conference on Aspect-Oriented Software Development*, (Enschede, The Netherlands), April 2002.
[135] S. Chiba and T. Masuda, "Designing an extensible distributed language with a meta-level architecture," *Lecture Notes in Computer Science*, vol. 707, 1993.
[136] D. Caromel, W. Klauser, and J. Vayssière, "Towards seamless computing and metacomputing in Java," *Concurrency: Practice and Experience*, vol. 10, no. 11–13, pp. 1043–1061, 1998.
[137] J. Ferber, "Computational reflection in class based object-oriented languages," in *Conference proceedings on Object-oriented programming systems, languages and applications*, pp. 317–326, ACM Press, 1989.
[138] M. Golm, "Design and implementation of a meta architecture for Java," Master's thesis, Friedrich-Alexander-University, Erlangen-Nurenburg, Jan. 1997.
[139] M. Golm and J. Kleinoder, "metaXa and the future of reflection," in *Proceedings of Workshop on Reflective Programming in C++ and Java*, pp. 1–5, 1998.
[140] A. Popovici, T. Gross, and G. Alonso, "Dynamic homogenous AOP with PROSE," tech. rep., Department of Computer Science, Federal Institue of Technology, Zurich, Switzerland, March 2001.
[141] A. Oliva and L. E. Buzato, "The implementation of Guaraná on Java," Tech. Rep. IC-98-32, Universidade Estadual de Campinas, Sept. 1998.
[142] T. Ledoux, "OpenCorba: A reflective open broker," *Lecture Notes in Computer Science*, vol. 1616, 1999.
[143] D. Conger, *Remoting with C# and .NET*. Wiley Publishing, Inc., Indianapolis, Indiana, 2003.
[144] M. Clarke, G. S. Blair, G. Coulson, and N. Parlavantzas, "An efficient component model for the construction of adaptive middleware," *Lecture Notes in Computer Science*, vol. 2218, 2001.
[145] M. Roman, F. Kon, and R. H. Campbell, "Reflective middleware: From your desk to your hand," *IEEE Distributed Systems Online*, vol. 2, no. 5, 2001.

[146] M. Roman, M. Mickunas, F. Kon, and R. H. Campbell, "LegORB and ubiquitous CORBA," in *Proc. IFIP/ACM Middleware'2000 Workshop on Reflective Middleware (RM2000)*, (New York), April 2000.
[147] J. C. Fabre and T. Perennou, "A metaobject architecture for fault-tolerant distributed systems: The FRIENDS approach," *IEEE Transactions on Computers*, vol. 47, no. 1, pp. 78–95, 1998.
[148] W. Cazzola and M. Ancona, "mChaRM: A reflective middleware for communications-based reflection," Tech. Rep. DISI-TR-00-09, Universita degli Studi di Milano, May 2000.
[149] R. Hayton, *FlexiNet Open ORB Framework*. APM Ltd., Oct. 1997.
[150] S. M. Sadjadi, P. K. McKinley, and E. P. Kasten, "Architecture and operation of an adaptable communication substrate," in *Proceedings of the Ninth IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, (San Juan, Puerto Rico), pp. 46–55, May 2003.
[151] S. M. Sadjadi, P. K. McKinley, R. E. K. Stirewalt, and B. H. Cheng, "Generation of self-optimizing wireless network applications," in *Proceedings of the International Conference on Autonomic Computing (ICAC-04)*, (New York, NY), pp. 310–311, May 2004.
[152] R. Klefstad, D. C. Schmidt, and C. O'Ryan, "Towards highly configurable real-time object request brokers," in *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, April - May 2002.
[153] G. T. Sullivan, "Aspect-oriented programming using reflection and metaobject protocols," *Communications of the ACM*, vol. 44, no. 10, pp. 95–97, 2001.
[154] A. Popovici, T. Gross, and G. Alonso, "Dynamic weaving for aspect-oriented programming," in *Proceedings of the First International Conference on Aspect-Oriented Software Development*, pp. 141–147, ACM Press, 2002.
[155] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1999.
[156] A. T. Campbell, G. Coulson, and M. E. Kounavis, "Managing complexity: Middleware explained," *IT Professional, IEEE Computer Society*, pp. 22–28, September/October 1999.
[157] Microsolft Corporation, *COM: Delivering on the Promises of Component Technology*, 2000. http://www.microsoft.com/com/default.asp.
[158] Microsoft Corporation, *Microsoft COM Technologies - DCOM*, 2000.
[159] Microsoft, http://www.microsoft.com/net/, *Microsoft .NET*.
[160] Object Management Group, Framingham, Massachusett, *The Common Object Request Broker: Architecture and Specification Version 3.0*, July 2003. Available at http://doc.ece.uci.edu/CORBA/formal/02-06-33.pdf.
[161] Sun Microsystems, http://java.sun.com/products/ejb/, *Enterprise JavaBeans Technology*, 2001.
[162] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements od Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, New York, NY: Addison-Wesley Publishing Company, 1995.
[163] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture*, vol. 2. John Wiley, 2001.
[164] A. Corsaro, D. Schmidt, R. Klefstad, and C. O'Ryan, "Virtual component a design pattern for memory constrained embedded applications," in *Proceedings of the Ninth Conference on Pattern Language of Programs (PLoP 2002)*, 2002.
[165] D. C. Schmidt, D. L. Levine, and S. Mungee, "The design of the TAO real-time object request broker," *Computer Communications*, vol. 21, pp. 294–324, April 1998.
[166] D. C. Schmidt, "The ADAPTIVE Communication Environment: An object-oriented network programming toolkit for developing communication software," *Concurrency: Practice and Experience*, vol. 5, no. 4, pp. 269–286, 1993.
[167] N. Wang, D. C. Schmidt, and M. Kircher, "Towards an adaptive and reflective middleware framework for QoS-enabled CORBA component model applications," *IEEE Distributed System Online, Special Issue on Reflective Middleware*, 2003.
[168] J. A. Zinky, D. E. Bakken, and R. E. Schantz, "Architectural support for quality of service for CORBA objects," *Theory and Practice of Object Systems*, vol. 3, no. 1, 1997.
[169] O. Othman, "The design, optimization, and performance of an adaptive middleware load balancing service," Master's thesis, University of California, Irvine, 2002.
[170] M. Cukier, J. Ren, C. Sabnis, D. Henke, J. Pistole, W. H. Sanders, D. E. Bakken, M. E. Berman, D. A. Karr, and R. E. Schantz, "AQuA: An adaptive architecture that provides dependable distributed objects," in *Proceedings of the The 17th IEEE Symposium on Reliable Distributed Systems*, p. 245, IEEE Computer Society, Oct. 1998.
[171] K. Lieberherr, *Adaptive Object-Oriented Software The Demeter Method*. PWS Publishing Company, Boston, 1996.

[172] C. Simonyi, "The death of computer languages, the birth of intentional programming," 1995.
[173] G. Kiczales, "Beyond the black box: Open implementation," *IEEE Software*, vol. 13, Jan. 1996.
[174] W. Harrison and H. Ossher, "Subject-oriented programming (a critique of pure objects)," in *OOPSLA'93*, 1993.
[175] M. Mezini, *Variation-Oriented Programming Beyond Classes and Inheritance*. PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany, 1997.
[176] A. R. Tripathi, N. M. Karnik, T. Ahmed, R. D. Singh, A. Prakash, V. Kakani, M. K. Vora, and M. Pathak, "Design of the Ajanta system for mobile agent programming," *Journal of Systems and Software*, vol. 62, pp. 123–140, May 2002.
[177] A. Murphy, G. Picco, and G.-C. Roman, "LIME: A middleware for physical and logical mobility," in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS'01)*, (Phoenix, Arizona), pp. 524–533, April 2001.
[178] G. Agha, *Actors: A model of concurrent computation in distributed systems*. MIT Press, 1986.
[179] R. Brandt and H. Reiser, "Dynamic adaptation of mobile agents in heterogeneous environments," in *Proceedings of the 5th International Conference on Mobile Agents*, pp. 70–87, Springer-Verlag, 2001.
[180] G. A. Cohen, J. S. Chase, and D. Kaminsky, "Automatic program transformation with JOIE," in *1998 Usenix Technical Conference*, June 1998.
[181] D. Sharp, "Reducing avionics software cost through component-based product line development," in *Proceedings of the Software Technology Conference*, (Salt Lake City, Utah), April 1998.
[182] P. Felber, B. Garbinato, and R. Guerraoui, "Towards reliable CORBA: Integration vs. service approach," in *Special Issues in Object-Oriented Programming* (M. Mühlhäuser, ed.), pp. 199–205, dpunkt-Verlag, 1997.
[183] R. Friedman and E. Hadad, "Client side enhancements using portable interceptors," in *Proceedings of the Sixth IEEE International Workshop on Object-oriented Real-time Dependable Systems*, January 2001.
[184] R. Baldoni, C. Marchetti, and A. Termini, "Active software replication through a three-tier approach," in *Proceedings of the 22th IEEE International Symposium on Reliable Distributed Systems (SRDS02)*, (Osaka, Japan), pp. 109–118, October 2002.
[185] C. Marchetti, L. Verde, and R. Baldoni, "CORBA request portable interceptors: A performance analysis," in *the 3nd International Symposium on Distributed Objects and Applications (DOA 2001)*, (Rome, Italy), Sept. 2001.
[186] S. M. Sadjadi and P. K. McKinley, "ACT: An adaptive CORBA template to support unanticipated adaptation," in *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, (Tokyo, Japan), March 2004.
[187] S. M. Sadjadi and P. K. McKinley, "Transparent self-optimization in existing CORBA applications," in *Proceedings of the International Conference on Autonomic Computing (ICAC-04)*, (New York, NY), pp. 88–95, May 2004.
[188] R. Koster, A. P. Black, J. Huang, J. Walpole, and C. Pu, "Thread transparency in information flow middleware," in *Proceedings of the International Conference on Distributed Systems Platforms and Open Distributed Processing*, Springer Verlag, Nov. 2001.
[189] R. Koster, A. P. Black, J. Huang, J. Walpole, and C. Pu, "Infopipes for composing distributed information flows," in *Proceedings of the International Workshop on Multimedia Middleware*, pp. 44–47, ACM, Oct. 2001.
[190] M. Geier, M. Steckermeier, U. Becker, F. J. Hauck, E. Meier, and U. Rastofer, "Support for mobility and replication in the AspectIX architecture," Tech. Rep. TR-I4-98-05, Univ. of Erlangen-Nuernberg, IMMD IV, 1998.
[191] S. Maffeis, "Adding group communication and fault-tolerance to CORBA," in *Proceedings of the Conference on Object-Oriented Technologies*, pp. 135–146, 1995.
[192] R. V. Renesse, K. P. Birman, B. B. Glade, K. Guo, M. Hayden, T. Hickey, D. Malki, A. Vaysburd, and W. Vogels, "Horus: A flexible group communications system," Tech. Rep. TR95-1500, Department of Computer Science, Cornell University, 23, 1995.
[193] K. P. Birman and R. van Renesse, "Reliable distributed computing with the Isis toolkit," *IEEE Computer Society Press*, 1994.
[194] IONA Technologies, *Orbix*. URL: http://www.iona.com/products/orbix.htm.
[195] IONA Technologies Inc., *ORBacus for C++ and Java version 4.1.0*, 2001.
[196] L. Moser, P. Melliar-Smith, P. Narasimhan, L. Tewksbury, and V. Kalogeraki, "The Eternal system: An architecture for enterprise applications," in *Proceedings of the Third International Enterprise Distributed Object Computing Conference (EDOC'99)*, July 1999.

[197] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, C. A. Lingley-Papadopoulis, and T. P. Archambault, "The Totem system," in *Proceedings of the 25th International Symposium on Fault Tolerant Computing*, (Pasadena, California), pp. 61–66, 1995.
[198] V. C. Zandy and B. P. Miller, "Reliable network connections," in *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking*, pp. 95–106, September 2002.
[199] Sun Microsystems, *EmbeddedJava Application Environment*. http://java.sun.com/products/embeddedjava/.
[200] "Distributed Extensible Open Systems (the DEOS project)." http://www.cc.gatech.edu/systems/projects/DEOS/, 2004. Georgia Institute of Technology - College of Computing.
[201] S. Adve, A. Harris, C. Hughes, D. Jones, R. Kravets, K. Nahrstedt, D. Sachs, R. Sasanka, J. Srinivasan, and W. Yuan, "The Illinois GRACE project: Global Resource Adaptation through CoopERation," in *Proceedings of the Workshop on Self-Healing, Adaptive, and self-MANaged Systems (SHAMAN)*, June 2002.
[202] A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau, "Information and Control in Gray-box Systems," in *Symposium on Operating Systems Principles*, pp. 43–56, 2001. http://www.cs.wisc.edu/graybox/.
[203] R. Rashid, R. Baron, A. Forin, D. Golub, M. Jones, D. Julin, D. Orr, and R. Sanzi, "Mach: A Foundation for Open Sytems," in *Proceedings of the Second Workshop on Workstation Operating Systems(WWOS2)*, September 1989. http://www-2.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html.
[204] B. N. Bershad and et al, "SPIN - An extensible microkernel for application-specific operating system services," tech. rep., Dept. of Computer Science and Engineering, University of Washington, February 1994. http://www.cs.washington.edu/homes/melody/os/spin.html.
[205] A. B. Montz, D. Mosberger, S. W. O'Malley, L. L. Peterson, T. A. Proebsting, and J. H. Hartman, "Scout: A Communications-Oriented Operating System," in *Operating Systems Design and Implementation*, 1994. http://www.cs.arizona.edu/scout/.
[206] J. Appavoo, K. Hui, C. A. N. Soules, R. W. Wisniewski, D. M. D. Silva, O. Krieger, D. J. E. M. A. Auslander, B. Gamsa, G. R. Ganger, P. McKenney, M. Ostrowski, B. Rosenburg, M. Stumm, and J. Xenidis, "Enabling autonomic behavior in systems software with hot-swapping," *IBM Systems Journal*, vol. 42, no. 1, 2003.
[207] M. B. Jones, "Interposition agents: Transparently interposing user code at the system interface," *Symposium on Operating Systems Principles (SOSP 14)*, pp. 80–93, December 1993.
[208] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer, "A Secure Environment for Untrusted Helper Applications," in *Proceedings of the 6th USENIX Security Symposium*, (San Jose, CA), 1996.
[209] T. Mitchem, R. Lu, and R. O'Brien, "Using kernel hypervisors to secure applications," *Annual Computer Security Application Conference (ACSAC '97)*, December 1997. http://www.securecomputing.com/khyper/.
[210] D. P. Ghormley, D. Petrou, S. H. Rodrigues, and T. E. Anderson, "SLIC: An Extensibility System for Commodity Operating Systems," in *USENIX 1998 Annual Technical Conference*, pp. 39–52, June 1998. http://now.cs.berkeley.edu/Slic/.
[211] A. D. Alexandrov, M. Ibel, K. E. Schauser, and C. J. Scheiman, "Ufo: A Personal Global File System Based on User-Level Extensions to the Operating System," *ACM Transactions on Computer Systems*, vol. 16, pp. 207–233, August 1998.
[212] T. Fraser, L. Badger, and M. Feldman, "Hardening COTS Software with Generic Software Wrappers," *IEEE Symposium on Security and Privacy*, pp. 2–16, May 1999.
[213] T. Fraser, "LOMAC: Low water-mark integrity protection for COTS environments," *IEEE Symposium on Security and Privacy*, pp. Oakland, CA, May 2000.
[214] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman, "Linux Security Modules: General Security Support for Linux Kernel," *USENIX Security Symposium*, 2002. http://lsm.immunix.org/.
[215] K. G. Anagnostakis, S. Ioannidis, S. Miltchev, J. Ioannidis, and J. M. Smith, "Efficient Packet Monitoring for Network Management," in *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, April 2002.
[216] P. Loscocco and S. Smalley, "Integrating flexible support for security policies into the Linux operating system," in *Proceedings of the FREENIX Track of the 2001 USENIX Annual Technical Conference*, 2001.
[217] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, "Coda: A highly available file system for a distributed workstation environment," *IEEE Transactions on Computers*, vol. 39, no. 4, pp. 447–459, 1990.
[218] C. Pu, T. Autrey, A. Black, C. Consel, C. Cowan, J. Inouye, L. Kethana, J. Walpole, and K. Zhang, "Optimistic Incremental Specialization Streamlinig a Commercial Operating System," *Symposium on Operating Systems Principles (SOSP)*, 1995. http://www.cse.ogi.edu/DISC/projects/synthetix/.
[219] J. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications," in *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, 1999.

[220] J. Andersson and T. Ritzau, "Dynamic code update in JDrums," in *Proceedings of the ICSE'00 Workshop on Software Engineering for Wearable and Pervasive Computing*, (Limerick, Ireland), 2000.
[221] A. Singhai, A. Sane, and R. H. Campbell, "Quarterware for middleware," in *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS'98)*, (Amesterdam, The Netherlands), pp. 192–201, May 1998.
[222] E. Wohlstadter, S. Jackson, and P. Devanbu, "DADO: enhancing middleware to support crosscutting features in distributed, heterogeneous systems," in *Proceedings of the International Conference on Software Engineering*, (Portland, Oregon), pp. 174–186, May 2003.
[223] P. P. Pal, J. Loyall, R. E. Schantz, J. A. Zinky, and F. Webber, "Building auto-adaptive distributed applications: the QuO-APOD experience," in *Proceedings of 3rd International Workshop on Distributed Auto-adaptive and Reconfigurable Systems, in conjunction with 23rd ICDCS*, (Providence, Rhode Island), May 2003.
[224] J. Zhang, Z. Yang, B. H. Cheng, and P. K. McKinley, "Adding safeness to dynamic adaptation techniques," in *Proceedings of the ICSE 2004 Workshop on Architecting Dependable Systems*, (Edinburgh, Scotland), May 2004.
[225] J. Zhang, Z. Yang, B. H. C. Cheng, and P. K. McKinley, "Adding safeness to dynamic adaptation techniques," Tech. Rep. MSU-CSE-04-11, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, March 2004.
[226] S. Kulkarni and K. Biyani, "Correctness of component-based adaptation," in *International Symposium on Component-based Software Engineering (CBSE7)*, May 2004.
[227] N. Amano and T. Watanabe, "A software model for flexible and safe adaptation of mobile code programs," in *Proceedings of the international workshop on Principles of software evolution*, pp. 57–61, ACM Press, 2002.
[228] A. Beugnard *et al.*, "Making components contract aware," *IEEE Computer*, vol. 32, pp. 38–45, July 1999.
[229] A. Tripathi, T. Ahmed, R. Kumar, and S. Jaman, "Design of a policy-driven middleware for secure distributed collaboration," in *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, pp. 393–400, July 2002.
[230] J. Keeney and V. Cahill, "Chisel: A policy-driven, context-aware, dynamic adaptation framework," in *Proc. of IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, (Lake Como, Italy), p. 3, June 2003.
[231] J. Jin and K. Nahrstedt, "QoS specification languages for distributed multimedia applications: A survey and taxonomy," *IEEE Multimedia*, 2004. to appear.
[232] P. P. Pal, J. Loyall, R. E. Schantz, J. A. Zinky, and F. Webber, "Open implementation toolkit for building survivable applications," in *Proceedings of DISCEX 2000, the DARPA Information Survivability Conference and Exposition*, (Hilton Head Island, SC.), Jan 2000.
[233] J. Lobo, R. Bhatia, and S. A. Naqvi, "A policy description language," in *The Eleventh Innovative Applications of Artificial Intelligence Conference on Artificial Intelligence(IAAI-2001)/Sponsored by AAAI*, pp. 291–298, 1999.
[234] K. Havelund and G. Rosu, "Monitoring Java programs with Java PathExplorer," in *Electronic Notes in Theoretical Computer Science* (K. Havelund and G. Rosu, eds.), vol. 55, Elsevier, 2001.
[235] T. A. Henzinger, R. Jhala, R. Majumdar, and M. A. Sanvido, "Extreme model checking," in *Verification: Theory and Practice, Lecture Notes in Computer Science 2772*, pp. 332–358, Springer-Verlag, 2004.
[236] M. Jackson and P. Zave, "Distributed feature composition: A virtual architecture for telecommunications services," *IEEE Transactions on Software Engineering*, vol. 24, pp. 831–847, October 1998.
[237] A. Back, U. Möller, and A. Stiglic, "Traffic analysis attacks and trade-offs in anonymity providing systems," *Lecture Notes in Computer Science*, vol. 2137, 2001.
[238] F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn, "Information hiding — A survey," *Proceedings of the IEEE*, vol. 87, no. 7, pp. 1062–1078, 1999.
[239] J. He, M. A. Hiltunen, M. Rajagopalan, and R. D. Schlichting, "QoS customization in distributed object systems," *Software Practice and Experience*, vol. 33, no. 4, pp. 295–320, 2003.
[240] S. Vinoski, "Integration with web services," *IEEE Internet Computing*, November-December 2003.
[241] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo, "Middleware to support sensor network applications," *IEEE Network Magazine*, January 2004.
[242] P. Levis and D. Culler, "Maté: A tiny virtual machine for sensor networks," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, (San Jose, California), October 2002.
[243] S. R. Madden, J. M. Hellerstein, and W. Hong, "TinyDB: In-network query processing in tinyos." http://telegraph.cs.berkeley.edu/tinydb, Sept. 2003.

[244] R. Kumar, M. Wolenetz, B. Agarwalla, J. Shin, P. Hutto, A. Paul, and U. Ramachandran, "DFuse: A framework for distributed data fusion," in *Proceedings of ACM Sensys 2003*, 2003.
[245] B. Li, W. Jeon, W. Kalter, K. Nahrstedt, and J. Seo, "Adaptive middleware architecture for a distributed omnidirect ional visual tracking system," in *Proceedings of SPIE Multimedia Computing and Networking 2000 (MMCN'00)*, January 2000.
[246] B. Li and K. Nahrstedt, "A control-based middleware framework for quality of service adaptations," *IEEE Journal of Selected Areas in Communications*, vol. 17, September 1999.
[247] P. Bridges, W.-K. Chen, M. A. Hiltunen, and R. D. Schlichting, "Supporting coordinated adaptation in networked systems." `ftp://ftp.cs.arizona.edu/ftol/papers/hotos.pdf`, May 2001.
[248] V. Poladian, J. P. Sousa, D. Garlan, and M. Shaw, "Dynamic configuration of resource-aware services," in *Proceedings of the 26th International Conference on Software Engineering*, (Edinburgh, Scotland), May 2004.
[249] *IBM Systems Journal, Special issue on Autonomic Computing*, vol. 42, 2003.
[250] M. Wang and T. Suda, "The bio-networking architecture: A biologically inspired approach to the design of scalable, adaptive, and survivable/available network applications," Tech. Rep. 00-03, Department of Information and Computer Science, Unversity of California, Irvine, California, February 2000.
[251] N. Arshad, D. Heimbigner, and A. Wolf, "Deployment and dynamic reconfiguration planning for distributed software systems," in *Proceedings of the 15th International Conference on Tools with Artificial Intelligence (ICTAI03)*, 2003.
[252] M. Castaldi, A. Carzaniga, P. Inverardi, and A. Wolf, "A lightweight infrastructure for reconfiguring applications," in *Proceedings of the 11th International Workshop on Software Configuration Management (Lecture Notes in Computer Science 2049*, Springer-Verlag, Berlin, 2003.
[253] P. Robertson and R. Laddaga, "A self-adaptive architecture and its application to robust face identification," in *Proceedings 7th Pacific Rim Conference on Artificial Intelligence*, Springer-Verlag, 2002. volume 2417 of Lecture Notes in Computer Science.
[254] J. Weng and W.-S. Hwang, "An incremental learning algorithm with automatically derived discriminating features," in *Proceedings Asian Conference on Computer Vision*, (Taipei, Taiwan), pp. 426–431, January 2000.
[255] D. Isla, R. Burke, M. Downie, and B. Blumberg, "A layered brain architecture for synthetic creatures," in *Proceedings of Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, (Seatle, Washington, USA), August 2001.
[256] T. Jebara and A. Pentland, "Statistical imitative learning from perceptual data," in *Proceedings of the 2nd International Conference on Development and Learning*, (Boston, Massachusetts, USA), pp. 191–196, June 2002.
[257] L. Bergmans, "The composition filters object model," tech. rep., Department of Computer Science, University of Twente, 1994.
[258] G. A. Cohen, J. S. Chase, and D. L. Kaminsky, "Automatic program transformation with JOIE," in *Proceedings USENIX Annual Technical Symposium*, (New Orleans, Louisiana), pp. 167–178, June 1998.
[259] M. Dahm, "Byte code engineering," in *Java-Informations-Tage*, pp. 267–277, 1999.
[260] S. Chiba, "Load-time structural reflection in Java," *Lecture Notes in Computer Science*, vol. 1850, 2000.
[261] E. Bruneton and M. Riveill, "Reflective implementation of non-functional properties with the JavaPod component platform," in *Proceedings of the European Conference on Object-Oriented Programming (ECOOP 2000): Workshop On Reflection and Metalevel Architectures*, (Sophia Antipolis and Cannes, France), June 2000.
[262] A. Duncan and U. Hölzle, "Load-time adaptation: Efficient and non-intrusive language extension for virtual machines," Tech. Rep. TRCS99-09, Department of Computer Science University of California, Santa Barbara, Santa Barbara, California, April 1999.
[263] I. Welch and R. Stroud, "Dalang — a reflective extension for java," Tech. Rep. CS-TR-672, University of Newcastle upon Tyne, East Lansing, Michigan, September 1999.
[264] G. Kniesel, P. Costanza, and M. Austermann, "Jmangler - a framework for load-time transformation of java class files," in *Proceedings of IEEE Workshop on Source Code Analysis and Manipulation (SCAM)*, pp. 100–110, November 2001.
[265] R. Keller and U. Hölzle, "Binary component adaptation," in *Proceedings of the European Conference on Object-Oriented Programming (ECOOP 1998)* (E. Jul, ed.), (Brussels, Belgium), pp. 307–329, Springer-Verlag, July 1998. volume 1445 of Lecture Notes in Computer Science.
[266] N. Amano and T. Watanabe, "An approach for constructing dynamically adaptable component-based software systems using LEAD++," in *OOPSLA International Workshop on Object Oriented Reflection and Software Engineering*, (Denver, Colorado), pp. 1–16, November 1999.

[267] "The TAILOR project." `http://javalab.iai.uni-bonn.de/research/tailor/`.
[268] M. Mezini and K. Lieberherr, "Adaptive plug-and-play components for evolutionary software development," *ACM SIGPLAN Notices, Proceedings of the conference on Object-oriented programming, systems, languages, and applications*, vol. 33, October 1998.
[269] I. Ben-Shaul, O. Holder, and B. Lavva, "Dynamic adaptation and deployment of distributed components in Hadas," *IEEE Transactions on Software Engineering*, vol. 27, no. 9, pp. 769–787, 2001.
[270] Sun Microsystems, *PersonalJava Application Environment*. http://java.sun.com/products/personaljava/.
[271] D. C. Schmidt and S. D. Huston, *C++ Network Programming: Mastering Complexity Using ACE and Patterns*. Addison-Wesley Longman, 2002.
[272] http://www.cs.wustl.edu/∼schmidt/ACE-overview.html.
[273] D. C. Schmidt and T. Suda, "The service configurator framework: An extensible architecture for dynamically configuring concurrent, multi-service network daemons," in *Proceedings of the Second International Workshop on Configurable Distributed Systems*, (Pitts-burgh, PA), pp. 190–201, March 1994.
[274] P. Narasimhan, L. E. Moser, and P. M. Melliar-Smith, "Transparent fault tolerance for enterprise applications," in *Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, (L'Aquila, Italy), July-August 2000.
[275] A. D. Alexandrov, M. Ibel, K. E. Schauser, and C. J. Scheiman, "Extending the operating system at the user-level: the Ufo global file system," in *Proceedings of USENIX'97*, pp. 77–90, 1997.
[276] S. McCanne and V. Jacobson, "The BSD packet filter: A new architecture for user-level packet capture," in *USENIX Winter*, pp. 259–270, 1993.
[277] D. Lafferty and V. Cahill, "Real world evaluation of aspect-oriented programming with Iguana," in *Proceedings of the International Workshop on Aspects and Dimensional Programming at the 14th European Conference on Object-Oriented Programming (ECOOP)*, (Sophia Antipolis and Cannes, France), June 2000.
[278] A. Oliva and L. E. Buzato, "The design and implementation of Guaraná," in *Proceedings 5th USENIX Conference on Object-Oriented Technologies and Systems*, (San Diego, California), May 1999.
[279] E. G. Parmelan, "Porting Kaffe to a new platform." `http://egp.free.fr/port-kaffe/port-kaffe-0.2.html`, 2001.
[280] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Indianapolis, Indiana: Addison-Wesley, 1995.
[281] K. Burton, *.NET Common Language Runtime*. Sams Publishing, U.S.A, 2002.
[282] http://www.cs.wustl.edu/∼schmidt/TAO.html.
[283] K. Nilsen, "Issues in the design and implementation of real-time java," *Java Developers Journal*, 1996.
[284] K. Nahrstedt, H. hua Chu, and S. Narayan, "QoS-aware resource management for distributed multimedia applications," *Special issue on multimedia networking, J. High Speed Network*, Dec. 1998.
[285] F. Kon, F. Costa, G. Blair, and R. H. Campbell, "The case for reflective middleware," *Communications of the ACM*, vol. 45, pp. 33–38, June 2002.
[286] G. S. Blair, G. Coulson, A. Andersen, M. Clarke, F. M. Costa, H. A. Duran, R. Moreira, N. Paralavantzas, and K. B. Saikoski, "The design and implementation of Open ORB version 2," *IEEE Distributed Systems Online*, vol. 2, no. 6, 2001.
[287] ITU-T/ISO, *Reference Model for Open Distributed Processing, Parts 1,2,3.*, 1995. ITU-T X.901-X.904 — ISO/IEC IS 10746-(1,3,3).
[288] T. Watanabe and A. Yonezawa, "Reflection in an object-oriented concurrent language," in *Proc. ACM Conference on Object-Oriented Programming, Systems, Languages and Applicaitons (OPSLA'88)*, (San Diego, California), pp. 306–315, Sept. 1988.
[289] R. Koster, *A Middleware Platform for Information Flows*. PhD thesis, Department of Computer Science, University of Kaiserslautern, Germany, July 2002.
[290] M. van Steen, P. Homburg, and A. S. Tanenbaum, "The architectural design of Globe: A wide-area distributed system," Tech. Rep. 422, Vrije Universiteit, Amsterdam, The Netherlands, March 1997.
[291] "Orbix+Isis programmer's guide." Technical report D071-00, IONA Technologies, 1995.
[292] P. Narasimhan, L. E. Moser, and P. M. Melliar-Smith, "The interception approach to reliable distributed CORBA objects," in *Proceedings of the Third USENIC Conference on Object-Oriented Technologies and Systems (COOTS)*, USENIX, 1997.
[293] IONA Technologies, *Orbix/E*. http://www.iona.com/products/orbix-e.htm.
[294] G. Brose and N. Noffke, "JacORB 1.4 documentation," tech. rep., Freie Universitt Berlin and Xtradyne Technologies AG, August 2002.

[295] R. Schantz, J. Loyall, M. Atighetchi, and P. Pal, "Packaging quality of service control behaviors for reuse," in *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-time distributed Computing*, (Washington, DC), April 2002.
[296] E. Hadad, *Architectures for Fault-Tolerant Object-Oriented Middleware Services*. PhD thesis, Computer Science Department, The Technion - Israel Institute of Technology, 2001.
[297] L. Capra, W. Emmerich, and C. Mascolo, "Reflective middleware solutions for context-aware applications," *Lecture Notes in Computer Science*, vol. 2192, 2001.
[298] H. Miranda, M. Antunes, L. Rodrigues, and A. R. Silva, "Group communication support for dependable multi-user object-oriented environments," in *SRDS Workshop on Dependable System Middleware and Group Communication (DSMGC 2000)*, (Nürnberg, Germany), October 2000.
[299] M. Roman and R. Campbell, "Gaia: Enabling active spaces," in *Proceedings of the ninth ACM SIGOPS European Workshop*, (Kolding, Denmark), 2000.
[300] F. Kon, R. Campbell, M. D. Mickunas, K. Nahrstedt, and F. J. Ballesteros, "2K: A Distributed Operating System for Dynamic Heterogeneous Environments," *9th IEEE International Symposium on High Performance Distributed Computing*, August 2000. http://choices.cs.uiuc.edu/2k/.
[301] M. Satyanarayanan, B. Noble, P. Kumar, and M. Price, "Application-Aware Adaptation for Mobile Computing," *Operating Systems Review*, vol. 29, no. 1, pp. 52–55, 1995.
[302] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker, "Agile Application-Aware Adaptation for Mobility," *Sixteen ACM Symposium on Operating Systems Principles*, pp. 276–287, October 1997.
[303] B. Noble, "System Support for Mobile, Adaptive Applications," *IEEE Personal Communications*, pp. 44–49, February 2000.
[304] "The SwitchWare project." `http://www.cis.upenn.edu/~switchware/`.
[305] "Distributed operating systems group: Computing communities project." `http://calypso.eas.asu.edu`.
[306] R. Litiu and A. Prakash, "DACIA: A mobile component framework for building adaptive distributed applications," in *Principles of Distributed Computing (PODC) 2000 Middleware Symposium*, (Portland, Oregon), July 2000.

Appendix A Classification of Projects

In this appendix, the projects listed in Table 1 are classified according to the taxonomy introduced in Section 4. Table 8 again lists the projects in Table 1 and classifies each by showing a check mark when a project supports a feature listed in the columns under categories *How*, *When*, and *Where*. We have expanded the *How* category: in addition to the particular technique(s) used to implement compositional adaptation, listed in Table 2, the *How* category includes four additional dimensions: transparency, granularity, coverage, and compatibility.

In the *transparency* columns, we distinguish projects that are transparent to the functional application source code, to the adaptive code, to the distribution middleware services (if applicable), and to the virtual machine (if applicable). In general, if an approach enables association of adaptive code to functional code, without modifying the functional code directly, we say the approach is transparent with respect to the functional code. If the association can be made without modifying the adaptive code directly, then we say that the approach is transparent with respect to the adaptive code. For example, an application might be adapted at run time in a "generic" way, for example, to apply encryption to all network communication. If the adaptive code is independent of the functional code of the application, then the adaptation is transparent with respect to the adaptive code.

Granularity refers to the degree to which interactions in an application can be intercepted, redirected, and modified. The granularity can be per system (*e.g.*, an ORB in a CORBA application), per class (*e.g.*, a class in a class-based object-oriented paradigm or a component type in a component-based paradigm), per object (*e.g.*, an object or an instance of a type), per method (*e.g.*, a method signature irrespective of the type or class where the method is introduced), and per method call (*e.g.*, a message being passed from one object to another as the result of a method call).

Coverage distinguishes projects according to whether adaptation can be applied to local and/or remote interactions. For example, many middleware approaches apply adaptation only to communication operations. Also, we identify projects that enable selection of a subset of interactions for possible adaptation, so as not to introduce overhead unnecessarily on other interactions.

Finally, in the *Support* columns, we identify projects that support the standard distribution frameworks: CORBA/CCM, Java RMI/J2EE, and DCOM/.NET Remoting.

This technical report is intended to be a living document. We expect Table 8 to grow and evolve over time, as additional projects are classified and as new capabilities are added to existing projects. We intend this document to serve as a tool for research in compositional adaptation, and we welcome input from the research community on projects not listed and on any necessary corrections. In the remainder of this appendix, we summarize a number of projects and briefly discuss the rationale for how they are classified. In this version, we do not discuss projects involving domain-specific services or cross-layer adaptation; those will be addressed later.

A.1 Language-Based Projects

We begin by discussing approaches that language-oriented techniques to support compositional adaptation. Considering Table 8, we see that all the approaches listed use some type of metaobject protocol, among other mechanisms. Moreover, since these solutions can be applied to arbitrary application code, their coverage of interactions among elements is very general (local, remote, selected). Mainly, differences are found in their granularity and the time at which recomposition occurs.

OpenJava. *OpenJava* [57] is a macro-based extension to Java that enables customization of class structures, alteration of the behavior of operations on objects, alteration of variable types, and extension of the Java syntax. OpenJava addresses the limitations of the Java reflection by enabling behavioral reflection facilities to be added to a Java program at compile time. The macros are defined to manipulate *class metaobjects* representing program entities. For example, these facilities enable inspection of a Java program (*e.g.*, to determine the class of a given object, along with the methods and fields of that class, and to perform limited operations at run time (*e.g.*, to get and set an object's field value and invoke one of its methods). OpenJava provides an openjava.mop.OJClass class, which extends the java.lang.Class class. The OJClass complements the Java structural reflection facilities at run time by extending the Class methods. Moreover, the OJClass provides a method, called transformClass(), that can be overwritten by a meta-level programmer to customize a Java program at compile time. A source-to-source compiler transforms a Java program according to the subclass of the OJClass provided by the meta-level programmer, using the transformClass() method.

We classify OpenJava as follows. To support compositional adaptation, OpenJava uses a meta-object protocol technique (*How*) at compile time (*When*) to customize a Java application (*Where*). Since the customization process is

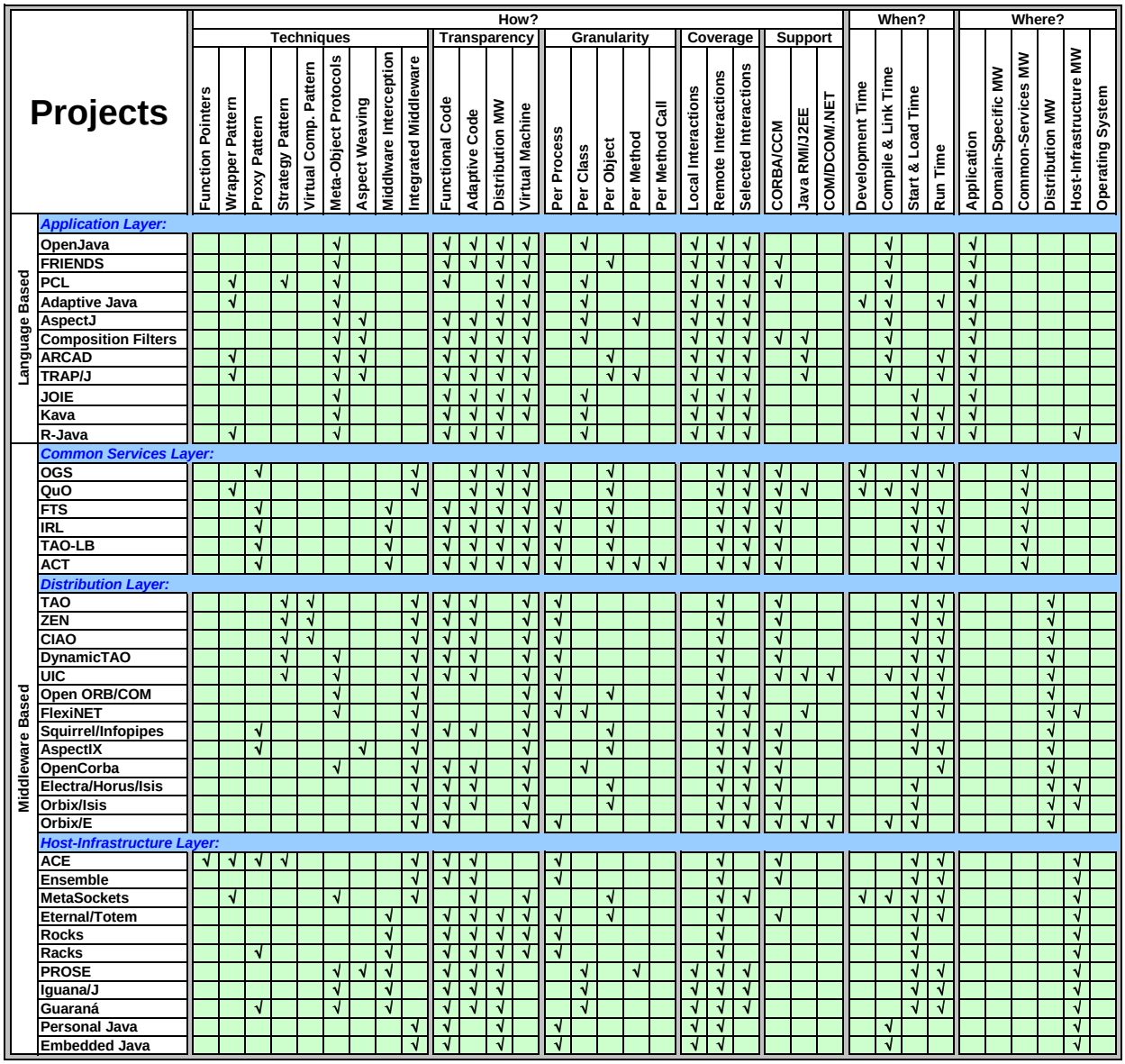


Figure 8: Categorizing projects according to the taxonomy introduced in Section 4

performed by the source-to-source compiler, OpenJava is transparent with respect to the functional code of the Java program. This process is also transparent with respect to the virtual machine, since the transformed source code is a Java program that can be compiled with a standard Java compiler and executed on a standard JVM. In addition, when adding "generic" types of new behavior, such as profiling the execution sequence, the meta-level programmer does not need to know in advance which base-level classes will be associated with the meta-level classes. Therefore, OpenJava supports transparency with respect to the adaptive code as well. OpenJava enables association of meta-level classes to the base-level *classes*, so the granularity of compositional adaptation is per class. Since this association can be for a selected set of base-level classes, coverage is general (open, remote, selected). Finally, while OpenJava does not provide any *explicit* support for CORBA, Java RMI, and COM/DCOM distribution middleware technologies, we note that it can be used to support adaptation for remote objects.

FRIENDS. The *FRIENDS system* [147], developed by Fabre et al., provides a meta-level architecture that programmers can use to develop dependable systems. The system comprises a set of metaobject libraries (written in both Open C++ and OpenJava) for fault tolerance, secure communication, and group-based distributed applications. FRIENDS

enables non-functional mechanisms to be implemented at the meta level. Reflection is used to address various non-functional requirements: fault tolerance using several replication strategies, security using ciphering and authentication protocols, and communication using atomic multicast protocols. Figure 9 shows the multi-level implementation of the client-server protocol using fault-tolerant, security, and communication meta objects. A specialized MOP enables interception of the interactions between CORBA objects. Using this MOP, FRIENDS provides fault-tolerant replicas as CORBA objects (or meta objects).

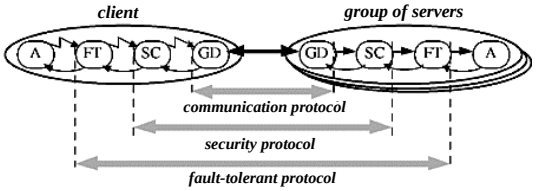


Figure 9: FRIENDS architecture [147].

FRIENDS use a meta-object protocol technique (*How*) at compile time (*When*) to adapt the code of a distributed application (*Where*). The use of an open compiler in FRIENDS makes it transparent with respect to both the functional code, adaptive code, and the virtual machine. While FRIENDS provides specific support for CORBA applications, the resulting transformation is transparent to the CORBA implementation, so we we consider FRIENDS transparent to the distribution middleware, Since FRIENDS enables association of meta-level classes to individual CORBA objects and clients, the granularity is per object.

PCL. *Program Control Logic (PCL)* [132] is a programming framework that enables design, development, and performance optimization of adaptive distributed applications. A source-to-source compiler is provided, which inputs meta code specified in either PCLC or PCLJ (languages very close to C++ and Java, respectively) and outputs a program source in C++ or Java that is then compiled and linked with the base program. The PCL project focuses on language support for run-time adaptability. PCL uses *Adaptors* to wrap and adapt original application classes. Adaptors are typically subclasses of targets and have access to many of the variables defined in the target class. Adaptors can change the behavior of a single target class through the use of variables and methods, called `ControlParameters` and `ControlMethods`, visible in the superclass to the Adaptor. The Adaptor can read and modify `ControlParameters` and call `ControlMethods` as part of the adaptation policy, enabling parameter observation and modification. Adaptation policies are specified using metrics associated with variables of the target class. Metrics can be sampled, timed or rate based, and are used to trigger events that execute adaptive processing.

PCL uses strategy pattern (*e.g.*, to enable selection of the best algorithm at run time among a list of algorithms known at compile time) and assigns controlling adaptors to application classes using a meta-object protocol technique (*How*) at compile time (*When*) to adapt a C++ or Java application (*Where*). Although PCL supports run-time adaptation through selection of algorithms and modification to the variables of a program at run time, a new algorithm or adaptation mechanism cannot be introduced after compile time. Therefore, we do not classify it as supporting compositional adaptation at run time. The adaptation process is performed by a source-to-source compiler that is transparent with respect to the program functional code. The transformed source code is either a C++ or Java program that can be compiled with a standard C++ or Java compiler, so transparency with respect to the virtual machine and distribution middleware (if used) is preserved. However, it seems that the adaptors in PCL must be aware of the variables of the class to which they are assigned, so the method is not transparent with respect to the adaptive code also. Because adaptors are assigned to application classes, we consider the granularity of PCL to be per class.

Adaptive Java. *Adaptive Java* [44] is an extension to Java that introduces new language constructs to support behavioral reflection. In its behavioral reflective meta-model architecture, Adaptive Java separates monitoring the behavior (introspection) from changing the behavior (intercession), using “refractive” and “transmutative” meta methods, respectively. By defining a reflection-based component model, Adaptive Java also supports run-time reconfiguration. Adaptive Java’s approach to composition using encapsulation can be used to instantiate a message filtering design, where components are extended and invocations added such that a call to an invocation would be filtered through subsequent encapsulation layers.

To support compositional adaptation, Adaptive Java uses the wrapper pattern to encapsulate application objects inside their meta-level objects and uses a meta-object protocol technique (*How*). These actions produce an *adapt-ready* version of a Java application (*Where*) at compile time that can be recomposed (including introduction of new

components) at run time (*When*). Unlike the projects discussed above, the developer is required to modify the source code of a Java program directly, so the column "Development Time" in Table 8 is checked. In addition, this direct modification means the method is transparent to neither the functional code nor the adaptive code. However, since Adaptive Java uses source-to-source compiler, the generated adapt ready program can be compiled with a standard Java compiler and can be executed by a standard JVM. The granularity is per class, since entire classes are metafied.

AspectJ. *AspectJ* [115] is an extension to Java that enables a developer to program crosscutting concerns of a Java program in separate modules, called *aspects*. At compile time, a number of such aspects can be selected to be woven into the Java source or class files of the application using the AspectJ compiler, called *aspect weaver*, to produce a special version of the application. An aspect in AspectJ has two parts: advice and pointcut. *Advice* is an implementation of a crosscutting concern and a *pointcut* is a set of joinpoints, at which the advice is woven. A *joinpoint* is an identifiable point in the execution path of an application, such as a method call or a access to a field.

AspectJ uses aspect weaving (*How*) at compile time (*When*) to modify a Java application (*Where*). We note that AspectJ also provides reflection facilities that add to the power of the Java reflection, so we have checked the "Meta-Object Protocol" box. The weaving process is transparent to the application functional code, the JVM, and any distribution middleware. Moreover, since it is possible to develop generic advice (for example, tracking program flow or performing security checks) that is independent of the application classes to which it is eventually applied, AspectJ supports transparency to the adaptive code. In terms of granularity, a joinpoint in AspectJ can be assigned to a class or to a method of a class. *[pkm: Masoud, please clarify the following sentence.]* Although, using the AspectJ reflection facilities, an aspect developer can associate an aspect to a single object or a single method call, we do not consider per object and per method call granularity for AspectJ, because such granularity is not directly supported in AspectJ. Finally, we note that while AspectJ itself is a compile-time method, it can be used to support run-time recomposition by weaving appropriate "hooks" into a program. This approach is used in ARCAD [124] and TRAP/J [62], among others.

Composition Filters. *Composition filters* [40, 56, 56, 257] provide a mechanism for disentangling the crosscutting concerns of the interactions among objects of an object-oriented program (*e.g.*, a C++ or Java program) or among remote objects of a distributed program (*e.g.*, a CORBA program in Orbix). Specifically, this system declares filters that intercept messages received and sent by objects. As such, messages can be massaged and checked before they are delivered to an object. In this manner, aspects such as security authentication or bounds checking can be separated from the objects that send and receive these messages. We note that one advantage of composition filters over AspectJ is that the code for filters is not (eventually) tangled with the the application code. Therefore, using the well-defined meta-object protocols defined in composition filters, a programmer can insert filters into, or remove them from, an application at run time. Preprocessing produces objects wrapped with a special *interface*. Depicted in Figure 10, *filters* can be applied to these interfaces, filtering incoming and outgoing messages, applying rules or passing them to the next *filterset* when no matches are found. Rules can reject or massage a message or dispatch it to an underlying object. Composition Filters implement *superimposition* by applying rules at application locations specified using wild cards or signatures. Superimposition can simulate dynamic inheritance, where program flow is redirected to filters by rules superimposed on classes or objects. Even functional concerns can be occluded and replaced by new implementations.

Composition filters use aspect weaving (*How*) that introduce filters at compile time (*When*) into the classes of a Java or C++ program (*Where*). As with several other approaches, the weaving process is transparent to the application functional code, to the virtual machine, and to any distribution middleware. Also, generic filters can be defined, providing transparency to the adaptive code. Filters are designed for entire classes. However, we note that filters can be developed to be effective with finer granularity such as per object, per method, or per method call. *[pkm: Masoud, explain what you mean above and why the boxes are not checked.]* Finally, composition filters have been applied to CORBA (using Orbix) and Java RMI distributed applications, so we check those boxes. *[pkm: per our macro discussion, is there a set of CFs designed explicitly for CORBA, etc? We should probably mention the specifics and give references.]*

ARCAD. The RNTL ARCAD project [124] uses a two-step approach to support compositional adaptation in existing Java applications (*Where*). In the first step, generic interception hooks are woven into an existing Java at compile time using the AspectJ aspect weaver. In the second step, intercepted operations are forwarded to the meta-level objects, which are programmable at run time. Therefore, we check both the compile time and the run time boxes (*When*). ARCAD uses a MOP library, called RAM, to attach or detach meta-objects to application objects (instances of classes

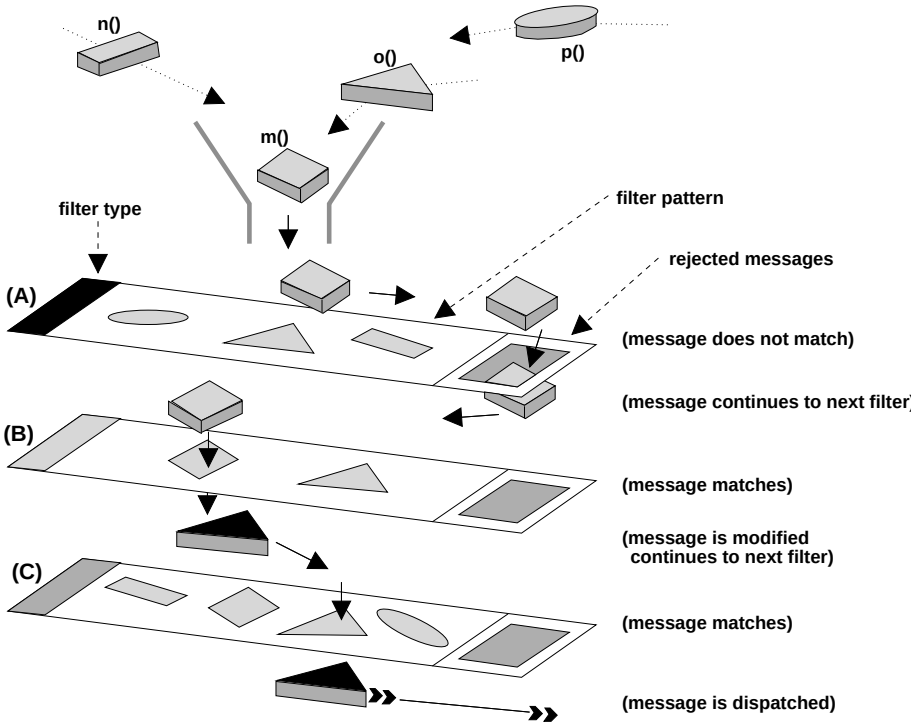


Figure 10: An intuitive schema of message filtering [257]. In this diagram, (A), (B) and (C) are three filters, while m(), n(), o() and p() are messages. Following message m(), filter (A) rejects m(), passing it to filter (B). Filter (B) matches m() and modifies m(). Filter (C) matches the modified message m() and dispatches it to a target object.

modified at compile time using AspectJ) at run time. The code for meta-objects in ARCAD is written in Java, while the code for filters in composition filters is written in a high-level language that can be reused for programs written in other languages such as C++, if a compiler is supported for that language.

Since the aspect weaving is used to wrap a selected set of classes, ARCAD combines three techniques: aspect weaving, wrapping, and meta-object protocols (*How*). to support composition technique supported by AspectJ and provides generic aspects that wrap a selected set of the classes As in AspectJ, the weaving process is transparent to the application functional code, to the virtual machine, and to any distribution middleware. The MOP library provides reusable meta-level classes that are developed without the need to know to which application classes they will be assigned, so ARCAD supports transparency to the adaptive code. The granularity of adaptation supported in ARCAD is considered per object, because meta-objects in ARCAD are attached to or detached from application objects at run time.

TRAP/J. *TRAP/J* [62] is a software tool that, similar to ARCAD, provides a two-step approach to dynamic adaptation in Java applications (*Where*). TRAP/J enables new adaptable behavior to be added to existing Java applications transparently (that is, without modifying the application source code and without modifying the JVM). Specifically, TRAP/J enables the developer to select, at compile time, a subset of classes in the existing program that are to be adaptable at run time. As illustrated in Figure 11, TRAP/J then generates aspects and reflective classes associated with the selected classes, producing an *adapt-ready* program. The aspects are generic in that simply provide hooks that intercept the program flow. Moreover, a generic MOP provides a means by which to introduce new code, referred to as a *delegate*, to be executed upon such an interception.

In terms of classification, TRAP/J uses the same three techniques as ARCAD (wrapping, weaving, MOPs) (*How*) and provides both compile-time and run-time composition (*When*). In fact, the only difference from ARCAD's classification is that TRAP/J's generic MOP enables the implementation of each individual methods of each instance of an adaptable class to be modified at run time, through insertion or removal of delegate classes. The delegate classes themselves can support generic functionality, so TRAP/J can be transparent to the adaptive code.

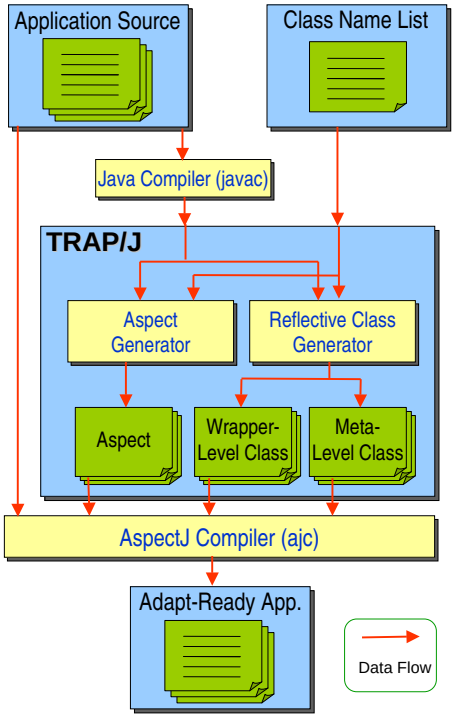


Figure 11: TRAP/J operation at compile time.

Java Object Instrumentation Environment (JOIE). The Java Object Instrumentation Environment (JOIE) [258] is a toolkit that enables existing Java class files to be modified when loaded by the JVM. JOIE provides a specialized Java class loader that enables the registration of user-specified *transformers*, which implement policies that modify Java byte code. The class loader constructs a reflective `ClassInfo` object, encapsulating the byte code of the class as it is loaded. This `ClassInfo` object is then passed to each of the registered transformers. `ClassInfo` objects provide an interface that enables the manipulation of byte code such that functionality can be modified or extended. As such, calls to debugging or tracing routines can be inserted, enabling existing compiled classes to be instrumented. Moreover, algorithms and variables can be modified through byte code splicing.

JOIE uses a meta-object protocol mechanism (*How*) at load time (*When*) to modify byte code of class files of a Java application (*Where*). The byte-code modification is transparent to the application functional code, to the virtual machine, and to any distribution middleware. The library of reusable class loaders that can be used to modify Java program in ways independent of the application code, so JOIE supports transparency to the adaptive code. Finally, the modifications are carried out on a class basis, so the level of granularity is the class.

Kava. Kava [118] uses load-time byte-code rewriting to support dynamic adaptation of Java programs at run time. Kava addresses the problems with byte-code rewriting toolkits such as JOIE [180] (discussed above), Byte Code Engineering Library [259], and Javassist [260] that do not support the express behavioral modifications in Java so that they can be compiled and verified as Java classes. Specifically, Kava uses the Byte Code Engineering Library [259] toolkit to incorporate its run-time meta-object protocol in a Java program. Kava provides an XML-like binding language that can be used to write configuration files to be used in the transformation process at load time. The configuration file plays the same role as the pointcuts in AspectJ [115]. Kava allows each class to be bound to a meta-level object, where behaviors such as method invocation, method execution, and field access can be modified dynamically. Traps inserted into class files at load time enable run-time redirection of the execution to the meta-level object.

Kava uses a meta-object protocol mechanism (*How*) to modify the Java program byte code at load time and to support dynamic adaptation at run time(*When*) at the application layer (*Where*). Kava supports dynamic adaptation by enabling dynamic manipulation of meta-level objects, but its adaptation is limited to the classes modified at load time (*i.e.,* Kava supports anticipated dynamic adaptation). The byte-code modification is transparent to the application functional code, to the virtual machine, and to any distribution middleware. Kava supports transparency to the adaptive code through the use of configuration files, which is used as a glue code between the application functional code (base-

level classes) and the adaptive code (meta-level objects). Since Kava meta-level objects are assigned to base-level classes, the level of granularity is the class.

R-Java. *R-Java* [58] is Java extension that supports a type of statically-typed metaobject called dynamic shells. Unlike many approaches, R-Java is designed to minimize the overhead when no adaptive behavior is required. However, the cost is a a new instruction, (chclass), added to the Java language that enables developers to explicitly change the class of objects at run time. (The new class must be a "subclass" of the original class.) Hence, the JVM requires this modification. Basically, R-Java is categorized similar to Kava, except that it is not support transparency with respect to the JVM. Also, the "Host-Infrastructure MW" column in the *Where* section is checked to indicate the need for a modified JVM.

Other Language-Based Approaches. Numerous other projects apply language-based approaches to compositional adaptation. Examples include Open C++ [135], Reflective Java [60], Hyper/J [116], Aspectual Components [117,171], Reflex [133], Handi-Wrap [134], JAC [59], JavaPod [261], Proteus [262], Dalang [263], Byte Code Engineering Library [259], Javassist [260], JMangler [264], Dynamic Weaver Framework (DWF) [120], Binary Component Adaptation (BCA) [265], DAS [266], TAILOR [267], Adaptive Plug-and-Play Components (APPCs) [268], DADO [222], and Hadas [269]. These and other projects will be classified at a later date.

A.2 Host-Infrastructure Middleware Projects

Projects involving compositional adaptation at the host-infrastructure layer generally fall in one of two groups. The first group includes those approaches that construct a layer of adaptable communication services, which can be used to adapt remote interactions (*e.g.*, ACE [166], Ensemble [42], MetaSockets [150], Eternal [196], Rocks [198], and Racks [198]). The second group includes approaches that are based on virtual machines. Unlike the first group, approaches in this group can be used to adapt *both* local and remote interactions, since a virtual machine intercepts all interactions. Approaches in this group may extend a standard virtual machine (*e.g.*, PROSE [140], Guaraná [141], and Iguana/J [61]), or may generate a customized version of a standard virtual machine (*e.g.*, PersonalJava [270] and EmbeddedJava [199]). Many benefit from facilities provided in the standard JVM, including Java reflection for inspection and (limited) manipulation of a Java program, the forName() static method for dynamic class loading, and command-line parameters (*e.g.*, -Xbootclasspath and various HotSpot options) for custom configuration. While providing transparency to the application code, however, approaches in this latter category reduce portability with respect to the standard virtual machines.

ACE. One of the earliest middleware projects in this category is Schmidt's *Adaptive Communication Environment (ACE)* [166, 271], a real-time object-oriented framework written in C++. ACE wraps many OS services in C++ wrappers and provides a variety of communication-related patterns for use by distributed applications. ACE is designed to support distributed applications with efficiency and predictability, including low latency for delay-sensitive applications, high performance for bandwidth-intensive applications, and predictability for real-time applications. Figure 12 illustrates the key components in the ACE framework. Note that the *OS Adaptation Layer* resides directly atop the native operating system APIs providing a platform-independent API, which is why we place ACE in the host-infrastructure layer (*Where*). ACE can be configured at startup time using configuration files and can be reconfigured at run time using a component configurator pattern [163] and C++ dynamic binding features (*When*).

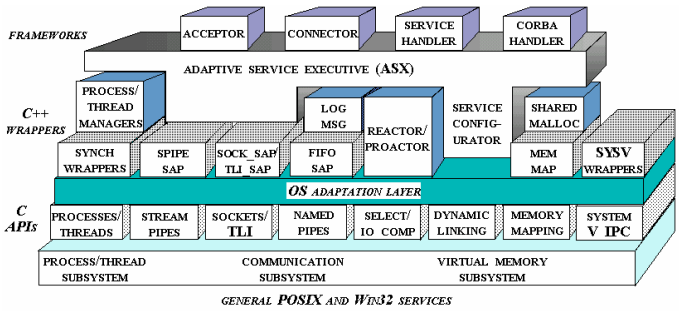


Figure 12: ACE architecture [272].

To support compositional adaptation, ACE employs C++ function pointers and software design patterns, and can be incorporated into a distributed application using the integrated middleware technique (*How*). Although ACE was designed to be used directly in distributed applications, with the development of TAO [165] (discussed later), ACE is more commonly used indirectly via TAO. In this mode, the ACE adaptive services are used explicitly by TAO and remain transparent to the application code. For example, TAO uses the ACE service configurator framework [273], which implements the component configurator pattern [163], to load and initialize a POA component (a CORBA portable object adapter that dispatches remote calls to CORBA servants) at run time. The specific implementation of a POA component is transparent to the application code, and a POA implementation does not need to have any specific knowledge of the application using it. Therefore, ACE provides transparency to both functional and adaptive code. If ACE were incorporated into an application directly (not through TAO), then it would not be transparent to the application code. Since the ACE core remains intact during adaptation, we consider it as repeatedly-tunable (but not mutable) middleware. Moreover, ACE provides a process-wide adaptation, since adaptations in ACE (such as the one discussed above) are applied to the entire program (as opposed to being applied to only a class of objects, a specific object, a method of a class, or a method call).

Ensemble. *Ensemble* [42] from Cornell University is a groupware communication toolkit that supports distributed applications with application-specific communication protocols (*Where*). Ensemble is a follow-on project to Horus [192] and is implemented in a variant of the ML programming language to facilitate formal verification (Horus was written in C). Central to the design is the construction of protocol stacks from fine-grained components, called micro-protocols. A set of communicating processes use a common stack configuration (represented as a protocol graph) to support group operations. The Ensemble framework is designed to provided QoS monitoring, reliability, high availability, fault-tolerance, consistency, security, and real-time responsiveness in distributed applications. For example, to support QoS monitoring, Ensemble enables insertion of detectors in the protocol graph. These detectors can trigger dynamic adaptation by distributing a new protocol-graph specification to all involved participants using a reconfiguration protocol.

Ensemble provides a number of reusable micro-protocols in its library, and new micro-protocols can also be developed and used in Ensemble. Ensemble can be configured at startup time by stacking a number of micro-protocols on top of each other and reconfigured at run time by rearranging the micro-protocol stack (*When*). Examples of dynamic adaptations supported in Ensemble include adaptation with respect to changes in network bandwidth and latency, to processor and network failures, and to changing security policies. The Ensemble core remains intact during adaptation, so we consider it to be repeatedly-tunable middleware. The classification of Ensemble is similar to that of ACE, with the exception of the mechanisms used for compositional adaptation. In Ensemble, these custom mechanisms are built into the framework, so we categorize them as integrated middleware (*How*). As with ACE, Ensemble can be incorporated into a distributed application either directly, where the application code uses Ensemble adaptive services explicitly, or through the a distribution middleware (Electra [191]) (discussed later), in which case the Ensemble adaptive services remain transparent to the application code. Finally, similar to ACE, Ensemble provides a process-wide adaptation.

MetaSockets. *MetaSockets* [150] from Michigan State University are adaptable communication components created from existing Java socket classes using Adaptive Java [44], a reflective extension to Java discussed earlier. As illustrated in Figure 13, MetaSockets support compositional adaptation through the use of a filter pipeline that allows insertion and removal of filters dynamically in response to external stimuli. A *filter* performs a single operation on a stream of data passing through the filter pipeline. Comparing MetaSockets to Ensemble, the filter pipeline and filters in MetaSockets play a role similar to that of the protocol graph and micro-protocols in Ensemble, respectively. However, the level of abstraction supported in MetaSockets is Java *sockets*, whereas the level of abstraction in Ensemble is network protocols.

MetaSockets use the reflection facilities and the wrapper pattern provided in Adaptive Java [44]. Typically, they are incorporated into an application using the integration middleware technique (*How*). That is, an application developer can use MetaSockets directly, in which case MetaSockets are explicitly declared in the application source code at development time and compiled by the Adaptive Java compiler at compile time (*When*). As such, the use of MetaSockets is not transparent to the application code. However, we note that tools such as TRAP/J can be used to weave MetaSockets into an application transparently. Similar to micro-protocols of Ensemble, the filters in MetaSockets can be developed by third parties and can be independent of a specific application code. For example, a pair of encryption/decryption filters can be developed independent of the type of data being secured or the application code using the filters. Therefore, MetaSockets provide transparency with respect to adaptive code. The code for MetaSock-

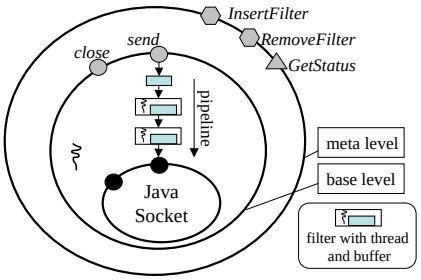


Figure 13: MetaSockets pipeline.

ets is compiled by the Adaptive Java compiler, which is a source-to-source compiler, so the resulting Java program can be compiled by the standard Java compilers and run by the standard JVM. A MetaSocket can be configured at startup time by inserting filters into its filter pipeline and can be reconfigured at run time by inserting, removing, or modifying the filters dynamically (*When*). Since the core MetaSocket code remains intact during the tuning process, we classify them as repeatedly-tunable middleware. Finally, the granularity of adaptation in MetaSockets is the individual Java socket object, since each MetaSocket can be configured differently at run time.

Eternal. *Eternal* [274] from UCSB and Eternal Systems, is a component-based middleware to provide fault tolerance to CORBA applications by replicating CORBA objects. As illustrated in Figure 14, Eternal is incorporated into a CORBA application using a middleware interception approach (*How*), which is transparent both to the application code and CORBA distribution middleware implementation. Eternal intercepts system requests originated from unmodified CORBA ORBs targeted for the kernel TCP/IP protocol stack using the operating system user-level extensions [275] (*Where*), and transfers them to a replication manager. Eternal uses a reliable multicast protocol (specifically, Totem [197]) in maintaining consistency among replicas.

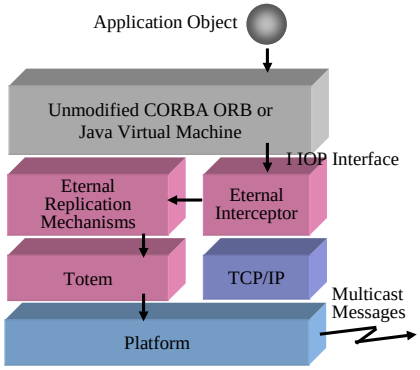


Figure 14: The Eternal architecture [196].

To use Eternal in an existing CORBA application, Eternal must be configured at application startup time to set up the monitoring and capturing of system calls. The system can be reconfigured at run time to modify the number of CORBA replicates and the replication strategy (*When*). The existence of the replication, and hence the configuration and reconfiguration, are transparent to the CORBA objects; hence, Eternal provides transparency to the application code. Moreover, using Eternal, one can choose alternative strategies (e.g., active or passive replication) to provide fault-tolerance. Since these strategies are independent of particular applications, Eternal provides transparency to the adaptive code. Finally, Eternal provides adaptation at the process level by intercepting all the calls to TCP/IP and at the CORBA object level using replicates.

Rocks. Zandy et al. [198] at the University of Wisconsin developed *reliable sockets (Rocks)* to protect socket-based applications from poor network conditions, specifically, connection failures in mobile computing environments. Examples include unexpected modem disconnections and IP address changes as a result of mobile device movements or DHCP lease expiration. Rocks resume sessions automatically after recovering from a period of disconnection. Using the "preloading" feature of the Linux loader [198], the Rocks library is interposed between the application code

and the kernel TCP socket (*Where*), as shown in Figure 15. Rocks monitor the TCP socket send and receive buffers and maintain a copy of in-flight packets to prevent data loss in the presence of connection failure. After reconnection, Rocks first resends the packets in the in-flight buffers and then resumes the TCP socket to continue its normal operation.

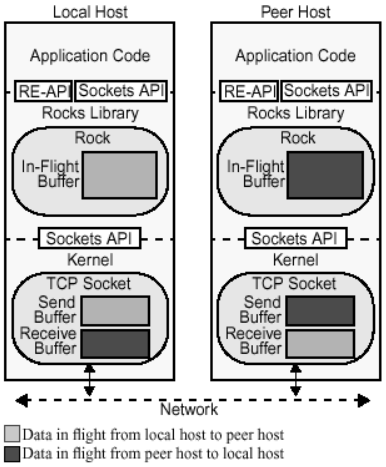


Figure 15: Rocks architecture [198].

The Rocks library exports the socket API, which is the same as the kernel socket API to be used transparently by the application. This middleware interception approach (*How*), similar to that of Eternal, means that Rocks reconfiguration is transparent to the application code, as well as to any distribution middleware or virtual machine. The reliability provided in Rocks is independent of specific applications; hence, Rocks is also transparency to the adaptive code. An enhanced API (RE-API) is available for use by Rocks-aware applications. We consider Rocks as configurable middleware because the interposition of the Rocks library must be done at the application startup time (*When*). Finally, since an adaptation applies to all connections of a process, the level of granularity is the process.

Racks. *Reliable packets (racks)* [198], also developed at University of Wisconsin, offer an alternative solution to Rocks that solves the following problems introduced by Linux preloading feature. First, the preloading feature depends on the dynamic linker while, for security reasons, the dynamic linker is disabled on "setuid" binaries. Second, system libraries may not correctly support preloading because they might have used static calls that cannot be trapped using preloading. Finally, it is possible that other interposed libraries coexist with the Rocks library; the ordering of the libraries affects the correct functioning of the Rocks library. As depicted in Figure 16, Racks are implemented as a separate daemon process (Rackd), as opposed to the in-process approach in Rocks.

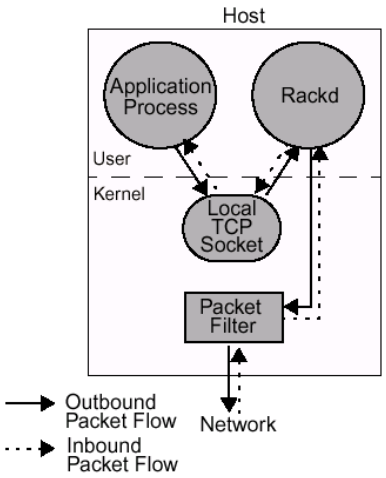


Figure 16: Racks architecture [198].

Instead of intercepting the socket calls as in Rocks or intercepting TCP calls as in Eternal, Racks intercept and manipulate packets using a "packet filter" approach [276], which is a kernel mechanism that enables user processes to select and intercept outgoing and incoming packets (*Where*). Although different than many middleware mechanisms, we classify this approach as a middleware interception technique (*How*). We consider Racks as configurable middleware because of the need to register a packet filter with the kernel during the application startup time (*When*). Similar to Rocks and Eternal, the adaptation supported in Racks is transparent to the application code, to the adaptive code, and to any distribution middleware and virtual machine. As in Rocks, Racks provides a per process adaptation granularity.

PROSE. *PROgrammable extenSions of sErvices (PROSE)* [140, 154] is an extension to the standard JVM (*where*) that supports dynamic weaving of aspects into Java programs. The authors describe their approach as *homogeneous*, which means the same language is used for both the application and aspects. (unlike AspectJ [115], which uses a different language to define aspects and pointcuts). In PROSE, aspects and joinpoints are constructed by extending the Aspect and Joinpoint classes, respectively. Weaving instructions are defined using the JVM debug interface (JVMDI), which is a middleware interception technique (*How*). When an application reaches a joinpoint, program execution is halted and the PROSE Java Machine Aspect Interface (JVMAI) receives notification of a joinpoint event. Then, PROSE executes the aspects associated with the joinpoint.

PROSE must be configured with the standard JVM to set the JVMDI at startup time and can be reconfigured at run time using its dynamic aspect weaving capabilities (*When*). The adaptation supported in PROSE is transparent to application code, since setting up the JVMDI at startup time and intercepting interactions at run time used in PROSE does not need any support from the application code. As in AspectJ, PROSE provides transparency to adaptive code also, by enabling development of generic aspects (*e.g.*, profiling aspects) that can be developed independent of specific application code. Such a generic aspect can be assigned to specific classes or methods of a Java program using joinpoints. PROSE provides adaptation at the class and method level for the same reasons as discussed in AspectJ. Finally, PROSE can be used to adapt both local and remote interaction, because it can intercept all interactions that go through the JVM.

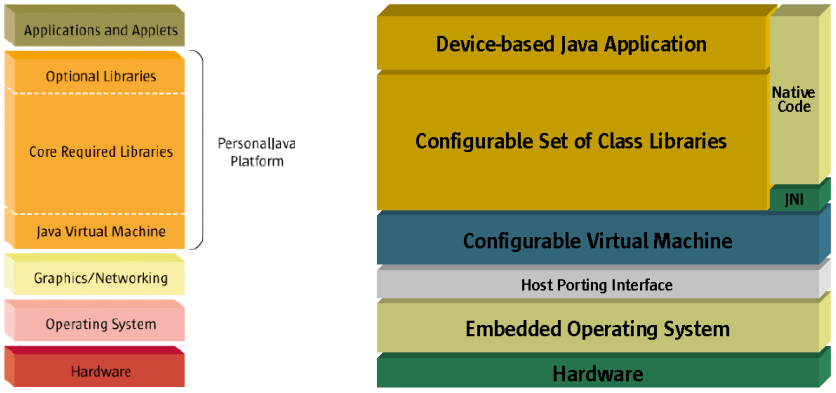
Iguana/J. *Iguana/J* [61] extends the standard JVM (*Where*) to intercept method invocation, object creation, and field reads and writes at run time. A key feature of Iguana/J is its support for *unanticipated* changes, where modifications are not foreseen during system design. Rather than focus on reweaving and superimposition as in PROSE [140] and Composition Fitlers [56], respectively, Iguana/J [61, 277] supports construction of applications that use reflection and metaobject protocols for behavioral modification (*How*). Specifically, Iguana/J intercepts and reifies base-level operations (*e.g.*, method invocations, object creations, and fields read and write), and redirects them to the corresponding metaobject.

Iguana/J must be configured with the JVM at startup time but supports dynamic adaptation at run time by assigning and reassigning metaobjects to classes (*When*). The adaptation is transparent to the application code, since similar to PROSE, the JVM configuration at startup time does not require any support from the application code. Iguana/J supports transparency with respect to the adaptive code, since generic metaobject protocols can be developed independent of the specific implementation of a Java program. Later, at run time, the generic MOPs can be assigned to various classes in the Java program. Finally, Iguana/J provides adaptation at the class level and, similar to PROSE, can be used to adapt both the local and remote interaction.

Guaraná. *Guaraná* [141, 278] also implements a metaobject protocol in Java by extending the JVM (*How*), Instead of configuring the standard JVM by Sun Microsystems, however, Guaraná extends the Kaffe OpenVM [279] (*Where*). The approach uses *composers* to delegate control to other metaobjects and to other composers. The composer structure forms a composite pattern [280] that allows multiple reflective views of a base-level object. The structure formed by metalevel objects for a base-level object is called the *metaconfiguration* of the base-level object. Like Composition Filters [56], Guaraná composers are capable of delegating an operation to multiple metaobjects for processing. The returned results are composed by the composer and returned to the caller. *Proxies* can be used to reincarnate an object from persistent storage or migrate an object to another machine. Similar to Iguana/J, Guaraná configures JVM at startup time to set the JIT interface and to disable the Java HotSpot, and can be used for dynamic adaptation at run time, by changing the metaconfiguration and issuing reconfiguration requests dynamically (*When*).

Personal Java and Embedded Java. *PersonalJava* [270] and *EmbeddedJava* [199] constitute a minimum Java that reimplements the full set of Java APIs in order to fit into smaller devices with limited memory (*Where*). PersonalJava

is designed for "web-connected" devices such as set-top boxes, smart phones, and hand-held devices like PDAs. As depicted in Figure 17(a), a minimum standard core is required on every PersonalJava-enabled device to enable the web functionality. Unlike PersonalJava, EmbeddedJava enables automatic creation of customized APIs relative to the requirements of one application, as opposed to one application domain, which results in smaller footprints. As such, EmbeddedJava enables Java applications on very limited memory embedded devices, including industrial controllers, process controllers, and scientific instruments. Every EmbeddedJava application may include different set of classes since there is no required core functionality for all embedded devices as depicted in Figure 17(b).



(a) Personal Java [270]. (b) Embedded Java [199].

Figure 17: Architecture of PersonalJava and EmbeddedJava

Both PersonalJava and EmbeddedJava use a middleware integration technique to be incorporated to a Java program (*How*). The incorporation is performed at compile time when a customized version of Java API is generated for a specific application (*When*). PersonalJava and EmbeddedJava support a static approach to compositional adaptation because they produce minimum applications at compile time and provide no specific support for dynamic adaptation. The adaptation supported in both PersonalJava and EmbeddedJava is transparent to the application code, since the customization takes place after the application is developed. Finally, both PersonalJava and EmbeddedJava provide adaptation at the process level and can be used to adapt both the local and remote interactions.

Other Host-Infrastructure Middleware-Based Approaches. Other host-infrastructure middleware-based approaches to compositional adaptation include Microsoft's .NET CLR [281], metaXa (also known as MetaJava) [138, 139].

A.3 Distribution Middleware Projects

Much of the research in adaptive middleware has focused on the distribution layer, which moves the portability issue closer to the application. The projects listed in Table 8 constitute only a representative sample. In considering the table, we note that while the projects employ a variety of techniques, all include integrated middleware, and therefore none is transparent with respect to the distribution middleware. However, all are transparent with respect to the virtual machine. The granularity in these projects is relatively coarse, and many apply to the entire process (for example, in a CORBA application, by requiring changes to all ORBs). In fact, the majority of the projects are based on CORBA. All apply to remote interactions, by definition, but many enable selecting subsets of interactions for adaptation. Most enable configuration at startup time and then enable reconfiguration at run time. Now, let us review each of the projects and discuss their classification.

TAO/ZEN/CIAO. Schmidt et al. [165] extended their ACE work to create *The ACE ORB (TAO)*, a CORBA compliant real-time ORB built atop the ACE components, shown in Figure 18. TAO enhances the standard CORBA event service to provide real-time event dispatching and scheduling required by real-time applications such as avionics, telecommunications and network management systems. Earlier versions of TAO employ the strategy design pattern [162] to encapsulate different aspects of the ORB internals, such as IIOP *pluggable protocols*, concurrency, request demultiplexing, scheduling, and connection management. A configuration file is used to specify the strategies used to

implement these aspects during startup time. TAO parses the configuration file and loads the required strategies.

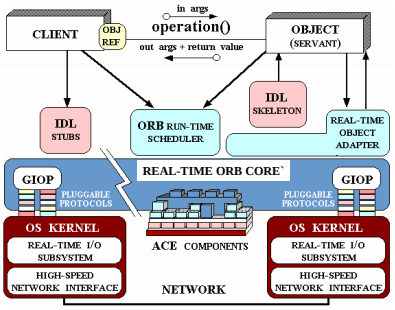


Figure 18: TAO architecture [282].

In terms of classification, the strategy and virtual component patterns of TAO are incorporated into an application using integration middleware (*How*). TAO is configured at startup and can load new components at run time using the virtual component pattern (*When*). We consider the earlier versions of TAO as configurable middleware and the recent version as repeatedly-tunable middleware, because it decomposes the C++ implementation of TAO into several core ORB components that can be dynamically loaded on demand using the virtual component pattern [164]. The adaptation supported in TAO is transparent to the application code the adaptive code, and the virtual machine, but not to the distribution middleware. The level of granularity is the process.

ZEN [152] is a TAO successor targeting real-time and embedded systems. ZEN is implemented in Java and Real-Time Java [283] and uses a micro-ORB architecture, illustrated in Figure 19, to minimize the footprint of the system. ZEN identifies several major ORB services, such as object adapters and transport protocols, that can be moved out of the micro-ORB kernel. Specifically, the virtual component pattern [164] is used extensively in ZEN to make services dynamically pluggable. Each ORB service itself is decomposed into smaller pluggable components that can be loaded into the ORB at run time only when required. Because of this feature, we consider ZEN as repeatedly-tunable middleware. ZEN also employs profiling and reflection techniques to monitor and inspect the optimized configuration found during the application tuning phase. The optimized configuration is written into a configuration file that can be used for future executions of the application. ZEN parses the configuration file during the application startup time and cofigures the middleware accordingly. As with TAO, ZEN uses strategy and virtual component patterns in an integration middleware technique (*How*), can be configured at startup time using configuration files and can load new components at run time using the virtual component pattern (*When*), and is a distribution middleware (*Where*).

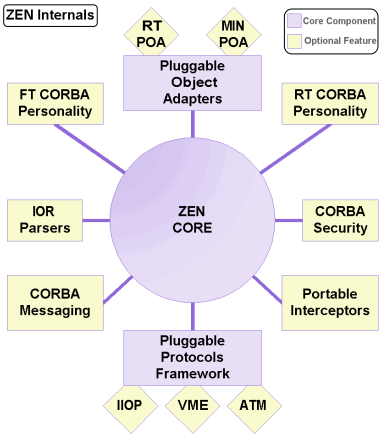


Figure 19: The ZEN architecture [152].

Component-Integrated ACE ORB (CIAO) [167], the TAO implementation of CORBA Component Model (CCM) [160], also resides in the distribution layer. CIAO is intended to provide component-based design to distributed real-time and embedded (DRE) system developers by abstracting systemic aspects, such as QoS requirements and composable meta-data units supported by the component framework. The classificationof CIAO is identical to that of TAO and ZEN.

dynamicTAO. Researchers at the University of Illinois have developed several adaptive middleware platforms [126, 145, 146, 221]. Kon et al. [126] extended an earlier version of TAO [165], which is configurable at startup/load time, by using computational reflection to construct a dynamically adaptive version. called DynamicTAO. To provide real-time services, DynamicTAO uses the Dynamic Soft Real-Time Scheduler (DSRT) [284], which provides QoS guarantees to applications with soft real-time requirements. Reification in DynamicTAO is achieved using the service configurator pattern [163], rather than metaobjects. In other words, reflection is mainly used to implement the service configurator pattern. Figure 20 illustrates the DynamicTAO reified structure. The DomainConfigurator, TAOConfigurator, and ServantConfigurator are all realizations of service configurator pattern in DynamicTAO. A service configurator in DyanimcTAO exports the DynamicConfigurator interface, which is a CORBA IDL interface, defined also as the MOP for inspecting, adapting, loading, and unloading "component implementations" dynamically. Component implementations are organized in categories representing different aspects of the TAO ORB packaged as dynamically loadable libraries (DLLs) that can be linked to the ORB at run time.

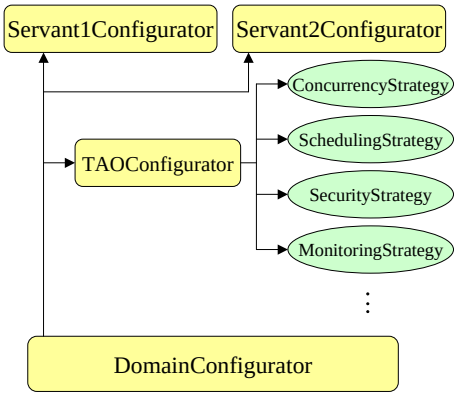


Figure 20: DynamicTAO reified structure [285].

As indicated in Table 8, DynamicTAO uses a strategy pattern and a meta-object protocol mechanism and is incorporated into an application using the integration middleware technique (*How*), can be configured at startup time using configuration files and can be reconfigured at run time using the configurator pattern (*When*). Of course, it is considered as distribution middleware (*Where*). Because component implementations can be introduced to the middleware at run time using DLLs, and because only a limited portion of the middleware core can be reconfigured at run time, we consider DynamicTAO to be repeatedly-tunable middleware. The adaptation supported in DynamicTAO is transparent to the application code to the adaptive code, and to the virtual machine, but not to the distribution middleware. DynamicTAO provides a ORB-wide adaptation, so the granularity is per process.

UIC. *Universal Interoperable Core (UIC)* [145] (previously called LegORB [146]) is the successor to dynamicTAO [126]. In addition to its small footprint (a UIC client-side ORB for PalmOS can be as small as 16KB [145]), UIC can adopt one or more *personalities* such as CORBA, Java RMI, and DCOM for interoperability purposes. Figure 21 illustrates the interaction between the UIC core and its personalities. UIC personalities can be either customized statically during the application compile time, or tuned dynamically using late composition during run time. The UIC minimum ORB core runs uninterrupted while ORB strategies and servants are dynamically updated. We classify UIC as both customizable and repeatedly-tunable middleware. UIC exploits customizable adaptation for rare and expensive changes during compile time, and exploits repeatedly-tunable adaptation for the frequent and inexpensive changes during run time. Using UIC, the same server objects can interoperate with different personalities without modifying their implementations. The main difference in our classification of UIC, compared to that of dynamicTAO, is the support for Java RMI and DCOM.

Open ORB and Open COM. Researchers at Lancaster University have conducted several projects in multimedia middleware [108, 127, 144, 286]. In the Adapt Project, Blair et al. [127] investigated middleware implementation for mobile multimedia applications that can be dynamically adapted in response to the environmental changes. In the OpenORB project [108], the successor to the Adapt project [127], Blair et al. focused on the role of computational reflection in middleware. More recently, Blair et al. [286] designed OpenORB v2, which adds a component-based design framework to the OpenORB reflective framework. OpenCOM [144] is the implementation of OpenORB v2,

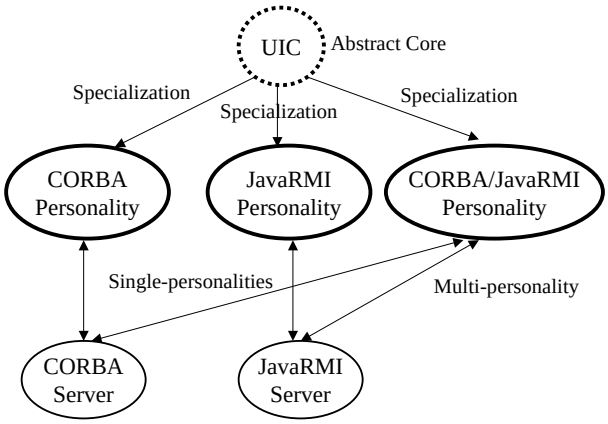


Figure 21: The UIC personalities [145].

designed for Microsoft COM systems. All above mentioned projects are greatly influenced by RM-ODP [287], a meta standard for multimedia applications. Unlike TAO [165] and DynamicTAO [126], none of the Adapt, OpenORB, or OpenORB v2 projects is CORBA compliant.

OpenORB uses meta-object protocols to realize an integration middleware solution (*How*) that supports dynamic adaptation at run time (*When*). The implementation of the OpenORB reflective architecture is based on the reflection model illustrated in Figure 22. OpenORB categorizes reflection into structural and behavioral reflection [285], a distinction first introduced in [288]. The *architecture meta-model* provides access to an object using its object graph. The *interface meta-model* provides access to the methods, associated attributes, and inheritance structure of each interface of an object. The *interception meta-model* provides interception hooks for each interface of an object including message arrival, dispatching, marshalling and unmarshalling interception hooks. The *resources meta-model* provides access to available resources per address space and enables resource reservation.

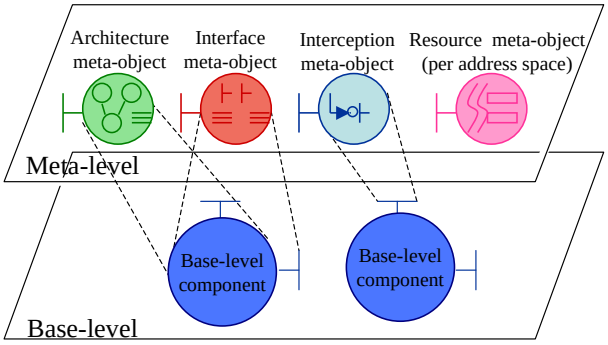


Figure 22: OpenORB reflection model [285].

Unlike DynamicTAO [126], which uses reflection mainly to implement the service configurator pattern, OpenORB provides an ORB-wide reflection. Therefore, we consider OpenORB as mutable middleware. The adaptation supported in OpenORB is not transparent to the application code, to the adaptive code, or to the distribution middleware, but it is transparent to the virtual machine. While adaptations can be ORB-wide, the reflective capabilities of OpenORB also enable adaptation of individual objects and selected sets of remote interactions.

FlexiNet *FlexiNet* [149] is a CORBA-compliant ORB implemented in Java that uses reflection (*How*) to provide dynamic adaptation at run time (*When*). FlexiNet is designed as a set of components that can be dynamically assembled. Similar to DynamicTAO [126], FlexiNet provides coarse-grained, ORB-wide adaptation via metaobject protocols. For example, FlexiNet can dynamically modify the underlying communication protocol stack through the replacement and insertion of layers. Similar to OpenORB [108], FlexiNet also provides fine-grained per-interface adaptation, as depicted in Figure 23. Therefore, we consider FlexiNet as repeatedly-tunable middleware. In FlexiNet, replaceable meta-objects can intercept requests in stubs and skeletons. These meta-objects realize channel configuration policies

that are used to adapt stubs and skeletons.

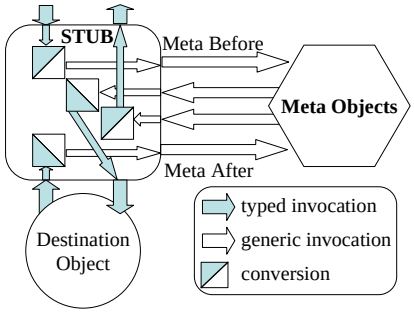


Figure 23: FlexiNet architecture [149].

Squirrel over Infopipes. Squirrel [188, 289] is QoS-oriented middleware specialized for distributed multimedia applications. Squirrel uses the Infopipes abstraction [189] to support streaming data. The designers argue that CORBA stubs and skeletons generated from IDL interfaces follow a standard protocol (marshalling and unmarshalling) that is not suitable for multimedia applications with different QoS requirements. To solve this problem, Squirrel introduces *smart proxies* [289], which are service-specific stubs that include adaptive code (*How*). A smart proxy for a specific application can be developed and shipped to the client program statically during compile time or dynamically during load time (*When*). Figure 24 illustrates dynamic smart proxy shipping in a live video application. Squirrel provides adaptation for CORBA objects, so it supports a per object adaptation granularity.

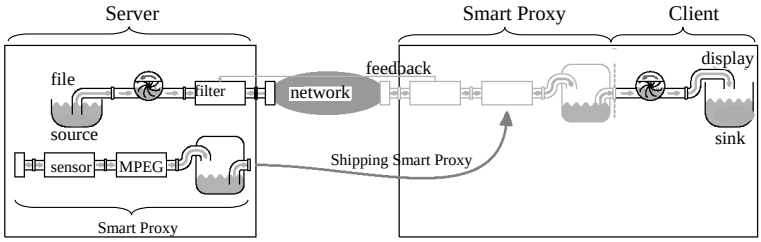
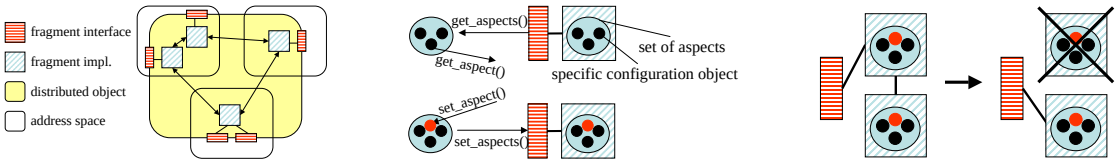


Figure 24: Squirrel: dynamic shipping of a smart proxy [289].

AspectIX. AspectIX [190] is an aspect-oriented middleware based on a fragmented (distributed) object model [290]. Figure 25(a) illustrates a distributed object that has four fragments distributed over three programs over a network. A fragment is divided into a fixed interface and a flexible implementation. A fragment implementation can be as simple as a CORBA stub or as complicated as a "smart" fragment (similar to smart proxies in Squirrel [188]) that can cache previous replies locally or change its behavior using dynamically inserted aspects. Figure 25(b) shows how an AspectIX-aware application can dynamically inspect and adapt the set of aspects residing inside a fragment implementation. Figure 25(c) shows how a fragment implementation can be dynamically exchanged, transparent to the application.



(a) distributed object model. (b) aspect configuration. (c) fragment exchange.

Figure 25: AspectIX: an aspect-oriented middleware based on the distributed object model [190].

AspectIX supports stub configuration at stub load time and reconfiguration at run time (*When*) using the proxy

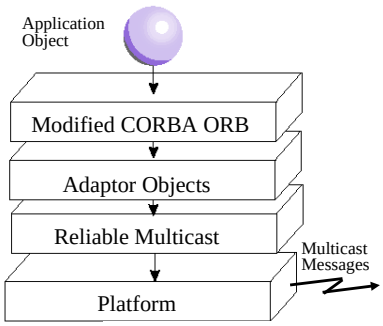


Figure 26: The middleware integration approach [292].

pattern and aspect weaving dynamic aspect weaving (*How*). AspectIX can be repeatedly tuned by insertion and removal of aspects at runtime. The adaptation supported in AspectIX is not transparent to the application code, to the adaptive code, or to the distribution middleware, but it is transparent to the virtual machine. AspectIX provides adaptation for CORBA objects, so it supports a per object adaptation granularity.

OpenCorba. *OpenCorba* [142] is a CORBA-compliant ORB that uses reflection to expose and modify certain internal characteristics of CORBA (*How*). OpenCorba is implemented in NeoClasstalk, a reflective language based on Smalltalk [51]. OpenCorba reifies various properties of the ORB through explicit meta-classes. Unlike DynamicTAO [126] and OpenORB [108], OpenCorba does not provide a global view of ORB, but similar to DynamicTAO preserves an intact ORB core during tuning process at run time, so it is repeatedly-tunable middleware (*When*). OpenCorba provides adaptation for application classes, finer-grained adaptation than DynamicTAO and coarser-grained adaptation than OpenORB.

Electra and Orbix+Isis. *Electra* [191] and *Orbix+Isis* [291] are two CORBA compliant ORBs that provide fault-tolerance by integrating an object replication mechanism inside their ORBs. As depicted in Figure 26, adapter objects are used to enable the modified ORB to use the services provided by the reliable multicast. Orbix+Isis uses Isis [193] and Electra can use either Horus [192] or Isis [193] as their reliable multicast service. The integration approach is transparent to the application code, but requires both sides of application to use the same modified ORB. Both Electra and Orbix+Isis are incorporated into an application using the integration middleware technique (*How*) that supports configuration of CORBA object replicas at startup time (*When*), and are considered as distribution and host-infrastructure middleware (*Where*). Since both can be configured during the application startup time, we consider them to be configurable middleware. Electra and Orbix+Isis provide adaptation for CORBA objects, so they supports a per object adaptation granularity.

Orbix/E Orbix/E [293] from IONA Technologies is a lightweight and high-performance CORBA ORB designed for embedded devices. The size of an Orbix/E can be as small as 100KB for the client and 150KB for server programs. Similar to PersonalJava (and unlike EmebeddedJava), Orbix/E requires a minimum fixed core functionality. We also consider Orbix/E as configurable middleware because of its ability to parse configuration files during the application startup time, for example, to load optional pluggable protocols. We consider Orbix/E as customizable middleware because it allows a developer to generate customized versions of Orbix/E. The adaptation supported in both Orbix/E is transparent to the application code, to the adaptive code, and to the virtual machine but not to the distribution middleware. Orbix/E provides adaptation at the ORB level, so the granularity is per-process. Finally, Orbix/E is used to adapt remote interactions.

Other Distribution Middleware-Based Approaches. Other distribution middleware-based approaches that support compositional adaptation include Orbix [194], ORBacus [195], JacORB [294], Horus [192], Isis [193], COM/-DCOM [157, 158], and .NET remoting [143].

A.4 Common-Services Middleware Projects

We divide projects and specifications supporting compositional adaptation at the common-services layer into two groups according to the transparency provided to the application source code. Approaches in the first category do

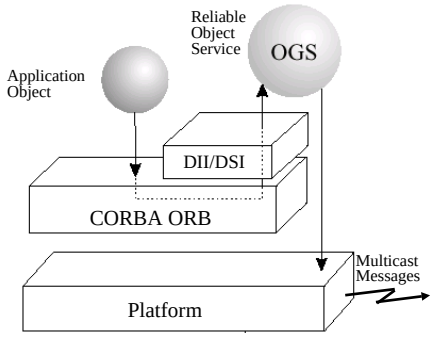


Figure 27: The service approach [292].

not provide transparency to the application code (*e.g.*, OGS [182] and QuO [168]), whereas approaches in the second group do (*e.g.*, FTS [183], IRL [184], TAO-LB [169], and ACT [186]). Projects in the former group use the integration middleware technique, where the services are required to be used explicitly by the applications. Projects in the latter group use a middleware interception technique (*e.g.*, CORBA portable interceptors [160]) to enables existing applications to benefit from the adaptive services.

As indicated in Table 8, all projects in common-services layer provide transparency with respect to adaptive code, and to any distribution middleware and virtual machine (as applicable). Finally, they all involve adaptation of *remote* interactions.

OGS. *Object Group Services (OGS)* [182] from Swiss Federal Institute of Technology is an implementation of fault-tolerant CORBA, supporting CORBA applications with reliability. It is implemented as an integrated middleware technique (*How*). OGS provides a group communication CORBA service[1], that must be used explicitly by a CORBA application at development time, can be configured at startup time by setting up members of object groups, and can be reconfigured at run time through a dynamic group membership (*When*). A *CORBA service* is a set of domain-independent CORBA objects, which can be used by any CORBA client through a CORBA compliant ORB. The OGS CORBA object uses its group communication protocol to communicate with the service replicas. It then returns the reply to the application object.

Three approaches, namely, integration, interception, and service, have been used in the literature to implement fault-tolerant CORBA [292]. In the *integration* approach, an implementation of a group management protocol and its use are integrated either with the application code (*e.g.*, FRIENDS [147] using a compile-time MOP) or with the CORBA ORB (*e.g.*, Electra [191] and Orbix+Isis [291] that provide a modified ORB, which must be compiled with the application). In the *interception* approach, a group management implementation intercepts the remote interactions and adapts them accordingly (*e.g.*, Eternal [196] that uses a middleware interception technique). In the *service* approach, a CORBA service provides a group management implementation of a group management protocol, such as OGS that provides a CORBA object for this purpose (*Where*). As depicted in Figure 27, the OGS object uses CORBA DSI and DII interfaces [160] to receive requests from the client object. Note that an OGS object can also be considered as a proxy for its assigned group of service replicas. Finally, OGS provides an object-level granularity for compositional adaptation, since the configuration of replicas and the replication style (active or passive) are independent of the specific CORBA object benefiting from them.

QuO. Researchers at BBN Technologies [168] have developed *QuO*, an adaptive framework that provides a high-level QoS abstraction on top of distribution middleware technologies such as CORBA and Java RMI. Figure 28 illustrates QuO components residing between the application and a distribution ORB (*Where*). QuO employs aspect-oriented programming [112] to separate the non-functional (systematic) aspects from the functional aspects of an application. QuO uses an integrated middleware technique (*How*) by wrapping stubs and skeletons of a remote object in an object called a *delegate*, which intercepts all the interactions targeted to or received from stubs and skeletons and redirects them to a QuO kernel (using the premethod and postmethod methods). The intercepted interactions may be monitored or adapted by the QuO kernel using a contract and several system conditions. A *contract* is written in the

[1]We note that as of the time of writing this technical report, a group communication service is not specified as a *standard* CORBA service by OMG. For a list of standard CORBA services, please refer to the following URL: http://www.omg.org/-technology/documents/formal/corbaservices.htm.

contract-description language (CDL) [295] and defines acceptable regions of operation in an application. *System conditions* can be considered as software "sensors" that record values representing the state of the execution environment. QuO combines the code for a contract, a number of system conditions, and a wrapper into a package called a *qosket*, which can be reused for different applications providing transparency to the adaptive code.

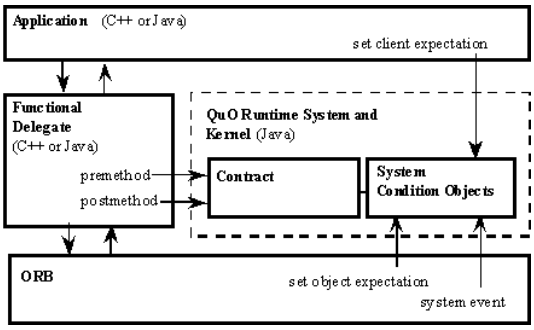


Figure 28: QuO architecture [295].

A delegate is generated from a program written in the aspect-oriented structural description language (ASL) [295]. Delegates are similar to a statically shipped smart proxies in Squirrel [188], except that delegates can also wrap skeletons on the server side, whereas smart proxies are only at the client side. To take advantage of QuO, a developer modifies the source code of the application at development time (to add code for creation and use of delegates in the application code explicitly) and uses the quogen tool at compile time to generate the code for delegates, contracts, and system conditions. Finally, QuO provides adaptations granularity at the level of remote objects.

IRL and FTS. *Interoperable Replication Logic (IRL)* [184, 185], developed by Baldoni et al., and *fault-tolerant service (FTS)* [183, 296], developed by Hadad et al., are two middleware projects that provide fault-tolerance CORBA using both the service and interception techniques (*How*). IRL and FTS both use CORBA request portable interceptors [160] to intercept requests (requests, replies, and exceptions) and redirect them to a group manager object that coordinates the interactions among clients and object replicas (*Where*). The basic IRL architecture is illustrated in Figure 29, where a client request portable interceptor forwards the request to a local proxy, which provides a fault-tolerant service using object replication.

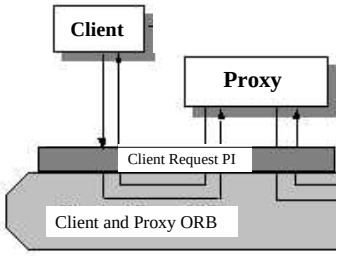


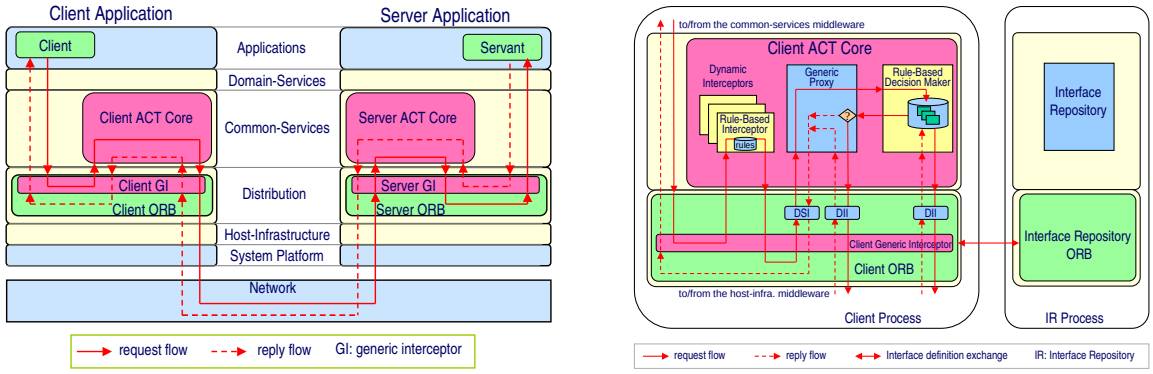
Figure 29: The IRL basic architecture [185].

To use IRL and FTS, a developer must configure a CORBA portable interceptor with the CORBA application at startup time (an ORBInitializer class must be introduced to the application ORB) and can reconfigure the object replicas at run time (*When*). Using the portable interceptors, both IRL and FTS support transparency to the application code and to CORBA ORBs. The replication style can be developed independent of the applications using it. As such, IRL and FTS provide transparency with respect to the adaptive code. Finally, both IRL and FTS provide adaptation granularity at the level of CORBA objects.

TAO Load Balancing. *TAO load balancing (TAO-LB)* [169], developed by Othman et al., adds a load balancing service to TAO [165] using CORBA portable interceptors (*Where*). TAO-LB employs an *adaptive on-demand architecture* that works as follows. First, a client receives a handle to the load balancer instead of the target object. Next, using CORBA standard LOCATION_FORWARD mechanism [160], the load balancer redirects the initial client request to the appropriate target object replica. The CORBA client continues using the new object reference (obtained

as part of the LOCATION_FORWARD message) to communicate with this replica directly until it is either done or redirected again. An adaptive load balancer that forwards requests on demand can monitor the load on each replica continuously. Using this load information and the policies specified for load balancing, the load balancer can determine whether the load is distributed evenly. When the load becomes unbalanced, the load balancer can communicate with the replica ORBs and ask them to redirect their clients back to the load balancer. The load balancer can then redirect the clients to less loaded replicas. Similar to IRL and FTS, TAO-LB is incorporated into an application using a middleware interception technique (*How*). It can be configured at startup time (to register its portable interceptor with CORBA ORBs and set up number of objects cooperating in load balancing) and can be reconfigured at run time (*When*). Similar to IRL and FTS, TAO-LB supports transparency to the application code, to the adaptive code, and to the CORBA ORBs. Finally, the granularity of adaptation is at the level of the CORBA object.

ACT. *Adaptive CORBA Template(ACT)* [186], developed at Michigan State University, is an adaptive CORBA framework designed to support transparent adaptation to (unanticipated) changes in a CORBA application's functional requirements or in non-functional concerns, such as quality-of-service, fault-tolerance, and security. The key insight into how to achieve this transparency is the concept of the *generic interceptor*, which is a particular type of CORBA portable request interceptor [160] (*How*). The generic interceptor provides a "hook" into the interaction between CORBA clients and servants, enabling the insertion of new adaptive functionality after deployment. Although the generic interceptor must itself be registered with the ORB of a CORBA application at startup time, it enables registration of other *specific* request interceptors to be postponed until run time (*When*). The interceptors can adapt requests, replies, and exceptions that pass through the ORB. As a result, an existing application need only be restarted with an argument identifying a generic proxy object to be used. There is no need to modify or even recompile the application. At run time, the generic interceptor can be used to incorporate specific interceptors that dynamically adapt the application behavior.



(a) ACT component configuration

(b) ACT Core components

Figure 30: ACT architecture [186].

Figure 30(a) shows the flow of a request/reply sequence in a simple CORBA application using ACT. The *client* generic interceptor intercepts all outgoing requests and incoming replies (or exceptions) and forwards them to its *ACT core* (*Where*). Similarly, the generic interceptor at the server side intercepts all the incoming requests and outgoing replies (or exceptions) and forwards them to its ACT core. Figure 30(b) shows the flow of a request/reply sequence within the client ACT core. The request is first processed by one or more *dynamic* interceptors. A *rule-based* interceptor (RBI) is an example of dynamic interceptors that uses "rules" to govern its operation. The rules can be inserted, removed, and modified at run time. A *generic proxy* is a surrogate for any CORBA object that used CORBA DII and DSI [160] to receive and send any request. The generic proxy consults with a *decision maker* in determining how to handle intercepted requests. Possibilities include sending a new request (possibly with modified arguments) to either the target object or to another object. Alternatively, and unlike a request interceptor, a proxy can reply to the intercepted requests using local data (*e.g.*, cached replies). In the case of exceptional situations, the decision maker may notify other objects by way of an *event mediator* [150], which propagates events to interested objects. Similar to other projects that use CORBA portable interceptors, ACT supports transparency to the application code, to the adap-

tive code, and to the CORBA ORBs. ACT provides adaptation at the ORB-, object-, method-, and method-call level through insertion or removal of rules at run time Finally, ACT can be used to adapt a selected set of CORBA object interactions.

Other Common-Services Middleware-Based Approaches. Other approaches that support compositional adaptation at the common services level include CorbaServices [160], AQuA [170], Context-Toolkit [13], Context-Aware [297], multi-channel reification model (mChaRM) [148], and CORE [95].

A.5 Other Related Projects

This technical report has attempted to classify some of the many projects in compositional adaptation. Numerous other projects, addressing one or more aspects of compositional adaptation, were not specifically discussed: Boeing Bold Stroke [181], Aura [11], MOOSCo [298], Oxygen [10], PlanetBlue [9], GRACE [201], DEOS [200], Graybox [202], Gaia [299], 2K [300], Odyssey [301–303], Puppeteer [37], SwitchWare [304], Computing Communities [305], Cactus [7, 30, 41, 247], and Dynamic Adjustment of Component InterActions (DACIA) [306]. We intend to classify more projects in future versions.