



**ITEA**  
Information Technology for European Advancement



**Opening mobile platforms for the development  
of component-based applications  
(VIVIAN – ITEA 99040)**

***System Architecture (v0.3)***

This document will be treated in strict confidentiality.  
It will be seen only by persons who have signed the Declaration of non-Disclosure  
(see [http://www.itea-office.org/button](http://www.itea-office.org/button/documents) “documents”).

*Edited by Titos Saridakis, July 2002*



# TABLE OF CONTENTS

<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
1.1	Background	1
1.2	VIVIAN System's Environment	2
1.3	Structure of the Document	3
<b>2</b>	<b>VIVIAN SYSTEM MODEL</b>	<b>4</b>
2.1	VIVIAN Component	4
2.2	VIVIAN-enabled Terminal	4
<b>3</b>	<b>VIVIAN ARCHITECTURE: STAGE 0</b>	<b>7</b>
3.1	Interoperability Core, Stage 0: IC0	7
3.2	Mapping IC0 to CORBA	9
3.3	Mapping IC0 to wireless CORBA	11
3.4	Interoperability Issues	12
3.5	The Gateway	13
3.5.1	Generic Gateway	13
3.5.2	Proxy-style Gateway	15
3.6	Platform Services, Stage 0: PS0	16
<b>4</b>	<b>CONCLUSION</b>	<b>18</b>
<b>5</b>	<b>REFERENCES</b>	<b>19</b>

## **CONTRIBUTORS**

- Titos Saridakis, NOKIA ([titos.saridakis@nokia.com](mailto:titos.saridakis@nokia.com))
- Harm Smit, PALMWARE ([harmsmit@altavista.net](mailto:harmsmit@altavista.net))
- Tom Suters, PHILIPS ([tom.suters@philips.com](mailto:tom.suters@philips.com))

## 1 Introduction

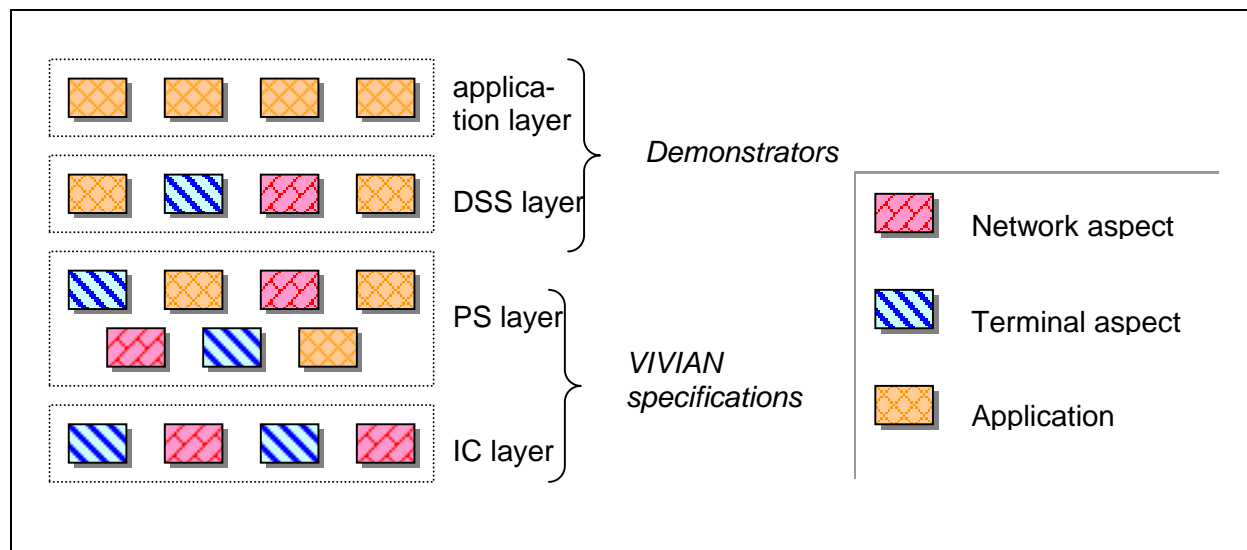
The aim of VIVIAN is to deliver a component-based middleware for component-based mobile applications. This document aims at identifying identify and describing the system architecture of the VIVIAN platform.

The purpose of this document is two-fold:

1. To serve as a reference for the developers of the VIVIAN platform and the services for it.
2. To be a deliverable report for ITEA.

### 1.1 Background

The requirements analysis in [4] has revealed a conceptual structure of VIVIAN consisting of 4 layers (see Figure 1): the interoperability core (IC), the platform services (PS), the domain specific services (DSS), and the applications.



**Figure 1** The four abstraction layers and the three architectural aspects of VIVIAN architecture.

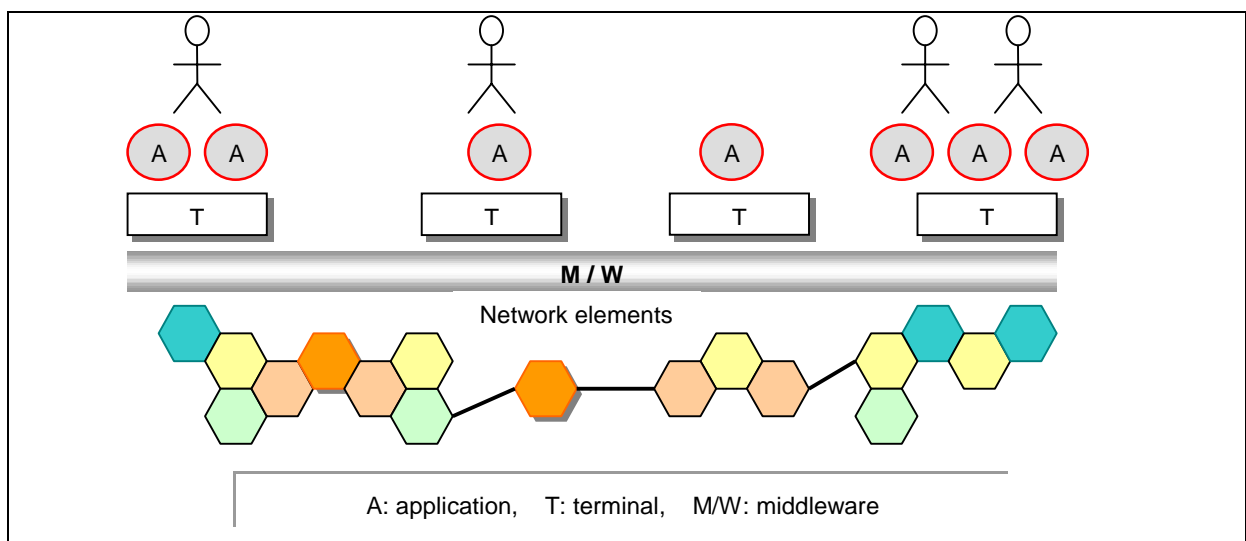
The IC layer contains the basic operations regarding the establishment of a network connection and the execution of a software component (e.g. addressing, messaging, and process/thread creation, etc). The PS layer is further divided into two groups: the *interoperability services* (IS) and the *generic services* (GS). The IS group contains operations that make easier the access to the IC layer and includes functionalities like naming, component registry, event handling, component installation, etc. The GS group contains services which are independent of application domains and which make easier the programming of the VIVIAN platform using the IC and IS functionalities. The functionalities provided by the GS group are persistent storage, atomic transactions, terminal profiling, etc. The DSS layer includes functionalities that are specific to certain application domains, e.g. GIS services, mCRM services, CSCW services, etc. Finally, the application layer contains application components. In general, application components may use more than one DSS layer but the VIVIAN platform does not guarantee the availability of any DSS components. It is left to the responsibility of the application developer to verify that the domain specific services that an application requires exist in a given instantiation of the VIVIAN platform.

To map these conceptual layers of VIVIAN functionalities to system architecture the project consortium has decided to follow an incremental architecting approach. The VIVIAN system architecture will be developed in 3 stages as described in the following:

- **Stage 0** provides a version of the VIVIAN platform which supports only remote access, i.e. interactions among components distributed on different VIVIAN-enabled terminals. In this stage the IC layer consists of the services that provide addressing and messaging functionalities which includes the mapping of addresses and the message format used by the application components to the addresses and the message format recognized by the actual transport protocol. The PS layer consists of the naming, service discovery and event handling services in the IS group and persistent storage, remote UI and connectivity management services in the GS group.
- **Stage 1** provides VIVIAN-enabled terminals with component hosting and execution capabilities. In this stage the IC layer is extended with process and memory management functionalities, the PS layer is extended with component installation, component version control, access control, communication encryption and user authentication services. Finally, the DSS and application layers focus on the application domains of GIS and mCRM.
- **Stage 2** will transform the VIVIAN platform into an agent-enabled system. The current understanding of the services and the functionalities provided at the different architectural layers by Stage2 is not clear yet, but the intention is to have Stage 2 delivering the final version of the VIVIAN platform.

## 1.2 VIVIAN System's Environment

From the user standpoint the VIVIAN context is a system of applications which run on mobile terminals (e.g. NOKIA 9210, PSION netBook, COMPAQ iPaq, etc) and which interact with other applications running either on mobile terminals or on terminal servers (e.g. any computer that plays the role of an internet-based Web server). Hence, the goal of the system architecture is to define a mapping of the 4-layer conceptual structure given above to the context like the one depicted in Figure 2 (borrowed from [4]).



**Figure 2** Physical context in which the 4-layer conceptual structure of VIVIAN has to map.

### 1.3 Structure of the Document

The remainder of this document is structured as follows: in section 2 we describe the system model that VIVIAN follows. The presented notions of component, interface, service are mostly borrowed from the ROBOCOP project and adjusted to the needs of VIVIAN. In addition to these, the VIVIAN-enabled terminal is explained in section 2.

Section 3 contains the main contribution of this report, the VIVIAN architecture in Stage 0, the architecture which is preoccupied with remote communication issues. This section elaborates on the structure and the layers of the VIVIAN architecture and provides details about how VIVIAN architecture Stage 0 can be mapped onto CORBA and wireless CORBA. The issues related to the interoperability between CORBA-based VIVIAN-enabled terminals and SOAP-based services and vice versa are also addressed in this section.

The report concludes in section 4 with a summary of the presented material and a reference to the work that will be covered in subsequent stages of the VIVIAN architecture.

## 2 VIVIAN System Model

The purpose of this section is to describe the physical and conceptual entities in the VIVIAN system model and present the associated terminology which is used in the remainder of this document.

### 2.1 VIVIAN Component

The concept of a component is the corner-stone of the VIVIAN project. Yet, the term component has been abusively used in the related literature with various meanings, sometimes closely related and sometime radically different. Our goal not being to provide a new definition of the term component, we base the VIVIAN system model on two related concepts: the general component and the VIVIAN component.

A *general component* is a coherent package of software artifacts that can be independently developed and delivered as a unit and that can be composed, unchanged, with other general components to build something larger. A general component may include a list of provided interfaces, a list of required interfaces, the external specification (i.e. the description of the general component's behavior as observed through accessing its interfaces), the executable code, the validation code, the source code, the documentation material, etc. When it does not produce any ambiguity in the context of its employment, the shorter term "component" is used to refer to general component.

An *interface* is a description of a set of operations related to the external specification of a component, i.e. a set of operations that a component needs to access in its surrounding environment (required interface), or a set of operations that the surrounding environment can access on the given component (provided interface).

An *operation* is unit of functionality implemented by a component. It usually maps to a method, a function or a procedure, depending on the nature of the programming language used for the implementation of the component.

A *service* is an abstraction of a comprehensive functionality that refers to the same purpose of use (e.g. a printing service embraces functionalities related to printing, an event service embraces functionalities related to the subscription of components' interest into specific events and their notification when these events occur). A service is represented as a set of interfaces and can be accessed through these interfaces.

A *VIVIAN component* is a subset of a general component centered around the executable code, which can interact with its environment (other VIVIAN components or the executable code of non-VIVIAN components) using the VIVIAN middleware. As part of the properties it inherits from the general component, a VIVIAN component may have a number of (provided and required) interfaces associated to it, through which it can access services offered by its surrounding environment and vice versa.

A *VIVIAN service* is a service specific to the VIVIAN middleware and it is defined by the VIVIAN project. A VIVIAN service is provided by a VIVIAN component.

### 2.2 VIVIAN-enabled Terminal

The VIVIAN middleware aims to support the seamless interaction of application components that reside on VIVIAN-enabled terminals. The term *VIVIAN-enabled*

*terminal* is used to signify the combination of a hardware device (handheld or workstation) plus the accompanying firmware and OS plus the VIVIAN middleware components. Application components on a VIVIAN-enabled terminal can *potentially* interact with applications components on remote VIVIAN-enabled terminals. The interaction must be possible when the following conditions are met:

- A network connection can be established between the two VIVIAN-enabled terminals. This means that:
  - In a point-to-point network configuration, the two terminals must use the same network bearer and protocol.
  - In a network configuration where a number of machines can be interposed between the two interacting VIVIAN-enabled terminals, a sequence of pairwise feasible network connections must exist among these machines (e.g. consider the interaction of terminals 1 and 4 in Figure 3).
- The VIVIAN middleware on each terminal can meaningfully communicate with its counterpart on the other terminal. This means that:
  - In a point-to-point network configuration, the VIVIAN middleware on both terminals is an instance of the VIVIAN platform type (see §3 for more information on VIVIAN platform types).
  - In a network configuration where a number of machines can be interposed between the two interacting VIVIAN-enabled terminals, the two terminals may have VIVIAN middleware of different type, provided that an appropriate *gateway* is placed between them. The role of the gateway is to bridge the two different types of VIVIAN middleware (see § for more information).

**N.B.:** According to the above definition of a VIVIAN-enabled terminal, it is possible that two VIVIAN-enabled terminals are not able to communicate, e.g. because of the absence of an appropriate gateway (e.g. consider the communication between terminals 1 and 4 in Figure 3, if terminal 1 has a CORBA-based VIVIAN platform and terminal 4 has a SOAP-based VIVIAN platform and there is no CORBA-SOAP gateway on terminals 2 and 3).

Besides the communication with other VIVIAN-enabled terminals, a VIVIAN-enabled terminal can access services on machines that are not VIVIAN-enabled, provided that:

- Network connection to these machines is possible.
- The services on the remote machines are accessible over the middleware used as the base for the VIVIAN-platform of the VIVIAN-enabled terminal.

For example, a VIVIAN-enabled terminal with a CORBA-based VIVIAN platform can access CORBA services on a server which has no VIVIAN specific software. More information on this topic is provided in the following section.

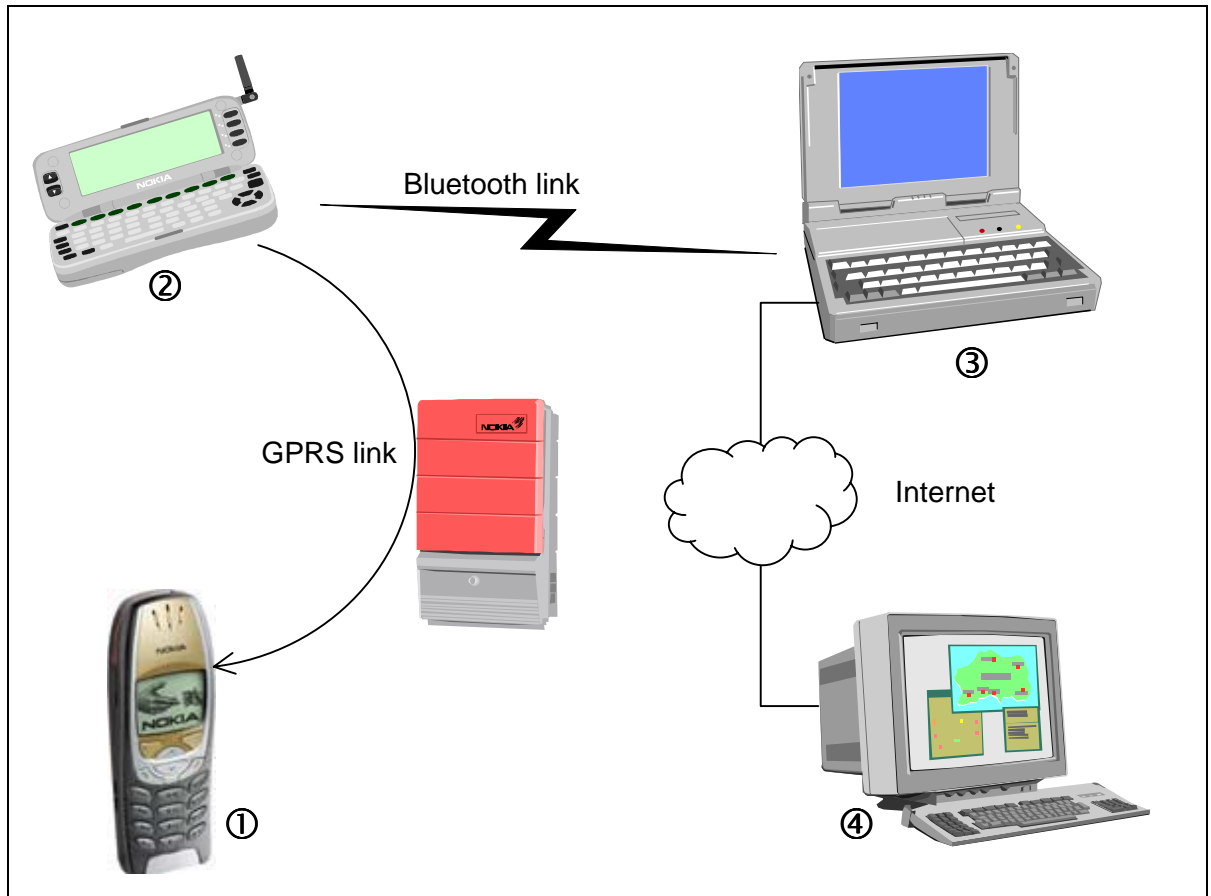


Figure 3 Interactions among VIVIAN-enabled terminals.

### 3 VIVIAN Architecture: Stage 0

This section defines the VIVIAN architecture independently from any middleware which can be used for implementing it. By mapping the VIVIAN architecture to implementations on different middleware architectures (e.g. CORBA, SOAP-based, etc) we obtain different VIVIAN platform *types*. Hence, a VIVIAN platform type refers to the mapping of the VIVIAN architecture to a specific middleware.

The purpose of VIVIAN architecture Stage 0 is to provide support for remote access, i.e. accessing from a VIVIAN-enabled terminal a service on another VIVIAN-enabled terminal or a server without any VIVIAN specific software which provides the given service over a suitable middleware.

This section describes the Stage 0 of the VIVIAN system architecture, focusing primarily on the interoperability core (IC), i.e. the part of the VIVIAN platform which is responsible for masking the diversity of parameters of the environment in which the VIVIAN platform operates. A mention about interoperability issues and a discussion about the role of the gateway completes the basis of the VIVIAN architecture. The section concludes with a rough outline of the Platform Services (PSs) in Stage 0 of the VIVIAN architecture, leaving the exact specification of these services and their APIs for later.

#### 3.1 Interoperability Core, Stage 0: IC0

Given that Stage 0 of the VIVIAN architecture is aimed at providing remote access of services, the core of the architecture is a software bus which ensures:

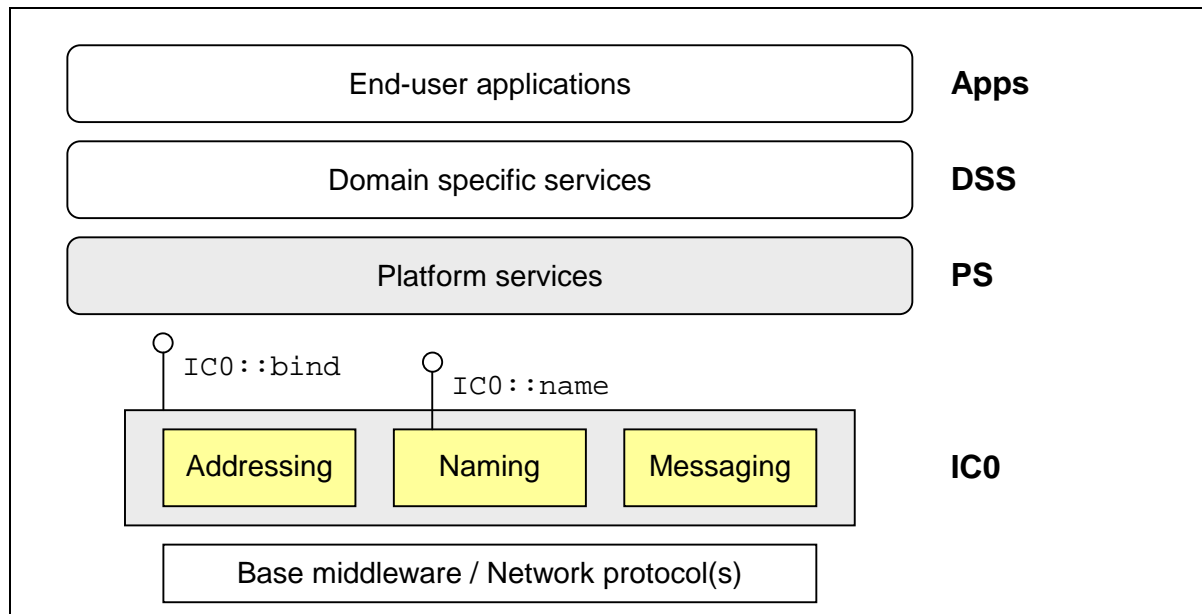
- the addressing of remote services (addressing),
- the resolution of the human-readable service names to such addresses (naming), and
- the communication of requests and replies to and from the remote services (messaging).

*Messaging*, in its broadest sense, refers to the creation, storage, exchange, and management of text, images, voice, telex, fax, e-mail, paging, and Electronic Data Interchange (EDI) over a communications network. In programming, messaging is the exchange of messages (specially-formatted data describing events, requests, and replies) to a messaging server, which acts as a message exchange program for client programs. In VIVIAN, messaging is the preparation (formatting) of service requests and replies exchanged between VIVIAN components. This message formatting must be independent of the underlying carrier and its associated protocols, i.e. the VIVIAN components must be able to prepare their messages always in the same way, independently whether these messages will be sent over a Bluetooth, WLAN, GPRS or Ethernet link and over L2CAP, PPP or TCP/IP.

*Addressing* in networking refers to the identification of a remote party (e.g. a node within a network, a process within a node, a thread within a process, etc) and the establishment of a communication link with it. For the purposes of identification, a unique identifier called an *address* is assign to every entity of interest (e.g. node, processes, thread, etc). In the VIVIAN context and at the IC level, addressable entities are the VIVIAN components, their interfaces and the operations within these interfaces. Services are not addressable entities at the IC level, but they are addressable entities at the PS level, i.e. a VIVIAN component at the PS level or above it (DSS level and application level) can request connection to a service only

by naming the service itself. It is the responsibility of the VIVIAN services related to naming, trading and service discovery to locate the component(s) offering the requested services provide the corresponding address(es) for connection establishment at the IC level.

Figure 4 captures graphically the essence of IC0.



**Figure 4** A graphical illustration of IC0 (Interoperability core in VIVIAN architecture, Stage 0)

In Figure 4 IC0 is placed right above the network and base middleware layer and it forms the support for the platform services (PSs) which subsequently support the domain specific services (DSSs) and the application layer.

IC0 provides two interfaces, the `IC0::bind` and `IC0::name` respectively. The former interface allows to a software component to bind to the software bus provided by IC0. Once bound to IC0, a component can access service on other components also bound on IC0. Vice versa, when a component is bound to IC0, the services this component provides can be accessed by other components also bound to IC0. The latter interface allows components bound to IC0 to register their services with the naming service provided by IC0. It also allows components bound to IC0 to lookup for the address of specific services which have already been registered with the naming service (i.e. which are available and can be contacted over IC0).

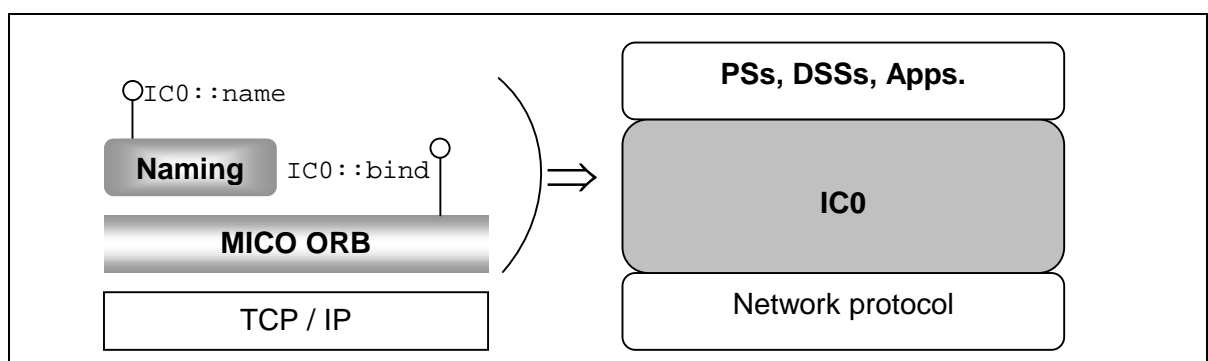
**N.B.:** Despite the apparent layers architecture in Figure 4, the aforementioned two interfaces provided by IC0 can be accessed not only by the components in the PSs layer but also by the components in the DSSs and the application layers.

Both the interfaces provided by IC0 must be represented by well-known entry points in the system, i.e. components that will use them do not have to query any mechanism to obtain reference to them. In practice this means that the application developer will have to insert two local method calls (one for each interface) in order to bind his application to IC0 and to (activate and) connect to the naming functionality of the IC0. The following subsection provides a concrete example of the implications this statement has when IC0 is mapped to CORBA middleware.

### 3.2 Mapping IC0 to CORBA

The architecture of the VIVIAN platform in Stage 0 is heavily influence by wireless CORBA [15]. Hence, it is of no surprise that IC0 maps to CORBA in a straightforward way. The functionality described by IC0 can be implemented by an ORB plus the naming CORBA Common Object Service (CCOS). In fact, only the GIOP engine from the ORB is sufficient for covering the addressing and messaging functionalities, and the naming functionality is mapped naturally to the naming COS.

Let's take for example the MICO implementation of CORBA specifications [19]. The ORB provided by MICO is built for a TCP/IP network, and thus contains an IIOB engine (Internet Inter-ORB Protocol). In the MICO distribution one can also find an implementation of the naming COS. The combination of the MICO ORB plus the naming CCOS provides an instantiation of the VIVIAN IC0 for TCP/IP networks.



**Figure 5** MICO ORB plus naming CCOS provide an implementation of the VIVIAN IC0 for TCP/IP.

When VIVIAN IC0 is mapped to the MICO implementation of CORBA, the two interfaces of IC0 (`IC0::bind` and `IC0::name`) can be defined as shown in Figure 6.

The interfaces defined in Figure 6 provide the basic information needed to use the VIVIAN IC0 when it is implemented on MICO. The `IC0::bind` interface contains three methods:

- The `ORB_init()` method permits the developer to instantiate the MICO ORB which provides the addressing and messaging functionalities of IC0.
- The `_narrow()` method is an auxiliary method which casts an object of type `CORBA::Object` to the implementation specific object type.
- The `resolve_initial_references()` method is a build-in method of the ORB which returns the CORBA IOR (Interoperable Object Reference, the equivalent of object reference for CORBA) of some well-known objects that offer essential services. In IC0 this method can take only one value as input argument, *"NameService"*, and it will return the CORBA reference to CORBA object which offers the naming functionality through the `IC0::name` interface.

The `IC0::name` interface contains only two methods: `bind` and `resolve`. The former allows servers to register with the naming service and the latter allows objects to lookup the CORBA reference of specific server.

```

module IC0 {

  typedef sequence<string> StringSeq;

  interface bind {
    CORBA::ORB_var CORBA::ORB_init ( in long argc,
                                     in StringSeq argv,
                                     in string ORBid );
    CORBA::_ptr_type _narrow ( in CORBA::Object_ptr obj );
    CORBA::Object_var resolve_initial_references ( in string name );
  };

  interface name {
    void bind ( in CORBA::CosNaming::Name name,
               in CORBA::Object obj );
    CORBA::Object_var resolve ( in CORBA::CosNaming::Name name );
  };

}; // module IC0

```

**Figure 6** A simplified definition of the IC0 interfaces for the VIVIAN platform mapping to MICO.

It is worth noticing that the two IC0 interfaces given in Figure 6 give only the indispensable methods that IC0 must provide when implemented on MICO. In practice, the application developer will have access to all MICO ORB functionalities (e.g. Basic Object Adaptor, Portable Object Adaptor, CORBA::release for destroying CORBA servers, etc). Similarly, the entire interface of the Naming CCOS as it can be found in [14], will be available since the MICO suite provides a complete implementation of the given OMG specification.

In a similar way, VIVIAN IC0 can be mapped to any other CORBA implementation besides MICO, such as Orbix from IONA ([www.iona.com](http://www.iona.com)), and e\*ORB from Vertel ([www.vertel.com](http://www.vertel.com)).

Mapping VIVIAN IC0 to a particular implementation of CORBA serves as a proof of concept that demonstrates the feasibility of VIVIAN. The argument that this proof of concept makes is that the application developer who want to program his mobile applications using VIVIAN IC0 (the strip down version of the VIVIAN platform), will face the task of programming in CORBA.

On the other hand, this very argument leads to the question: if the application developer has to program his mobile application in CORBA then what is the added value of VIVIAN? The answer to this question has two legs.

- First, the VIVIAN platform is not only about CORBA. What was shown in the previous section §3.2 was an example (admittedly, it is the predominant example in the VIVIAN project) of mapping the VIVIAN IC0 to a well-known middleware. There is nothing preventing future mappings of VIVIAN IC0 to other middleware platforms. The VIVIAN project is already considering a SOAP mapping.
- Second, the VIVIAN platform is not only IC0. The platform services which are VIVIAN specific, are not directly mapped to CORBA services. Hence, although

the VIVIAN core can be mapped to CORBA, the entire VIVIAN platform is expected to be quite more powerful than CORBA, especially for the purpose of developing mobile applications.

Another advantage of the mapping of VIVIAN IC0 to CORBA is the fact that it made possible the development of VIVIAN platform services, domain specific services and application demonstrators before the prototype of the VIVIAN platform was delivered.

### 3.3 Mapping IC0 to wireless CORBA

CORBA itself is not specifically intended for mobile applications. However, recent evolutions of CORBA have led to the specifications of wireless CORBA [15] which is well suited for mobile applications. The differences that wireless CORBA has from its "standard" counterpart as mainly centered around the use of a network protocol for wireless media and the relaxation of the assumption of a continuous and reliable network connections such as that provided by TCP/IP. The latter network protocol is the basis of the IOP, by far the most popular version of messaging and addressing protocol used by existing CORBA ORB implementations.

In order to make the VIVIAN platform a competitive choice for the developer of mobile applications, the mapping of VIVIAN IC0 to CORBA was not enough; wireless CORBA must be considered.

The good news for VIVIAN is that the wireless CORBA does not change the ORB API at all. Handling the wireless network bearer and the associated protocol is done at the GIOP engine, while the application developer does not have to learn any new CORBA primitive specific to the wireless version of the middleware.

For VIVIAN this means that the mapping of IC0 to CORBA described in §3.2 is still valid for wireless CORBA. The contribution of the VIVIAN project to the wireless CORBA work is focused on the specification of the GIOP tunneling over L2CAP, i.e. the description of how to implement a GIOP engine for the Bluetooth wireless protocol without necessitating the interposition of a stack of protocols (e.g. PPP, IP, TCP) in order to allow the use of the popular IOP engine.

Hence, the mapping of VIVIAN IC0 to wireless CORBA is very similar to the "standard" CORBA case, as show in Figure 7. The task of developing the prototype of the VIVIAN IC0 mapped to wireless CORBA is assigned to workpackage No.2 and is elaborated in another VIVIAN deliverable.

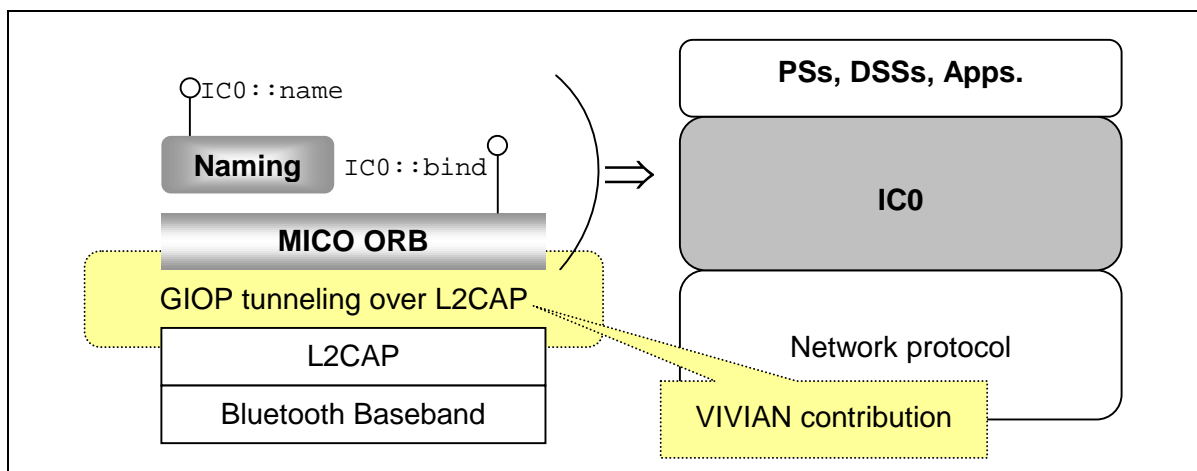


Figure 7 Mapping VIVIAN IC0 to wireless CORBA and highlighting VIVIAN contribution.

### 3.4 Interoperability Issues

The essence of the VIVIAN platform is to enable the access of services from mobile terminals, mainly handheld computers like PDAs and smart-phones. To achieve this, the (abstract) VIVIAN architecture can be mapped to a number of middleware platform, notably CORBA and SOAP. This means that the mobile terminal will be equipped with an instance of the VIVIAN platform, be it CORBA-based, SOAP-based or something else, which the developer of the client-part of the application (the part that will run on the mobile device) will be able to program.

On the other hand, the service developer is not bound to VIVIAN platform. This is especially true for services that are available on internet servers. These services can be CORBA-based, or SOAP-based or event based on some other platform.

The question that VIVIAN is called to answer is how the interoperability between the mobile, VIVIAN-based client and the server offering a service is accomplished. The answer to this question is simple in the case of homogeneous clients and servers. For example, a client that uses a CORBA-based instance of the VIVIAN platform will be able to access a CORBA-based service in a straightforward way, as described by the CORBA specifications. The same is true for a client that uses a SOAP-based instance of the VIVIAN platform and accessing a SOAP-based web service.

More challenging is the case of heterogeneous clients and servers. In general, a mobile terminal  $\mathbb{T}$  may have an instance of the VIVIAN platform based on middleware  $\mathbb{X}$  and a service  $\mathbb{S}$  may be available over a middleware  $\mathbb{Y}$ . For  $\mathbb{X}$  being SOAP and  $\mathbb{Y}$  being CORBA (i.e. accessing a CORBA-based service from a SOAP-based client), OMG has already made a suggestion (see CORBA Web Services [16]). Now VIVIAN is called to answer the interoperability problem for the general case of  $\mathbb{X}$  and  $\mathbb{Y}$  middleware platforms.

The answer that VIVIAN provides for the interoperability of heterogeneous clients and servers is the introduction of a *gateway* between a pair of heterogeneous client and server. In the context of this document we adopt the following definition of a gateway:

*A **gateway** is a network point that acts as an entrance to another network.*

In the VIVIAN case, a network is defined by the use of a given middleware platform. Hence, the gateway in our case is an entity which can use both middleware platforms  $\mathbb{X}$  and  $\mathbb{Y}$  and enables the transfer of a service request made on middleware  $\mathbb{X}$  to the equivalent service request on middleware  $\mathbb{Y}$ .

The communication between the terminal using an instance of the VIVIAN platform based on middleware  $\mathbb{X}$  and the gateway takes place as a service request according to the specifications of middleware  $\mathbb{X}$ . Then, the gateway translates the received service request to its equivalent for middleware  $\mathbb{Y}$  and issues it to the server which provides the desired service over middleware  $\mathbb{Y}$  as a service request according to the specifications of middleware  $\mathbb{Y}$ . The server received the request, serves it and send the reply to the gateway which performs the opposite translation (from  $\mathbb{Y}$  to  $\mathbb{X}$ ) and send the results of the translation as a reply to the terminal that has issued the initial service request. Table 1 summarizes the way VIVIAN deals with the interoperability issues addressed in this section.

	Service x	Service y
Terminal x	✓	Gateway
Terminal y	Gateway	✓

**Table 1** Interoperability issues in VIVIAN (X and Y are different middleware platforms)

### 3.5 The Gateway

The development of a gateway that enables the interoperability between VIVIAN-enabled terminals and services which are based on different types of base-middleware is not in the scope of this document. However, we provide two different implementation directions that can be used for the development of the gateway in order to clarify the role and the responsibilities of the gateway in the context of accessing services from VIVIAN-enabled terminals.

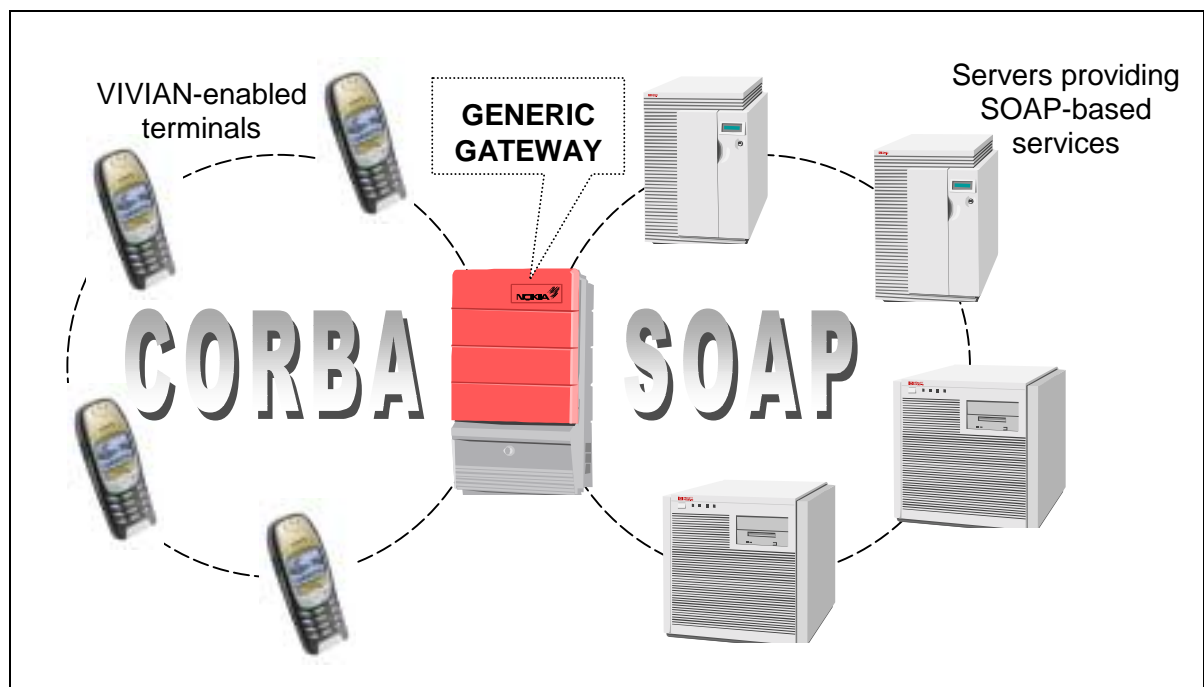
#### 3.5.1 Generic Gateway

One approach to the development of a gateway for the VIVIAN purposes described above is to construct a translation engine which can translate *arbitrary* requests from one base-middleware to another. Let's consider for example such a translation engine which translates CORBA requests to SOAP requests. This means that a VIVIAN-enabled terminal which is equipped with a CORBA-based instantiation of the VIVIAN platform will be able to use this gateway to access SOAP-based services.

Since the generic gateway must be able to translate arbitrary CORBA requests to SOAP request, it must be able to translate service requests, service addresses and parameter types from CORBA to SOAP and service replies from SOAP to CORBA. More precisely, the generic gateway must provide at least the following functionalities:

- Translate a CORBA service request to a SOAP service request in a well-defined way. The straightforward approach to this translation would be to keep the name of the service unchanged across different types of middleware and make the appropriate mapping of request from one middleware to the other. On this issue standardization bodies will offer major contributions (e.g. the OMG work on CORBA Web services [16]).
- Translate a SOAP service reply to a CORBA service reply in a well-defined way. The straightforward approach to this translation can follow the rationale presented above. Similarly to the previous functionality, standardization bodies are expected to specify well-defined ways to achieve such translations (e.g. see [16]).
- Translate a CORBA type to a SOAP type in a well-defined way. Again, standardization bodies are expected to specify well-defined ways to translate variable types (e.g. see [16]).
- Translate the address of a CORBA service used in the initial service request to the address of the SOAP service that will be used in the translated counterpart of the initial request.

The role of the generic gateway is graphically illustrated in .



**Figure 8** The generic gateway for CORBA-based terminals and SOAP-based services.

The last point in the list above makes the construction of the generic gateway a complicated task. In order to enable the translation of service addresses the gateway should provide a naming functionality that would allow the SOAP services to register and their addresses to become known to the gateway. The SOAP services can either register explicitly with the gateway or the gateway could implicitly register their addresses after it has looked them up in the naming / trading service available in the SOAP-based middleware network (e.g. UDDI).

Alternatively, the gateway should have a very close interaction with the naming / trading service on the SOAP-based middleware in order to obtain the addresses of the desired SOAP-based services every time it receives a request for them.

Another challenge that arises with the translation of service addresses is related to the fact that the gateway is not explicitly addressed by the terminal which issues the request for service. Rather, the gateway must be able to capture requests for services in the CORBA-based middleware that cannot be served by the available CORBA services. Then the gateway can check whether there is a corresponding SOAP service which can serve the "hanging" request and if so translate the request and forward it to that SOAP service.

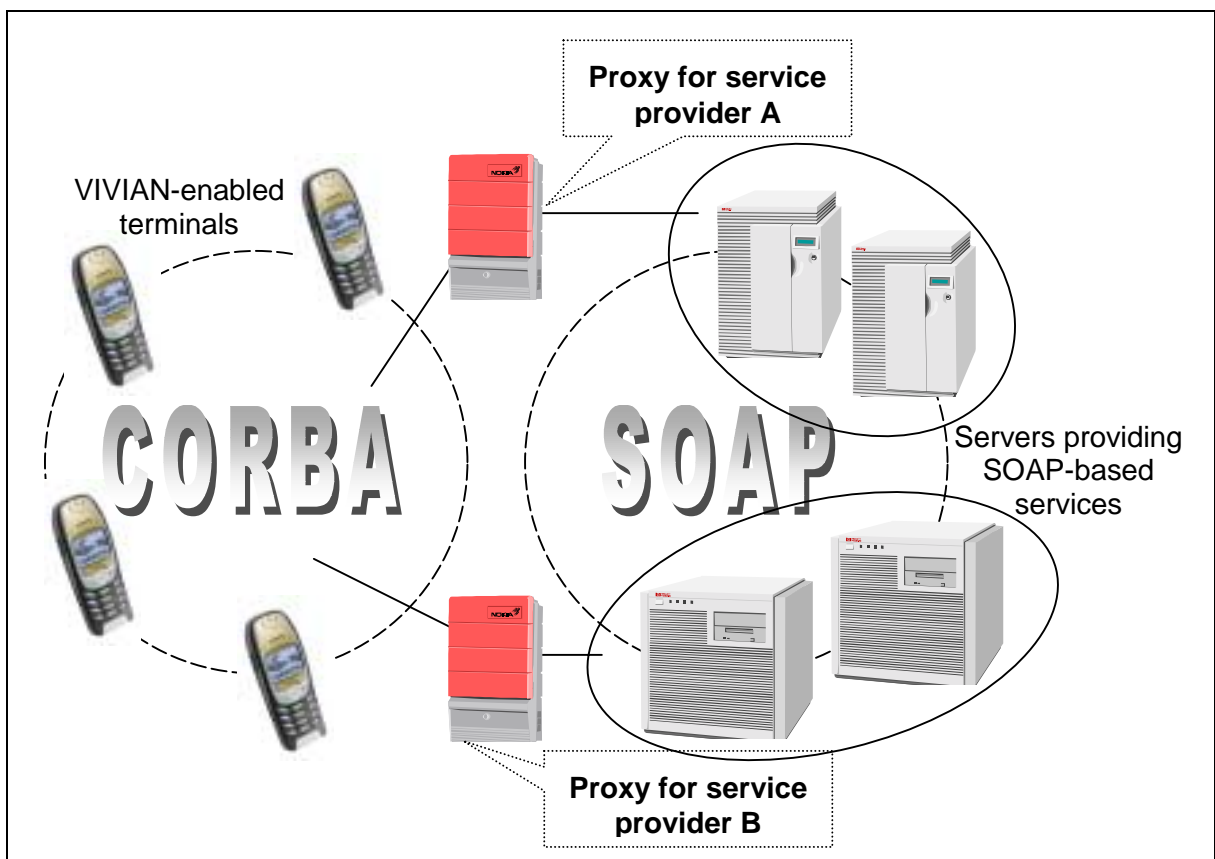
The challenge in this case comes from the fact that a single gateway that captures "hanging" CORBA requests represents the number one bottleneck for accessing SOAP service in the given case under study in our example. The lack of scalability with the number of the VIVIAN-enabled terminals that will use the gateway as well as with the number of SOAP service the gateway can serve would be a major problem for the developer of the gateway.

Alternatively, a number of generic gateways can be available for the same CORBA-based middleware, and they can share the load of translating CORBA request to SOAP and SOAP replies to CORBA. However, in this case a coordination mechanism must be employed to control the load balancing activity and the parallelism of catching and serving "hanging" CORBA requests.

### 3.5.2 Proxy-style Gateway

One way to circumvent the complexity associated to the generic gateway is to sacrifice the general aspect of the gateway. Instead of having a generic gateway that captures the non-served CORBA request, identify the equivalent SOAP-based service, translate the original request to a SOAP request and forward it to the appropriate SOAP-based server, the functionality of the gateway can be trimmed down to a *proxy* (as it is described in [10]). The proxy acts as a gateway which is specific to one service, one server, or preferably, one service provider.

In general, a service provider has a number of servers each providing a number of services. The usual practice is that the services offered by the same service provider are developed, tuned and optimized for a specific middleware, for example SOAP. However, in order to increase the number of his customers, the service provider can develop his own gateway which cannot translate any arbitrary CORBA request to its SOAP equivalent but it can do that only for the SOAP-based services offered by the given service provider. Figure 9 provides a graphical illustration of the proxy-style gateway in the VIVIAN context.



**Figure 9** The proxy-style gateway for CORBA-based terminals and SOAP-based services.

The proxy-style gateway is fairly simpler than its generic counterpart since its role is restricted in providing the following functionalities:

- Translate a CORBA service request to a SOAP service request in a well-defined way. Since the purpose is not to translate any arbitrary request but only those requests intended for the services provided by a given service pro-

vider, this translation engine can be optimized for the given set of service requests it can receive.

- Translate a SOAP service reply to a CORBA service reply in a well-defined way. Similar to the above case, the translation engine can be optimized for the given set of service request it can receive.
- Translate a CORBA type to a SOAP type in a well-defined way. The developer of the gateway can chose to follow a standard way for doing this (as described in the generic gateway in §3.5.1) or a custom way that is optimized for the types used in the service requests and replies that the gateway can receive.

**N.B.:** In the proxy-style gateway, the gateway does not have to capture non-served CORBA requests. Rather, the gateway is explicitly addressed in the original request. In fact, for the VIVIAN-enabled terminals that are equipped with a CORBA-based instance of the VIVIAN platform, the gateway of service provider X looks exactly like a server which provides all the services offered by provider X as CORBA-based services. Hence, the CORBA request issued by such a terminal as those mentioned above is explicitly addressed to the proxy-style gateway of the provider who offers the desired service.

In the proxy-style gateway, there is no need for the gateway to interact with a naming / trading service because it already knows all the (SOAP-based) services for which it can translate CORBA requests.

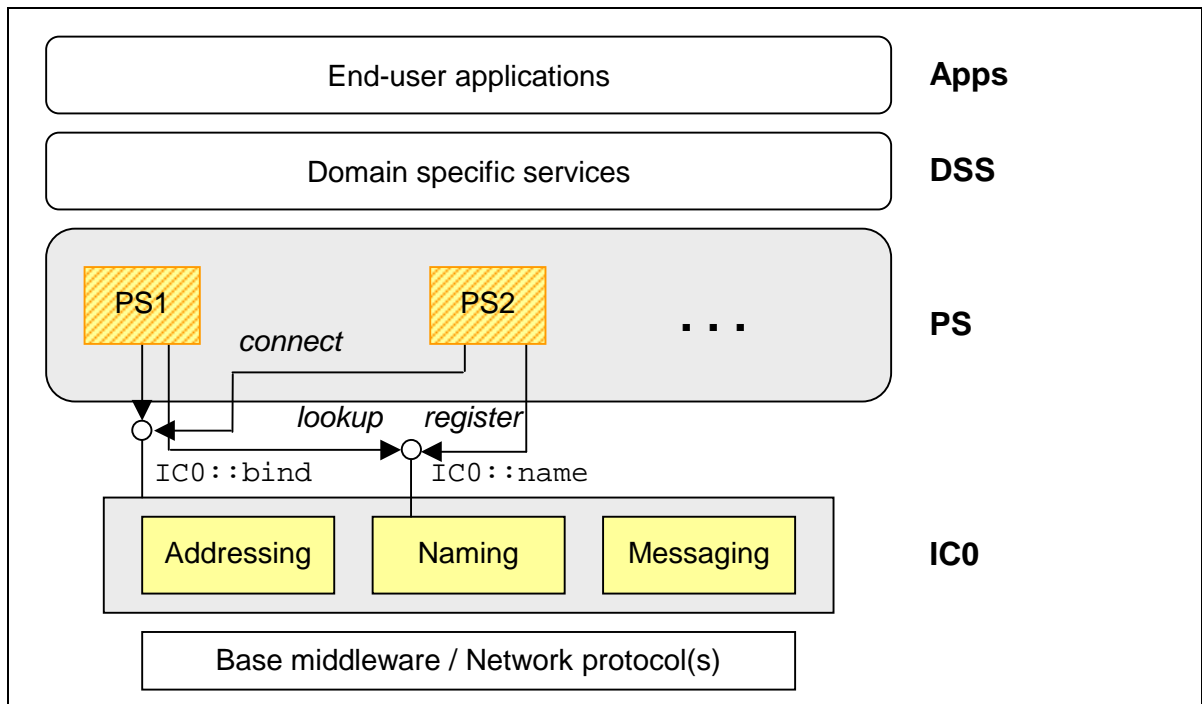
Another advantage of the proxy-style gateway is that the use of multiple gateway by the same provider can be coordinated by dividing the (SOAP-based) services offered by that provider to the different gateways he possesses. On the other hand, the use of one proxy-style gateway by each service provider on the same (SOAP-based) network does not introduce any need for coordination since the gateway is explicitly addressed in the request for service and hence only one of the many gateways will actually receive a given request.

### 3.6 Platform Services, Stage 0: PS0

Based on the VIVIAN IC0, a number of services can be built and used by the developer of mobile applications for VIVIAN-enabled terminals. For CORBA-based instances of the VIVIAN platform all the CCOSs (e.g. Trading service, Object Transaction service, Persistent State service, and all the other CORBA services defined by the OMG) can be adopted and used in a straightforward CORBA way. The same statement is true instances of the VIVIAN platform that are specific to some other base middleware and services that are developed from that other middleware (e.g. SOAP-based VIVIAN instances and UDDI [20]).

In addition to the use of *services-off-the-shelf* that VIVIAN IC0 enables, a number of VIVIAN-specific services, the Platform Services (PSs), are specified and developed in the VIVIAN project. The specifications of these VIVIAN-specific services are given in a separate document [5]. This section provides only a rough outline of the way the VIVIAN PSs will integrate with the VIVIAN IC0.

Figure 10 below reproduces the contents of Figure 4 that illustrates the abstract architecture of VIVIAN IC0. According to this figure, IC0 provides two interfaces which are available to the components from any of the layers laid above IC0. The VIVIAN platform services will use these two interface to bind to IC0 and to make their presence known to other components (application components, components providing domain specific services or component providing other platform services).



**Figure 10** Graphical illustration of the integration of PSs with the VIVIAN IC0.

Since the VIVIAN PSs use the interfaces provided by IC0 to integrate with the VIVIAN platform, they must contact these interfaces in a way specific to the base middleware for which the instance of the VIVIAN platform that will be used is developed. Hence, VIVIAN PSs in Stage 0 are specific to the base middleware used for the development of the IC0 on which the PSs will be available. This does not mean that the very same PS cannot be available for different IC0 implementations. However, it does mean that it is the responsibility of the PS developer to enable the integration of the service he develops with more than one type of middleware.

In practice, the above notice is not very restrictive for the VIVIAN platform. Since the VIVIAN platform will be running on a mobile device, it is most likely that the scarce resources of the mobile device will not allow the co-existence of more than one type of middleware. Hence, an instance of a PS will not need to integrate with different types of base middleware.

On the other hand, the developer of a PS will have to provide for the instantiation of his service on different VIVIAN platform instances, probably run on different types of mobile terminals. This inconvenience will be addressed in future Stages of the VIVIAN architecture.

Another noteworthy point is that VIVIAN IC0 does not provide any explicit support for the installation of components on the mobile terminal. That counts both for application components but also for components that provide platform service. Hence, the person who will be responsible for the installation and configuration of the VIVIAN platform on a given terminal has to employ the primitive provided by the operating system and/or the base middleware in order to install all the necessary components for a given instantiation of the VIVIAN platform, including IC0 specific components, PS specific components and application components.

## 4 Conclusion

This document provides the description of the abstract VIVIAN system architecture as well as the description of one possible mapping of this architecture to a CORBA base middleware.

Direct communication among CORBA-based VIVIAN-enabled terminals (i.e. interaction based on a point-to-point link) is shown to be straightforward. However, VIVIAN architecture Stage 0 does not provide any support for direct communication between two VIVIAN-enabled terminals that each runs a different instance of the VIVIAN platform (e.g. a CORBA-based terminal and a SOAP-based terminal).

On the other hand, the access from a VIVIAN-enabled terminal of a service that is available on a base middleware different than the one used for the implementation of the VIVIAN platform on the terminal can be achieved through the use of the appropriate gateway.

Given these remarks regarding the influences of VIVIAN architecture Stage 0 and the interconnection capabilities of the VIVIAN-enabled terminals, we can deduce the following properties of the VIVIAN-enabled terminal in Stage 0 of the VIVIAN architecture:

- A VIVIAN-enabled terminal is seen by the application developer as a black-box which provides a number of interfaces for accessing the box's capabilities. These are mainly interconnection capabilities offered by IC0 and a number of other functionalities offered either by services-off-the-shelf (e.g. OMG CCOSs for CORBA-based instances of the VIVIAN platform) or by VIVIAN specific services, also called Platform services and described in [5].
- The VIVIAN IC0 allows the application components on a VIVIAN-enabled terminal to interact with remote services in a transparent way regarding the interconnection network and the transport protocols.
- The VIVIAN platform in Stage 0 does not provide any support for the installation of the application components on the VIVIAN-enabled terminal.

## 5 References

- [1] VIVIAN Consortium. *Opening mobile platforms for the development of component-based applications (VIVIAN)*, ITEA - 99040. Full Project Proposal v3.2.1, July 2001.
- [2] VIVIAN Consortium. *Requirements Document v0.8 (deliverable report D1)*. May 2001.
- [3] VIVIAN Consortium. *Technology Survey (deliverable report D1b)*. May 2001.
- [4] VIVIAN Consortium. *Requirements Analysis v0.1 (deliverable report D1a)*. December 2001.
- [5] VIVIAN Consortium. VIVIAN Services Specifications. *To appear*.
- [6] I. Sommerville, P. Sawyer. *Requirements Engineering: a good practice guide*. Wiley, 1997.
- [7] M. Jackson. *Software Requirements & Specification: a lexicon of practice, principles and prejudices*. Addison – Wesley 1995.
- [8] D. F. D'Souza, A. C. Wills. *Objects, Components, and Frameworks with UML: the Catalysis<sup>SM</sup> approach*. Addison-Wesley, December 1998.
- [9] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal: *Pattern-Oriented Software Architecture: a System of Patterns, Volume 1*; John Wiley & Sons, 2001.
- [10] Erich Gamma *et.al.* *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [11] D. E. Comer. *Internetworking with TCP/IP. Volume I; Principles, protocols and architecture*. Prentice Hall, 1991.
- [12] J. W. Stamos and D. K. Gifford. Remote Evaluation. *ACM Transactions on Programming Languages and Systems*, 12(4):537-564, October 1990.
- [13] CORBA. Common Object Request Broker Architecture. <http://www.omg.org/>
- [14] OMG. The Naming Service Specification. <http://www.omg.org>
- [15] OMG. The Wireless CORBA Specification. March 2001. <http://www.omg.org/cgi-bin/doc?telecom/01-03-01>
- [16] OMG. CORBA Web Services. <http://cgi.omg.org/cgi-bin/doc?orbos/01-06-07>
- [17] SOAP. Simple Object Access Protocol. <http://www.w3.org/TR/SOAP/>
- [18] WSDL. Web Services Description Language. <http://www.w3.org/TR/wsdl/>
- [19] A. Puder and K. Römer. MICO: An Open Source CORBA Implementation. Morgan Kaufmann Publishers, 2000. [www.mico.org](http://www.mico.org)
- [20] UDDI. Universal Description, Discovery and Integration. [www.uddi.org](http://www.uddi.org)