An Integrated Approach to Satisfy Application Requirements

J. Becher, G. M. Solvie
University of Karlsruhe, Institute of Telematics
Karlsruhe, Germany

Abstract
The requirements of distributed applications concerning configuration and communication vary greatly. Therefore, flexible architectures are needed which can satisfy these varying requirements. ARCADE and XOSI offer such a flexibility. The integration of both architectures is described in this paper.

1 Introduction
In the future the number of distributed applications executed in a network of autonomous hosts or workstations will grow. Besides that, increasing costs for software development will lead to an extensive reuse of existing software. This means, that larger applications will not be coded using a single programming language, but will consist of components realized in different programming languages and with different programming paradigms. The realization of applications using different languages and paradigms is called mixed-language programming, the resulting applications mixed-language applications.

Distributed applications consist of autonomous, cooperating components. At run-time, the distribution of the components as well as interactions between them have to be supervised and controlled. These aspects of an application are called the configuration whereas an entity for supervising and controlling the configuration is called a configuration manager.

ARCADE [1] realizes a configuration management system for distributed, mixed-language applications. A prototype configuration language, COLA, has been realized. In contrast to other configuration management systems, ARCADE strictly separates configuration from implementation aspects and is open ended concerning supported programming languages, configuration management strategies, used communication architectures, supported transparency mechanisms, and operating systems.

Communication is an important part of distributed applications. Depending on the application and its configuration the communication requirements may vary. Therefore distributed applications require a flexible communication system, especially those with varying configurations.

The eXtended OSI architecture, XOSI, defines a flexible communication system architecture based on the OSI architecture. The main difference between the conventional OSI architecture and XOSI is the additional flexibility of XOSI concerning application support and the transport system used. Because of its flexibility, a communication system based on the XOSI architecture can offer any services required. Services conforming to OSI may be offered, but also future communication requirements (e.g., multimedia applications) will be satisfied [3].

Since a great number of transport systems are available today, XOSI is independent of the transport system. Therefore it may be used on any host where application components are located.

ARCADE and XOSI have been developed independently. The flexibility of both architectures, however, allows easy integration thus leading to a powerful system providing both, configuration management and flexible communication. In this paper, we describe the integrated approach of ARCADE and XOSI.

2 Model
In ARCADE each application is constructed from autonomous entities, the so-called application components. Application components are connected to each other by interaction points which represent communication and transparency of an interaction. This makes application development easier, because an application programmer can concentrate on either functionality or transparency and communication and does not need to mix up these different concepts.

Analogously to the object oriented approach, application components and interaction points are grouped into classes. New application components and interaction points are created by instantiating them from a class.

2.1 Application Components
Application components represent the functionality of an application. Analogously to the object based approach, each application component offers a set of services and perhaps requires a set of other services to perform well. We call these services exported and imported services. Exported services can be accessed by receiver interfaces. Imported services are represented by sender interfaces.

An interface specification in COLA consists of the name of the interface and optionally transparency requirements. For each interface, several required transparencies can be specified. The current realization of ARCADE, however, only supports access transparency, which is defined by a specification of the exchanged data types and the way of the data exchange (e.g., call-by-value). Examples are shown later in section 4.

2.2 Interaction Points
Interaction points represent the communication and transparency requirements of interactions. The advantage of this approach is, that changing application requirements can easily be satisfied by using the proper interaction point. An example will illustrate this:

In an application a component A calls over the interface A-call an offered service DB-query of a database component DB to make queries to the database. The application programmer specifies the connection between A-call and DB-query as follows:

\[
\text{links A.A-call to DB.DB-query via IP-safe; endlinks}
\]

This means, that the runtime system creates an interaction point from the class IP-safe and links the two interfaces of A and DB to it. If IP-safe provides safe communication and the application programmer later wants fast communication rather than a safe one he can simply instantiate the interaction point from another class of interaction points which provides the required communication. Therefore, the configuration can easily be adapted to changing application requirements without affecting the application.

The communication requirements can be specified when the class of an interaction point is defined. If the specification of the communication requirements is omitted, either
3 Architecture

3.1 ARCADE

In this section, we shortly describe the relevant components of the ARCADE runtime system. A more detailed description can be found in [1].

The major concept of the ARCADE runtime system is, that it is open ended with regard to supported programming languages, configuration management strategies, used communication architectures, supported transparencies, and operating system details. This is achieved by separating these aspects and encapsulating them in components representing the functionality of a configuration management system, the so-called metacomponents. The ARCADE runtime system consists of a global, distributed configuration service based on configuration service entities. The configuration service provides means for communication and data exchange among the metacomponents (see figure 1).

**Figure 1: ARCADE runtime system.**

*Interface Generators* derive language independent interface descriptions from the declarations of the services in the respective programming language. Interface generators therefore parse the source-code of the implementation of an application component and generate suitable interface definitions in COLA.

*Preparation Components* take a set of application components and prepare so-called interpretation units which are processed by virtual machines (see below). Interpretation units are the analogon to programs which can be executed as a process. Interpretation units, however, are an abstraction of the process concept to be independent from the underlying system.

*Virtual Machines* process interpretation units. They provide services for initialisation and migration of application components and for processing of interpretation units. This defined interface guarantees the independency of the ARCADE runtime system from supported operating systems.

*Interconnection Managers* realise and manage interactions between application components. They instantiate and initialise the appropriate interaction points and provide any information the interaction points need to set up a connection between the interacting application components. Because an interaction point provides means for transparency and communication, an interconnection manager generally consists of two parts, a transparency manager to set up the desired transparency and a communication manager to provide the required communication support.

*Configuration Managers* control the actions of all other metacomponents. They coordinate the metacomponents and realise the desired configuration of application components. Configuration Managers can be either those provided by ARCADE or those of programming languages with integrated configuration support. Each component suitable to handle the determined protocol of configuration managers can be used.

*Initiation Agencies* initiate the execution of an application and determine the configuration. An example for an initiation agency is the user which starts the application, but a program can also be used as an initiation agency. It is noteworthy, that each component processing the service primitives of the configuration service in a correct manner can be used to substitute the component provided by the system. Therefore, components providing special transparency mechanisms or special communication support can be added by building an interaction manager. New programming languages can be supported by providing the appropriate interfaces generating configuration components and new configuration management strategies by providing configuration managers.

3.2 XOSI

Communication systems may be divided into two parts, a transport oriented part responsible for reliable transfer of data (in OSI communication systems layers 1 - 4) and an application oriented part satisfying specific application requirements (in OSI communication systems layers 5 - 7), e.g. access to a remote file.

In the extended OSI architecture flexible application support is realised by configuration of services and protocols in the application oriented part of the communication system. The services are offered by components called autonomous service-elements. Each autonomous service-element uses a protocol to realise the services offered. The services required for transfer of the associated protocol data units (PDU) of an autonomous service-element must be offered by other autonomous service-elements or the transport system. The set of services offered to the application is a composition of the services offered by the autonomous service-elements.

There are three different kinds of service-elements defined in the extended OSI architecture—OSI, non-OSI, and service-converter autonomous service-elements. OSI autonomous service-elements use OSI conforming protocol to offer OSI conforming services. Use of these autonomous service-elements has the advantage of being completely OSI conforming and therefore compatible with future standard communication systems. But, already existing applications using communication services of existing communication systems will not be satisfied by OSI services. Therefore non-OSI autonomous service-elements offer appropriate non-OSI communication services, using non-OSI protocols (e.g. SNA services and protocols).

All services and protocols specified today are inappropriate to satisfy future application requirements completely (e.g. multimedia document transfer). Therefore new services and protocols may be added to XOSI communication systems by realising new autonomous service-elements. All autonomous service-elements should be combined arbitrarily. If autonomous service-elements use services of other autonomous service-elements originating from the same communication system, e.g. OSI, no problems result. But, if services of a service-element of another communication system are used, the number and format of parameters may vary. Therefore service-converter autonomous service-elements are to be defined offering a mapping of services of different autonomous service-elements, where necessary.

In favor of maximum flexibility the autonomous service-elements must be independent of each other. Therefore another component is needed to control the coordination of their use, the coordinator. The coordinator contains the status object managing all information that may be
accessed or manipulated by more than one autonomous service-element. An example of that information is the presence and ownership of a token (session layer) and the state of a connection. The main task of the coordinator is routing of events—service primitives and PDUs—to the appropriate autonomous service-elements, the application, and the transport system. This routing is done according to specific rules specifying the destination of each event and the allowable sequence of use of events.

The configuration of autonomous service-elements—consisting of the set of autonomous service-elements used and the rules specifying their coordination—depends on the services required. To offer exactly the services required, the application must specify the requirements in a formal specification language. For XOSI communication systems the specification language LARS was defined. In LARS application requirements are specified using keywords for specific communication services. For each service attributes may be used to specify options of the required service, e.g. confirmed service or minimum length of user data.

Selection of suitable autonomous service-elements satisfying the application requirements specified with LARS is the task of the contextmanager. It also specifies rules for coordination of the selected autonomous service-elements, which are controlled by the coordinator. Figure 2 gives an overview of the architecture and its components.

![Figure 2: The eXtended OSI architecture.](image)

The specification of the selected autonomous service-elements and the rules for their coordination is called a context. A separate context is specified for each connection (association).

In the extended OSI architecture a context is composed hierarchically. First, the autonomous service-elements selected by the contextmanager with respect to the application requirements and rules for their coordination belong to the context of the topmost connection. In a next step the context is extended by those autonomous service-elements which are selected by the contextmanager to satisfy the communication requirements of the autonomous service-elements selected in the first step, and rules for coordination of these. If a new connection is required then a new context is specified, similar to that of the connection above it. This process of context composition continues until all requirements of selected autonomous service-elements may be reduced to the services offered by the transport system. Here, any transport system may be used, e.g. OSI and TCP.

### 3.3 An Integrated Approach

The integration of XOSI and ARCADE offers flexible communication as well as configuration support to distributed applications. Though the two architectures have been de-
A specified. In the example, only one integer parameter (the flight number) is transferred. When the application is started, the configuration manager asks the interaction manager to establish the interaction between the two components. First, the transparency manager generates components realizing access transparency. These transparency components require two communication services, one to establish an association and another to invoke remote operations. The interaction manager specifies those requirements in LARS. Here a connection oriented service for initiation of an arbitrary number of remote operations is required. The requirements specification for this example is given in figure 4.

In this example, a connection oriented service is specified by the two keywords a-BEGIN and a-END, meaning association establishment and association release respectively. After an association has been established the process OP specifies further services allowed. When process OP exits the only service allowed is a-END. Since an association may be aborted anytime, a possible interrupt a-ABORT is specified with the interrupt operator >].

The process of service OP allows the initiation of a remote operation (operation) which returns information in case of successful remote operation initiation as well as in case of an error occurring during procedure initiation (succ.err). If a remote operation has been initiated the procedure OP is called recursively. But, no remote operation must be initiated because the choice operator [] allows exiting the procedure (exit) alternatively.

At runtime the interaction manager calls its communication manager—which is the context manager of XOSI— with the specification of the components realizing access transparency. The context manager selects the appropriate autonomous service elements according to the requirements specification. The only difference is that remote operations may be initiated only when an association has been established (tA-BEGIN) after that. After any number of remote operations have been initiated within a transaction, a service completing the transaction (tA-END) must be used. If no operation has been initiated within a transaction, a service must be executed which allows returning to the pre-process state.

In this paper an integrated approach of two flexible architectures satisfying application requirements for configuration and communication was presented. We illustrated that the integration process is quite easy. This is a result of the flexibility of both architectures. The integrated approach satisfies all requirements of existing distributed applications. Besides, the requirements of future applications can also be satisfied because of the extensibility of both architectures.

5 Conclusion

In this paper an integrated approach of two flexible architectures satisfying application requirements for configuration and communication was presented. We illustrated that the integration process is quite easy. This is a result of the flexibility of both architectures.

The integrated approach satisfies all requirements of existing distributed applications. Besides, the requirements of future applications can also be satisfied because of the extensibility of both architectures.

References

