



Information Technology for European Advancement

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

**Deliverable Report D1**

***Requirements Document (v0.8)***

**May 2001**

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 (NOKIA) in May 2001*

## Table Of Contents

1 Introduction .....	1
1.1 Purpose of the document .....	1
1.2 Scope of the VIVIAN system .....	1
1.2.1 Target market .....	1
1.2.2 Business model .....	2
1.2.3 Stakeholders .....	2
1.3 Definitions, acronyms, abbreviations .....	3
1.4 References .....	5
1.5 Structure of the remainder .....	5
2 Overview .....	6
2.1 Platform perspective .....	6
2.2 Platform functions/services .....	7
2.3 User characteristics .....	8
2.4 General constraints .....	8
2.5 Assumptions and dependencies .....	8
3 Requirements Specification .....	9
3.1 General .....	9
3.2 Component Interoperability .....	10
3.2.1 Parser / generator tools for common data format .....	11
3.2.2 Common open data format for different application categories .....	12
3.2.3 Common open extensible data exchange format .....	13
3.2.4 Component model definition .....	14
3.2.5 Compliance test suite .....	15
3.2.6 Forward Compatibility .....	16
3.2.7 Minimal set of Interoperability APIs .....	17
3.2.8 Event generation and notification .....	18
3.2.9 SW component version control .....	19
3.3 Service Discovery .....	20
3.3.1 Service discovery .....	21
3.3.2 Location of SW components on the network .....	22
3.3.3 Registry of (SW) Components .....	23
3.3.4 Peer-to-peer networking .....	25
3.3.5 Ad-hoc networking over different networks .....	27
3.4 Security .....	28
3.4.1 User authentication to a service provider .....	29
3.4.2 Service Provider authentication .....	30
3.4.3 Explicit access/denial to services .....	31
3.4.4 Encrypted communication .....	32
3.4.5 Encrypted data .....	33
3.4.6 API access control .....	34
3.4.7 User authentication to the terminal .....	35
3.4.8 Store in protected form the user's security information .....	36
3.4.9 Digital signatures for transactions .....	37

3.4.10	Able to hook up different encryption systems .....	38
3.4.11	User authentication to the application .....	39
3.4.12	Not affected too much by security / privacy .....	40
3.4.13	Terminal should support different customised user authentication methods .....	41
3.5	Network Transparency .....	42
3.5.1	Common API for network access .....	43
3.5.2	Atomic transactions, resync and roll-back mechanisms .....	45
3.5.3	Transparency to network disconnection .....	46
3.5.4	Resuming long downloads .....	49
3.5.5	Graceful degradation against component failures .....	50
3.6	Terminal Resource Management .....	51
3.6.1	Adaptive resource management of the terminal .....	52
3.6.2	Peaceful co-existence between applications .....	57
3.6.3	No unnecessary connections .....	58
3.6.4	Resource sharing mechanisms .....	59
3.6.5	Power save mode and application restart .....	60
3.6.6	Memory protection .....	61
3.7	Terminal Configuration Management .....	64
3.7.1	Profiles for different terminals .....	65
3.7.2	Terminal capability enquiry mechanism .....	66
3.7.3	Profiles for application families .....	67
3.8	Future Proof & Extendibility .....	68
3.8.1	HW & SW Independence .....	69
3.8.2	Based on open specifications/standards .....	70
3.8.3	Installable (3 <sup>rd</sup> party) SW Components .....	71
3.8.4	Plug and Play: no cables to connect to peripherals and other terminals .....	73
3.8.5	User Input Device Independence .....	74
3.9	User Interface .....	75
3.9.1	Graphical manipulation .....	76
3.9.2	UI Interaction .....	77
3.9.3	Defined set of (G)UI elements .....	78
3.9.4	Consistent look, easy to learn, intuitive .....	80
3.10	Specific Components .....	81
3.10.1	Persistent Data Storage .....	82
3.10.2	Terminal Specific Events .....	83
3.10.3	Remote User Interface .....	84
3.10.4	Stream Control .....	86

## 1 INTRODUCTION

The main objective of the VIVIAN project is to create an open, mobile platform that promotes the development of component-based applications [1]. A proven way to reach such a goal is to elicit the requirements placed on the platform by analyzing the needs of a multitude of actors that will get in contact with the platform. The composition of the VIVIAN consortium with members coming from the large scale industry, the SME category and the academia gave us an ideal basis for analyzing the platform requirements from a number of viewpoints including those of the application developer, the value-added service provider, the platform provider, the terminal manufacturer and the terminal software developer. The result is a set of some 50 requirements organized in 9 categories.

### 1.1 Purpose of the document

This document contains a summary of the requirements elicited during a period of 6 months when the project consortium analyzed the needs that different actors have placed on the VIVIAN system. Besides serving as a deliverable report to the ITEA Office, this document is also a reference guide for the next phase of the project which focuses on the specification of the VIVIAN platform and the APIs provided by the platform and by the associated services defined in the VIVIAN system.

The contents of the document have resulted after a review process that took place in two phases: first, requirements were elicited and their description was elaborated and then requirements referring to the same functionality (e.g. this was the case when more than one actor had identified the same a closely similar requirement) were grouped together to lead to a clearer requirement description. We bring to the reader's attention that this is not the final version of the VIVIAN requirements, since we expect that the specification of the APIs (for which this document serves as a guide) might cause some modifications in the requirements description. These modification might be the further merging of requirements or the splitting of a requirement to two or more subparts depending on the way the functionality captured by the APIs is mapped to them.

### 1.2 Scope of the VIVIAN system

In order to better comprehend the rationale of the requirements described in this document, we first describe in brief the VIVIAN scope in terms of the target market for the open platform, the associated business model and the stakeholders which influence the purposes as much as the technical characteristics of the platform.

#### 1.2.1 Target market

The VIVIAN platform is meant to facilitate the development of 3<sup>rd</sup> party applications which run in a distributed way on a number of mobile terminals and a number of web terminals. Such applications may include access to web-based services where the application is deployed in a client/server fashion (the client running on the mobile terminal and the server on the web terminal). For example, an application which enables the user to access an information database on the internet in order to retrieve various kinds of information.

However, client/server applications for accessing web-based services is not the only kind of applications that the VIVIAN platform can support. It also provides support for collaborative applications that reside on a number of mobile terminals and that may or may not involve the access of some centralized server. Examples of such applications are the electronic wallet based shopping applications and multiparty gaming.

Hence, the VIVIAN platform has the potential to form the development basis for a wide spectrum of 3<sup>rd</sup> party distributed applications that involve heavy interactions of mobile terminals and this is a fact that relates to the openness of the platform. But the openness characteristic of the VIVIAN platform has also another aspect, the one

relating to the variety of technologies it can accommodate. In order for the platform to be really useful for 3<sup>rd</sup> party application development for mobile terminals, it should function for the majority of mobile terminals regardless the differences in their resource characteristics and the operating system they are running. Moreover, the platform should allow a diversifying multitude of communication technologies to be used for the application interactions.

To summarize, the VIVIAN platform is target for a market of 3<sup>rd</sup> party application developers which develop applications in various domains, for a wide range of mobile terminals that may use of a wide variety of communication technologies.

### 1.2.2 Business model

There is no single model that can capture all the possibilities of market penetration for the VIVIAN platform. However there are two basic parameters that make the business model presented in the remainder of this section the predominant one. The first parameter is the fact that the VIVIAN platform is not meant to be a commercial product but rather an attempt to promote a number of existing and upcoming commercial products offered by some of the project partners. The second parameter is the fact that the VIVIAN platform is a set of specifications and a software that can be classified as middleware which means that its role is to combine a number of possibly heterogeneous terminals and connectivity technologies (i.e. communication media and communication protocols) and allow the deployment of distributed applications over them.

Based on the above facts, the business model that underpins the VIVIAN platform lies on two cornerstones: the mobile terminal and the 3<sup>rd</sup> party applications which can be either end-user applications or value-added services (VASs) accessible from a mobile terminal.

The market roles linked to the mobile terminal (i.e. the manufacturer and the providers of the OS and the peripherals) offer to the customer a VIVIAN enabled handset that allows the customer to access VIVIAN compatible VASs as well as to execute VIVIAN compatible applications. In order to achieve that, the mobile terminal related roles provide with the handset a number of platform functionalities which can be accessed through a set of service interfaces and APIs that conform to the VIVIAN specifications. The customer is not charged extra for the VIVIAN functionalities on the handset and he/she does not have to install and/or explicitly configure them since they come integrated in the handset packet he/she purchased.

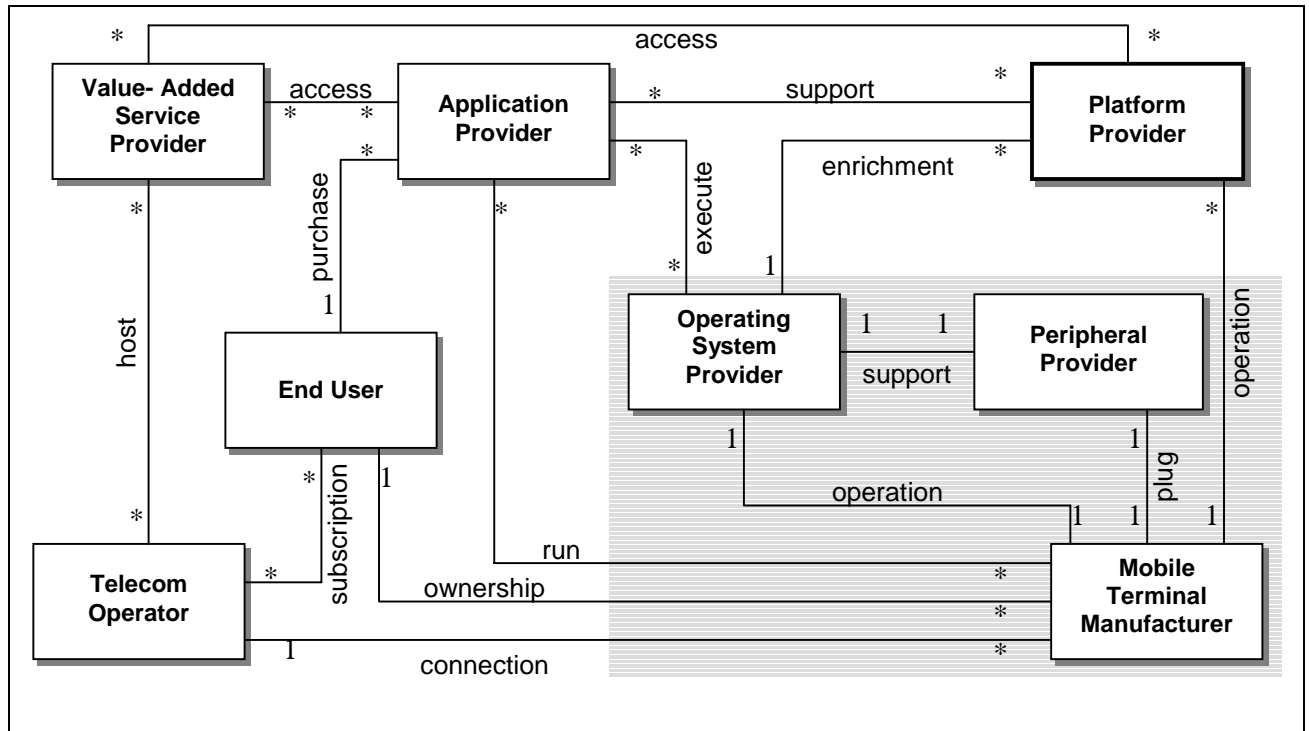
One the other hand, the 3<sup>rd</sup> party developers of applications or VASs for VIVIAN enabled mobile terminals can take advantage of the features of the VIVIAN platform to their full extend since the VIVIAN specifications, service interfaces and APIs are publicly open. Hence, VIVIAN compatible applications and VASs can be offered to VIVIAN enabled handsets with no more installation and configuration effort than what is required for launching the applications and connecting to VASs. Moreover, in case that some VIVIAN applications use optional VIVIAN functionality that is not available in "standard" distribution that comes with the handset, the 3<sup>rd</sup> party developers may distribute that optional functionality with the application package.

To better comprehend the aforementioned business model and the implications it has on the requirements elicitation and analysis of the VIVIAN platform, in the following subsection we elaborate the designated stakeholders and their relations.

### 1.2.3 Stakeholders

The identification of the system stake holders is an essential step in the process of requirements elicitation and specification of a system [2]. The business model briefly described above defines six major roles which represent the stakeholder in the market where the VIVIAN platform is targeted: the mobile terminal provider, the application and VAS providers, the telecom operator, the platform provider and, of course, the end-

user. In addition, the role of the mobile terminal provider can be seen as a composite role consisting of the roles of the mobile terminal manufacturer, the OS provider and the peripheral manufacturer<sup>1</sup>. These roles along with the relations that hold among them are graphically illustrated in the picture below and elaborated in the remainder of this subsection. As is the case with the majority of the business models, this one is end-user centric too.



The Entity-Relation diagram of the VIVIAN business model.

The end-user may own one or more terminals, may have purchased one or more applications for these terminals and may have subscriptions to one or more telecom operators. The telecom operator may provide telecom and internet connectivity to a number of mobile terminals and may host a number of VASs. The same VASs can be offered to the end-user by a number of different operators and can be accessed by a number of different user applications. The applications may run on different mobile terminals operating different OSs and having plugged on various peripherals. Finally, the VIVIAN platform may provide access to a number of different VASs, support to a number of different applications enriching the functionalities of the OS installed on the mobile terminal on which the platform is operating.

What the reader should retain from the business model description given in the previous subsection and the stakeholders E-R diagram here above is the fact that the platform provider (the VIVIAN platform in our case) is closely related to the 3<sup>rd</sup> party applications and VASs as well as to the mobile terminal provider. The later three roles may in fact provide the platform (or parts of it) to the their customers.

### 1.3 Definitions, acronyms, abbreviations

So far we have abusively used a number of terms with a meaning that may not correspond exactly to the readers intuition. The most important of these terms which where used to described our work are *platform*, *system*, *specification*, *service*, and 3<sup>rd</sup>

<sup>1</sup> In this context, the peripheral provider is the role of the one who provides both the hardware and the low level software (e.g. device drivers) which is related to a specific application domain (e.g. the provider of smart-card related hardware and software for banking applications).

*party applications*. The following list contains the definition of these terms according to the meaning with which they are employed in this document.

- *3<sup>rd</sup> party application*: a piece of software developed by an entity other than the end-user and the mobile terminal provider. In the context of this report the term is used to refer to the software that various software houses may develop for the (VIVIAN enabled) mobile terminals.
- *Value-added service (VAS)*: a piece of software which provides some functionality that enhances the functionalities offered by the mobile terminal and the applications executed on it and which is accessible to end-user under some fee which depends on the VAS and its provider. In the context of this report this term refers to various internet services which will be made available to the VIVIAN enable mobile terminals.
- *Platform*: a piece of software which enhances the functionalities offered by the mobile terminal and enables the development of 3<sup>rd</sup> party applications and the access of VASs. This term, present also in the project title, refers to the implementation of the functionalities that allow to different types of mobile terminals to host the same 3<sup>rd</sup> party applications and to access the same VASs without any modifications/adjustments of the applications or the VASs.
- *System*: a group of interrelated hardware and software components that provide some meaningful functionality to its surrounding environment. In this document the term system is used to refer to the combination of a mobile terminal and the VIVIAN platform.
- *Specification*: the detailed description of the functionalities offered by a system or a platform. In the context of this report the term specification is often used in conjunction with the term platform and it refers to the definition of the functionalities and the service interfaces offered by the VIVIAN platform. It also appears as requirements specification and it refers to the detailed description of the requirements placed on the VIVIAN platform by the project consortium.

Besides the above terms, this report also contains a number of abbreviations which are listed below:

- API: application programmer interface
- CORBA: common object request broker architecture
- COS: common object service
- GIOP: general inter-ORB protocol
- GPRS: general packet radio services
- GSM: global system for mobile communication
- GUI: graphical user interface
- IIOP: internet inter-ORB protocol
- OMG: object management group
- ORB: object request broker
- OS: operating system
- PDA: personal digital assistant
- SOAP: simple object access protocol
- UI: user interface
- VAS: value-added services

## 1.4 References

- [1] The VIVIAN web-site, [www.nrc-nokia.com/Vivian](http://www.nrc-nokia.com/Vivian). VIVIAN consortium, June 2000.
- [2] I. Sommerville, P. Sawyer. *Requirements Engineering: a good practice guide*. Wiley 1997.
- [3] M. Jackson. *Software Requirements & Specification: a lexicon of practice, principles and prejudices*. Addison – Wesley 1995.
- [4] IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, IEEE-SA Standards Board, 1998.

## 1.5 Structure of the remainder

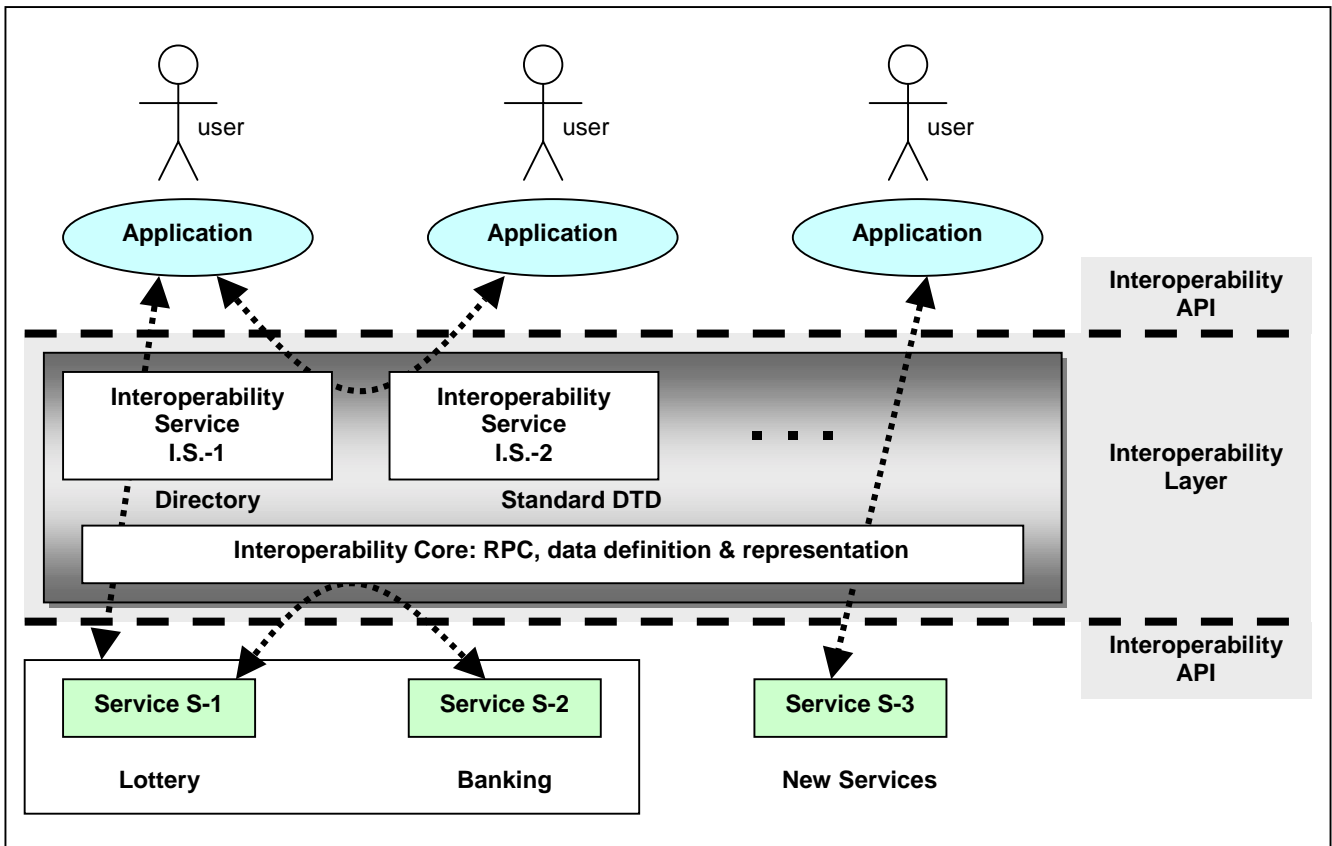
The remainder of this report is structured according to the IEEE recommendations for software requirements documents given in [4]. Next section contains an overview of the system requirements identified by the project consortium and a brief description of their classification into nine categories. Section 3 elaborates on these categories and the requirement contained in each category. The system requirements are presented in a uniform format which succinctly indicates the nature target and meaning of each requirement, the relation with other requirements and some other information regarding the author the status of the requirement.

## 2 OVERVIEW

### 2.1 Platform perspective

The objective of the VIVIAN project is to provide an adequate platform for handheld terminals and PDA devices, which will enable the production of third-party software applications. Hence, the VIVIAN platform will offer the necessary support for access to third-party services from user applications running on the target devices, as well as for the development of new services and applications over such devices.

The following figure gives an overview of the VIVIAN platform from a logical perspective.



Graphical representation of the logical view of the VIVIAN platform

The VIVIAN platform is not aimed at a specific kind of terminal nor at a specific networking infrastructure. Instead, services and applications may run over various terminals and interact via diverse networks (e.g., Internet, Bluetooth, GSM, GPRS, ...). However, the VIVIAN platform specifically concentrates on enabling interaction from mobile, resource-constrained terminals where it is aimed to not restrict the underlying OS nor wireless network.

As depicted in the figure, the VIVIAN platform enables interaction among *applications* running on mobile terminals and third-party *services*, also possibly running on mobile terminals. This leads defining the platform as an *interoperability layer*, which comprises an *interoperability core* and a number of *interoperability services*. Applications and services using the VIVIAN platform to interact will run over a variety of terminals and do so via a variety of networks depending on the physical location of the interacting parties. In that context, the VIVIAN specification embeds functions relating to enabling the execution of applications and services over mobile, resource constrained terminals. This part of the specification is referred to as *terminal modules*. Finally, the

specification of the VIVIAN platform includes the one of a *component model*, which is to be exploited for the development of applications and third-party services using the VIVIAN platform to interact.

The following further defines the aforementioned terms.

**Interoperability layer:** The interoperability layer corresponds to the mandatory part of the VIVIAN platform, and as such is the main constituent of the VIVIAN specification. The layer subdivides into:

- **The interoperability core:** This defines the basic elements required for achieving *interoperability* and thus comprises base messaging facilities (e.g., RPC mechanism) and support for data exchange. This core relates to existing middleware platforms such as JINI, CORBA ORB and SOAP, from which it differs by enabling running a variety of applications/services over mobile resource-constrained terminals.
- **The interoperability services:** The interoperability services enhance the core by offering means for *cooperation* among actors interacting using the VIVIAN platform. It is part of the VIVIAN specification to prescribe the set of services to be provided, some of them possibly being optional. As for the interoperability core, the definition of the VIVIAN interoperability services relates to the ones offered by existing middleware platforms (e.g., COS of the OMG architecture).

The interoperability layer comes along with the **interoperability API**, henceforth part of the VIVIAN specification, which is to be used by third-party services and applications for interacting using the VIVIAN platform.

**Services:** Services correspond to functionalities invented, developed and deployed by unknown parties. Services may possibly be distributed as, e.g., a news service. Services are not part of the VIVIAN specification. However, it is the aim of the VIVIAN platform to ease the use of such services from mobile terminals and the development of new services specifically aimed at mobile terminals.

**Applications:** Applications refer to the software that run on the mobile terminals with which the user may interact through the associated User Interface. Applications are developed by unknown parties, are not distributed, and integrate multiple services. Applications are not part of the VIVIAN specification. However, it is the aim of the VIVIAN platform to ease their development.

**Terminal modules:** These give applications and services access to terminal resources. This facility is part of the VIVIAN specification, and by its very nature is not distributed. Access to terminal resources is further enabled through a remote access module. Offered access to resources is generic in that accessible resources are dependent upon each terminal.

**Component model:** VIVIAN components abstractly characterize services and applications that are to interact using the VIVIAN platform, as well as terminal modules with which applications and services interact to possibly adapt their behavior according to available resources. Briefly stated, a component is defined through its required and provided interfaces, hence giving the interfacing point of the given component.

## 2.2 Platform functions/services

Functions of the VIVIAN platform are detailed in Section 3, and subdivided into the following categories:

- **User interface:** The VIVIAN platform abstracts the user interaction methods as part of its interoperability API, and leaves it up to the user or device vendor to choose which input method to use to communicate information.

- **Security:** The VIVIAN platform provides different authentication, encryption and access control to applications, services and end-users as part of its interoperability services.
- **Terminal configuration management:** NO SUMMARY AVAILABLE
- **Resource management:** The VIVIAN platform offers functions dealing with resource constraints of mobile devices as part of its terminal modules. These enable the dynamic adaptation of resource consumption over the terminal and fair sharing of resources among the applications/services running on the terminal.
- **Service discovery component:** Service discovery is an important feature in future network scenarios. From the administrator's point of view, it simplifies the task of building and maintaining the network and in particular the one of introducing new devices and new services. The VIVIAN platform offers a service discovery component as part of its interoperability services, which enables clients of the platform to discover needed services, components and devices.
- **Network transparency:** NO SUMMARY AVAILABLE
- **Future proof/extendibility:** The VIVIAN platform supports applications/services that are based upon open standards, and which may be installed/uninstalled on demand in a secure way.
- **Component interoperability:** The VIVIAN platform promotes the development of applications/services based on software components. Interoperability among components is achieved by specifying one common API as part of the VIVIAN interoperability API, which is implemented in all components and allows them to exchange information with an Object Request Broker (ORB). This API enables a basic negotiation for further interfaces between a component and the ORB as well as the registration of the component on the platform.
- **Others:** : NO SUMMARY AVAILABLE

### 2.3 User characteristics

Intended users of the VIVIAN platforms are developers of application services that may be remotely accessed. In general, developers having experience in the use of existing middleware platforms (e.g., CORBA, JAVA-based middleware, COM) will be able to exploit the VIVIAN platform quite directly.

### 2.4 General constraints

It is not expected that the VIVIAN platform will limit the developer's option when developing new services/applications (not considering availability of implementation on a given target). However, this will depend upon the VIVIAN interoperability services that are actually available. In particular, services supporting the development of applications/services offering enhanced quality properties (e.g., fault tolerance, handling soft real-time constraints appertained to the exchange of multimedia data) may not be offered by the first releases of the VIVIAN platform.

### 2.5 Assumptions and dependencies

There is a priori no factor that affects the requirements stated in this document.

### 3 REQUIREMENTS SPECIFICATION

#### 3.1 General

This chapter contains the actual requirements of the Vivian system within the constraints set by the scope as explained above. The requirements have been divided into nine different categories that contain a coherent set of related requirements:

- Component Interoperability
- Service Discovery
- Security
- Network Transparency
- Resource Management
- Terminal Configuration Management
- Future Proof & Extendibility
- User Interface
- Specific Components

Within each category, the following functional and non-functional aspects were considered when defining the requirements, taking into account the interests of users, service providers, application developers and terminal manufacturers:

- Functionality: What does the system have to do
- Ease of use: How does the user interact with the feature
- Robustness
- Product differentiation
- Cost effectiveness

This approach should lead to a system that can create a viable open market of services and applications that are developed by 3<sup>rd</sup> parties and have commercial value to end-users

Requirements are described as **features** according to a standard template. These features are sometimes further elaborated in one or more **use cases**.

### **3.2 Component Interoperability**

The Vivian project promotes the development of applications based on software components. This chapter is describing different requirements that are necessary to ensure the interoperability of those components. In the Vivian system, the component interoperability is needed to allow components to discover each other in the system, exchange data if needed or use each others functionality to create a valuable service to the user.

## 3.2.1 Parser / generator tools for common data format

<b>Feature name</b> F11: parser / generator tools for common data format
<b>Priority</b> Optional
<b>Scope</b> System
<b>Dependencies</b> Requires Common open data format for different application categories (3.2.2) Common open extensible data exchange format (3.2.3)
<b>Description</b> There should be a system, which reads “documents” in the common data format and accepts structured queries on the data within the “document”. The system should allow constructing a “document” by inserting structured building blocks to an empty or existing “document”.
<b>Behavior</b> After reading a document, the system allows structured queries on its content. For creating a new document the system starts with an empty document, which can be build up by inserting building blocks of content. Further editing like moving and deleting blocks of content would be nice to have. After manipulating a document, the system allows to output it in the common data format. The system should allow handling multiple documents simultaneously.
<b>Appearance</b> There is no direct need for a GUI to this feature. A GUI-based Editor would be nice to have. If XML (or a subset) is chosen as common data format, the definition of the parser/generator could be SAX or DOM.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Jörn Eisenbiegler (CAS), Michael Przybilski (Nokia)

## 3.2.2 Common open data format for different application categories

<b>Feature name</b> I07: Common open data format for different application categories
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Modifies Based on open standards (3.8.2) Registry of components (3.3.3) Installable 3rd party WS components (3.8.3) Forward compatibility (3.2.6) parser / generator tools for common data format (3.2.1)
<b>Description</b> Different applications shall exchange data in a common, standardized, vendor-neutral and machine-neutral format, in order to harmonize the data manipulation tools used by these applications.
<b>Behavior</b> The structuring of information is expressed in a common formal language, according to a set of grammars, preferably expressed in the same language. Each grammar or family of grammars corresponds to one particular application or application domain, e.g. a family of grammars that defines the rules for representing medical information. The actual physical recording of the data is not necessarily the same as the logical representation, e.g. the latter may be mapped to a relational database. There exist several standardized data formats, which can be further subdivided into two categories: binary (e.g. ASN.1 and XDR) and textual (in particular, all the mark-up languages derived from SGML). Binary formats are more compact but more difficult to manipulate, textual formats are easy to parse but verbose and have a problem with the inclusion of binary information. It is quite clear, however, that XML has become the data representation format of choice for Web-based applications. An extensive set of domain-specific XML-based grammars (or "XML vocabularies") exists and is being extended regularly; ref. e.g.: <a href="http://wdvl.internet.com/Authoring/Languages/XML/Specifications.html">http://wdvl.internet.com/Authoring/Languages/XML/Specifications.html</a>
<b>Appearance</b> No UI required
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Harm Smit (Cime), Michael Przybilski (Nokia), Tom Suters (PHI, reviewer)

## 3.2.3 Common open extensible data exchange format

<b>Feature name</b> I08: Common open extensible data exchange format
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Modifies Based on open standards (3.8.2) Registry of components (3.3.3) Installable 3rd party WS components (3.8.3) Forward compatibility (3.2.6) parser / generator tools for common data format (3.2.1)
<b>Description</b> Within a given application category, the logical representations of all data exchanged over the network between peer application entities shall be based on a common standardized, vendor-neutral and machine-neutral format, in order to facilitate interoperability between these application entities, potentially originating from different vendors. This format shall consist of a set of predefined data types plus a set of rules for constructing composed types from the initial type set.
<b>Behavior</b> The structuring of data is expressed in a common formal language, using the predefined type set and a set of grammar rules, preferably expressed in the same language. It must be easy to extend such a grammar, without jeopardizing existing applications. There exist several standardized data formats, which can be further subdivided into two categories: binary (e.g. ASN.1 and XDR) and textual (in particular, all the mark-up languages derived from SGML). Binary formats are more compact but more difficult to manipulate, textual formats are easy to parse but verbose and have a problem with the inclusion of binary information. It is quite clear, however, that XML has become the data exchange format of choice for Web-based applications.
<b>Appearance</b> No UI required
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Harm Smit (Cime), Michael Przybilski (Nokia), Titos Saridakis (NOKIA)

## 3.2.4 Component model definition

<b>Feature name</b> I09: Component model definition
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Requires Future Proof & Extendibility
<b>Description</b> This feature expresses the ability of the Vivian platform to propose concepts and the corresponding artifacts to component-based development. The goal of the component model is to allow 3rd party providers (that are not partners of the Vivian project) to build software components that can execute on the Vivian platform.
<b>Appearance</b> No UI required
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Guy Bernard (INT), Michael Przybilski (Nokia)

## 3.2.5 Compliance test suite

<b>Feature name</b> I05 / R06: Compliance test suite
<b>Priority</b> Optional
<b>Scope</b> System
<b>Dependencies</b> None
<b>Description</b> Executable test suite
<b>Behavior</b> A set of test specifications should be part of the Vivian specificatoion to enable the verification of the conformity of a terminal to Vivian platform specification. Testing a system to Vivian conformance should be as much automatic as possible. Tests should cover all the Vivian API except those that are not automatically testable like graphic user interface.
<b>Appearance</b> Minimal console based UI required
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> David Mentré (INRIA), Michael Przybilski (Nokia), Tom Sutera (PHI, reviewer)

## 3.2.6 Forward Compatibility

<b>Feature name</b> I01: Forward Compatibility
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Modifies Minimal set of interoperability APIs (3.2.7) Terminal capability enquiry mechanism (3.7.2) Profiles for application families (3.7.3)
<b>Description</b> The Vivian specification and APIs shall be extensible with new functionality using well-defined extension procedures while not disabling existing applications that rely on functionality and APIs provided by older versions of the Vivian specification.
<b>Behavior</b> Existing applications shall be able to offer their full functionality on newer Vivian terminals implementing the extended functionality and APIs, giving the same or better user experience. New applications that use the extended functionality and APIs shall be capable of offering some functionality on existing Vivian terminals but may have a reduced performance and user experience. New applications shall not cause existing Vivian terminals to crash.
<b>Appearance</b> No UI required
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Tom Suters (PHI), Michael Przybilski (Nokia)

## 3.2.7 Minimal set of Interoperability APIs

<b>Feature name</b>	Minimal set of Interoperability APIs
<b>Priority</b>	Mandatory
<b>Scope</b>	System
<b>Dependencies</b>	none
<b>Description</b>	There should be a minimal set of API that needs to be implemented in every component to allow the communication and negotiation with other components and access to their APIs.
<b>Behavior</b>	
<b>Appearance</b>	No UI required
<b>Release</b>	0.8
<b>Status</b>	Draft
<b>Author</b>	Michael Przybilski (Nokia), Tom Sutera (PHI, reviewer)

## 3.2.8 Event generation and notification

<b>Feature name</b> Event generation and notification
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Requires Location of SW components on the network (3.3.2) Registry of SW components (3.3.3) Based on open specification/standards (3.8.2) Modifies Common open extensible data exchange format (3.2.3)
<b>Description</b> The VIVIAN platform should allow the definition of events that are of interest for VIVIAN applications and services, which should also be able to register their interest in these events and get a notification when these events occur.
<b>Behavior</b> The VIVIAN platform should allow the definition and monitoring of a number of event including hardware events (e.g. mouse and keyboard signals), network events (appearance/failure of a network element) and user-defined events. Software components in the VIVIAN platform which have registered their interest in specific events receive a notification upon the occurrence of an event.
<b>Appearance</b> The defined events, the components which have registered interest in them and a log-file with the occurrence of these events should be conveniently accessible to the VIVIAN applications and services which need to monitor the event activity in a VIVIAN system.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Titos Saridakis (NOKIA)

## 3.2.9 SW component version control

<b>Feature name</b> SW component version control
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Requires Location of SW components on the network (3.3.2) Registry of SW components (3.3.3) Modifies Service discovery (3.3.1) Profiles for application families (3.7.3)
<b>Description</b> The VIVIAN platform should allow the co-existence of multiple versions of the same software component without any ambiguities regarding their addressing. More than one versions of the same software component may be active at any time on the same machine or in a configuration of machines that execute the same VIVIAN applications.
<b>Behavior</b> Multiple versions of software components might be useful in a number of cases. In these cases the VIVIAN platform should allow the different versions of the same software component to register and interact with a VIVIAN application or service without introducing any conflicts in their addressing or otherwise causing confusion to the executing system. The latter should be also able to choose the precise version of a software component with which it wishes to interact.
<b>Appearance</b> Access to the mechanism that satisfies the version control requirement should be provided through an appropriate (G)UI which should allow the easy manipulation (registry, extraction, use) and identification of the various versions of software components that may co-exist in the same VIVIAN system. Existing version control systems can provide useful ideas about how to structure the version control UI (e.g. see <a href="http://www.cvshome.org">www.cvshome.org</a> )
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Titos Saridakis (NOKIA)

### 3.3 Service Discovery

Service Discovery (SD) simplifies finding and using services. From the network administrator's point of view, service discovery simplifies the task of building and maintaining a network, especially to introduce new services and new terminals. Let us make a short definition of "service" vs. "terminal". "*Terminals*" are represented by software; "*terminals*" export zero or more "*services*", "*services*" involve one or more "*terminals*".

SD is also one of generic interoperability (IOP) services in Vivian. Pieces of a distributed system must be able to find each other before they can start any type of interaction. SD is providing this ability - users of the system can discover services and terminals by means of SD. The following are key characteristics of the Service Discovery Component:

- it maintains a list of currently available services;
- all parts may participate directly without a central lookup service;
- there should be a notification of arrivals/departures of services or terminals;
- it needs to support different types of device profiles;
- it should be independent from underlying transport protocol;
- must work without or with minimum effort from administrator;
- the user should be able to search services by specifying the name or other attributes of a service.

Service discovery plays an essential role in ad-hoc communications and in ad-hoc networks management, where no fixed network infrastructure is present. In other words SD is the main component for building ad-hoc networks and components of the system can find each other without a central lookup service.

Another idea of utilization of SD is to find and download software components from the network and install them on a terminal.

## 3.3.1 Service discovery

<b>Feature name</b> F1: Service discovery
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Requires Future Proof & Extendibility and Network Transparency Modifies E12: Ad-hoc networking over different networks
<b>Description</b> Terminals should be able to discover available services on the network using names or other attributes of the services. Services can be provided by desktop server computers on the network or by any other kind of terminals . Terminals should be able to locate each other on the network in order to create ad-hoc networks and peer-to-peer connections.
<b>Behavior</b> Applications can send a query to the Service Discovery service in order to obtain references for specified (by name, properties, attributed etc.) services. Also Service Discovery must provide an ability to find other terminals. Terminals can interact directly and independently from the Service Discovery service after they have found each other. Service Discovery should notify the user of the system if new services become available.
<b>Appearance</b> User is interacting with "Service Discovery" feature by sending queries for service discovery.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Eugene M. Bourmakin (HUT), Tom Suters (PHI, review)

## 3.3.2 Location of SW components on the network

<b>Feature name</b> F16: Location of SW components on the network
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Requires Future Proof & Extendibility Modifies F1: Service discovery, F12: Registry of (SW) components
<b>Description</b> Applications running on a terminal should be able to locate SW components on the network using names or other attributes of components. This implies the need for directories and/or registries of available SW components in the network.
<b>Behavior</b> An application (game etc.) started on terminal can locate and, if necessary, download SW components (drivers, libraries etc.) from the network.
<b>Appearance</b> User can interact with "Location of SW components on the network" feature indirectly through a GUI of application.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Eugene M. Bourmakin (HUT). Reviewed by Bruno Bretelle (Review, INT), Christian Abeln (Review, CAS), Tom Suturs (PHI, review)

## 3.3.3 Registry of (SW) Components

<p><b>Feature name</b></p> <p>F12: Registry of (SW) components</p>
<p><b>Priority</b></p> <p>Mandatory</p>
<p><b>Scope</b></p> <p>System</p>
<p><b>Dependencies</b></p> <p>Requires Future Proof &amp; Extendibility, Network Transparency</p>
<p><b>Description</b></p> <p>There should be a system, which can be asked for components available in the system and their characteristics. A component should be defined by Name; the characteristics should at least include the version. It should be allowed to ask for the latest version, a specific Major version or an exact version.</p> <p>Components can also define proprietary characteristics that require a-priori knowledge to use them.</p> <p>The user shall be able to access a list of all available components, or those matching certain characteristics and be given the possibility to interact with them, e.g. start/subscribe to services or download and install a SW component.</p>
<p><b>Behavior</b></p> <p>When asked for a component with certain characteristics, the system either returns "not available" or all information that is required to use the component. The results of queries must reflect the dynamic availability of components in the network.</p> <p>When roaming with his Vivian terminal, the user may enter an area where a new component is offered or no longer accessible. Unless the user has explicitly inhibited this, an unsolicited alert is given by the terminal. Inhibiting an unsolicited alert can be based on certain characteristics of the component (e.g. personal, commercial, emergency...)</p> <p>The user shall at all times be able to explicitly ask for the list of all available components or those matching certain characteristics. Available components shall be listed irrespective of whether an unsolicited alert was disabled. The user shall be able to start all the services in the list or otherwise interact with it (e.g. insert it in a filter).</p>
<p><b>Appearance</b></p> <p>The user is alerted by the terminal e.g. via a sound or display of a message on the screen that a new component is now available or no longer available.</p> <p>The list of available components is presented via a the UI and the user can manipulate them, e.g. start them or enable/disable unsolicited alerts for the availability of these components.</p>
<p><b>Release</b></p> <p>0.8</p>
<p><b>Status</b></p> <p>Draft</p>

**Author**

Jörn Eisenbiegler (CAS), Michael Przybilski (Nokia), Tom Sutera (PHI, reviewer)

## 3.3.4 Peer-to-peer networking

<b>Feature name</b> F9: Peer-to-peer networking
<b>Priority</b> Mandatory
<b>Scope</b> System.
<b>Dependencies</b> Requires Network Transparency F1: Service Discovery F12/I6: Registry of (SW) components
<b>Description</b> Peer-to-peer networking allows users to communicate directly with one another, without any centralized server being involved. It allows users to set up collaborative working groups with access to each other's files or with automatic replication of shared information amongst the members of the group.
<b>Behavior</b> Use Case Collaboration between dynamic groups of multiple persons.
<b>Appearance</b> N/A.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Heike Messerschmidt, adisoft, H Smit (CIM), B Andréasson (CIM), Eugene M. Bourmakin (HUT, review), Tom Suters (PHI, review)

## 3.3.4.1 Use Case: Collaboration between dynamic groups of multiple persons

<b>Use case</b>	Collaboration between dynamic groups of multiple persons
<b>System under discussion</b>	System
<b>Primary actor</b>	An application
<b>Level</b>	System wide-specific
<b>Supporting actors</b>	Vivian terminals
<b>Stakeholders</b>	The end-user
<b>Pre-condition:</b>	An application is running on a Vivian terminal
<b>Minimal guarantee</b>	Peer-to-Peer connecting
<b>Success guarantee</b>	To start an interaction.
<b>Main success scenario</b>	This feature enables two or more users share the same (graphical) data. Viewing and manipulation of the data is allowed by all of them. A copy of the shared data will be stored locally at will as the participant resigns. Users with Vivian terminals should be able to provide different kind of interactions for each other in an ad-hoc network. A group of people is working with a collaborative application like a Whiteboard or business cards exchanging in an ad-hoc network.
<b>Exceptions</b>	
<b>Release</b>	0.8
<b>Status</b>	Draft
<b>Authors</b>	J Tuominen & M. Przybilski (Nokia). Eugene M. Bourmakin (HUT, review), Tom Sutera (PHI, review),

## 3.3.5 Ad-hoc networking over different networks

<b>Feature name</b> E12: Ad-hoc networking over different networks
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Requires Network Transparency F1:Service Discovery
<b>Description</b> This feature enables two or more devices to form a communication channel in between them and take it down whenever and without any warning.
<b>Behavior</b> A number of Vivian enabled terminals, which are using different transport protocols organizing an ad-hoc network. Service Discovery components needs to installed on these terminals.
<b>Appearance</b> As the connection is up, some kind of notification to the user should appear somewhere in the user-interface.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> J. Tuominen & M. Przybilski (Nokia). Reviewed by Eugene M. Bourmakin (HUT).

### 3.4 Security

The security features of the Vivian platform provides different authentication, encryption and access control mechanisms to applications, services and users..

Features 'S1: User authentication to a service provider' , 'S2: Provider Authentication' , 'S9: User Authentication to the Terminal' , 'S10: Store in protected form the user's security information' , 'S14: User Authentication to Applications' and 'E10: Platform should offer different customized user authentication methods' deal with authentication of various parties.

Features 'S11: Digital signatures for transactions' , 'S4: Encrypted communications' , 'S5: Encrypted Data' and 'S13: Able to hook up different encryption systems' offer encryption support to the Vivian platform.

Features 'S3: Explicit access/denial to services' , 'S6 : API Access Control' limit access to services.

Finally 'E9: Not affected too much by security/privacy' controls that the security features do not disturb the user with unnecessary security matters.

## 3.4.1 User authentication to a service provider

<b>Feature name</b> S1: User authentication to a service provider
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Requires Service Discovery User Interface S9: User authentication to terminal S3: Explicit access/denial to services S4: Encrypted communication E9: Not affected too much by security
<b>Description</b> To grant explicit access/denial to services it is necessary to identify the accessing user. Further necessities result from the potential costs of services that are to be charged to the user, and the possibility to individualize services.
<b>Behavior</b> Different means (HW and SW) of authentication can be applied depending on the service. The simplest way is usually a user name and a password. Any hardware identification shall not be mandatory..
<b>Appearance</b> The UI depends on the required information the user has to provide. It should be kept in mind that the terminal capabilities for input may be restricted.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Heike Messerschmidt (ISO)

## 3.4.2 Service Provider authentication

<b>Feature name</b> S2: Service Provider Authentication
<b>Priority</b> Optional
<b>Scope</b> System
<b>Dependencies</b> Requires S11: Digital Signatures.
<b>Description</b> There should be a defined system for a user to check and hold service provider certificates.
<b>Behavior</b>
<b>Appearance</b> There is no direct need for a user interaction with the system. A graphical interface to list, delete and view stored certificates would be nice to have.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Jörn Eisenbiegler (CAS)

## 3.4.3 Explicit access/denial to services

<b>Feature name</b> S3: Explicit access/denial to services
<b>Priority</b> Optional
<b>Scope</b> System
<b>Dependencies</b> Requires S1: User authentication to a service provider S6: API access control S9: User authentication to terminal User Interface
<b>Description</b> There should be a way to configure explicit access or denial to services for users. This feature guarantees control of costs and contents.
<b>Behavior</b> When the configuration is called up, a list of available and, if existing, already configured services must appear. The user should then be allowed to configure the service access control on various levels: <ul style="list-style-type: none"> <li>• Which services shall be allowed on what devices?</li> <li>• Who is allowed/excluded to a service at all?</li> </ul>
<b>Appearance</b> The appearance of the UI should fulfil all specified necessities. For the intended effect of access control to services it is vital that the user has to authenticate to the tool.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Heike Messerschmidt (ISO)

## 3.4.4 Encrypted communication

<b>Feature name</b> S4: Encrypted communications
<b>Priority</b> Optional
<b>Scope</b> System
<b>Dependencies</b> Requires S5: Encrypted data
<b>Description</b> A secure communications channel between two parties that can't be eavesdropped without knowing a secret key.
<b>Behavior</b> All data sent through a network channel is encrypted.
<b>Appearance</b>
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Tero Lyytikäinen (PAR)

## 3.4.5 Encrypted data

<b>Feature name</b> S5: Encrypted Data
<b>Priority</b> Optional
<b>Scope</b> System
<b>Dependencies</b> Requires S13 : Able to hook up different encryption systems
<b>Description</b> Data can be encrypted so that it's contents are disguised with a secret key.
<b>Behavior</b> Data is encrypted with a key. Data can then be decrypted to the original form with the same key (symmetric algorithm) or different key (asymmetric algorithm).
<b>Appearance</b>
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Tero Lyytikäinen (PAR)

## 3.4.6 API access control

<b>Feature name</b> S6 : API Access Control
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Requires Specific Components User Interface S2: Service Provider Authentication S9: User Authentication to the terminal
<b>Description</b> Depending on the identification of the user, or the service provider, the application may or may not be allowed to use certain APIs of the Vivian terminal, e.g. a phone API to prevent unauthorized dialing of (expensive) phone numbers or APIs to access credit-card or other personal information. For each Vivian API it shall be stated which access rights are needed (e.g. free, trusted provider, ...)
<b>Behavior</b> A Vivian terminal shall check whether for each application if it has the required access rights for the APIs it needs to access. This check may be done at installation-, starting- or run-time. If the application does not have the required access rights, the Vivian platform shall inform the user who can then decide to stop or continue the application. If a user decision is not possible (e.g. the terminal is left unattended) or if the user has not been authenticated, the application will be stopped. Only an authenticated user shall be able to permanently set or change access rights for APIs or to temporarily grant them for a particular application.
<b>Appearance:</b> If an application tries to access an API for which it has no privilege, the user will be notified and the application will not be installed or started or will be stopped.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Tom Sutera (PHI)

## 3.4.7 User authentication to the terminal

<b>Feature name</b> S9: User Authentication to the Terminal
<b>Priority</b> Mandatory
<b>Scope</b> Terminal
<b>Dependencies</b> Requires User Interface
<b>Description</b> There should be a mechanism to identify the person using the terminal at the moment.
<b>Behavior</b> The behavior of this feature depends on the authentication mechanism available on the platform and chosen by the user. This can vary from none (Whoever turns on the terminal is said to be a certain person) to complex biometrics systems like eye-scans. As soon as the user is identified to the terminal the applications have access to the user-application-authentication service.
<b>Appearance</b> The appearance of this feature strongly depends on the authentication methods used.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Jörn Eisenbiegler (CAS)

## 3.4.8 Store in protected form the user's security information

<b>Feature name</b> S10: Store in protected form the user's security information
<b>Priority</b> Optional
<b>Scope</b> Terminal
<b>Dependencies</b> Requires S5: Encrypted data S9: User authentication to the terminal User Interface
<b>Description</b> User security information is stored so that only a specific user can access it.
<b>Behavior</b> User data, like passwords, private keys, are encrypted so that only the user can use it by giving some authentication, like a PIN-code, password etc.
<b>Appearance</b> User is prompted to authenticate when user's security information is needed.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Tero Lyytikäinen (PAR)

## 3.4.9 Digital signatures for transactions

<b>Feature name</b> S11: Digital signatures for transactions
<b>Priority</b> Optional
<b>Scope</b> System
<b>Dependencies</b> Requires S5 : Encrypted data S13 : Able to hook up different encryption systems Component Interoperability Future Proof & Extendibility
<b>Description</b> In order to prevent unauthorized modifications of information, a signature will be appended to data.
<b>Behavior</b> A hash-code is computed from data and it is encrypted with asymmetric cipher algorithm.
<b>Appearance</b> No UI needed
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Tero Lyytikäinen (PAR)

## 3.4.10 Able to hook up different encryption systems

<b>Feature name</b> S13: Able to hook up different encryption systems
<b>Priority</b> Optional
<b>Scope</b> System
<b>Dependencies</b> Requires Future Proof & Extendibility
<b>Description</b> Application should be able to install new encryption components that are needed.
<b>Behavior</b> An extendable CryptoAPI that uses secured encryption components that are installable by software.
<b>Appearance</b>
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Tero Lyytikäinen (PAR)

## 3.4.11 User authentication to the application

<b>Feature name</b> S14: User Authentication to Applications
<b>Priority</b> Optional
<b>Scope</b> Terminal
<b>Dependencies</b> Requires User Interface Specific Components
<b>Description</b> The system could have mechanism that allows to get the authentication of the current user in different ways. The application needing an authentication of the user should be able to get it in a transparent way.
<b>Behavior</b> An application needing a user authentication just calls the authentication service. The user can pre-configure this service or enter the password/certificate at run time. There is a difference between the authentication of the user to the authentication service and the authentication the application wants to get. The authentication has to store the certificates or passwords for each application. It might be useful to be able to store more then one certificate or password for an application and user. In this case the application has to provide a parameter which describes the desired password.
<b>Appearance</b> There must be a user interface to configure the authentication service and to enter passwords or certificates.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Jörn Eisenbiegler (CAS)

## 3.4.12 Not affected too much by security / privacy

<b>Feature name</b> E9 : Not affected too much by security/privacy
<b>Priority</b> Optional
<b>Scope</b> Terminal
<b>Dependencies</b> Requires Future Proof & Extendibility Specific Components
<b>Description</b> The user is notified only by uncommon tasks. The user is in confidence with the device and for its data. The device accepts some common tasks concerning its security and privacy.
<b>Behavior</b> Notifications are uncommon and relevant. The device knows which kind of personal data or resources maybe given by default.
<b>Appearance</b>
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Dominique Dutoit, Patrick de Torcy (MEM)

## 3.4.13 Terminal should support different customised user authentication methods

<b>Feature name</b> E10: Terminal should support different customized user authentication methods
<b>Priority</b> Optional
<b>Scope</b> Terminal
<b>Dependencies</b> Requires S1: User authentication to a service provider User Interface Future Proof & Extendibility
<b>Description</b> The terminal should support installable authentication components which implement different authentication methods.
<b>Behavior</b> Different service providers might use different authentication methods. Each authentication system supplier can provide a component for the authentication subsystem. Each application that runs on the terminal will use one of the authentication methods the terminal supports. The user may choose from the authentication methods that he currently prefers and that the application supports. Example authentication methods are: <ul style="list-style-type: none"> <li>• Username / Password based authentication</li> <li>• Card reader based authentication (Smart card, Credit card)</li> <li>• Biometrics authentication (Fingerprint, Voiceprint, facial features)</li> </ul>
<b>Appearance</b> Each authentication component may present its own UI..
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Christian Abeln (CAS)

### 3.5 Network Transparency

Vivian is a distributed system consisting of multiple terminals connected via a network of (wireless and fixed) networks. Applications and services executing on these terminals want to see a 'distributed processing environment' where either all or none of the required components are available and executing as expected.

This 'Network Transparency' does not exist in the real world. Network connections between terminals may fail temporarily e.g. caused by the handoff between cells in a wireless cellular network like GSM. Applications and services themselves may fail because of abnormal abortion or user initiated actions. Network Transparency defines how the Vivian system should deal with these imperfections of the 'real' distributed processing environment.

Another aspect of 'Network Transparency' is to give applications and services a uniform network access mechanism based on the quality of service (QoS) they offer but that hides the differences between the underlying networks.

## 3.5.1 Common API for network access

<b>Feature name</b> F15/P3: Common API for Network Access
<b>Priority</b> Mandatory
<b>Scope</b> Terminal
<b>Dependencies</b> Modifies: E15/E4: Transparency to network disconnection.
<p><b>Description</b></p> <p>The terminal should have a simple to use set of APIs for applications to manage network (dis)connections. Nonetheless the API should allow to choose between the QoS (bandwidth, delay, error rates, cost, ...) of these network connections.</p> <p>An abstraction layer should hide different network protocols, authentication and encryption methods from the applications. The terminal should wherever possible not be bound to a specific network (e.g. GSM, UMTS, Bluetooth, IR) or network provider.</p> <p>The terminal may support one ore more of the network protocols (terminal dependent):</p> <ul style="list-style-type: none"> <li>• TCP/IP for Serial Link Networking(PPP/SLIP) and LAN connections</li> <li>• IrDA (OBEX)</li> <li>• Bluetooth</li> </ul> <p>The API instead should offer specialized support for Internet Protocols, Remote File access, Internet Control Message Protocol (ICMP) and Telephony.</p>
<p><b>Behavior</b></p> <p>The system allows to open (and close) a network connection with a chosen set of characteristics. The set of characteristics available can be asked for.</p>
<p><b>Appearance</b></p> <p>There is no direct need for a GUI to this feature. A GUI based configuration tool for network characteristics is nice to have.</p>
<p><b>Release</b></p> <p>0.8</p>
<p><b>Status</b></p> <p>Draft</p>
<p><b>Author</b></p> <p>Jörn Eisenbiegler, CAS Software AG, Tom Suters (PHI, review))</p>

## 3.5.1.1 Use case "Network status change notification"

<b>Use case name (goal)</b>	Network status change notification.
<b>System under discussion</b>	VIVIAN terminal.
<b>Primary actor</b>	Terminal, system, application.
<b>Level</b>	System wide - specific.
<b>Supporting actors</b>	
<b>Stakeholders</b>	The end-user.
<b>Pre-condition:</b>	Changes at the network possible status (reserved or available bandwidth, latency, ...) announced by QoS.
<b>Minimal guarantee</b>	No application affected.
<b>Success guarantee</b>	Getting notification from QoS in time.
<b>Main success scenario</b>	<ol style="list-style-type: none"> <li>1. The client application sends request to network for getting current QoS parameters.</li> <li>2. Client has got notification about QoS parameters.</li> <li>3. The application is trying to execute its goals.</li> </ol>
<b>Exceptions</b>	<ol style="list-style-type: none"> <li>1a QoS does not exist at the network <ol style="list-style-type: none"> <li>1a1 The client application is trying to work anyway.</li> </ol> </li> </ol>
<b>Release</b>	0.8
<b>Status</b>	Draft
<b>Author</b>	Lora Samarina (INR)

## 3.5.2 Atomic transactions, resync and roll-back mechanisms

<b>Feature name</b> R7/R10 : Atomic transactions, resync and roll back mechanisms
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Modifies Security E7/O3 : Resuming long downloads E15/E4: transparency to network disconnection R1/R2/R3/R8/R9: Graceful degradation against failure of components
<b>Description</b> In order to protect data transfers from failures of network or VIVIAN components and resulting loss/corruption of data, such a feature should be implemented. These mechanisms allow to group several modification events in a single sequence and to obtain the guarantee that such a sequence will be applied completely or not at all.
<b>Behavior</b> Beginning and end of an atomic part of the transaction must be clearly indicated. If a roll back is necessary, the whole atomic part must be repeated completely, with respect to security issues. As a roll back always causes additional data transfers, the size of an atomic transaction should be carefully considered. Resync must be launched automatically, sync points must be stored connection-dependent.
<b>Appearance</b> No UI needed.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Heike Messerschmidt (adisoft), Bruno Bretelle (Review, INT), Tom Suters (PHI, review)

## 3.5.3 Transparency to network disconnection

<b>Feature name</b> E15/E4: Transparency to network disconnection
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Requires User Interface Modifies F15/P3/F14: Common API for network access R1/R2/R3/R8/R9: Graceful degradation against failure of components E7/O3: Resuming long downloads R7/R10: Atomic transactions, resync and roll-back mechanisms
<b>Description</b> A disconnection from the physical network should be transparent for an application as long as a logical connection is still established. The state of the connection is maintained even through network failures.
<b>Behavior</b> After a disconnection and reconnection sequence, voluntary or not, state of connections in the Vivian system just before disconnection is automatically restored, without user intervention. A UI status monitor can optionally appear on screen. Use case Voluntary disconnection by the application Use case Involuntary disconnection
<b>Appearance</b> A GUI dialog box to the user of the terminal to notify him(her) a disconnection. Optional UI status monitor.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Authors</b> Guy Bernard, Bruno Bretelle and Denis Conan (INT), David Mentré (Review, INR), Tom Suters (PHI, review)

## 3.5.3.1 Use case “Voluntary disconnection by the application”

<b>Use case name</b>	Voluntary disconnection by the application/service
<b>System under discussion</b>	Vivian terminal
<b>Primary actor</b>	An application/service
<b>Level</b>	System wide-specific
<b>Supporting actors</b>	Vivian terminal
<b>Stakeholders</b>	The end-user
<b>Pre-condition:</b>	An application/service is running on a Vivian terminal.
<b>Minimal guarantee</b>	Data integrity and application/service shutdown.
<b>Success guarantee</b>	Connections closed
<b>Main success scenario</b>	<p>The part of the application which is running on the terminal decides to close all its connection with the network</p> <p>it notifies to the Vivian terminal the close of its connections</p> <p>The Vivian terminal releases resources used by connections of this application and sends a acknowledgement message to the application</p>
<b>Exceptions</b>	
<b>Release</b>	0.8
<b>Status</b>	Draft
<b>Authors</b>	Guy Bernard, Bruno Bretelle and Denis Conan (INT), David Mentré (Review, INR), Tom Suturs (PHI, review)

## 3.5.3.2 Use case “Involuntary disconnection”

<b>Use case name</b>	Involuntary disconnection
<b>System under discussion</b>	Vivian terminal
<b>Primary actor</b>	The network
<b>Level</b>	System
<b>Supporting actors</b>	Running applications/services
<b>Stakeholders</b>	The end-user
<b>Pre-condition:</b>	An application/service is running on a Vivian terminal.
<b>Minimal guarantee</b>	Data integrity and application/service shutdown.
<b>Success guarantee</b>	Connections closed
<b>Main success scenario</b>	<ol style="list-style-type: none"> <li>1. The network cuts a connection</li> <li>2. The Vivian terminal detects a disconnection</li> <li>3. It tries to reconnect to the network and uses a timeout</li> <li>4. Before the timeout expires, the reconnection succeeds</li> </ol>
<b>Exceptions</b>	<ol style="list-style-type: none"> <li>1. The timeout expires and no reconnection is possible, the terminal signals blocked applications/services <ol style="list-style-type: none"> <li>1.a. The Vivian terminal sends a notification of “impossible connection” to the user</li> </ol> </li> </ol>
<b>Release</b>	0.8
<b>Status</b>	Draft
<b>Authors</b>	Guy Bernard, Bruno Bretelle and Denis Conan (INT), David Mentré (Review, INR), Tom Suturs (PHI, review)

## 3.5.4 Resuming long downloads

<b>Feature name</b> E7/O3: Resuming (long) downloads
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Requires User Interface F15/P3/F14: Common API for network access E15/E4: Transparency to network disconnection R7/R10: Atomic transactions, resync and roll back mechanisms R1/R2/R3/R8/R9: Graceful degradation against failing components
<b>Description</b> It should be possible to resume downloads after a broken connection instead of re-initializing them. The main issues are cost reduction and transfer acceleration.
<b>Behavior</b> After re-establishment of a broken connection the download is resumed automatically.
<b>Appearance</b> No UI needed here, but a message box announcing the re-established connection is a nice-to-have.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Heike Messerschmidt (adisoft), David Mentré (Review, INR), Tom Suters (PHI, review)

## 3.5.5 Graceful degradation against component failures

<b>Feature name</b> R1/R2/R3/R8/R9: Graceful degradation against component failures
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Modifies Resource Management Security Service Discovery F15/P3/F14: Common API for network access E15/E4: Transparency to network disconnection
<b>Description</b> Component failure can be caused by (a) resource scarcity on the terminal, or (b) an order sent by the user, or (c) an order sent by another component (for a security or privacy reason) or (d) because of a network failure. This feature ensures that when a component fails, its users and all other components that interact with it are informed of it. The notified components and their users should continue working in a stable way, possibly with reduced functionality. Dependent on the severity of the failure, components may ask to be notified again when the failed components re-appear in the system.
<b>Behavior</b>
<b>Appearance</b> A GUI dialog box to the user of the terminal to notify him(her) of a (partially) failed component. The user might notice reduced functionality. The user can be notified (at application will) if the failed component has been recovered.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Authors</b> Guy Bernard, Bruno Bretelle and Denis Conan (INT), Eugene Bourmakin (HUT), Michael Przybilski (NOK), Heike Messerschmidt (ADI), David Mentre (INR), Tom Sutera (PHI, review)

### 3.6 Terminal Resource Management

This chapter presents resource management features of the Vivian terminals such as PDAs (Personal Digital Assistants). PDAs are constrained by their resources: memory, network access (cost) and battery life are limited. It is therefore crucial to exploit those resources in a wise manner, otherwise usability of the PDA could be impacted and this could even block its wide acceptance. Exploiting resources wisely means knowing the terminal environment and adapting dynamically resource consumption of the terminal according to this environment.

Those requirements are addressed by features "F18: adaptive resource management of the terminal", They allow a Vivian terminal and its applications/services to work in any environment, even if terminal resources such as memory and external connectivity are temporarily unavailable at the cost of reduced performance or functionality. Applications and services may also temporarily close a connection to save on air-time, while logically they remain connected.

Another constraint on resource management is that several applications on the same terminal (i.e. the same PDA) should share resources optimally and, at the same time, they should not impact each other, intentionally or not. It is therefore necessary to have mechanisms that explicitly allow this resource sharing while at the same time protecting an application from the others running on the same terminal.

Features "R4: Peaceful coexistence between applications", "R5: Resource sharing mechanisms", "R11: Power save modes and application restart" and "S7: Memory protection" allow several applications on the same terminal to share terminal resources.

## 3.6.1 Adaptive resource management of the terminal

<b>Feature name</b> F18: Adaptive resource management of the terminal
<b>Priority</b> Mandatory
<b>Scope</b> Terminal
<b>Dependencies</b> Modifies R4: Peaceful coexistence between applications R5: Resource sharing mechanisms R11: Power save modes and application restart Network Transparency Service Discovery Specific Components
<b>Description</b> Terminals should handle gracefully variations in terminal working conditions and in terminal environment, with minimal user interaction. Parameters that can change are: (a) connectivity, (b) memory, (c) bandwidth, (d) processor load and (e) power. Under new working conditions, terminal's applications and services should reconfigure accordingly. Vivian terminal takes appropriate action to adapt to new working conditions.
<b>Behavior</b> Use case resource reallocation Use case resource negotiation
<b>Appearance</b> None
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> INRIA (INR)

## 3.6.1.1 Use case 'Resource reallocation'

<b>Use case name (goal)</b>	Adapt to resource starvation through resource reallocation
<b>System under discussion</b>	Vivian terminal
<b>Primary actor</b>	An application (or service)
<b>Level</b>	System wide – specific
<b>Supporting actors</b>	None
<b>Stakeholders</b>	User
<b>Pre-condition</b>	Applications are running on a Vivian terminal. A new resource is requested but the terminal cannot offer it.
<b>Minimal guarantee</b>	No application is killed.
<b>Success guarantee</b>	No application is stopped.
<b>Main success scenario</b>	<ol style="list-style-type: none"> <li>1. An application asks the terminal for a new resource</li> <li>2. The terminal cannot handle the request due to resource starvation</li> <li>3. The terminal reallocates resources amongst all running applications</li> <li>4. Initial resource request satisfaction is possible</li> <li>5. The terminal answers initial demand</li> <li>6. The requesting application runs correctly</li> </ol>
<b>Exceptions</b>	<ol style="list-style-type: none"> <li>4.a. Answering initial request is not possible <ol style="list-style-type: none"> <li>4.a.1. The terminal suspends a running application</li> <li>4.a.2. Initial resource request satisfaction is possible</li> <li>4.a.3. The terminal answers initial demand</li> <li>4.a.4. The requesting application runs correctly</li> </ol> </li> <li>4.b. Answering initial request is not possible <ol style="list-style-type: none"> <li>4.b.1 The terminal suspends the requesting application</li> </ol> </li> </ol>
<b>Release</b>	0.8

<b>Status</b> Draft
<b>Author</b> INRIA (INR)

## 3.6.1.2 Use case 'Resource Negotiation'

<b>Use case name (goal)</b> Adapt to resource starvation through negotiation
<b>System under discussion</b> Vivian terminal
<b>Primary actor</b> An application (or service)
<b>Level</b> System wide – specific
<b>Supporting actors</b> - Vivian terminal - Other applications
<b>Stakeholders</b> User
<b>Pre-condition</b> Applications are running on a Vivian terminal. A new resource is requested but the terminal cannot respond to it.
<b>Minimal guarantee:</b> No application is killed.
<b>Success guarantee:</b> No application is stopped.
<b>Main success scenario</b> <ol style="list-style-type: none"> <li>1. An application ask the terminal for a new resource</li> <li>2. The terminal cannot handle the request due to resource starvation</li> <li>3. The terminal negotiate with available application to redistribute current resources</li> <li>4. Initial resource request satisfaction is possible</li> <li>5. The terminal answers initial demand</li> <li>6. The requesting application runs correctly</li> </ol>
<b>Exceptions</b> <ol style="list-style-type: none"> <li>4.a. Answering initial request is not possible <ol style="list-style-type: none"> <li>4.a.1. The terminal suspends a running application</li> <li>4.a.2. Initial resource request satisfaction is possible</li> <li>4.a.3. The terminal answers initial demand</li> <li>4.a.4. The requesting application runs correctly</li> </ol> </li> <li>4.b. Answering initial request is not possible <ol style="list-style-type: none"> <li>4.b.1 The terminal suspends requesting application</li> </ol> </li> </ol>
<b>Release</b> 0.8

<b>Status</b> Draft
<b>Author</b> INRIA (INR)

## 3.6.2 Peaceful co-existence between applications

<b>Feature name</b> R4 : Peaceful co-existence between applications
<b>Priority</b> Mandatory
<b>Scope</b> Terminal
<b>Dependencies</b> Requires User Interface F18: adaptive resource management of the terminal R5: Resource Sharing Mechanism
<b>Description</b> An application (or service) shall not be able to permanently, maliciously or inadvertently deny access to the resources of a Vivian terminal by other applications. It shall always be possible for a user to switch between all available and running applications.
<b>Behavior</b> The Vivian terminal shall be able to reclaim an application's resources at any time and assign them to other applications.
<b>Appearance</b> The user has the possibility via a simple UI mechanism to get an overview of all applications (either running or not) and start or switch to them.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Tom Suters (PHI)

## 3.6.3 No unnecessary connections

<b>Feature name</b> C2: No unnecessary connections
<b>Priority</b> Mandatory
<b>Level</b> Terminal
<b>Dependencies</b> Requires: Network Transparency
<b>Description</b> A physical connection is only established if an application wants to transfer data. As soon as no more data needs to be transmitted, or the QoS falls under a certain threshold the network connection is terminated (cost & power optimization).
<b>Behavior</b> The logical connection between the mobile device and the central application is handled independently of the transmission capability of the network. Temporary message storing and signaling of sending readiness is done automatically. The logical connection will remain open as long as the application is still active.
<b>Appearance</b> The user may be notified that the connection has been temporarily closed..
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Heike Messerschmidt (adisoft), Tom Sutera (PHI, review)

## 3.6.4 Resource sharing mechanisms

<b>Feature name</b> R5: Resource sharing mechanisms
<b>Priority</b> Mandatory
<b>Scope</b> Terminal.
<b>Dependencies</b> Modifies F18: Adaptive resource management of the terminal R4: Peaceful co-existence between applications
<b>Description</b> Based on PDA Operating System this feature shall provide resources sharing for applications
<b>Behavior</b> Possible resource sharing mechanisms: shared libraries, shared memory segments, application specific scheduler, shared objects, file system, ...
<b>Appearance</b> None
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Michael Samarin (HUT), INRIA (INR)

## 3.6.5 Power save mode and application restart

<b>Feature name</b> R11: power save mode and application restart
<b>Priority</b> Mandatory
<b>Scope</b> Terminal
<b>Dependencies</b> Modifies F18: Adaptive resource management of the terminal C2: No unnecessary connections
<b>Description</b> The terminal shall monitor user activity and disable unneeded terminal features (hardware and software components) automatically in order to reduce energy consumption. This reduction will extend terminal usability in autonomous mode.
<b>Behavior</b> When the user is no longer using a feature of the terminal, components (hardware and software) implementing this feature are automatically disabled, without user intervention. Symmetrically, when the user starts to use a feature, corresponding components are reactivated. Deactivation and reactivation of components should be done with minimal user impact, i.e., this mechanism should be transparent for the user. If an application is suspended (i.e. it goes in "sleep" mode), it should be restored in exactly the same state when reactivated. Power save modes should not impact the behavior of other terminals.
<b>Appearance</b> None
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> INRIA (INR)

## 3.6.6 Memory protection

<b>Feature name</b> S7: Memory protection
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b>
<b>Description</b> Vivian platform should offer a memory protection such that a memory fault in an application (or service) does not impact other running applications or the system.
<b>Behavior</b> Use case: Application memory fault
<b>Appearance</b> A GUI dialog to signal a fault
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> INRIA (INR)

## 3.6.6.1 Use case 'Application memory fault'

<b>Use case name (goal)</b> Protect system from a memory faulting application
<b>System under discussion</b> Vivian terminal
<b>Primary actor</b> An application (or service)
<b>Level</b> System wide - specific
<b>Supporting actors</b> Vivian terminal
<b>Stakeholders</b> - other applications - user
<b>Pre-condition:</b> One or more applications are running on a Vivian terminal.
<b>Minimal guarantee</b> Non faulty application are not impacted.
<b>Success guarantee</b> Faulty application is killed and an error is signaled to the user.
<b>Main success scenario</b> 1. A running application triggers a memory fault (memory access outside its allowed memory area) 2. An exception is triggered by hardware 3. Hardware exception is caught by Vivian terminal and used to find the faulting application 4. The faulty application is killed 5. An error message is sent to the user
<b>Exceptions</b> 4a. The faulty application is not stopped but restarted 4a1. The application context is cleaned up 4a2. The application is restarted in a new context 4a3. An information message is sent to the user 4b. The faulty application has been check-pointed 4b1. The last checkpoint is restored 4b1a. Maximum number of restorations for this application has been reached 4b1a1. The faulty application is killed 4b2. The application is restarted from last checkpoint 4b3. An information message is sent to the user

5a. No error message is sent to user 5a1. The error message is logged in a system specific record
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> INRIA (INR)

### 3.7 Terminal Configuration Management

The category studies :

- The description of each terminal hardware with a standard profile
- The description of a terminal according to its capabilities and accessibility
- The accessibility by the applications to the hardware and OS profile
- The combination of application and hardware

## 3.7.1 Profiles for different terminals

<b>Feature name</b> I3 : Profiles for different terminals
<b>Priority</b> Mandatory
<b>Scope</b> Terminal.
<b>Dependencies</b> Modifies Component Interoperability I4/E13: Terminal capability enquiry capability P5/P6: Profiles for application families
<b>Description</b> Profiles for different terminal hardware shall be specified in order to allow the deployment of the same application or component on terminals with slightly different hardware (e.g. devices with touch screen, keyboard, integrated mobile phone, etc).
<b>Behavior</b>
<b>Appearance</b>
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Michael Przybilski (NOK), Dominique Dutoit (MEM), Tom Sutera (PHI, review)

## 3.7.2 Terminal capability enquiry mechanism

<b>Feature name</b> I4/E13: Terminal capability enquiry mechanism
<b>Priority</b> Mandatory
<b>Scope</b> Terminal.
<b>Dependencies</b> Requires I3: Profiles for terminal capabilities
<b>Description</b> This feature enables an application or service running on a Vivian terminal to retrieve the different terminal capabilities (e.g. on User Interface devices)
<b>Behavior</b>
<b>Appearance</b>
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Michael Przybilski (NOK), Dominique Dutoit (MEM), Tom Suters (PHI, review)

## 3.7.3 Profiles for application families

<b>Feature name</b> P5/P6: Profiles for application families
<b>Priority</b> Mandatory
<b>Scope</b> System.
<b>Dependencies</b> Requires I3: Profiles for different terminals
<b>Description</b> Since terminals have different hardware capabilities, different versions of the same application should be specified, allowing the same application to be used on different hardware. This feature would also allow a very good means of product differentiation, since it could be specified, that a certain member of an application family is only available for a certain hardware device.
<b>Behavior</b>
<b>Appearance</b>
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Michael Przybilski (NOK), Dominique Dutoit (MEM), Tom Sutera (PHI, review)

### 3.8 Future Proof & Extensibility

In the future new hardware and software become available. Therefore, the Vivian specification must support a wide range of hardware and software. This is achieved by the requirements of the future proof & extensibility category.

Feature "P1: HW / SW independence" requires that the Vivian specification shall not imply the use of any particular HW or SW products.

Feature "P4/O4: Based on open specifications/standards" requires that the Vivian specification does not imply the use of technologies with unreasonable licensing conditions.

Feature "O1: Installable SW" requires that installable software is component-based and it can be installed and updated as easily as possible.

Feature "E2: Plug and Play: no cables to connect to peripherals and other terminals" allows using wireless connections are to connect to other terminals and devices.

Feature "E3: User Input Device Independence" requires that the Vivian specification shall not imply the use of any particular input device.

## 3.8.1 HW &amp; SW Independence

<b>Feature name</b> P1: HW & SW Independence
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Modifies Component Interoperability Requires E3: User Input Device Independence P4/O4: Vivian based on open standards
<b>Description</b> The Vivian specification shall not imply the use of any particular HW or SW products that are owned and sold on the market by specific vendors such as processor(s), wired or wireless communication interfaces, I/O devices, storage medium, (Real-Time) Operating Systems, Database Systems, etc.
<b>Behavior</b> Specific terminal capabilities and the HW and SW products that it uses shall be hidden from the APIs of the Vivian terminal to give the terminal manufacturers the freedom to choose the most appropriate HW and SW products as technology develops without compromising the inter-operability with existing applications and services.
<b>Appearance</b>
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Tom Suters (PHI)

## 3.8.2 Based on open specifications/standards

<b>Feature name</b> P4/O4: Based on open specifications/standards
<b>Priority</b> Mandatory
<b>Scope</b> System
<b>Dependencies</b> Modifies: Component Interoperability
<b>Description</b> Technical specifications that are part of or referenced by the Vivian specification and that are needed to make any Vivian compliant product, service or application as allowed by the specification, shall be freely accessible and available, and the right to implement and apply these technologies for both non-commercial and commercial purposes such as a product, service or application will be licensed on reasonable and non-discriminatory terms.
<b>Behavior</b>
<b>Appearance</b>
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> INRIA(INR), Jari Hennilä(UNI, review), Tom Suturs (PHI, review)

3.8.3 Installable (3<sup>rd</sup> party) SW Components

<p><b>Feature name</b></p> <p>O1/F2/F13/E6/E16: Installable (3<sup>rd</sup> party) SW Components</p>
<p><b>Priority</b></p> <p>Mandatory</p>
<p><b>Scope</b></p> <p>Terminal</p>
<p><b>Dependencies</b></p> <p>Requires Component Interoperability</p>
<p><b>Description</b></p> <p>Installable software is composed of one or more components. (Un)Installable components are pieces of software that can be (un)installed on the terminal. Once installed they become available. The components can be uninstalled when they are no longer used. The Vivian platform provides a mechanism to check for permissions, select options, query information from the user, download, install, uninstall, configure and finally execute the software.</p>
<p><b>Behavior</b></p> <p>Executable code</p> <p>There must be a standard executable code format that is supported by all terminals. Supporting other executable code formats is optional.</p> <p>Installable software</p> <p>Installable software is composed of one or more components. They may require the use of other components. It must be possible that one component can be used by more than one component.</p> <p>SW Components</p> <p>(Un)Installable SW components are pieces of software that can be installed on the terminal. They must be stored locally and assembled on the terminal to make them available. The components may require the use of other components and they must be able to be used by other components. The access mechanism to other components is provided by the system. The components provide information about the components they provide and require. They also contain other information: description, manufacturer, version, digital signature, terminal requirements, etc.</p> <p>Installation</p> <p>The software must be installed before it can be used. As the software is composed of one or more components the installation may require the installation of one or more components. The installation mechanism is not a part of the software but it is provided by the system. The installation should be as easy as possible. The configuration should be as automatic as possible. However, the procedure may include a dialogue with the user. The user may be asked to select options and confirm the actions or enter some information. If the installation includes downloading the software from the service provider the user and the service provider must be able to authenticate each other. If the authentication fails the installation can be cancelled by either of the parties. Possible signatures and other information stored in the components are checked and the installation can be cancelled if problems are encountered. The components are assembled to make them available. The software can</p>

also be automatically executed after the installation.

#### Uninstallation

The software can be uninstalled when it is no longer used. The uninstallation mechanism is provided by the system. As the software is composed of one or more components the uninstallation may include the uninstallation of one or more components. The component is uninstalled if its presence is no longer required.

#### Updating

There must be a mechanism to update the components. As the component is updated it is replaced by another version without affecting the components that need not be updated. Updating the component version may also require installing other components unless they are present.

#### Appearance

(Un)Installable components may enhance the software by adding new functionality to it, for example special multimedia components. They can also appear as terminal-specific device drivers.

The installation mechanism is consistent and easy for the end user. In addition, the software developers can use the installation mechanism provided by the Vivian platform.

#### Release

0.8

#### Status

Draft

#### Author

Erno Alhoniemi (UNI), Bruno Bretelle (INT, review), Tom Sutere (PHI, review)

## 3.8.4 Plug and Play: no cables to connect to peripherals and other terminals

<b>Feature name</b> E2: Plug and Play: no cables to connect to peripheral devices .
<b>Priority</b> Optional
<b>Scope</b> Terminal
<b>Dependencies</b>
<b>Description</b> The end user will access applications based on VIVIAN via mobile terminals. Various types of external peripheral devices may be connected to the mobile terminal, preferably as "plug and play devices". These devices need not be restricted to pure slave devices but may also include devices that deliver a service or application themselves such as a vending machine.
<b>Behavior</b> For the end user's convenience, the connected devices should integrate – from the end user point of view -as easily as possible. A typical example is a smart card reader (if not integrated as a standard part of the mobile device). The smart card reader is used for communication with, for example, the end-users bank/credit/electronic purse/multi-application (loyalty, ticketing) card, which is envisaged to be a JavaCard in the near future. The smart card reader is physically connected to the mobile terminal for example by a PC Card or via the USB (Universal Serial Bus). It may also be connected via a wireless interface like Bluetooth. In an ideal situation the smart card reader is automatically recognized, when inserted or connected, and integrated to be used by the application. The terminal should not need a-priori knowledge in order to be able to interact with it. Preferably, this process should be finalized without a need of restarting the mobile device. On a second level, a smart card inserted in the reader should also be handled as smoothly as possible with clear messages to the user. In a similar way, other peripheral units should be handled, for example, printers, biometrics identification devices, external keyboards, docking stations for communication with a desktop PC, storage devices like compact flash cards, cameras, etc.
<b>Appearance</b>
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> H Smit (CIM), B Andréasson (CIM), Tom Suturs (PHI, review)

## 3.8.5 User Input Device Independence

<b>Feature name</b> E3: User Input Device Independence
<b>Priority</b> Mandatory
<b>Scope</b> Terminal
<b>Dependencies</b> Modifies User Interface Other Components P1: HW & SW Independence
<b>Description</b> The Vivian specification shall offer a generic API that abstracts from the type of the input device. That eliminates the need to develop and maintain multiple different versions of applications to handle the diversity of input devices such as a mouse, keyboard, stylus or speech because some of them are not available on all terminals.
<b>Behavior</b>
<b>Appearance</b>
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Tom Suters (PHI), Erno Alhoniemi (UNI, review)

### 3.9 User Interface

With mobile devices we have a broad range of circumstances that affect the requirements for the user interface of an application. Applications that are used in automotive computing devices need to interact completely different with the user in comparison to Smart Phones, mobile internet driven music players, or even semi-mobile devices that are used very similar to a laptop.

We should rate the numerous devices that already exist and the applications they are running along two ratios of user interface experience.

- Limitation of user input capabilities
- Limitation of output capabilities

The limitations may be forced by the desired usage environment (effects of natural forces: e.g. dirt – no mechanical interaction, noise – no vocal interaction, network availability), user capability draw backs (user focused on driving or working, physical disabilities: e.g. no display or keypad interaction) or hardware restrictions caused by technology or design issues (e.g., small display, small keys).

As a rule of thumb we can understand that the existing limitations for an user interaction type should be compensated by a different interaction method that even may need a higher level of processing intelligence on the client itself or in a remote server application.

Examples of currently available Input methods are:

- Physical Interaction: Keypad, Pen, Touch sensitivity, Accelerometer, Fingerprint detection (identification)
- Acoustic interaction: Voice recognition
- Interaction with face/eyes: facet features (camera identification)

The Output methods currently available are:

- Physical Interaction: Braille, Vibrating (e.g. telephone ringing)
- Acoustic interaction: Sound, Voice generation, Text2Speech (server based)
- Interaction with face/eyes: Display

We can also distinguish several classes of information:

- Textual information: in form of Text or Voice
- Picture and sound data
- Input method specific data: Positioning of a cursor/caret, Button press, navigation

A generic platform for different mobile devices must abstract the user interaction methods and leave it up to the user or the device vendor to choose which input method to use to communicate an information class.

Moreover, each Input / Output method should look familiar to the user over different operating systems and applications.

## 3.9.1 Graphical manipulation

<b>Feature name</b> F8: Graphical manipulation
<b>Priority</b> Optional
<b>Scope</b> Terminal
<b>Dependencies</b> Requires E14/E8/E11: defined set of (G)UI elements Modifies E1: Consistent look, easy to learn, intuitive
<b>Description</b> The graphical appearance is made up by a "Canvas". It abstracts the display area to a 2 dimensional area for drawing. It provides basic information: <ul style="list-style-type: none"> <li>• Size of the drawing area</li> <li>• Capabilities of the drawing area: resolution, colors</li> </ul> It implements basic functions: <ul style="list-style-type: none"> <li>• Drawing (Pen, Brush, Line, Circle, Rectangle, Shapes, Color Palettes, Bitmap)</li> <li>• Formatted text output</li> <li>• Drawing and positioning of a "Cursor"</li> <li>• Drawing and positioning of a "Caret"</li> </ul> The "graphical Manipulating system" should be extendable with 3 D capabilities
<b>Behavior</b> <ul style="list-style-type: none"> <li>• Whiteboard application</li> <li>• Importing drawing: only the ability to import drawings and pictures and add them to the initial drawing</li> <li>• Altering drawings: the ability to manipulate a drawing</li> <li>• Making annotations</li> <li>• Adding shapes</li> </ul>
<b>Appearance</b>
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Michael Przybilski (NOK), Christian Abeln (CAS, reviewer)

## 3.9.2 UI Interaction

<b>Feature name</b> O2 : UI Interaction
<b>Priority</b> Mandatory
<b>Scope</b> Terminal
<b>Dependencies</b> Modifies Future Proof & Extendibility E1: Consistent Look, Easy to Learn, Intuitive E14/E8/E11: Defined Set of GUI elements F8: Graphical Manipulation
<b>Description</b> A Vivian terminal has full control over how to map the requested user interaction as provided by an application, to the User Interaction facilities of the terminal. The application is completely independent from the actual mechanisms that a particular terminal offers for user interaction (e.g. speech, sound, graphics, stylus, keyboard, etc). The Vivian specification shall specify a minimum set of User Interaction facilities that a Vivian terminal must provide. All user interactions specified by Vivian shall be implementable using this minimum set..
<b>Behavior</b> The user interacts with applications in the way that the terminal allows
<b>Appearance</b> When a remote application requests to present a panel containing several interactions and links to other panels, it is the terminal that decides how to present this. E.g. the panel links could be mapped to dedicated navigation buttons by one terminal or as buttons on the screen by another terminal. A value range can be presented as a number or as a slider, etc. Where and how the user interaction is shown on the screen is also determined by the terminal and may depend on the usage of the I/O facilities by other applications or capabilities such as 2D or 3G graphics. Each terminal shall use a mapping that provides a consistent and intuitive look and feel to the user.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Tom Suters (PHI)

## 3.9.3 Defined set of (G)UI elements

<p><b>Feature name</b></p> <p>E14/E8/E11: defined set of (G)UI elements</p>
<p><b>Priority</b></p> <p>Mandatory</p>
<p><b>Scope</b></p> <p>Terminal</p>
<p><b>Dependencies</b></p> <p>Modifies O2: User Interaction F8: Graphical Manipulation</p>
<p><b>Description</b></p> <p>There should be a defined set of often used building blocks for input and output of data.</p> <p>There must be a way to enter alphanumerical text on every VIVIAN platform.</p>
<p><b>Behavior</b></p> <p>The application can display and request data in different forms. Depending on the user interaction (edit an item, select an item, etc.) the application gets corresponding messages. The set of supported UI elements should at least consist of:</p> <ul style="list-style-type: none"> <li>• display and edit text: <ul style="list-style-type: none"> <li>• Entering text on the Vivian platform can be done in several ways</li> <li>• Keyboard</li> <li>• Virtual keyboard on a touch-screen</li> <li>• Recognition of handwriting</li> <li>• Voice-Recognition</li> <li>• Telephone-“keyboard” (T9)</li> <li>• Menu structure and navigate to menu items</li> </ul> </li> <li>• display and edit record sets</li> <li>• display lists and select an entry</li> <li>• interface, which allows to input or output sound, pictures, or video sequences. (optional) <ul style="list-style-type: none"> <li>• The application can ask the terminal which multimedia facilities are available on the device. There is also a classification of available input or output quality (resolution, colors, mono/stereo) and of input or output formats (mpegX, etc.)</li> <li>• The terminal provides the application with it's specific representation of the media content, which may be different on different terminals</li> </ul> </li> </ul>
<p><b>Appearance</b></p> <p>The appearance of the UI elements depends on the User Interaction facilities of the terminal. It could be possible to output data via display or sound. Input can be done via the text entry system or pointing devices.</p> <p>The appearance of the text entry mechanism depends on the output</p>

capabilities of the device. If there is no display the request to the user to enter some text must be transported via voice. In some cases it might be useful to repeat the text entered by the user for control. Special concern may be given to handling of hidden text link being used in password fields.

The appearance of the Multimedia UI for the user strongly depends on the hardware the application is running on. Therefore the application has first to question the terminal for available User Interaction facilities.

**Release**

0.8

**Status**

Draft

**Author**

Christian Abeln (CAS), Tom Suters (PHI, reviewer)

## 3.9.4 Consistent look, easy to learn, intuitive

<b>Feature name</b> E1 : Consistent look, easy to learn, intuitive
<b>Priority</b> Mandatory
<b>Scope</b> Terminal
<b>Dependencies</b> Modifies Future Proof & Extendibility E14/E8/E11: Defined Set of GUI elements F8: Graphical Manipulation O2: UI Interaction
<b>Description</b> <p>All applications for a Vivian terminal must follow certain guidelines as to how a user interacts via his terminal with an application to prevent unnecessary and confusing differences between applications.</p> <p>Each terminal has complete freedom to determine the Look-and-Feel (LAF) of its UI. This gives each Vivian terminal the possibility to optimize the user experience to its User Interaction facilities of the terminal and at the same time prevents that application developers need to develop and maintain many different versions of their UI to accommodate the differences between terminals.</p> <p>Each terminal itself shall offer a consistent LAF for all the user interaction mechanisms that it supports to make the operation of the terminal and its applications by the user both intuitive and easy to learn.</p>
<b>Behavior</b> N/A.
<b>Appearance</b> <p>E.g. a guideline should force all applications to use equal names for menu titles that are common for all applications such as the term “preferences” to indicate the possibility to set certain application parameters.</p> <p>The LAF of a text entry box will be different on a terminal with a Q-CIF graphical display compared to one with a 2-line text-only display and may depend on the mechanism that is used to input the text (keyboard, stylus).</p> <p>The setting of a value for a value range or the selection of one of several items from a list on a particular terminal shall have the same LAF for all applications.</p>
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Tom Suters (PHI)

### 3.10 Specific Components

This section contains the requirements for specific components (Applications, Services and Terminal Modules) and their APIs that are defined by Vivian. Some are mandatory and some are optional. These components represent a specific functionality available to other components either locally on the same terminal or over the network.

## 3.10.1 Persistent Data Storage

<b>Feature name</b> F10 : Persistent Data Storage
<b>Priority</b> Optional
<b>Scope</b> Terminal
<b>Dependencies</b> Modifies Security
<b>Description</b> This feature gives applications and services via the Terminal API access to persistent data storage on the terminal where they execute. The Persistent Data Storage is an optional Terminal Module. This requirement does not describe the need for persistent data storage as a service.
<b>Behavior</b> Applications and services can store, retrieve and delete data. Data remains stored even after the application that stored it has stopped. The data can be retrieved, changed or deleted at a later time by other (possibly instances of the same) applications if properly authorized. Multiple applications or services executing on a terminal shall not be aware of each other nor have access to each other's data when simultaneously using the persistent data storage unless properly authorized. Applications cannot access the Persistent Data Storage of another terminal. Encryption of the data can on request of the application be performed by the Persistent Data Storage. The data store should at least accept simple SQL-queries.
<b>Appearance</b> There is no GUI to this feature. If Java is the language for the VIVIAN platform, a JDBC-Interface is nice to have. Examples of already implemented persistent data storage technologies are database products for existing mobile terminals such as Oracle Light, SQL Anywhere, MS SQL Server2000 WinCE edition
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Jörn Eisenbiegler(CAS), Tom Sutera (PHI, reviewer), Christian Abeln (CAS, rewritten)

## 3.10.2 Terminal Specific Events

<b>Feature name</b> F6: Terminal Specific Events
<b>Priority</b> Mandatory
<b>Scope</b> Terminal
<b>Dependencies</b> Requires User Interface  Modifies Network Transparency Component Interoperability
<b>Description</b> A terminal; informs applications and services executing on a terminal when important terminal conditions occur (e.g. low battery power, loss of network connection, hardware hot plug-in/out)..
<b>Behavior</b> All applications and services shall on request be notified of terminal specific events they are interested in How a application or service handles the event is application- and service specific. Applications may initiate some user interaction asking the user e.g. to close connections or stop the application. Besides the notification of all applications and services, a general notification should be given to the user if the terminal supports user interaction
<b>Appearance</b> Notifications could display textual information in a striking manner or could generate sound, voice or physically perceptible output.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Christian Abeln(CAS), Tom Suters (PHI, reviewer)

## 3.10.3 Remote User Interface

<p><b>Feature name</b></p> <p>F3 : Remote User Interface</p>
<p><b>Priority</b></p> <p>Mandatory</p>
<p><b>Scope</b></p> <p>System</p>
<p><b>Dependencies</b></p> <p>Requires Component Interoperability Service Discovery User Interface</p>
<p><b>Description</b></p> <p>An application or service executing on a terminal shall be able to request another terminal to present information to and retrieve information from a user in a way that abstracts from the actual user interaction mechanisms that are supported by the other terminal (e.g. pen, mouse, keyboard, speech, display ...)</p>
<p><b>Behavior</b></p> <p>Each terminal shall enable remote applications and services to discover and access this Terminal Module over the network.</p> <p>Once a session between a remote application or service and the terminal module has been initiated over a communication link, the terminal retrieves the 'initial panel' and starts an interaction with the user using the actual user interaction mechanisms of the terminal. What user interaction is requested (text-entry, selection, ..) is communicated to the terminal by the remote application via a panel-description. Which mechanisms are used to effectuate the user interaction is determined by the terminal.</p> <p>Besides a panel description, the terminal receives information from the application or service on all other panels to which the user must be able to navigate from this panel and shall enable the user to actually navigate to all these linked panels. In this way the application or service can guide the user through a maze structure of panels. This structure is completely defined by the application or service.</p> <p>The terminal does not have any knowledge about the meaning of the user interaction, nor does it maintain any state concerning the remote application or service.</p>
<p><b>Appearance:</b></p> <p>The user will see a "normal UI" and he is not aware that the actual application or service is not running on his terminal. The terminal should use the user interaction mechanisms available to it in a consistent way for all remote applications and services.</p>
<p><b>Release:</b></p> <p>0.8</p>

**Status:**

Draft

**Author:**

Tom Suters (PHI)

## 3.10.4 Stream Control

<b>Feature name</b> F4 : Stream Control
<b>Priority</b> Optional
<b>Scope</b> Terminal
<b>Dependencies</b> Modifies Component Interoperability
<b>Description</b> A (A/V) streaming module (e.g. MP3 decoder) on a terminal offers via the Terminal API a standard control interface that allows an application or service to connect this module to external I/O of the terminal (e.g. a communication link, a display or a speaker) or to other streaming modules in the same terminal.
<b>Behavior</b> An application or service will determine which streaming modules a terminal supports (e.g. the supported data formats, data rates, etc). Based on this knowledge the application or service then creates an instance of each module it needs and connects them in the way it needs, resulting in a streaming graph. Finally the application starts the flow of (A/V) data in this graph. An application or service can temporarily stop the flow of (AV) data in a graph and modify the behaviour of its streaming modules (e.g. change certain parameters, allocate additional resources). This may also be possible while (A/V) data is flowing on the graph. Finally the application or service can tear down a graph and release the streaming module instances.
<b>Appearance:</b> The user is not directly involved in creating these graphs but only via other applications.
<b>Release</b> 0.8
<b>Status</b> Draft
<b>Author</b> Tom Sutera (PHI)