Yasmin: a Component Based Architecture for Software Applications

Luca Deri
IBM Zurich Research Laboratory[1]
University of Berne[2]

Abstract

Object-oriented programming (OOP) has changed significantly the way to produce software applications and allowed many problems that affected traditional programming to be overcome. Unfortunately several object-oriented frameworks misused OOP techniques, failed to address issues such as application extensibility, and have produced monolithic systems hard to manage and tailor.

This paper describes Yasmin, a new object-oriented, component-based architecture for software applications. It allows one to build applications which use system resources efficiently and which can be easily extended and configured in addition to being simple to program and to compose. This is achieved by means of droplets, a new type of software components that can be replaced and modified at runtime and by exploiting novel techniques such as collaboration and delegation.

Keywords: Object-Oriented Programming, Software Components, Application Frameworks.

1. Introduction

The popularity of object-oriented programming (OOP) has increased considerably over the past several years. OOP offers many benefits over traditional programming such as allowing programmers to define objects that can be easily extended and composed in order to build a software application. Although powerful, OOP lacked a standard framework through which software objects created by different vendors can interact with one another. The major result of this trend has been the production of a "sea of objects" that cannot interact across application boundaries in a meaningful way. At the beginning of this decade, the software industry realised that the ability to tie objects together into a closer unit would result in a much more powerful system. For this reason, many frameworks and architectures have been developed to address this problem. Unfortunately applications based on such frameworks often had a monolithic structure mostly because object-oriented techniques such as inheritance have been misused by introducing cross dependencies among classes. The obvious consequence has been that objects were so tightly coupled that even the simplest application had to link the entire system. This has been the reason for the decline or limited diffusion of highly celebrated applications systems [27] [6].

Software components [21] [17] [25] seem to be the answer to all these problems. As its name suggests, component software is based on the notion of a component, which is a reusable piece of software that can be "plugged into" other components from other vendors with relatively little effort. In contrast, traditional applications are monolithic, which means they come pre-packaged with a wide range of features, most of which cannot be removed or replaced with alternatives.

This paper presents Yasmin, a component-based, object-oriented architecture and framework for software applications. Yasmin has been developed with the intent to simplify the implementation of software applications with particular emphasis on distributed networking applications for network management.

Yasmin defines a new style of building applications, based upon established technologies such as OOP [20], software components and novel concepts like cooperation and delegation. This is part of the effort to make computer software easy to use and develop in addition to overcoming typical problems that affect network management applications such as being monolithic and hard to configure and extend. The experience gained by applying Yasmin to selected network management problems allowed us to refine and to transform it into a more general architecture for software applications.

[1] IBM Research Division, Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland. Email: lde@zurich.ibm.com, WWW: http://www.zurich.ibm.com/~lde/.
[2] Universität Bern, Institut für Informatik und angewandte Mathematik, Software Composition Group, Neubrückstrasse 10, CH-3012 Bern, Switzerland. Email: deri@iam.unibe.ch, WWW: http://iamwww.unibe.ch/~deri/.

2. Yasmin Fundamentals

This section introduces Yasmin: it defines the scope of the architecture, the basic components and how they interact from a high-level perspective.

2.1. Understanding Frameworks

Yasmin is built upon OOP abstractions known as frameworks. A framework is an extensible library of cooperating classes that make up a reusable design solution for a given problem domain. Nevertheless a framework is more than a class hierarchy: a framework is an object-oriented class hierarchy plus a built-in model which defines how the objects derived from the hierarchy interact with one other [18].

When the class hierarchy is constructed prior to running the program, the relations among object classes are static. At runtime, the objects are constructed during program execution, hence the relation is dynamic. The framework defines the architecture itself and provides both dynamic and static parts of the application.

An application framework goes beyond class hierarchies defining a preassembled generic application with a dynamic and static structure that a developer can modify to create a real application. A programmer, instead of writing a main program, instantiates objects that are part of the framework's class hierarchy and provides methods for the framework to call by refining the implementation of some classes. In this way, a developer tailors the generic framework by defining highly specialised classes and methods that are called by the framework itself. Because the developer starts with robust code, the new code is more likely to work reliably, hence reducing the testing and debugging phase.

2.2. Yasmin's Background

The idea to design Yasmin derived from several years of network management application development [7]. Despite the fact that many frameworks and architecture for building software applications are available on the market, most of them are tailored only for GUI development [19] [16] [12]. In the network management world, applications are usually built following the craftsman principle without the adoption of application frameworks. Some companies have developed huge application systems [15] that are composed of several applications and libraries that address every network management need. Although these systems are very powerful and rich in terms of functionality and tools, they do not address problems relating to application development. In fact in order to build network management applications based on those systems, developers need to know in detail many different libraries that have not always been designed to work together and that very seldom are based on OOP concepts. Additionally, due to the interdependencies among those libraries, user applications require the installation of a large subset of the application system in order to run. The natural consequence is that applications are monolithic, difficult to tailor and configure and are system resources-hungry, preventing them from running on hosts of limited power.

Beside this, network management applications quite often have to support different management protocols and object models other than being open to extensions and updates. Since network services should be available most of the time, it is necessary to identify mechanisms which allow to selectively upgrade applications while they remain partially available in order to guarantee a minimal level of service. Additionally, it is necessary to build applications in such a way that it is possible to add new pieces when new hardware devices have to be supported or when users demand new services.

Yasmin attempts to address these issues and to overcome problems that affect conventional network management applications by defining an applications framework characterised by the following properties:

1. light and simple kernel;
2. based upon pluggable software components;
3. built from the ground up on object-oriented technology;
4. founded on cooperation and delegation;
5. extensible, easy to tailor and distribute.

During the design phase, the author has analysed several frameworks and architectures available on the market, in order to verify whether it would have been possible to implement Yasmin using one of them. The result of that survey is that even established standards such as CORBA [26] are not fully suitable for this task in terms of:

- independence of the CORBA implementation being used: there is no CORBA implementation which allows one to write the code once and to deploy it on several platforms ranging from personal computers to powerful Unix workstations;
- interoperability of Yasmin-based applications: CORBA implementations are usually unable to interoperate, although this problem is supposed to be fixed in CORBA V2;
- ease of development: CORBA is rather difficult to use especially in terms of memory management and datatypes generated by the IDL compiler;
- runtime application evolution: it is not simple to develop facilities which allow the behaviour of CORBA objects to be modified at runtime.

The idea behind Yasmin is to build component-based applications that can be composed by the user, who can add or replace components at runtime. By enforcing the

component boundaries, Yasmin prevents components from making assumptions on other components, hence reducing component inter-dependencies and making them easy to reuse on different contexts [4]. Additionally, Yasmin loads the components on demand only when they are really needed and unloads them when no longer in use according to the component lifetime defined by its developer. The efficient use of system resources is quite important because it enables complex applications to run on hosts of limited computation power like mobile computers.

An effective way to limit the application size is through cooperation: every component that implements a service of general use makes it available through a well-defined interface. Often, large network management systems need to use a common set of services that sometimes require significant resources. A typical example are encoding/decoding (enc/dec) services needed to transmit information that are often replicated in different applications. As those services are often implemented using shared libraries, there is only one copy of the enc/dec logic in memory. Nevertheless this is not efficient enough because every time the enc/dec service is instantiated in the application, it needs a lot of memory for the allocation of the enc/dec tables. In this case, a single (multithreaded) enc/dec that cooperated with all the local applications would be able to serve all of them without the need to replicate the services and hence require a larger amount of memory.

3. Inside Yasmin

The following picture depicts Yasmin's components.

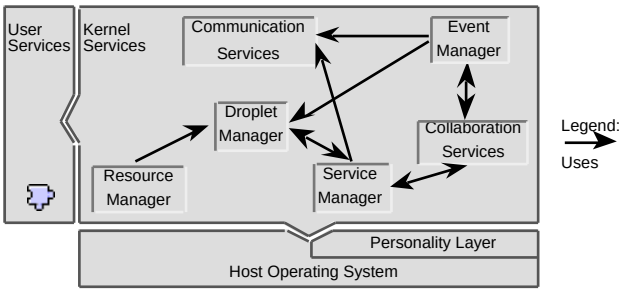

Figure 1. Yasmin's Components

The design goal of Yasmin is to define a style of building applications that make efficient use of system resources whose behaviour can be modified and tailored at runtime.

Yasmin's main characteristic is its simplicity. The kernel is very light and contains only those services that are needed by every Yasmin-based application. Every other service not widely needed is not included in the kernel. This is to prevent adding to the kernel some superfluous logic that takes up memory and disk space. The kernel being rather generic, it is possible to build very different applications that share the same architecture: the applications differ for the number and the nature of their software components[3]. Extensibility and runtime behaviour modification is realised using *droplets* [8], software components implemented using shared libraries having the following specific properties:

- they are not statically linked to the application but loaded at runtime;
- they have the ability to be replaced (i.e. a new version of the droplet can replace a previous one) at runtime while the application is running[4];
- they have a well-defined interface (droplet interface) that makes it possible to communicate with other droplets independently from the type of services provided.

Usually object-oriented frameworks provide a set of object classes that have to be subclassed in order to implement the final application. Yasmin instead uses droplets to implement the application itself. Internally, droplets can be based on different object models and frameworks, and implemented using procedural or object-oriented languages. Basically Yasmin provides the glue between different droplets and makes sure that they can interoperate and cooperate to define the application behaviour. The droplet interface guarantees that droplet internals are not exposed hence that they appear like black boxes which prevents other droplets from using specific droplet characteristics which may change in future releases or which may not be available all the time. Besides this, the droplet interface pushes droplet developers to have a clean design exporting the public services and shielding droplets from using services and resources not available through the interface. Basically developers are free to implement droplets using the tools and technologies that they prefer, leaving to Yasmin the task of ensuring application consistency.

From this perspective Yasmin can be seen as a generic architecture for compound applications because it allows one to build very different applications sharing the very same structure. Nevertheless its lack of specialisation in a specific domain, e.g. GUI, does not allow people to use it as the final solution for all their problems. This is beyond the scope of Yasmin, as its root are in the network management field where there are different standards, object models and APIs that often need to be combined in order to construct the final application [24]. This is Yasmin's problem domain: the integration of many heterogeneous programming models, sometimes deeply different, in a

[3] Despite this characteristic of Yasmin, this does not mean that it is possible to efficiently build every class of applications using a single framework since frameworks are specialized for a narrow range of applications because each model of interaction is domain-specific.

[4] In order to activate a droplet it is necessary to drop it into a certain directory monitored by the running application. Hence the term droplet.

compound object-oriented application. This is achieved through the droplet interface which prevents different application components from influencing each other and hence from creating a monolithic structure. For instance, multiple inheritance, used to combine different classes, creates interdependencies among all such classes with the result that even a minimal application has to include most of the code needed to implement the whole system. Through cooperation, different classes can be associated to perform a certain task without the limitation of creating a monolithic system.

Additionally, by using droplets the system can load on demand the parts it needs and purge them when no longer in use. This mechanism allows Yasmin-based applications to make efficient use of system resources enabling them to run in systems of limited computation power.

Finally the ability to replace droplets at runtime, hence to modify the behaviour of an application while it is running, is quite important on network management. In this field, application lifetime is very long. When an application has to be stopped for maintenance, services that rely on such application become unavailable. This is clearly a major problem when many users rely on those services, like in the case of telephone or network communications. Droplets give the chance to developers to incrementally upgrade their applications without the need to stop them hence to interrupt services that rely on such applications.

3.1. Personality Abstraction Layer

The framework kernel services are designed to run on different operating systems. For this reason it is necessary to abstract the operating system through a thin layer called *personality*. Personalities provide abstractions for low-level services performed by the host operating system. Those services include, but are not limited to:

- threads and semaphores;
- loadable shared libraries with metadata;
- interprocess communication.

If a potential operating system does not support all of these capabilities, it is often possible to run Yasmin too. For instance an application can run in single threaded mode if threads and semaphores are not supported.

The use of personalities allows to keep a single source code tree for different operating systems, whereas the personality contains operating system dependent code. This makes simple to port the framework on different operating systems and to maintain the different versions.

3.2. Droplet Manager

The droplet manager (DM) is responsible for handling droplets. Namely it:

- loads droplets on demand;
- maintains a droplet reference counter that allows droplets to be purged;
- is responsible for detecting new droplet versions or further droplets added at runtime;
- collaborates with the service manager, informing it of newly available services.

A Yasmin-based application stores droplets in a well defined directory (usually named `Droplets`). The user puts (or drops if drag-and-drop is used) droplets in such directory and is free to replace them during program execution. The DM at start-up time searches for a file named `index` which contains the name of the droplets and the services they implement. If such file is found, the DM verifies that it is newer than all the droplets (this is done by checking the file modification time) to make sure that it is consistent with the current droplet set. If not, or if there is a new droplet, the index file is rebuilt. This operation similar to the registration of the OpenDoc [1] part editors, is done by loading and unloading in sequence each droplet in order to build the list of droplets available and of the services provided by such droplets that are not visible at the file system level because they are usually stored inside the droplet itself[5].

Inside each droplet the droplet lifetime is specified. The lifetime, which ranges from one second to infinity, specifies how long the droplet has to be kept in memory since the last time it was used. This facility is used to avoid keeping droplets in memory that are no longer needed and indicate to the system when it is necessary to purge some resources in case of low memory conditions. If a droplet's lifetime has expired, the DM unloads it and releases all the memory and resources allocated by the droplet by calling the droplet termination function.

The DM is also responsible for detecting new droplets and new versions of them. In this case the `index` file is updated and the service manager (SM, covered in 3.4.), responsible for handling the services, is notified of the new services available and the ones no longer available (this happens when a new version of a certain droplet does not implement all the services implemented by the former version). Besides this, droplet versioning prevents the system from loading and using droplets which have been developed for a different application version which may introduce problems or spurious errors. Similar to the versioning system used by DSOM [11], droplets have a minor and a major version number. The major version number specifies what application version can use such droplet version, whereas the minor number it is used to implement the droplet versioning.

[5] In some systems such as MacOS™ it is possible to know the set of implemented services by using the resource manager, supposing that for each service there is a code resource that represents it.

It is worth noting that droplets cannot be unloaded when in use, whereas it is possible to have one or more different versions of the same droplet active at the same time. This technique works because the DM is the only entity that maintains direct references (i.e. pointers) to droplets, whereas other entities such as the service manager simply access services through the DM. For this reason, whenever there is a new droplet version, the DM loads it without checking whether someone is still using the old version. In case the old droplet version was no longer in use, then such a version is unloaded and the new version is loaded, otherwise the DM flips the pointer to the droplet. This operation:

- allows the new droplet to be used whenever a new request for such a droplet has to be processed;
- prevents new requests from being processed with the old droplet while the operations in progress (that make use of the old droplet) can continue;
- allows the old droplet to be unloaded whenever all the operations involving the old droplet have been completed.

The DM collaborates with the SM in order to guarantee this behaviour by keeping track of the requests currently in progress for each droplet. A request is considered "in progress" since the time the SM issues a new service request to the DM until the SM notifies the DM that such request has been completed. This mechanism works because services and droplets are accessed only through managers which shield them from the rest of the system.

The access to resources and services exclusively through managers:

- contributes to the global system robustness;
- prevents droplets from being directly dependent on each other, i.e. by means of direct function or method calls;
- allows droplets to be selectively plugged and unplugged at any time because they have no cross dependencies of any type.

3.3. Event Manager

Events are by nature asynchronous and usually indicate that something has occurred. Typical events are mouseUp/mouseDown or network events. Normally, events are processed when they occur and their type is well defined so the program can handle them. In Yasmin, these limitations are relaxed, hence:

- events can specify when they have to be processed;
- droplets can define new event types.

Yasmin represents events as information records containing the type of the event (event type), when it has occurred (event time), and additional information relative to the event itself (event info). Yasmin adds a new field to this record which specifies when the event has to be processed; it may contain an absolute time or a displacement with respect to the time the event occurred. If this field is set, the event is called a *delayed event.* A delayed event is used to implement repetitive tasks and activities that have to be performed at a certain time. Typical examples are chime clock events that have to be executed every hour or system backups that have to be performed every Sunday at 1 am when almost nobody is expected to use the system.

The Event Manager (EM) is responsible for:

- delivering events to the various components;
- handling delayed events;
- allowing different droplets and services to cooperate and interact by means of the events they exchange.

Yasmin defines a set of basic event types and allows droplets to define their own custom event types, specified inside the droplet itself. Hence, whenever a new droplet is loaded/unloaded, the DM notifies the EM about the event types that can be handled. As different droplets can handle the same event type, a string called event destination specified inside the event record is used to identify the type of the received event. Such string can have three values: a droplet name, a star ("*"), or a null value. In the first case the event is delivered to the specified droplet, in the second to all the droplets that handle such an event, in the last to a droplet that handles the event, if any. If an event cannot be handled it is discarded and the memory used by the event is freed.

Yasmin's event flexibility allows communications to be implemented between droplets easily and in a clean way. Delayed events facilitate the implementation of periodical tasks whereas custom events allow different droplets to interoperate in an appropriate way to send a specific event for a certain situation instead of using generic ones that need to be jeopardised in order to express peculiar situations. Additionally, the event destination enables droplets to implement a multi/broadcast facility which is useful when multiple droplets have to be informed of a certain event that is important for all of them (for instance a resourcePurge event which is broadcast by the system in low memory situations).

3.4. Service Manager

The Service Manager (SM) interacts with the DM to handle the services provided by the droplets. When a droplet is loaded, the DM notifies the SM of the services provided by the droplet which are made available to the whole system. When a droplet is unloaded, the DM informs the SM of the services that are no longer available. Services, identified by a unique string, can be of two types: local or

remote. A local service can be used only locally whereas a remote service can be used both locally and remotely in an RPC[6]-like way [3] by exploiting the Communication Services (CS, covered in 3.6.). The main difference between local and remote services is that for remote services, the input/output parameters are both strings (hence the service is responsible for marshalling/unmarshalling data) whereas local services can use any type they want. Each service is specified through an entry inside the droplet information record. Such entry contains the name of the service, information about the service and about the input and output parameters specified as strings using the C language convention. Remote services contain "char*" in both input and output parameters whereas local services parameter contain the real type. For instance a local service that takes as input a record containing the name and the age of a person has a service input parameter that looks like "char*, "unsigned short". Service parms are mandatory and are useful for developers whenever they want to access services provided by a droplet written by third parties[7].

When a droplet has to invoke a service request, it cannot call the service directly so the SM does this on its behalf. The droplet provides the service name and the input parameters to the SM, and it gets back from the SM the result of the service invocation or an error if the service cannot be found. This design choice derives once more from the plug 'n play principle which specifies that information access cannot be gained directly (e.g. through a pointer) but has to be mediated by the entity responsible for managing such information.

In Yasmin, service requests must be processed within a limited amount of time in order to leave processing time to other requests. This means that a service request has to terminate and it cannot last for an infinite amount of time (e.g. an endless loop). Services that may need a long amount of time in order to process requests (for instance they may need some resources not yet available) should be divided into subservices that are activated sequentially by means of events. The requirement to complete service requests in a finite amount of time is very important in single-threaded systems, because the whole system is blocked until the service has been completed. In a multi-threaded system, although the system can continue to work, long lasting services occupy resources (for instance threads) hence reduce the global application performance. The SM has no way to guarantee that services do not last for too long since programmers are responsible for this. The only way for SM to prevent application deadlock or wild resource usage, is to run a sort of garbage collector that kills the threads that are apparently in a infinite loop or that are running since too long. Although this solution is not very clean because resources in use by threads may not be freed when the thread is killed, SM has no other choice to guarantee a minimal quality of service and to prevent application deadlock.

[6] RPC stands for Remote Procedure Call.

[7] In a future version, service parms will be used to do transparent marshaling/unmarshaling, allowing just one type of service accessible both locally and remotely.

3.5. Resource Manager

The Resource Manager (RM) cooperates with other managers to use system resources efficiently. Such resources include but are not limited to memory, communication sockets, and droplets. The RM makes sure that system resources are not wasted and that it is activated periodically like a sort of garbage collector to purge resources no longer needed. The RM:

- informs the DM when droplets have expired their lifetime so they can be unloaded;
- makes sure that threads are used efficiently not starting too many threads, which would decrease the overall application performance;
- is responsible for purging memory and other system resources (including droplets) periodically or when it is required to perform a certain task and the available resources like MacApp does.

Although the RM is a 'hidden' component, it is very important because it allows the system to be run with very limited resources and prevents wasting them. For instance, Liaison, a Yasmin-based network management platform, can perform complex network management tasks using a very low amount of memory because the RM contributes to scale-down network management applications from large hosts to standard machines.

3.6. Collaboration Services

CollaBoration Services (CBS) enable droplets and services to cooperate in order to perform a certain task. CBS, exploiting SM and EM, allow a task to be broken down into many small cooperative subtasks. This solution enhances performance because these subtasks can be performed concurrently. This helps keep the complexity low because each subtask is simpler than the original task. CBS provide facilities for:

- sending requests in multicast/broadcast mode and collecting results;
- synchronising tasks by means of events.

It is worth noting that Yasmin implements collaboration not only by means of CBS but also through the SM. In fact droplets collaborate with the rest of the system by providing services that can be of general interest. This

avoids services being replicated, which saves development time and keeps the system slim and efficient.

3.7. Communication Services

Communication Services (CS) allow droplets and services to communicate with remote entities (local communications are performed via events). Since Yasmin has been designed with portability in mind, external communications are based on TCP/IP because it is a protocol supported by most platforms especially after the explosion of Internet. In view of the great diffusion of the World-Wide Web, CS includes also support for the HTTP protocol used by the Web to retrieve documents and which is becoming increasingly important even outside this context. Its popularity derives from the fact that it is a simple and efficient protocol which can be used to retrieve information other than to communicate with remote peers possibly located behind firewalls. Developers can exploit CS in order to:

- register/deregister sockets;
- be notified when data is available;
- issue HTTP requests and retrieve results.

CS is also used internally by other architecture components such as the SM which uses it to send transparently remote service requests and to receive responses. In fact an important characteristic of CS is that they allow one to send data in a reliable way and to handle transparently socket and protocol errors, shielding droplets from differences among socket implementations available on various platforms.

4. Yasmin at Work

Yasmin has been designed as an open and general architecture for software applications. Although the base principles of Yasmin have been applied to different fields such as a system for rapid GUI application development and a C++-like interpreter for the manipulation of distributed objects [2], Yasmin has been designed with particular emphasis on distributed networking applications for network management. This section covers the implementation of a Yasmin-based application for multidomain network management and shows how the adoption of Yasmin simplified the design of the application and allowed multiple network management protocols to be supported in a seamless way.

4.1. Liaison: a Yasmin Based software application

Liaison [9] is a proxy application [22] [13] which allows users to manage network resources through the Web using CMIP [5] and SNMP [23], the two dominant protocols for network management. Issues encountered during the design and development of Liaison are mostly related to:

- the great difference between CMIP and SNMP, other than their complexity;
- the need to leave room in order to accommodate future network management protocols/services and to modify the existing ones at runtime;
- the ability to run Liaison on hosts of limited computing power and under different operating systems.

Following Yasmin's principles, Liaison has been divided into two parts: general and protocol-related services. General services constitute the kernel of Liaison and are necessary for an application's execution. They include Yasmin's kernel services and additional ones such as HTTP services that are necessary to allow Web browsers to interact with Liaison, this being the only way for clients to interact with the application.

Protocol-related services, stored inside droplets, implement basic CMIP/SNMP services besides other facilities needed to allow users to manage network resources easily. The great difference between CMIP and SNMP prevented them from having a shared set of services shared by both protocols. Nevertheless the dynamic nature of droplets allowed services to be loaded on demand and to unload them when no longer needed. This has contributed a small application memory footprint and to avoiding the need to mix different protocols, for instance by means of object-oriented techniques such as inheritance. This design solution allowed Liaison's core part to be totally independent from the management protocols and simplified the integration of additional services and protocols that will be implemented using additional droplets.

Yasmin's collaboration services allowed us to reduce the amount of code needed to implement Liaison. Instead of implementing CMIP and SNMP separately, the few services shared by the two protocols have been implemented in a droplet in order to be used by both of them. This allowed us to avoid code duplication and also prevented these services that may not be needed by other protocols from being included statically into Liaison.

Yasmin's event services have been used to handle asynchronous protocol events. Network resources may emit asynchronous notifications when they change state or when a special situation arises. Management applications usually perform some actions prior to receipt of such notifications. Liaison allows droplets to register event handlers that are called when a notification is received. The droplet responsible for receiving notifications posts an event containing the notification itself. Event services then process the event by invoking the corresponding event handlers, if any. This mechanism allows asynchro-

nous events to be handled properly, independently of the type of event and the protocol used to receive it.

The usage of personalities contributed to obtaining a highly portable application with almost no effort. Every platform specific functionality such as thread and shared library management has been encapsulated inside the personality without affecting most of the Liaison's code. This solution allowed Liaison to be ported over different platforms with very little effort.

Liaison has been used to validate Yasmin and to apply the architecture to a real case with strong requirements in terms of resource utilisation and problem complexity. The use of Yasmin has significantly contributed to creating a highly extensible and customisable application, which makes moderate use of system resources and which allows network management resources to be managed easily from the Web.

5. Final Remarks

This paper presented a new object-oriented, component-based architecture for software applications called Yasmin. Its main characteristics are that it is:

- highly portable, configurable and extensible;
- built upon dynamic software components called droplets;
- founded on cooperation and delegation, used to glue components together;
- a slim and efficient kernel which relies on a fast event handler;
- an efficient use of system resources, which enable Yasmin-based applications to run on environments of limited computing power.

Despite Yasmin's native ability to work in a networked environment, this architecture is general enough to be used not only on this applications field. So far it has been successfully applied to break large monolithic applications and to create a new class of web-based applications based on Liaison that integrate heterogeneous network management object models in a single homogeneous application [10]. Finally, the ability to modify and extend its applications at runtime, makes it attractive in dynamic environments where new services and resources need to be supported while the original application must remain active and ready to serve incoming requests.

6. Acknowledgements

The author would like to thank Dr. Wolfgang Kleinöder, Prof. Oscar Nierstrasz and the users and early adopters of Liaison[8] who have greatly contributed with all their comments and suggestions.

References

[1] Apple Computer, *Components Made Easy*, OpenDoc Technical White Paper, March 1995.

[2] B. Ban, *A Generic Management Model for CORBA, CMIP and SNMP*, PhD thesis, University of Zurich, Institut für Informatik, 1997.

[3] A. Birrell and B. Nelson, Implementing Remote Procedure Calls, *ACM Transactions on Computer Systems*, Vol. 2, February 1984.

[4] D. J. Chen and D. T.K. Chen, An Experimental Study of Using Reusable Software Design Frameworks to Achieve Software Reuse, *Journal of Object-Oriented Programming Languages*, May 1994.

[5] International Standards Organization, Information Technology - OSI, *Common Management Information Protocol (CMIP) - Part 1: Specification*, CCITT Recommendation X.711, ISO/IEC 9596-1, 1991.

[6] Cotter and M. Potel, *Inside Taligent Technology*, Addison-Wesley, ISBN 0-201-40970-4, 1995.

[7] L. Deri and E. Mattei, An Object-Oriented Approach to the Implementation of OSI Management, *Computer Networks and ISDN Systems*, Vol. 27, 1995.

[8] L. Deri, *Droplets: Breaking Monolithic Applications Apart*, IBM Research Report RZ 2799, September 1995.

[9] L. Deri, *Network Management for the 90s*, Proceedings of ECOOP '96 Workshop on System and Network Management, Linz, Austria, July 1996.

[10] L. Deri, *Surfin' Network Management Applications Across the Web*, Proceedings of 2nd Int. IEEE Workshop on System and Network Management, Toronto, Canada, June 1996.

[11] IBM Corporation, *DSOM Development Toolkit*, October 1994.

[12] A. Weinand, E. Gamma and R. Marty, *ET++: An object-oriented application framework in C++*, ACM OOPSLA '88 Conference Proceedings, San Diego, CA, September 1988.

[13] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[14] T. Berners-Lee, R. Fielding and H. Nielsen, *HyperText Transfer Protocol - HTTP/1.0*, Internet Draft, October 1995.

[15] IBM Corporation, *IBM TMN Products for AIX: Gen-*

[8] A version available for public download can be found at http://misa.zurich.ibm.com/Webbin/.

eral Information, Release 2, GC 31-8016-00, March 1996.

[16] M. Linton and P. Calder, *The design and implementation of Interviews*, Proceeding of USENIX C++ Workshop, Santa Fe, NM, November 1987.

[17] A. Joch, Killer Components, *Byte Magazine*, January 1996.

[18] T. Lewis et al., *Object Oriented Applications Frameworks*, Manning Publications, ISBN 0-13-213984-7, 1995.

[19] Apple Computer Inc., *MacApp 2.0: Programmer's Guide*, 1989.

[20] B. Meyer, O*bject Oriented Software Construction*, Prentice Hall, Englewood Cliffs, NY, 1988.

[21] O. Nierstrasz, S. Gibbs and D. Tsichritzis, Component-Oriented Software Development, *Communications of the ACM*, 35(9), September 1992.

[22] M. Shapiro, *Structure and Encapsulation in Distributed Systems: the Proxy Principle*, 6th Int. Conference on Distributed Computing Systems, Boston, Mass., May 1986.

[23] J. Case, M. Fedor, M. Schoffstall and C. Davin, *Simple Network Management Protocol (SNMP)*, RFC 1157, May 1990.

[24] D. Tsichritzis, *Object-Oriented Development for Open Systems*, Proceedings of IFIP '89, North-Holland, San Francisco, August 1989.

[25] J. Udell, ComponentWare, *Byte Magazine*, May 1994.

[26] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, July 1995.

[27] Symantec Corporation, *Bedrock Architecture*, 1993.