Multidimensional Separation of Concerns in Middleware

Isabelle Rouvellou, Stanley M. Sutton Jr., and Stefan Tai
IBM T. J. Watson Research Center, New York, USA
{rouvellou, suttonsm, stai}@us.ibm.com

1 Introduction

Middleware is an essential element in large software systems such as those that support enterprise applications that require the interoperation of multiple components. The components may be distributed, independently operated, and heterogeneous with respect to language, data model, environment, architecture, and protocols. Middleware is required to integrate these diverse software components and allow them to interoperate effectively. One active area of middleware research and development in recent years has been distributed object systems. Some examples of major efforts in this area include CORBA [1,2] and Enterprise Java Beans (EJB) [3,4]. Another major focus of middleware efforts is in messaging systems such as IBM's MQ Series and MSMQ [5].

Multidimensional separation of concerns (MDSOC) [6] provides a new and more flexible way of managing the separation of concerns in software. In particular, MDSOC overcomes the limitations of the typical programing-language mechanisms for separating concerns, namely that the available mechanisms typically do not separate many concerns very well, resulting in entanglements that significantly complicate software maintenance, evolution, integration, and reuse. MDSOC is based on the premise that independent concerns should be represented independently, and that programs can be developed by the principled composition of separate concerns according to systematic rules.

Previous work has focused on MDSOC within a single application or language, although there has been some application of these ideas across the software life cycle [7]. In this paper we consider the extension of MDSOC to middleware and "middleware-mediated systems" (MMS) such as enterprise applications. This approach seems natural because middleware often exists precisely to address specific concerns in such applications. Additionally, enterprise applications are typically developed by composition of separate, if not independent, components (which may also represent concerns). Thus, a "compositional" style of system construction applies in MMS in a way that may be analogous to intra-linguistic or intra-application composition by MDSOC mechanisms. Furthermore, issues such as integration, evolution, and reuse are important in MMS, and these are the sorts of activities that MDSOC is intended to facilitate.

In this paper we consider middleware and MMS from a MDSOC perspective. More particularly, in Section 2 we look at some of the ways that concerns are currently separated (or not separated) in middleware and MMS, and in Section 3 we summarize significant issues and indicate some further topics for research.

2 Concerns in Middleware and MMS

In analyzing the dimensions of concern for middleware and MMS, it is necessary to keep in mind the general nature of middleware and its role in software systems:

- First, middleware is software; therefore, middleware is subject to concerns like software in general.
- Second, the general function of middleware is to interconnect other software components (possibly software systems) to create larger software systems. Therefore, it is possible to distinguish the concerns of the middleware specifically from the concerns of other components and of the system overall.
- Third, the specific functions of middleware typically address specific concerns of a system, for example, communication, security, or transactions. Therefore, middleware itself can be seen as concern-separation mechanism for software systems.
- Finally, although the concerns of middleware and the concerns of an overall system can be separated, the way in which middleware addresses various concerns determines whether, how, and how well it can address various concerns in a larger system. (Conversely, the concerns of a system help to constrain the use of middleware in the system.)

With this perspective, we identify approximately one dozen dimensions along which concerns about middleware can be separated. These dimensions are grouped into four main categories: Architecture, Middleware Services, MMS Life Cycle and Development, and Middleware Products. The categories reflect semantic associations among the grouped dimensions and also represent the different interests that various groups of stakeholders have.

2.1 Architecture

Middleware is used to connect components into systems and architecture is a fundamental way of describing systems. Also, architecture conditions, and is conditioned by, the kinds of properties that middleware uses or requires. The primary stakeholders in an architectural view of middleware are system architects and middleware developers.

We believe that the "standard" classifications of object-oriented or message-oriented middleware architectures can be analyzed in terms of the fundamental dimensions of *components*, *connectors*, and *topologies*. (A similar taxonomy in terms of components, busses (connectors), systems (analogous to topologies), and properties of these is proposed for concern identification in [8].) In fact, the standard object-oriented and message-oriented models reflect significant entanglements of concerns along these other dimensions.

2.1.1 Components

Components are the main constituents of an architecture. Generally, it is the function of middleware to connect other components, although the middleware itself may also be represented by components in the larger system. Examples of middleware components include clients, servers, agents, containers, and queues.

In most middleware architectures, there is a strong relationship between the middleware components and other aspects of the middleware or application architecture. For example, servers imply clients, queues imply messages, and containers imply "beans." Additionally, as described below, there is a strong connection between components and connectors. Thus, identifying the components of an architecture often serves to imply many other features of the architecture. Because of entanglements between components and other architectural dimensions, the variety of architectural patterns is restricted, and the opportunities to separate and reuse components are limited.

2.1.2 Connectors

Connectors are the elements by which components are interrelated, communicate, and interact. Connectors may be characterized in terms such as communication paradigms, interaction patterns, and protocols.

The two main communication paradigms are object-oriented and messaging. In general, object-oriented communication is synchronous, requires that the recipient be identified, and requires that the sender and recipient be active concurrently. Messaging, on the other hand, is asynchronous, does not require that the recipients be known, and does not require that the sender and recipient be active concurrently.

The split between object-oriented communication and messaging communication defines the dominant decomposition in middleware systems. Because the communication paradigms are so closely aligned with other aspects of middleware architecture and function, and because the two communication paradigms cannot be very easily mixed, we consider this decomposition to be tyrannical. Because of the significance of middleware to overall system architectures, the entanglement of concerns in middleware architecture tends to promote corresponding entanglements throughout a system. Thus, we believe that separating these entanglements is a major issue for research in advanced middleware.

2.1.3 Topologies

Topology is the arrangement of components and connectors. Specific topologies reflect not just the kinds of components and connectors but also their numbers and arrangements.

Both middleware systems and MMS have topologies. The way in which MMS topology is built up from middleware is most strongly influenced by the kinds of middleware connections and components. Generally, a variety of MMS topologies may be built up using a given kind of middleware.

An issue related to middleware and MMS topology is how elements that support middleware services (Section 2.2) are distributed about the system. Consider, for example, transaction processing in CORBA systems. The OMG's Object Transaction Service (OTS) requires clients and servers to provide and use particular interfaces for transaction processing and to interact with each other and with an OTS-system and any persistent data store involved according to specific protocols. Middleware services like the OTS, that are distributed throughout a system, impact the topology of an MMS and influence other factors of middleware use because they become entangled with multiple dimensions of concern.

2.2 Middleware Services

Services are a natural characterization of what middleware does or provides. They reflect, from an application perspective, the purposes for which middleware is used. The services dimensions of middleware also provide analogs to the programming-language dimensions of concern regarding functions and features. The primary stakeholders in the services dimensions of middleware are application developers, middleware developers, and system architects.

The main dimensions we distinguish are kinds of service, properties (which include effects produced by services), and representation of services.

2.2.1 Kinds of Service

For any particular kind of middleware, the services are usually well advertised. For example, CORBA standards have been defined for naming, life cycle, transactions, security, persistence, events, and other numerous services. Enterprise JavaBeans dictates services for concurrency, transactions, persistence, distribution, naming, and security. MQ Series defines a messaging service specifically (but it includes numerous features that can be considered as services in support of messaging). Each individual kind of service may be considered a separate dimension, as services are not generally exclusive, so "kinds of services" is probably best considered a concern subspace.

Some middleware models make a distinction between basic and complex services. For example, CORBA distinguishes common object services (such as those named above) from common facilities (such as information and task management) that are built on the common services. Middleware supporting basic services may be considered "well separated' if it can be readily used in constructing middleware to support complex services. Middleware for complex services can be considered well separated if it can be decomposed or reconfigured in support of other (simple or complex) services.

We also distinguish domain or application independent versus domain or application dependent middleware. However, for brevity we omit discussion of this topic.

2.2.2 Properties

Consistent with definitions proposed elsewhere (e., g., [9]), we regard properties as the externally visible aspects of a component or system (which in an architectural sense may be a component or a connector). Typical examples of properties include scalability, reliability, transactionality, portability, persistence, and security.

Properties, like services, are not exclusive. Thus, properties, like services, constitute a subspace of dimensions rather than a single dimension. However, not all properties are orthogonal, as there may be covariance or contravariance between them. For example, transactionality in a message delivery system may improve reliability of delivery but reduce some measures of throughput.

Properties, like architecture, is a concept that applies both to middleware and to MMS. Middleware services are generally used specifically to effect or support properties in MMS. (In system development, properties are often the ends for which services are the means.) Additionally, the incidental properties of middleware also affect the properties of MMS.

"Quality of service" is an important characteristic of applications. Typical QoS goals are reliability up to fault tolerance, performance in terms of throughput, client response time, and so on. We generalize the usual notion of QoS to that of *qualities* of service, which may reflect any desired combination of properties of interest relative to the concerns of any stakeholder. Qualities of service reflect properties that emerge from an application and that are often supported through (or influenced by) middleware. This effect may be obtained directly, through a specific service, or indirectly, for example, through architecture properties.

2.2.3 Representation of Services

Middleware services can be represented in a number of ways. For example, a naming service that is used to find object references typically is represented through a centralized naming server component to which application components connect. For other services, a representation through server components is not possible or desirable. For example, a centralized transaction server as representation of a transaction service would be a single-point-of-failure and potential bottleneck in the system. Thus, a transaction service is typically represented through libraries, which are linked to application components. Different kinds and numbers of components are thus possible for service representation. In addition, a middleware service imposes different interaction protocols (connectors), for example to register objects under certain names, or for transaction processing. These connectors may be prescriptive or suggestive. Again, the numbers and kinds of connectors for service representation are varied. Essentially, service representation relates to the architectural dimensions of components, connectors, and topologies as described above. Due to the complexity (and power) of some services, the entanglement with other dimensions is particularly inherent here.

2.3 MMS Life Cycle and Development

Concerns under this heading relate to the use of middleware primarily on the application level. They reflect when, where, and how middleware services or properties are incorporated into or determined for an application as the application is developed, deployed, and executed. The main stakeholders for these concerns are application developers, software engineers (i.e., concerned not with specific applications but with general methods, models, and processes), and middleware developers. The main dimensions of concern in this group include the life cycle stage at which middleware services are incorporated into an application, and the "programming model" by which middleware properties or functions are incorporated.

2.3.1 Life Cycle Stage

An application system is developed through a number of stages. Of particular importance for MMS are the early life cycle stages of specification and design, where selections of middleware technology are made; the runtime-related stages of deployment and execution, where middleware services are specified and used; and the later stages of maintenance, evolution, and reuse, which are more or less complicated or enabled according to middleware use.

Life-cycle stage is a dimension of concern for any system because decisions made earlier in the life cycle tend to constrain options and opportunities at later stages of the life cycle. For MMS in particular, because middleware tends to have such a profound effect on the architecture and properties, decisions related to middleware are particularly significant.

For example, if language independence (or heterogeneity) is an initial requirement for a system, then CORBA is a natural choice for middleware. However, once CORBA is selected, then the interfaces between components must be specified through the CORBA IDL (and are thus enabled or restricted accordingly), middleware services are component-managed, and access to middleware services is through a particular programming model (see below). Decisions about specific services may need to be hard coded. In contrast, a Java-based system may choose to use Enterprise Java Beans. In this case the middleware services are (by default) container-managed, and access to the services is by a different programming model; decisions about which services are used and how they are used are fixed at deployment time.

One goal for research in advanced middleware is better support for separation of concerns so as to minimize the restrictions that earlier decisions about middleware impose on later decisions about middleware or the system overall. Also, The more separable that middleware concerns can be made, the greater the possible range of development approaches for MMS.

2.3.2 Programming Model

The middleware programming model addresses aspects of the use of middleware in application development, especially as related to the writing of code. It addresses interfaces and mechanisms for specifying, accessing, controlling, and customizing middleware services.

Key aspects of the programming model include whether services are accessed or specified declaratively or operationally, and whether they are accessed or specified explicitly or implicitly.

CORBA provides operational interfaces for the access of middleware services. Applications access middleware services through calls on an API. These calls are explicit in the application code, which complicates the code (and development and maintenance) but allows precise control by the application. In the component/container model (e.g., Java Beans and Enterprise Java Beans), containers provide middleware services on behalf of components, and the services to be provided are specified in a declarative deployment descriptor that is separate from the components. From an application perspective, these services are provided implicitly. This simplifies the programming of applications (and components), and may facilitate site-specific adaptations, but it means that middleware services used are not directly apparent in an application or component.

2.3.3 Discussion of Life Cycle and Development

With respect to engineering life cycle and programming model, there is currently considerable entanglement of concerns in the provision of middleware services. Notably, the explicit approaches are operational and are determined at development time, whereas the implicit approaches are declarative and determined at deployment time. Additionally these approaches tend to be associated with particular middleware products (or product classes). We believe that these two "styles" of middleware constitute another dominant, tyrannical "dimension" of middleware decomposition (actually covariant clusters of dimensions). We take as a major goal of advanced middleware research to discover better means of decoupling these dimensions and allowing them to be combined in more meaningful and useful ways.

2.4 Middleware Products

This group views middleware components from the enterprise system level. It relates to actual middleware and application products and to components of those products. The main stakeholders are middleware developers, system architects, and also possibly marketing and procurement. The main dimensions of concern that we put under this heading are product features (especially vendor-specific), compatibility with application and (other) middleware systems, and modularity of middleware systems.

2.4.1 (Vendor-specific) Product Features

This is important from a business standpoint since these are the units in which middleware systems are bought, sold and (more or less) used. Products often attempt to distinguish themselves based on unique features or combinations of features. For example, the IBM WebSphere products come at three levels, “standard”, “advanced”, and “enterprise”, based on the richness of features offered.

Because of accepted standards, either institutional (e.g., CORBA) or defacto (e.g., MQ Series), the choice of a general engineering approach does not totally constrain the selection of a middleware product. However, it is often by the non-standard features that vendors will hope to sell their products. If the development of a MMS comes to depend on the non-standard features of a selected product, then ongoing use and evolution of that system become entangled with (and dependent on) the product.

2.4.2 Compatible Application Systems

These are the application system components with which a given middleware product works. Especially in the case of legacy applications or existing IT infrastructure, this may constrain the selection of middleware for system evolution.

2.4.3 Compatible Middleware Systems

These are the other middleware systems with which a given middleware system may usefully interoperate. Constraints on interoperability may be imposed especially according to some of the tyrannical dimensions of separation noted above.

2.4.4 Modularity of Middleware Systems

The modularity of a middleware system reflects the potential for selective reuse of components from the system, possibly in combination with components from another system. The logical and physical separation of middleware components can be an important means for separating concerns that are addressed by components (analogous to “modules” that allow concern separation in a programming language). If the components within a middleware system are highly separable, then the entanglement between them is low, and thus so is the overhead of using any individual component. Of course, the converse, problematic situation occurs if the components cannot be readily separated.

3 Summary and Discussion

We have identified about one dozen dimensions, or subspaces of dimensions, along which concerns in middleware and MMS can be usefully separated. These separations address a variety of purposes, reflecting the interests of a variety of kinds of stakeholder.

There is a good separation of concerns along some dimensions, but covariant or contravariant linking is common among others. The most dominant (and tyrannical) dimensions of decomposition that we find are 1) the split of “object-oriented” and “message-oriented” middleware, which has strong architectural and service implications, and 2) the split between declarative, deployment-time specification and operational, development-time specification, which has implications for the middleware development life cycle and middleware compatibility.

One important goal of our research in middleware is to find ways to disentangle the concerns that are associated with these dominant decompositions. That includes research on the interoperability and integration of object-oriented and message-oriented communication as one example [10]. Some additional research topics related to middleware concern dimensions include:

- What additional middleware services (such as rules and consistency management) might be useful?
- How can middleware semantic models be better integrated?
- How should middleware-related concerns be specified and modeled in MMS?
- How can middleware modularity be improved for greater separability and composability?

In this paper we have considered separation of concerns in middleware primarily in terms of the major dimensions of concern. However, there are other contexts (dimensions?) in which separation of concerns in middleware may be considered. One is the relationship between system-oriented, inter-component concern management (as addressed by middleware) and application-oriented, intra-component, concern management (as addressed by linguistic mechanisms). Some example questions from this area:

- How are system-level concerns represented within components?
- How does system-level concern management depend on component-level concern management?
- What concerns arise in applications (within components) in expectation of use with middleware?
- How can (or should) applications be designed to take advantage of middleware-managed concerns?

Still more research topics in separation of concerns for middleware exist in the areas of architecture, methodology, tools, and environments (among others).

Better support for the separation of concerns in and through middleware is needed if advances in application or component-level concern management are to be extended system-wide. For better or worse, the required areas of research are just as varied and entangled as the middleware concerns themselves.

4 References

[1] OMG, The Common Object Request Broker, Object Management Group, 1998.

[2] OMG, CORBA Services, Object Management Group, 1998.

[3] SUN MICROSYSTEMS, *Enterprise JavaBeans^{TM} Specification*, Copyright 1998 by Sun Microsystems, Inc.

[4] MONSON-HAEFEL, RICHARD, Enterprise Java Beans, O'Reilly & Associates, Inc., Sebastopol, CA, 1999.

[5] LEWIS, RHYS, Advanced Messaging Applications with MSMQ and MQSeries, Que Corporation, Indianapolis, Indiana, 2000.

[6] TARR, PERI AND OSSHER, HAROLD AND HARRISON, WILLIAM AND SUTTON JR., STANLEY M., *N degrees of separation: multidimensional separation of concerns*, in Proceedings of the 21st International Conference on Software Engineering, ACM, New York, 1999, pp. 107--119.

[7] First Workshop on Multi-Dimensional Separation of Concerns in Object-oriented Systems at OOPSLA '99, at http://www.cs.ubc.ca/~murphy/multid-workshop-oopsla99/.

[8] GRÜNBACHER, PAUL, EGYED, ALEXANDER, and MEDVIDOVIC, NENAD, "Dimensions of Concerns in Requirements Negotiation and Architecture Modeling", in Proceedings of the Workshop on Multi-Dimensional Separation of Concerns in Software Engineering (held in conjunction with the ACM 22nd International Conference on Software Engineering), Limerick, Ireland, June, 2000 (to appear).

[9] KLEIN, MARK and KAZMAN, RICHARD, "Attribute-based architectural styles", Carnegie-Mellon University/Software Engineering Institute Technical Report, CMU/SEI-99-TR-022.

[10] TAI, STEFAN AND ROUVELLOU, ISABELLE, *Strategies for integrating messaging and distributed object transactions*, in Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000), New York, Springer LNCS 1795, pp. 308-330, April, 2000.