Introduction 1

Rapid software development is essential for enterprises that seek innovative product leadership and enhanced revenue opportunities. To succeed in today's increasingly fast-paced global networked business environment, agile enterprises require agile software solutions. Equally, the inertia associated with traditional, monolithic applications must be overcome so that products can be developed and marketed expediently, resulting in an increased market share and profitability. Finally, understanding and meeting customers' requirements for individual applications and delivering the best solution for their needs will help position a company to achieve a dominant position in its industry and enable it to continuously adapt to changing market requirements.

Software Development for Agile Enterprises

Taking enterprise computing to a new level in productivity, SunSoft's™ WorkShop NEO is engineered to be the vehicle for progressive software developers building state-of-the-art, flexible, custom, networked applications. WorkShop NEO is part of the realization of Sun's vision of a global network of open systems transparently interacting with each other to form an efficient conduit for distributing corporate knowledge.

The foundation technology behind WorkShop NEO is networked objects. Networked object technology builds on, and takes full advantage of, the client-server model for distributed applications. Snap-together application building blocks allow application prototypes to be developed, tested, and deployed very quickly. In essence, networked objects provide a completely new — and

extremely powerful — paradigm for network computing. 'Programming the Network' using objects takes software development for enterprise computing to new levels.

Networked Objects Better Represent Business Processes

One of the most powerful benefits of object technology is its ability to greatly streamline information flow, as well as provide new perspectives on information and the corporate processes which manage it. In effect, *objects are the business*.

Some subtle benefits to organizational representation lie in the philosophy behind object technology. Today's corporate trends toward globalization and decentralization can profit from the modularity, robustness and flexibility of objects. Organizations changing to meet new competition, or to address new markets will find that their information infrastructure is flexible enough to change with them.

The Software Development Crisis

Over the last 20 years, application software development has increased in complexity by fifteen-fold. Application backlog — the time it takes to develop new applications — has grown to over two years. Maintenance costs for applications have steadily increased. And application quality too frequently fails to meet desired targets (see Figure 1). Finally, user expectations are not being met because applications take too long to become available, and when they do, they fail to meet the user's current needs.

Conventional software development techniques have proven increasingly inadequate as the complexity of applications have increased. Structured design techniques, while providing a suitable approach for managing complexity, tend to break down during implementation and revision, as developers are compelled to use their own approaches for dealing with complex software component interactions.

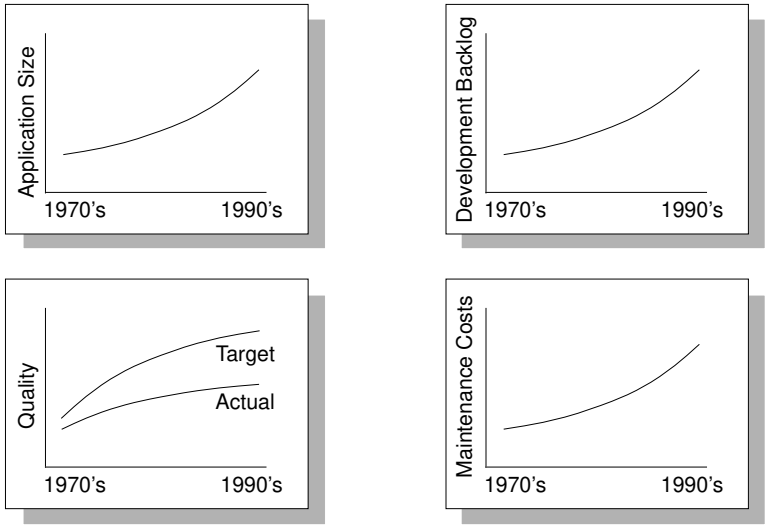

Figure 1 Trends in software development

In contrast, object technology provides a more sustainable approach to managing project and application complexity. Objects are generally better able to isolate the functionality of individual software components, permitting software developers to focus on the application rather than on details of integration.

Software development in the near future will enable programmers to build applications using components from existing catalogs of objects, provided both by SunSoft and third-party vendors. Developers will connect objects and customize their application solution using object-oriented development tools to exchange information and request services via messages passed through the object computing environment (see Figure 2).

In this new object-based environment, the independence of objects will free developers to focus on application issues rather than deal with system specific details. SunSoft's WorkShop NEO development environment delivers a rich set of capabilities isolating the developer from the underlying operating system and specifics of the network. In this new computing environment, applications are no longer packages of software code statically linked together, but rather

are object-based applications built upon dynamic assemblies of other objects — each continually making and breaking connections with other objects to serve the user's momentary needs.

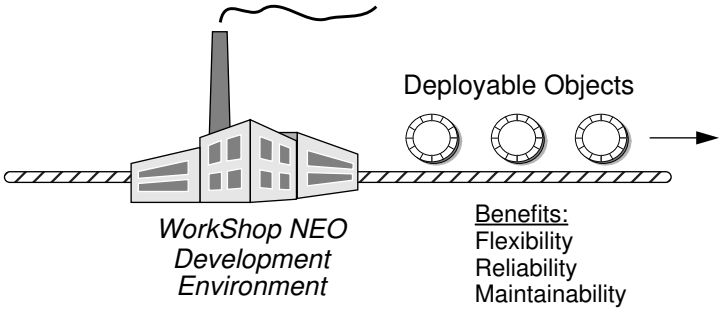

Figure 2 Much like factory automation, WorkShop NEO fosters the efficient development of networked applications

Encapsulation promotes graceful updating of functionality, as well as reliability and maintainability. As a result, object technology can reduce software maintenance effort through the production of better quality systems. Where errors do appear, they would likely be highly localized, and hence be easier to fix. Objects also encourage information hiding, leading to more clearly defined interfaces between modules. In a well designed and implemented object-based system, correcting errors should avoid the common domino effect to other software modules, since there is less direct interaction and no knowledge of other object's internals.

Because object-based applications are inherently designed to be re-engineered, there is freedom to make changes, promoting an atmosphere that emphasizes a relentless commitment to quality and targets the best possible solution, rather than settling for a mediocre one. Well-defined object interface definitions free developers from dependence on a particular language and greatly increase the reuse of existing objects for building new applications.

Object reuse allows new applications to be built on the solid foundations of others, resulting in less development effort and higher quality, more robust code. Most significantly, SunSoft's Solaris NEO operating environment further extend encapsulation around *network interoperability*, thus allowing software to be built that is also free from dependence on location, operating system, or vendor.

By delivering reusable, superior application solutions, new opportunities emerge for establishing a market that sells object software components. Objects allow more than just software module reuse, as new subclasses created from existing objects inherit the functionality of their ancestor, as well as incorporating their own unique functionality.

Development Model for the Networked Enterprise

Today, many companies employ a two-tiered model for building applications, consisting of a graphical front-end and back-end database environment. While popular development tools such as Microsoft's® Visual Basic™ have enabled companies to easily develop simple applications, much of the application's logic is tied to the user interface and the database access logic. The difficulty of this approach lies in that as application requirements change and evolve, it becomes increasingly difficult to modify the source code in an effective manner. Further, changes to the user interface, now tied inexorably to the application logic, are significantly more difficult to make. Precisely the same problem exists on the back-end, where the data has complex ties to the application code. Application backlog, quality, size, and maintenance costs all continue to suffer.

In a nutshell, *the conventional two-tiered model fails to keep pace* with the flexibility and scalability required in rapidly-changing MIS environments. Fast prototyping of applications leading to timely deployment of effective solutions requires a new approach — one which enables the application logic to be altered to provide new features, without impacting the user interface or data storage.

Coincident with the availability of networked object technology, SunSoft sees the solution to the limitations of today's two-tier approach as a three-tier model which effectively moves the application or business logic into a middle layer between the user interface and data storage layers (see Figure 3). This middle level, or *business tier*, fosters a much more flexible and scalable approach to building missions-critical applications. Because application logic is more independent of the user interface, rapid prototyping is easier to perform. At the same time, application logic once bound to the data storage can be moved to the business tier, effectively removing the constraints on database storage and permitting significantly greater scalability of their applications.

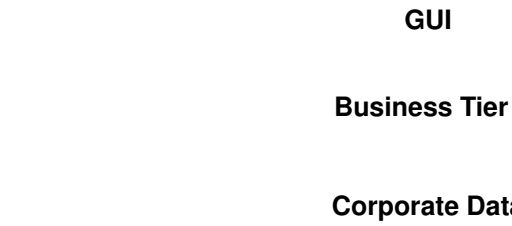

Figure 3 Three-tier Development Model

WorkShop NEO provides tools to implement the three-tier computing model based on networked objects. It enables the software development process itself to be partitioned into three well-defined tiers:

- *A consistent, visually rich, highly intuitive user interface* for both generic as well as custom applications provides an environment that helps leverage the user's time and productivity. The consistency of the interface reduces the learning time for new applications, integrates well with others, and results in improved quality, and look and feel. Building on proven user interface objects helps to facilitate fast prototyping and shortened development cycles. This ease of changing and testing the interface improves the interaction between the end user and the developer, as revisions of an application are fine-tuned to enable the application to perform precisely as it was designed to.
- *Generic business objects can be developed first* followed by more specific sub-classes built from these original objects and inheriting functionality from their parents. These objects are then used as sub-class objects to build custom applications precisely tailored to meet very specific needs.
- *Data access can also be isolated* in the third tier of this model. A common interface to a variety of storage options ensures that the developer can choose a method that fits the needs of the application. Whether using the built-in persistence offered by Solaris NEO, or the ability to access legacy data stored in Relational Database Management Systems (RDBMS) or Object Oriented Database Management Systems (OODBMS), objects may be built to handle any data requirement.

Need for Powerful Tools

SunSoft's mission is to provide next generation development tools for corporations that are seeking to achieve strategic leadership positions within their industries. To make this possible, software development teams must work synergistically using tools that are engineered to work together.

SunSoft's WorkShop NEO ia an innovative software development solution which includes a rich set of tools for managing application development and deployment, building interfaces, and increasing developer and team productivity. These tools provide the basis for the development and deployment of powerful, mission-critical object-based applications.

Importance of Standards

Once satisfied with just application portability that allowed common source code to be compiled and linked for multiple architectures, today's truly heterogeneous enterprise network now requires complete interoperability of objects and applications across multiple platforms, operating systems, languages, and legacy data.

SunSoft's dedication to open industry standards makes this interoperabilty achievable today, and will benefit the organizations that utilize WorkShop NEO to build applications that will last long into the future. Object Management Group (OMG) standards for object interface definitions and Object Request Broker (ORB-to-ORB) interoperability between vendor implementations make this vision a reality.

Use of OMG's Interface Definition Language (IDL) makes applications programming language-independent. Through the ORB's ability to enable a client to access an object knowing only an object reference, location independence is achieved. Further, tools to integrate legacy databases and code provide a means to protect investments in information infrastructures, while freeing the enterprise to make buying decisions without restricting itself to current vendors.

A Comprehensive, Scalable Environment

The scalable performance offered by Sun's advanced multiprocessing and multithreading technology and the built-in, industrial strength, transparent networking capabilities of Solaris supply the foundation for integrating a

heterogeneous mixture of workstations, PCs, NetWare® clients, and mainframes into a dynamic, flexible, and efficient networked object environment. The networking infrastructure encompasses not only UNIX™, but also includes OLE™ compliant MS-Windows™ applications as well as seamless access to both relational and object oriented databases.

NEO Product Family Overview 2

A Comprehensive Solution

The built-in networking of Solaris is the foundation of SunSoft's object environment, and provides a transparent, high performance, distributed computing infrastructure. Industry-leading multiprocessing and multithreading technology ensures that Solaris NEO can scale to meet growing demands of enterprise-wide networked applications, while bundled Solstice NEO management tools allow total control of the object network from any location.

Advanced productivity tools available in WorkShop NEO enable team software development with an object-oriented interface builder that speeds and simplifies the building of prototypes. Promotion of, and conformance to, industry standards further ensures connectivity with mainframes as well as with MS-Windows based desktops. Further, database connectivity and built-in persistence and encapsulation methods provide easy access to legacy data.

Figure 4 illustrates the five key dimensions of the NEO Product Family. Together, Solaris NEO, Solstice NEO, WorkShop NEO, and complementary MS-Windows and database connectivity technology, provide a comprehensive solution for enterprise application systems.

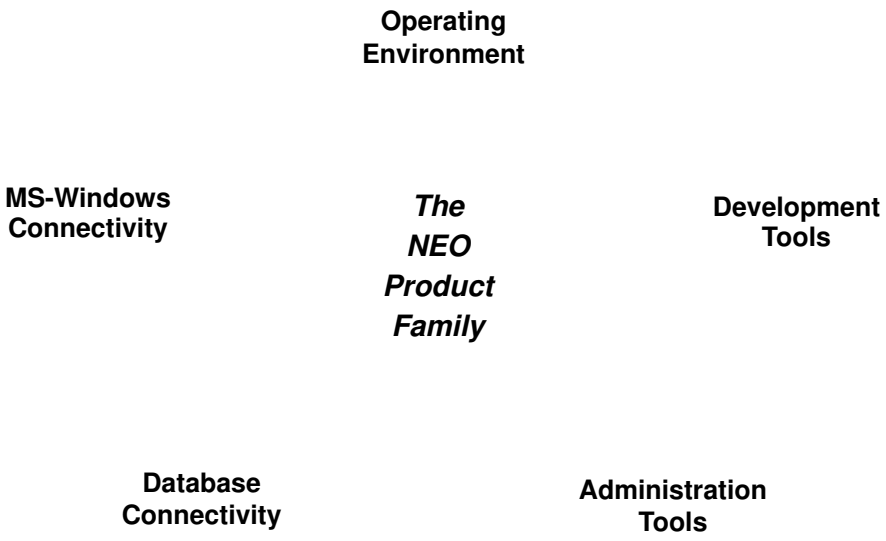

Figure 4 Solaris NEO Product Family

As illustrated in Figure 5, Solaris NEO delivers the means to implement and deploy a comprehensive set of services for the networked enterprise. With Solaris NEO, a three-tier computing architecture can be efficiently implemented that provides scalable and flexible networked applications.

Figure 5 Solaris NEO components supports efficient three-tier computing

Standards for Interoperability

With a membership of over 500 software vendors, developers, and end users, the Object Management Group (OMG™) continues to develop the standards that will ensure a common architecture for heterogeneous, distributed, object-oriented applications. SunSoft was a founding member of the OMG, and has contributed many key specifications designed to ensure portability and interoperability between object system products from multiple vendors.

A major benefit of truly heterogeneous networked objects is the ability to integrate disparate platforms, allowing information resources to be shared across the network, independent of specific operating systems, languages, or implementation techniques. The cornerstone of this effort is the method by which interfaces to objects are defined, allowing behavior to be characterized without specifying the method used to implement it. Evidence of its commitment to both standards and object technology, SunSoft contributed the Interface Definition Language (IDL) to the OMG.

Through the availability of a standardized Inter-ORB Protocol (IOP), vendors can supply objects that interoperate in a heterogeneous environment. OMG's CORBA 2.0 standard Internet IOP (IIOP) is based on TCP/IP, and SunSoft provides a sample reference implementation free of charge, thus enabling vendors to be certain that their systems, objects and applications are interoperable.

Solaris NEO Operating Environment

Solaris NEO is a comprehensive operating environment that includes an intuitive rich graphical desktop, a networked object infrastructure, an advanced runtime environment for shared services, and administration and management tools. Figure 6 illustrates the component structure of Solaris NEO. The shaded boxes represent shared runtime libraries.

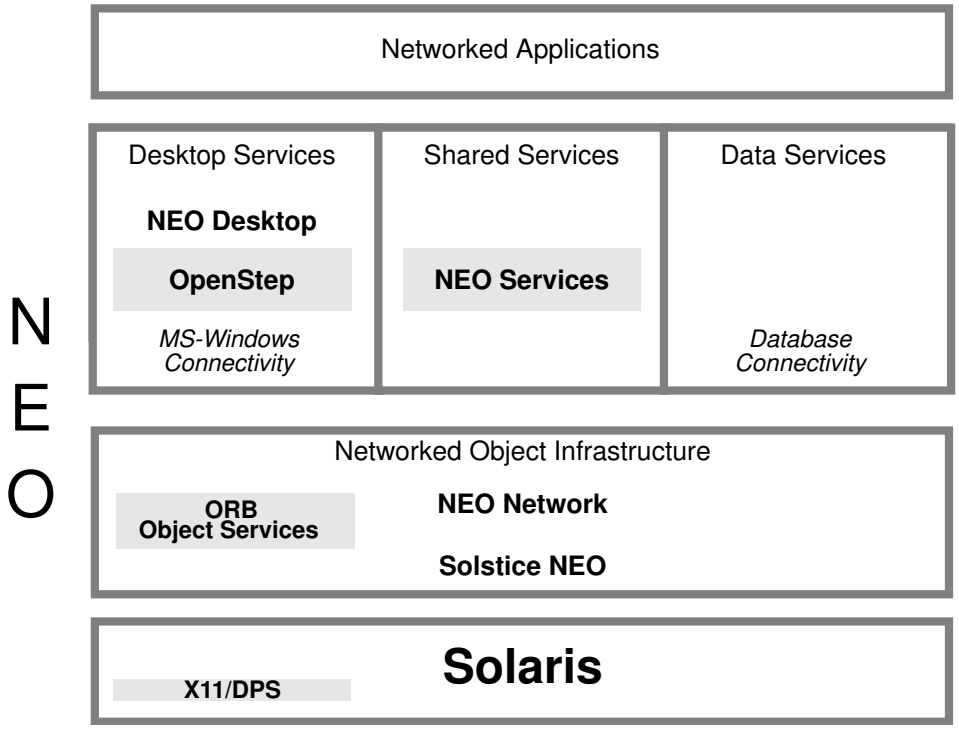

Figure 6 Solaris NEO components

NEO Desktop

The NEO Desktop is a user-friendly OpenStep™ based application environment consisting of the following elements:

- Window Manager
- Desktop Applications
- Desktop Services

The ready-to-use desktop applications (accessories) build on capabilities available to all OpenStep-based applications. These capabilities facilitate application consistency and interoperability in areas such as fonts, color, and on-line hypertext-based help. The common desktop services are usable by any OpenStep application at runtime. These services, accessed programmatically via OpenStep frameworks, include pasteboard (for cut and paste), drag and drop, hyperlinks, spell checker, and print.

NEO Desktop provides the user with the same functionality as the NEXTSTEP™ Application Environment. In addition, NEO Desktop enables the continued use of the thousands of existing applications available for Solaris today. As an integral part of SunSoft's "universal desktop", this includes use of Common Desktop Environment (CDE) applications, MS-Windows applications using Wabi™, and Apple® Macintosh® applications running under the Macintosh Application Environment (MAE™). Interoperability such as drag-and-drop and cut-and-paste is supported between applications from all of these different environments. In addition to OpenStep applications, WABI, MAE, and CDE-Motif applications can be launched from the NEO Desktop. In turn, OpenStep applications can be launched from the CDE desktop.

NEO Desktop delivers a rich visual environment based on X11 with Display PostScript™ extensions and provides an intuitive, easy to use, high quality, multimedia user environment. Informative active icons, advanced drag and drop support, embedded multimedia capabilities, and integrated help facilities enhance the user experience and productivity. Interface consistency aids users in adapting to employing new applications quickly, and productivity is improved because the same functionality is available for all desktop applications. As an example, the NEO Desktop spell checker may be invoked from the text edit and electronic mail desktop applications, or any other custom application.

OpenStep

The OpenStep component of NEO consists of OpenStep-compliant development frameworks and associated shared runtime libraries. The OpenStep frameworks, supplied as Objective C class libraries, comprise the following:

- Graphical User Interface (GUI) Framework
- Application Framework
- Foundation Framework
- Enabling Framework

OpenStep frameworks provide powerful reusable application building blocks that can be used by developers in conjunction with the WorkShop NEO Graphical Application Builder.

NEO Network

NEO Network is an OMG CORBA-compliant networked object infrastructure. It essentially provides an "operating system" for networked objects that is particularly suitable for pre-emptive multitasking, multithreading environments. NEO Network includes an Object Request Broker (ORB) and set of object services.

NEO Network's advanced technology is scalable, high performance, and designed to form a solid platform for shared service computing. Networked administration and management facilities are built-in. In addition, NEO Network is architected to enable future system evolution.

NEO Services

NEO Services are key to enabling the shared service computing model. NEO Services are a comprehensive development framework and associated shared runtime libraries for networked objects and shared services. The NEO Services Development Framework is employed by the WorkShop NEO Networked Object Constructor.

NEO Services automate and make transparent functions that an application developer would normally need to write for areas including shared services, server availability, persistent object availability, concurrent requests, server management, and application installation. It enables full, simplified use of the NEO networked object infrastructure.

Completely unique to Solaris NEO, NEO Services are fully compatible with CORBA specifications, and provide the extended services required to facilitate rapid development of networked applications.

Solstice NEO

Management tools are key to the success of distributed computing environments. By providing network management from any location, enterprise management tools improve administrative capabilities and responsiveness while reducing costs.

Solstice NEO, bundled with Solaris NEO, complements this strategy and adds capabilities for the administration and management of networked applications and shared services to the Solstice product family. With Solstice NEO, resources can be monitored and controlled from any point on the network. System, workgroup, shared service, and application management capabilities are supported in the areas of:

- System Installation and Management
- Application Installation and Administration
- Workgroup and Shared Service Administration

A full set of tools provide for one-step system and application installation including incremental upgrades and extensive on-line help.

WorkShop NEO Development Environment

A rich environment for the development of applications is a major component of the NEO product family. SunSoft's award winning WorkShop has been extended to include tools unique to NEO. WorkShop NEO is an integrated development environment that includes tools for building networked objects and shared services and tools for building custom applications including GUI front-ends.

Figure 7 illustrates the component structure of WorkShop NEO.

NEOworks™

The NEOworks component of WorkShop NEO consists of:

- CORBA networked object development tools and the NEO Services Development Framework
- OpenStep graphical application development tools and the OpenStep-compliant GUI, Application, Foundation, and Enabling Frameworks

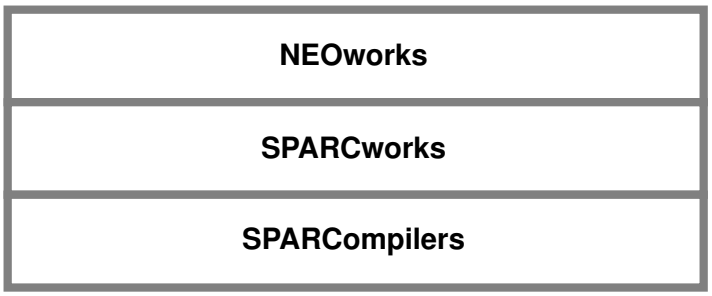

Figure 7 WorkShop NEO components

Tools for Constructing Networked Objects

Included in NEOworks are the Networked Object Constructor, IDL compiler, Networked Object Debugger, and NEO Services Development Framework. Coupled with the OpenStep graphical application development tools and frameworks, and SunSoft's SPARCompilers and SPARCworks tools, NEOworks enables developers to rapidly design, implement, test, and deploy shared services and complete networked applications.

Tools for Building Graphical Applications

SunSoft's NEOworks OpenStep development tools leverage the industry-leading NEXTSTEP object development environment. As part of NEOworks, they deliver a competitive edge to enterprises relying on custom software applications. Familiar tools such as Interface Builder, Project Builder, Header Viewer, and Icon Builder facilitate the prototyping and deployment of graphical front-end applications with less effort and at lower cost.

SPARCworks and SPARCompilers

WorkShop NEO also includes advanced SPARCworks developer productivity tools and SPARCompilers. This includes tools for C, C++ and Objective C programming, tools for multithreaded programming, and tools to support team development.

MS-Windows Connectivity

Today's enterprise environment includes MS-Windows desktops, mainframes with legacy databases, as well as workgroup LANs. A successful software environment must provide access to, and seamlessly integration of these technologies into a coherent conduit for information exchange. SunSoft's own experience in deploying heterogeneous environments has provided us with the needed vision to begin forging the standards and relationships with key technology leaders to ensure this transparent connectivity.

Complementary technology from IONA Technologies, in combination with Solaris NEO, provides connectivity with MS-Windows desktops including interoperability with OLE and the underlying Component Object Model (COM). This technology, based on IONA's Orbix™ product, allows MS-Windows applications to access NEO objects and shared services and act as networked enterprise application front-ends. Network communication is based on OMG's CORBA 2.0 Interoperability specifications (see Figure 8).

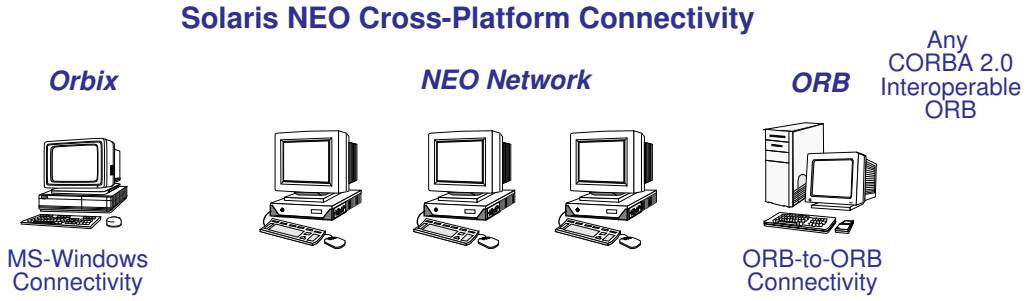

Figure 8 Solaris NEO cross-platform connectivity

2

Database Connectivity

A wide range of options are available in NEO to meet data storage, and database access and integration needs. The built-in Solaris NEO persistence support, based on technology licensed from Object Design, Inc., provides a mechanism for applications with simple object data storage requirements. For more complex needs, application developers can employ existing Relational Database Management Systems (RDBMS), or Object Oriented Database Management Systems (OODBMS).

Complementary technology from Persistence™ Software, Inc., in combination with Solaris NEO, provides mechanisms for automatically mapping objects to relational tables, ensuring data integrity by enforcing object constraints and controlling relational transactions, as well as scaling to multiprocessor and multiple database implementations.

As a founder of the Object Database Management Group (ODMG), SunSoft is promoting standards for interoperability between OODBMS providers. The ODMG-93 specification defines interfaces to object databases that ensure heterogeneous interoperability, object portability and reuse, as well as concurrent access.

NEOworks — Tools for Constructing Networked Objects 3

WorkShop NEO provides an innovative, comprehensive, integrated set of tools called NEOworks for developing networked applications. These tools fit into two general categories: *tools for constructing networked objects* and *tools for building graphical applications* (described in the next chapter). Coupled with SunSoft's advanced SPARCompilers and SPARCworks development tools, developers can rapidly design, implement, test, and deploy powerful networked applications and shared services.

NEOworks tools for constructing objects include the Networked Object Constructor, the OMG-compliant Interface Definition Language (IDL) Compiler, the Networked Object Debugger, and the NEO Services Development Framework.

Networked Object Constructor

The NEOworks Networked Object Constructor is central to networked object and shared service development in WorkShop NEO. It effectively frees the developer from tedious implementation details, allowing them to focus on the actual semantics of the application. The Networked Object Constructor is one of the main factors in achieving the large scale programmer productivity improvements that can be realized using the WorkShop NEO development environment.

Simplified Development

The Networked Object Constructor simplifies the development of networked applications composed of networked objects. It automates the creation of object implementations, in particular those required for providing shared services. In doing so, it manages the networking complexity of the shared service computing model. Networked objects and shared services developed using the Networked Object Constructor take full advantage of the common NEO Services Runtime System, making many runtime support functions completely transparent.

With the Networked Object Constructor, an application or shared service can be written as if for a non-networked, unshared environment. The developer need only supply object method behavior. No additional work is then required to make the code work in a networked, shared environment. The Networked Object Constructor also simplifies the development of services that are **not** networked (i.e., those intended to be used locally to a computer) where object interfaces are still expressed in IDL to allow programming language independence.

Advanced Features

Advanced features of the Networked Object Constructor include:

- *Automated housekeeping code.* Most of the housekeeping code necessary in a robust networked object environment is automated. By generating implementations of standard protocols supported at runtime by the NEO Services Runtime System, the Networked Object Constructor simplifies the installation, administration and debugging of objects. The approach focuses on supporting C++ implementations that use the NEO Network ORB Basic Object Adapter (BOA) and (optional) transparent persistence.
- *Satisfies shared service requirements.* Shared services require the handling of multiple concurrent clients, server and persistent object availability (activation and state persistence), multithreading (required for sharing and scalability), and application component management (e.g., installation and upgrade). These are fully handled by the Networked Object Constructor and NEO Services.
- *Standardized protocols for common operation and reduced complexity.* Implementations of standardized protocols are automatically generated and thus every server program supports these protocols. Standardization means

that networked objects and shared services can be debugged, installed, administered and managed together through a common set of APIs and Solstice NEO tools. In this way, the NEO Services Runtime System forms an *intelligent layer* between the application code and the BOA that insulates the application from networking complexity through a combination of transparent functions and APIs to support specific protocols.

- *Rapid iterative development of interfaces and implementations.* Deployment and development versions can coexist on the same computer, and a developer can use both. Support is also provided for the graceful evolution of deployed interfaces and implementations. Developers can create and test new interfaces and implementations without impacting deployed objects.

Specific Areas of Support

The Network Object Constructor provides specific automated support in the following areas:

- *NEO BOA support.* Coupled with the NEO Server and Persistent Object Availability capabilities, this includes support for the creation of BOA-based objects, transparent activation and destruction of objects, and automatic object deactivation and server shutdown.
- *Installation support.* SVR4 installation packages are automatically generated. These can be NFS-mountable packages installed on a server computer, and mounted and enabled on client computers. Upgradable application packages can also be built.
- *Management and debugging support.* Built-in server management support is provided through the NEO Server Management capability. Also provided is support for distributed debugging, request tracing, and exception logging.
- *C++ programming support.* This includes basic support for atomic operations (locking/commit), support for C++ inheritance in servant classes, and support for object reuse through en-masse and per-operation delegation. This allows the flexible binary reuse of object implementations
- *Optional persistent data support*. Transparent persistence of object state is provided using the NEO Data Store Manager. Full hooks are also provided to allow custom persistence schemes.
- *Shared service support.* Code generated by the Networked Object Constructor conforms with the NEO Workgroup Support name space organization and policy for registering and finding shared services.

Object Server Language

The Object Server Language (OSL) is used to define object implementation and server process and program characteristics. The language addresses such areas as concurrency control and locking policies, object creation operations, object installation details, and (optional) persistent data management.

A separate Data Definition Language (described below) allows the developer to define the persistent data that is to be transparently read and written by the NEO Data Store Manager.

Data Definition Language

The Data Definition Language (DDL), whose syntax is a subset of OMG IDL, is used to define data to be made persistent. DDL provides a standardized way of describing the persistent state of an object in an architecture and programming language-independent fashion. State is defined as data objects that only have attributes. Data object interfaces are grouped into schemas. DDL supports the complete range of IDL data types with the exception of **`Any`**.

OSL Compiler

The Object Server Language Compiler (OSL Compiler) generates classes that inherit or use the NEO Services Development Framework and, at runtime, work with the NEO Services Runtime System. These classes address the following areas:

- *Installation* (e.g., registration with the BOA and hooks for reconfiguration)
- *Object creation and initialization functions* (including default initialization for servant objects and persistent state)
- *Server and object activation and deactivation functions* (to be invoked by the BOA)
- *Transparent persistence of object state*
- *Server process startup, timeout, and shutdown*
- *Locking policy*
- *Tracing and logging of status information* (controlled and recorded in a centralized fashion)
- *Subobject support*

- *Administration and installation support*

The OSL Compiler also generates sample code to which application-specific code can be added. This includes C++ servant classes that implement the object's operations in terms of C++ member functions. The optional automatic filemerge of newly generated code with existing customized source code files is supported.

DDL Compiler

The DDL Compiler reads the data description for an object expressed in DDL and produces all of the code necessary to read and write the object's state using the NEO Data Store Manager. The compiler also produces the header files necessary for accessing the data, for use in the object implementation source code.

Object Server Maker

The Object Server Maker generates (UNIX make) makefiles — based on an Imakefile definition — that control the development process.

NEO Services Development Framework

In general the NEO Services Development Framework is not used directly by developers, but rather provides supporting APIs and code for the Networked Object Constructor tools. The NEO Services Runtime System, consisting of shared libraries, provides the necessary runtime support for code generated by the Network Object Constructor and for calls made directly from the object implementation.

The NEO Services Development Framework is described in more detail in the following sections.

IDL Compiler

The OMG IDL-compliant NEOworks IDL Compiler provides a single front end and multiple back ends supporting C and C++ language mappings and the loading of information into the NEO Interface Repository. At development

time, the IDL Compiler generates client stubs and server skeletons, and associated header and other files. It also generates typecodes for the typecode interpreter, and interface and logical IDs for use in stubs.

The NEOworks IDL Compiler provides additional, extended support for data types including 64-bit ints, long dbls, wchars, wstrings and interface versioning.

Networked Object Debugger

The NEOworks Networked Object Debugger provides the full capabilities of the standard SPARCworks MT debugger. It handles multiple processes, servers, and thread contexts, and the state of running server processes can be examined and multiple threads inspected and debugged. The debugger can graphically manipulate several server process from a single GUI.

In addition, the Networked Object Debugger adds the ability to follow an operation invocation on a networked object, stepping from a client to an object running in a remote server process and back again. This essentially provides distributed program stepping in which a single step may cross process and machine boundaries.

NEO Services Development Framework

NEO Services are a comprehensive development framework and associated shared libraries for networked objects and shared services. The development framework is employed by the NEOworks Networked Object Constructor. The shared libraries make up the NEO Services Runtime System in Solaris NEO.

Unique to NEO, NEO Services are fully compatible with CORBA and enable full, simplified use of the NEO networked object infrastructure. NEO Services automate and make transparent functions that an application developer would normally need to write in the areas of:

- Workgroup Support
- Shared Service Finder
- Server Availability
- Persistent Object Availability
- Data Store Manager
- Concurrent Requests

- Implementation Support
- Server Management
- Application Installation

Table 1 summarizes the functions of NEO Services.

Function	Description
Workgroup Support	Establishes and implements standardized workgroup/computer naming policies, and allows access to shared services
Shared Service Finder	Registers and finds services based on NEO Naming Service, and enables relocation of services without changing or recompiling client code
Server Availability	Simplifies server activation, provides transparent management of object implementations grouped in a server program, and handles housekeeping functions
Persistent Object Availability	Provides transparent management of persistent objects, manages application-independent ORB-related housekeeping, automates support for life-cycle create and destroy and object instance and implementation activation
Data Store Manager	Provides transparent persistence for object state
Concurrent Requests	Provides transparent management of concurrent requests to multithreaded object implementations, and supports multiple locking policies
Implementation Support	Provides functions that simplify and make easier development of object implementations
Server Management	Provides server and application management functions and transparent support for management objects installed as part of applications
Application Installation	Provides transparent support for application installation

Table 1: NEO Services functions

Workgroup Support

The NEO Workgroup Support establishes and implements standardized workgroup and computer resource naming policies. The workgroup is the unit for administration, management, and sharing in NEO. A workgroup is a collection of computers that can include workstations and server machines. Each user may select which of their own resources may be shared by the

workgroup, and which workgroup services are to be used instead of local resources. A user can have a private part of the workgroup resources reserved for their development or other purposes.

Workgroup Support allows access to shared services by enabling services to be registered in a well known place and found using the NEO Shared Service Finder. In this way, the need for applications to have a detailed knowledge of the system name space is eliminated.

Shared Service Finder

The NEO Shared Service Finder employs a federated approach to registering and finding services based on the NEO Naming Service. Building on the Workgroup Support resource naming policies, it uses a federation of predefined and application-specific naming contexts.

Services (i.e., named objects) are registered at install time or runtime in a well known place. Applications can then dynamically find a service that is available to run in an appropriate server process. A service can be local to a computer or shared by a workgroup. A mode can be selected that controls whether a deployed service or a version under development is found.

The Shared Service Finder enables the relocation of services without needing to modify or recompile client code. In this way, service requests can be distributed to the most appropriate resource in a workgroup and dynamic load balancing is facilitated.

Server Availability

The NEO Server Availability builds on and simplifies NEO Network ORB server process activation. It provides the transparent management of the availability of object implementations grouped in a server program and takes care of application-independent ORB-related housekeeping functions that the server program developer would otherwise need to provide. With NEO Services, server program developers need only write minimal application-specific “server availability” code if needed.

Server Availability automates server process startup on arrival of a request for any object in a server program as well as server process shutdown after a period of inactivity. The timeout (i.e., maximum idle period) is configurable

per server process. The process waits for all objects in the server process to be deactivated before shutdown occurs. The wait interval and shutdown retry cycle time are configurable.

Because server availability is handled in a standardized way, the system can automatically manage and recover resources such as memory. In conjunction with the NEO Persistent Object Availability, the use of system resources is minimized.

Persistent Object Availability

The NEO Persistent Object Availability builds on and simplifies NEO Network ORB object activation. It provides transparent server-side management of the availability of persistent objects and takes care of application-independent ORB-related housekeeping functions that the object developer would otherwise need to provide. With NEO Services, object developers need only write minimal application-specific "persistent object availability" code if needed.

Persistent Object Availability automates support for object life-cycle create and destroy, and object instance and implementation activation. It automates the activation of the object implementation and instance on the arrival of a request. A *servant* C++ object implementing the operations of the object's interface, transient data, and other functions is automatically instantiated when an object is activated, and destroyed when the object is deactivated.

Persistent Object Availability also automates the deactivation of objects after a configurable period of inactivity. The timeout (i.e., maximum idle period) is configurable per object implementation. It transparently handles thread quiescing and waits for all pending operations to complete using a configurable wait and retry cycle time.

Optional transparent persistence of an object's state is provided based on the NEO Data Store Manager. This includes automatic atomic update at timed intervals and before deactivation of the object. The time interval is configurable per implementation. In addition, an infrastructure is provided to support custom persistence. All the hooks necessary to incorporate custom persistence mechanisms are provided to address cases where more control is needed over performance and the handling of data formats and legacy data.

Data Store Manager

Used in conjunction with Persistent Object Availability, the NEO Data Store Manager provides transparent persistence for the state of an object. It supports the model that allows networked objects to be implemented by code organized as servant objects.

Based on technology provided by Object Design, Inc., the Data Store Manager is designed to efficiently support development models in which applications are composed of many fine-grain objects comparable in size and complexity to typical C++ objects. Data objects can contain pointers to other data objects, allowing users to create persistent representations of complex, linked data structures in a natural manner.

A subset of IDL, called the Data Definition Language (DDL), is used to define the persistent state in terms of "data objects". The IDL language binding approach is used to provide client bindings for a wide range of architectures and programming languages. Stored data maintains the same level of type safety as that provided by IDL.

Applications can create "data stores" (the unit of storage) in any part of the file system to which they have access. Transparent to clients, data objects are cached in local memory. Access to individual attribute values is essentially at the speed of native programming language calls. Modifying the persistent state of an object is done by simply modifying the C++ state of the object in memory — the rest is automatic and transparent.

Concurrent access to clusters within a data store is supported, making it ideally suited for supporting NEO multithreaded server programs. Automatic atomic updates provide a two-phase commit protocol, in anticipation of distributed transactions involving multiple services. Either all the changes made within an update are recorded, or none of the changes are recorded. A server process can be atomically updating several clusters with different schema definitions concurrently.

Based on the OMG CORBAservices Persistent Object Service specification, the Data Store Manager is a upwards-compatible subset of the object database specification developed by the Object Database Management Group (ODMG). An object implementation can therefore be easily upgraded as application requirements evolve.

Concurrent Requests

The NEO Network ORB spawns a new thread for each incoming request to a server process. NEO Concurrent Requests provides transparent management of concurrent requests to multithreaded object implementations, ensuring the integrity of shared data. This frees developers from the implementation details of multithreaded programming, allowing them to focus on the application, while gaining the significant performance advantages and scalability of Solaris' advanced multithreading technology.

Three different locking policies are supported: *mutex* (default), *reader-writer*, and *fine grain*. The locking policy is selectable on a per object implementation basis. Scoped locks are supported: with mutex and reader-writer locking, locks are automatically released when out of scope of the locking variable, or when the variable is destroyed.

- *Mutex Locking*

  Only one request at a time is allowed for each object (i.e., only one method in the object implementation is active at any time): a single thread gains exclusive access to an object's persistent state. Other objects (instances as well as implementations) in a server process can be servicing requests simultaneously.

- *Reader-writer Locking*

  Operations that only read shared data (persistent and transient) are distinguished from those that (potentially) may write or change data. Each operation in an object's interface is defined as either reader or writer. This locking policy enables multiple concurrent reader operations but writer operations gain exclusive use.

- *"Fine grain" Locking*

  Higher level locking can be turned off and developers can implement custom fine grain locking policies. Shared data can be protected at the thread mutex level within a method.

Implementation Support

The NEO Implementation Support provides functions that simplify and make easier the development of object implementations. This includes:

- *Servant Support*: constructor, destructors, locking, and deactivation

- *Smart Object References*: automatic memory deallocation for smart object references when they go out of scope or upon assignment
- *Reference Data Manipulation*: simplified manipulation and management of reference data associated with object references
- *Exception Handling and Message Text*: including an internationalized message text catalog, extensible by developers
- *Object Tracing*: standardized, automated way of tracing and logging; controlled on a per-server process basis
- *Message Logging*: predefined logging macros (in addition to trace messages) and runtime configurable control of message source, message destination, and output format on a per-server process basis
- *Utility Support*: ease-of-use support for custom persistence, subobjects, and other NEO Network and NEO Services features

Server Management

Individual custom applications can readily capitalize on the advanced built-in administration and management capabilities of Solaris NEO. The NEO Server Management provides server and application management functions and transparent support for management objects that are installed as part of an application. Management objects are automatically generated by the NEOworks Networked Object Constructor and incorporated into the server program along with application objects.

Server Management completely takes care of server management and application management functions that a server program developer would otherwise need to provide. It enables the status of computers, server processes, and objects to be interactively monitored, administered, and managed by Solstice NEO tools.

With NEO Services, server processes can be activated and shutdown manually, the current status of a server process can be queried, object tracing and message logging can be turned on or off, and the persistent state (transparent and custom persistence) of objects in the server process can be backed-up and restored without any programming by the application developer.

Application Installation

NEO Application Installation provides transparent support for the application installation process. Application installation code is automatically generated by the NEOworks Networked Object Constructor and accessed and controlled by Solstice NEO tools. Application Installation completely takes care of installation functions that the application developer would otherwise need to provide.

Installation steps that are automated include the registration of server programs with the NEO Network ORB, the installation of IDL information associated with server programs in the NEO Interface Repository, and the registration of shared services for access via the NEO Shared Service Finder.

The NEO Application Installation builds on the easy-to-use SVR4 packaging concept, providing a familiar and reliable method for installing and deinstalling software packages. Applications and objects are installed and their availability automatically registered, allowing seamless upgrading with minimal impact on users.

NEOworks — Tools for Building Graphical Applications

4

NEO's support of OpenStep provides functionality and interoperability which leverages the industry-leading NEXTSTEP development environment. The powerful set of OpenStep development tools included in WorkShop NEO, provides a development platform that delivers a competitive edge to enterprises that rely on custom software applications. Familiar tools such as Interface Builder, Project Builder, and Icon Builder are supplied to facilitate the development of reusable objects, as well as the rapid prototyping and deployment of customized networked applications that include GUI front-ends to shared services.

OpenStep

The OpenStep component of NEO consists of OpenStep-compliant development frameworks and shared runtime libraries. The OpenStep frameworks, supplied as Objective C class libraries, comprise a Graphical User Interface (GUI) Framework, Application Framework, Foundation Framework, and Enabling Framework.

OpenStep frameworks provide powerful reusable application building blocks that can be used by developers in conjunction with the NEOworks Graphical Application Builder. The frameworks are extensible by subclassing or delegation (via hooks to dynamically add new components) and feature the following capabilities:

- Rich data-type support including images and multi-attributed text suitable for supporting multimedia documents

- Same imaging model for screen and hardcopy (via DPS)
- Common fonts and printing management
- Consistent user help model
- Abstractions for cross-platform program portability
- Support for Sun “Level 3” internationalization for end-user, developer and administrator

The AppKit class library, SunSoft’s implementation of the OpenStep GUI and Application Frameworks, is based on the X11 Display PostScript (DPS) capability supplied as part of Solaris. AppKit provides a rich set of standard objects useful for all applications, with a wide variety of customization options. Included is a complete set of user interface objects and controls, as well as objects that support data sharing and inter-application communication.

AppKit is similar in overall function to the CDE-Motif toolkit. It incorporates DPS rendering within top level X Windows and supports Alpha compositing. It provides extremely sophisticated shading and transparency effects beyond the simple 3D shading provided by most X Windows toolkits. Other extended features include support for lower level data types available in the OpenStep Foundation Framework (implemented as the FoundationKit class library).

Graphical Application Builder

The NEOworks OpenStep Graphical Application Builder component of WorkShop NEO includes:

- Project Builder
- Interface Builder
- Header Viewer
- Icon Builder
- OpenStep Frameworks

Project Builder

The NEOworks Project Builder is a graphical tool for building, debugging and maintaining OpenStep applications (see Figure 9). It manages all files and resources associated with an application. A *project* (corresponding to an application or shared library) is a collection of source code files, make files,

localization files, icon and image files (TIFF or EPS format) and persistent object files. Persistent object files are output from the Interface Builder and define GUI objects that are dynamically created at runtime.

Figure 9 Project Builder

Project Builder is a complete development environment in which the developer can define new projects, start the Interface Builder and edit, compile, link, run and debug generated code. It supports multiple subprojects and the ability to search for strings in source code. Project Builder and InterfaceBuilder work together so that most developers never have to write makefiles and rarely need to directly use the OpenStep AppKit Objective C classes. Project Builder also automates the linking in of code for CORBA networked objects.

Project Builder has an easy-to-use point and click interface, reducing the ramp-up time for new developers, and forms a valuable asset to the development team as a repository for project-wide information and resources.

Interface Builder

The NEOworks Interface Builder provides a complete user interface development environment that greatly improves developer productivity, allowing the developer to focus on the application, and not on the interface

implementation details (see Figure 10). Because user interface changes are fast and simple, rapid prototyping is encouraged allowing multiple iterations to obtain the very best interface possible. This approach to enhancing the design cycle with feedback helps to significantly improve usability and the quality of human-software interaction. The ability to define and refine the interactions between objects through an object editor, makes testing and refining applications a quick and painless process.

A consistent user interface for both general-purpose and custom applications provides standard panels for common functions such as file opening, saving, and printing, thus reducing the time required for a user to become productive with a new application. A single Display PostScript model for viewing and printing enhances the end user experience as well as the development cycle.

Figure 10 Interface Builder

Applications may gain consistent, advanced features and functionality with little developer effort by building on an extensive library of objects that provide the common functions required by applications. Hooks for adding customized help for an application enable the developer to take advantage of a well organized, easy to use, consistent help facility with hyperlink capabilities.

The NEOworks Interface Builder is a graphical tool for OpenStep GUI generation and object editing. It allows the developer to graphically design the relationship between objects in an application. Interface Builder includes a user interface layout tool and facilities for subclassing existing objects. Unlike most X Windows GUI generators, developers never have to look at the underlying generated OpenStep code.

Key features of the Interface Builder include:

- *Icon-based palette panel for interface composition.* The palette panel contains icons of user interface objects (e.g., menu items, new top level windows, controls, browser and text objects) that can be deposited onto the main window or main menu to compose the user interface.
- *Palettes for built-in, third-party, and custom objects.* In addition to providing graphical palettes for built-in objects, third party and custom palettes can be loaded. New palettes can be defined by the user and dynamically loaded into the palette panel.
- *File panel for managing file objects.* This includes making additions to the four standard types of file objects: images, sounds, classes and objects. New classes can be defined using the classes subpanel. Interface Builder automatically generates appropriate header files and methods stubs.
- *Extensive tools submenu.* The tools submenu includes an object inspector to set properties for objects. Properties include connections between user interface objects and custom application-specific objects. An example of this is connecting a button to code that increments a counter.

Header Viewer

The NEOworks Header Viewer is a class browser for navigating class hierarchies. It provides access to class information and documentation, and displays classes, methods, protocols, and other language constructs. Selecting a particular construct causes the associated documentation or header file to be displayed.

Header Viewer handles Objective C library headers, making it very useful for working with AppKit and FoundationKit headers.

Icon Builder

The NEOworks Icon Builder is a simple, general purpose, editing tool for creating TIFF files. In common with most graphical editing tools, Icon Builder supports simple graphical painting, geometric figures (e.g., line, circle, oval, and rectangle), area fill, and text. Icon Builder can also be used to pixel-edit icons for use on the NEO Desktop.

In addition, Icon Builder supports the creation of color transparency and shading effects using a color panel. It also supports TIFF files containing multiple bit depths (i.e., bits per pixel).

OpenStep Frameworks

Used in conjunction with the Graphical Application Builder tools and with the SPARCompiler Objective C++, the OpenStep frameworks comprise the following:

- OpenStep GUI Framework
- OpenStep Application Framework
- OpenStep Foundation Framework
- OpenStep Enabling Framework

These frameworks are described in the following sections.

OpenStep GUI Framework

The OpenStep GUI Framework provides reusable, customizable objects for building and using windows, panels, structural views, control views, images, colors, text, and fonts. Table 2 lists the GUI Framework objects and their classes.

Objects	Classes
Window	Window, Screen, Event
Panel	Panel, Menu, ActionCell, MenuCell, PopUpList,
Structural View	View, Box, SplitView, ScrollView, ClipView
Control View	Control, Cell, BrowserCell, Browser, Button, PopUpButton, ButtonCell, Slider, SliderCell, Scroller
Image	Image, ImageRep, BitmapImageRep, EPSImageRep, CustomImageRep, CachedImageRep, Cursor
Color	Color, ColorList, ColorPicker, ColorPanel, ColorWell
Text	Text, CStringtext, TextField, TextFieldCell
Font	Font, FontManager, FontPanel

Table 2: OpenStep GUI Framework objects and their classes

OpenStep Application Framework

The OpenStep Application Framework provides reusable, customizable objects for managing and manipulating workspaces, forms, help, filing, printing, spelling, data exchange (for cut and paste, and drag and drop), and data links (hyperlinks). Predefined window panels for incorporation in applications are provided for help (enabling consistent, context-sensitive, hypertext linked, on-line help), opening and saving files, printing documents, creating and navigating hyperlinks, and for checking spelling and building user dictionaries. Table 3 lists the Application Framework objects and their classes.

Objects	Classes
Workspace	Workspace
Form	Matrix, Form, FormCell
Help	HelpPanel
File	SavePanel, OpenPanel
Print	PrintInfo, PrintOperation, Printer, PrintPanel, PageLayout
Spell	SpellChecker, SpellServer, SpellPanel
Data Exchange	Pasteboard
Data Link (Hyperlink)	DataLink, DataLink Manager, DataLink Panel, Selection

Table 3: OpenStep Application Framework objects and their classes

OpenStep Foundation Framework

The OpenStep Foundation Framework provides reusable, customizable objects that support basic program functions. This includes operation processing, creating and accessing basic structured and stored data, thread control, and obtaining information about the program runtime environment. Table 4 lists the Foundation Framework objects and their classes.

Objects	Classes
Operation Processing	Object, Invocation, MethodSignature, Exception, AssertionHandler RunLoop Notification, NotificationCenter, NotificationQueue Proxy, DistantObject
Structured Data	Array, MutableArray, Number, Value, Enumerator String, Scanner, MutableString Set, MutableSet, CountedSet Data, MutableData, Serializer, Deserializer CharacterSet, MutableCharacterSet
Stored Data	BTreeBlock, BTreeCursor ByteStore, ByteStoreFile Dictionary, MutableDictionary Coder, Archiver, Unarchive Date, CalandarDate TimeZone, TimeZoneDetail
Thread Control	Thread, Lock, Connection, RecursiveLock, ConditionLock
Program Environment	ProcessInfo, Timer, UserDefaults, Bundle, AutoReleasePool

Table 4: OpenStep Foundation Framework objects and their classes

OpenStep Enabling Framework

The OpenStep Enabling Framework provides reusable, customizable objects that enable the integration of networked objects.

Callback Facility

The Callback Facility supports the ability for OpenStep-based Objective C programs to call networked objects and properly handle the return results through the use of callback objects. A callback object is a limited form of networked object.

AppKit Synchronizer

NEO application code can be multithreaded, but most libraries are not MT-safe, including the AppKit. Only a single thread in a process may safely access the AppKit at any one time. On its own, AppKit cannot properly handle multiple threads arising from separate NEO Network ORB requests from the same process. The AppKit Synchronizer can be used by applications to serialize the request processing. This is sufficient for simple updates.

SPARCworks and SPARCompilers 5

SPARCompilers

Objective C++

The SPARCompiler Objective C++ included with WorkShop NEO provides full support for Objective C using an extended native ANSI C++-compliant compiler. Objective C++ is not a new programming language — the compiler provides support for both Objective C **and** C++.

Key features of SPARCompiler Objective C++ include:

- *Objective C and C++ binaries can be mixed in the same process.*
- *Objective C and C++ source code can be intermixed in the same program.* Some minor restrictions exist, such as nesting C++ classes in Objective C classes and vice versa. C++ and Objective C class hierarchies also cannot be mixed.
- *Seamless debugging of Objective C and C++ programs.* NEOworks and SPARCworks debuggers are language sensitive and can understand both Objective C and C++. Breakpoints may be set within Objective C methods and on Objective C method invocations. Stepping through a method invocation is possible without stepping into the method's lookup runtime. Class and instance methods may be invoked interactively. The contents and superclass instance variables of an Objective C object can also be printed.

- *SPARCworks tools understand both Objective C and C++ source, headers, etc.* For example, the SPARCworks SourceBrowser can answer queries containing Objective C syntax. The SourceBrowser can also generate a class graph of Objective C classes.
- *Integration of OpenStep and networked object programming.* Objective C programs can directly invoke operations on networked objects using C++ calls (using the IDL C++ mapping). C and C++ code can call selected OpenStep Objective C objects residing in the same process via IDL-style stubs. In addition, Objective C code can be made network accessible by wrapping it in C++ code that supports an IDL interface.

C and C++

WorkShop NEO includes the SPARCompiler C, providing K&R and ANSI compliant C (ANSI CX3.159-1989). As part of the SPARCompiler Objective C++, a full ANSI C++-compliant compiler is provided.

SPARCworks Developer Productivity Tools

SPARCworks developer productivity tools are an integrated set of development tools providing support for the development, deployment, and evolution of applications. The tools are engineered to improve individual developer productivity as well as facilitate improved application performance and quality.

Tools for C, C++, and Objective C Programming

SourceBrowser

A powerful Source Browser facilitates examination and modification of source code. A flexible search facility can filter based on definition, use, and type of symbols. CallGrapher displays C++ class inheritance hierarchies to help visualize relationships between object classes, while ClassBrowser helps navigate through classes, displaying data and function members.

PerformanceAnalyzer

PerformanceAnalyzer provides a wealth of graphical and textual information that helps in the tuning of applications for maximum performance. Information concerning processing time, memory usage, and system utilization is provided. A unique code ordering capability can increase performance by organizing the program to minimize runtime page faults.

FileMerge

Software development teams must often work on the same parts of code simultaneously and FileMerge is essential for successfully comparing and combining source code changes. Both versions of code are shown side-by-side, and the differences are graphically highlighted. Changes can be made automatically or manually to combine the two sources into a common, merged, file.

MakeTool

MakeTool automates the UNIX make facility and provides a graphical user interface, reducing errors and development time.

SPARCworks Manager

SPARCworks manager provides a common focal point for the SPARCworks toolset, launching and managing tools through a single graphical interface.

SPARCworks/iMPact

SPARCworks/iMPact tools are designed to get the most of multithreaded and multiprocessing applications.

Tools for Multithreaded Programming

MT Debugger

The MT Debugger supports dynamic access and control of threaded programs. ThreadInspector graphically views an applications threads at runtime. The debugger can seamlessly traverse system boundaries, since client and server objects may reside on different systems. A powerful 'fix and continue' capability saves valuable time in the debug stage of development by eliminating the linking phase in the edit-compile-link-test loop.

LockLint

LockLint statically analyzes code for the most common MT synchronization problems, deadlocks and data races.

ThreadAnalyzer

ThreadAnalyzer collects and graphs profiling information at the thread level, showing system resource usage and the thread metrics such as time waiting for locks.

SPARCworks/TeamWare

SPARCworks/TeamWare allows the software development team to visually coordinate their efforts for maximum productivity and effectiveness. SPARCworks/TeamWare frees a development effort from geographical bounds by coordinating their efforts on the network, and allows human resources to be placed where they are needed most (see Figure 11).

Figure 11 SPARCworks/TeamWare

Tools to Support Team Development

SPARCworks/TeamWare provides capabilities for visually tracking, integrating, and managing multiple version and architecture projects including tools to coordinate simultaneous code updates into a common, integrated version. Visual tools allow developers to graphically navigate workspaces and provides for drag and drop manipulation of individual files as well as entire source code trees. Besides organizational and intuitive gains in productivity, SPARCworks/TeamWare provides capabilities that take advantage of Sun's advanced multiprocessor technology and benefit from the biggest savings in development costs — the developer's precious time and energy. SPARCworks/TeamWare tools include the CodeManager, VersionTool, FreezePoint, and ParallelMake.

CodeManager

Developing applications for multiple hardware architectures and software environments, as well as multiple versions that must be maintained over time is a challenge to development teams. CodeManager provides a rich visual environment to monitor and control these issues effectively and easily, conveying information quickly and accurately, as well as speeding the software development and maintenance efforts.

VersionTool

Teams often need to work on multiple releases of a software product simultaneously. A new version may be quickly populated and begin its own existence by starting with a current version, and progressing from it independently. Browsing allows viewing and comparing of previous versions with intuitive operations.

FreezePoint

The ability to build a previous release from history aids in maintaining previous versions without interfering with the current version. FreezePoint allows the re-creation of previous versions, creates bill of materials for a given workspace, and tracks files even if they have been renamed.

ParallelMake

Taking advantage of SunSoft's advanced multiprocessing capabilities, ParallelMake builds software faster because it can utilize multiple processors to work in parallel to compile and link programs. ParallelMake is compatible with make and MakeTool, and is customizable, specifying the maximum number of parallel jobs, and other loading parameters.

References

The Common Object Request Broker: Architecture and Specification, Object Management Group, 1994.

NEO Programming Guide, SunSoft, Inc., May 1995.

NEO Systems Management Guide, SunSoft, Inc., May 1995.

NEO Tutorial, SunSoft, Inc., May 1995.

NEO System Installation, SunSoft, Inc., May 1995.

NEO Programming Interfaces Reference, SunSoft, Inc., May 1995.

Solaris NEO Operating Environment, Product Overview, SunSoft, Inc., January 1996.

Sunsoft's NEO Product Family, Product Overview, SunSoft, Inc., January 1996.

Solaris OpenWindows: OpenWindows V3 Collection: Release Reports and White Papers, Part Number 91021-0, SunSoft Inc.

Solaris SunOS 5.0: SunOS 5.0 Multithreading and Real-Time, Part Number 91025-0, SunSoft Inc.

Solaris ONC: Design and Implementation of Transport-Independent RPC, Part Number 91028-0, SunSoft Inc.

Solaris SunOS: SunOS 5.0 Release Report, Part Number 91023-0, SunSoft Inc.

The ToolTalk Service, Part Number 91022-002, SunSoft Inc.

Introduction to the ToolTalk Service, Part Number 91031-002, SunSoft Inc.

The ToolTalk Service: An Inter-Operability Solution, Part Number ISBN 013-088717-X. SunSoft Press/Prentice Hall, Englewood Cliffs, NJ.

ToolTalk and Open Protocols: Inter-Application Communication, Part Number ISBN 013-031055-7, SunSoft Press/Prentice Hall, Englewood Cliffs, NJ (June 1993).