We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

5

A Distributed Multilayer Software Architecture for MIMO Testbeds

José A. García-Naya, M. González-López and L. Castedo
Universidade da Coruña
Spain

1. Introduction

The use of multiple antennas at both transmission and reception, also known as Multiple Input Multiple Output (MIMO) transmission systems, has received a lot of interest from the wireless communications industry during the last years. Communications in wireless channels using MIMO technologies exhibits a superior performance in terms of spectral efficiency, reliability and data rate when compared to conventional single antenna technologies (Foschini & Gans, 1998; Telatar, 1999). Existing and emerging standards for wireless communications such as IEEE 802.11 (WiFI), IEEE 802.16 (WiMAX) and Long Term Evolution (LTE), support multi-antenna transmission in their highest performance profiles.

In spite of their potential performance-enhancing capabilities, most of the research on MIMO technologies up to the moment is based on theoretical studies. Typically, the expected gains of MIMO technologies are only shown under ideal conditions since most analysis rely on simulations. Experiments in real-world scenarios by means of hardware implementations are necessary to measure the actual performance of multi-antenna transmission methods. Hardware implementations not only take into account the real multipath propagation in wireless channels but also the implementation impairments so often ignored during the simulations.

Hardware implementations can be split into three groups (Rupp et al., 2006). The first one is constituted by *demonstrators* which are frequently designed having in mind a particular standard or specification. Demonstrators usually exhibit good technical features for real-time implementations but they are extremely expensive and present poor flexibility and modularity. The second group is formed by *prototypes* of a final product. A prototype is a real-time implementation of a system specifically developed to support an industrial need. Prototypes often constitute a preliminary stage where the system is implemented and debugged and later on implemented as a consumer product. Finally, the third set is formed by *testbeds* that support real-time transmission capabilities while data is generated and post-processed off-line.

In addition, hybrid solutions can be devised. As an example, a testbed can carry out some operations in real-time with the purpose of speeding up the measurement process. Usually, candidate signal processing operations to be implemented in real-time are those that operate at sample level and/or do common tasks for all experiments, i.e. I/Q modulation, up-sampling, pulse-shaped filtering, etc. Throughout this chapter we will focus on testbeds because they use open designs and are more often found in public research centres and academia.

Various MIMO testbeds have been reported in the literature (Borkowski et al., 2006; Caban et al., 2006; Fabregas et al., 2006; Haustein et al., 2006; Nieto et al., 2006; Ramírez et al., 2008;

Rao et al., 2004; Wilzeck et al., 2006; Zhu & Fitz, 2005). Some of them have been constructed to evaluate a particular standard or specification while others have been designed for general purpose. Flexibility, development time consumption, throughput or costs are important features when comparing existing testbeds. Also, it should be noticed the educational possibilities of testbeds that open the door to many teaching opportunities. In the literature, however, there is a lack of up-to-date guides and tutorials useful for the construction of a new testbed from the scratch. Indeed, there exists few contributions (Caban et al., 2006; García-Naya et al., 2008a; Rao et al., 2004; Rupp et al., 2007) that contain a detailed description of the constructed MIMO testbed and, except for (García-Naya et al., 2008a; Rupp et al., 2007), the information contained on them is already outdated.

Based on the previous experience acquired by the research group of the authors in building and setting up MIMO testbeds, as well as performing indoor measurements (Pérez-Iglesias et al., 2008; Ramírez et al., 2008), we can assert that once the testbed hardware is available and properly configured, accessing the testbed becomes the main and also frequently ignored issue. When a research team decides to start the process of acquiring and/or constructing a new testbed, they need to take into account numerous aspects related both with the hardware and its technical features, and the extensibility possibilities for the future (García-Naya et al., 2008a; Rupp et al., 2007). Usually, most of the efforts are devoted to the testbed setup, which results in equipment that is hardly usable by people not involved in its design and later configuration. This makes extremely difficult to access to the testbed.

As a result, the migration of an algorithm from a simulation environment to a testbed involves cumbersome low-level programming to access the hardware as well as a very detailed knowledge of the hardware. Additionally, hardware implementation problems frequently ignored by simulations arise, such as time and frequency synchronization, I/Q imbalances, non linear distortions caused by the power amplifiers, etc. All these issues make difficult to assess new MIMO transmission methods in a testbed. For this reason, it is desirable to make the testbed accessible to final users at a reasonable abstraction level. This goal represents an important challenge due to the large amount of heterogeneous technologies and development environments that have to be integrated together. However, if this challenge is accomplished, the final result is a very attractive product for the user, who can focus on the development of new transmission techniques that can be easily translated to the testbed and later evaluated in realistic scenarios.

In this chapter we describe how to solve all previously mentioned limitations by using a distributed multilayer software architecture. This architecture enables the testbed to be easily accessible by the researchers and to be integrated in the development environment they are using. Although all designs and results herein presented are particularized for our MIMO testbed (García-Naya et al., 2008b; Ramírez et al., 2008), they are easily adaptable to most of the existing testbeds, making even possible the integration of heterogeneous testbed nodes in order to build a multi-terminal testbed.

The proposed software architecture consists of three different layers:

1. **The middleware layer (MWL)** is the lowest-level layer that interacts with the testbed hardware. It makes the testbed accessible through standard TCP socket connections.
2. **The signal processing layer (SPL)** performs the necessary operations required to convert the discrete-time sequences provided by the final user into discrete-time signals suitable to be transmitted by the hardware. At the receiver, it is usual to perform signal processing operations like time and frequency synchronization in case they are not carried out by the testbed hardware. The tasks compounding the signal processing layer

can also be executed in real-time by the testbed hardware. For example, digital up and down converters are frequently available in the libraries of programmable hardware modules.

3. **The testbed interface layer (TIL)** is the highest-level layer and presents the testbed to the user at an adequate abstraction level. The TIL has to be designed and implemented for a specific development environment. For example, if the final user makes use of Matlab, then a specific implementation of the TIL for Matlab has to be developed. The main purpose of the TIL is to provide a simplified interface to access the testbed. This does not prevent from providing mechanisms to control the hardware in detail from the TIL. It is very important to emphasize that there is no logic in the TIL except that necessary for adapting the data format from the specific environment used by the SPL and vice versa. The only requirement for the TIL to be implemented is the availability of a standard TCP socket library.

The whole design and implementation of the proposed software architecture is done under the following premises:

1. **Layers have to be as decoupled as possible**. This is a fundamental idea in modern software design and structured programming. The key idea is not to replicate functionalities that are already present in another layer, which would be a symptom of a bad design. The basic premise is to design everything to be fully decoupled and reusable; and later on introduce some small violations of this principle only if they are strictly needed to increase the overall system performance.
2. **Each layer can be extended and/or customized** to be able to adapt them to future specifications and/or heterogeneous hardware environments. It is important that this layer upgrade be done without needing a complete remake of the software, which is frequently the only solution in case of monolithic non-layered systems.
3. Finally, **layers should be distributed**, which means that they use remote connections to interact among them. On the one hand, this helps to ensure the decoupling and independence principles whereas, on the other hand, allows the testbed to be remotely accessible from the user Personal Computer (PC).

The remaining of this chapter is structured as follows. Section 2 and Section 3 provide a global description of the testbed hardware and software possibilities, respectively. Section 4 describes the software technologies that were used to develop the proposed distributed multilayer software architecture. In Section 5 the software architecture is presented and described. The interaction among the different layers and components of the architecture is studied in Section 6 and Section 7. Finally, Section 8 is devoted to the conclusions.

2. Basic Testbed Hardware

Testbeds are often used to verify if a new signalling technique (or even a complete standard system) that has been proven useful by simulations is also valid in realistic wireless scenarios. Testbeds allow the dimensioning of the hardware needs for real-time implementations, not only for the digital signal processing modules (mainly, DSP and FPGA) but also for the analogue Radio Frequency (RF) front-ends. Testbeds allow capturing the requisites for the hardware used in the final real-time implementation. Consequently, evaluating performance with testbeds allows knowing whether a given technique is feasible from the real-time hardware and implementation requirements.

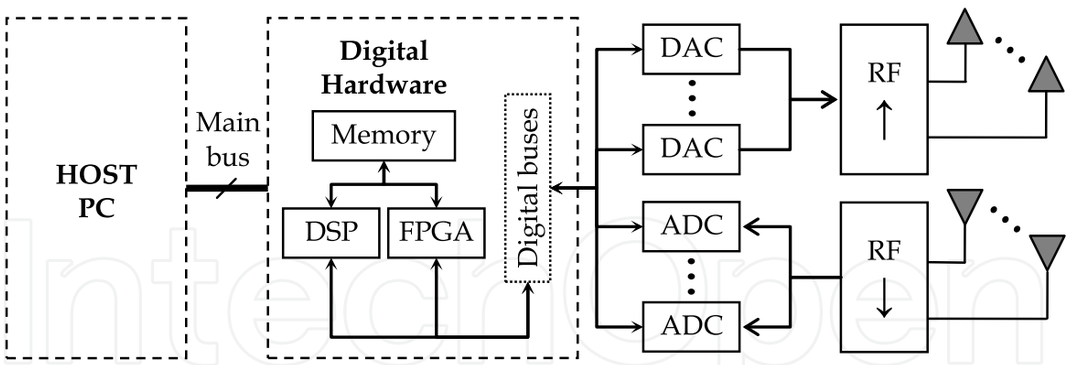


Fig. 1. Block diagram of the basic hardware configuration of a testbed.

Among the three different types of hardware implementations described above (demonstrators, prototypes and testbeds), testbeds present the following advantages:

- **Flexibility**. Testbed hardware is meant to be used for off-line processing. Only the signals are sent and acquired in real-time. This implies that testbed hardware is not subject to the real-time restrictions even though the hardware can include some sort of real-time capabilities.
- **Modularity**. Usually, the minimum modularity found in a testbed is given by the separation between the digital hardware (up to the D/A and A/D converters) and the RF hardware. Sometimes, the digital hardware can be split in different modules performing specific operations: D/A and A/D conversion, digital up and down conversion, signal buffering, etc. For the RF section it is possible to find self-made solutions or commercial products. Lastly, commercial RF front-ends also permit some degree of flexibility, for example allowing dual-band operation, RF carrier selection for each band or adjusting the gains at the transmitter and receiver amplifiers.
- **High-level language development**. Having in mind that most of the processing tasks are carried out off-line, general-purpose processors are the most adequate for processing the generated/acquired signals. This allows the utilization of high level programming environments (e.g. Matlab, C/C++, Java) and the simplification of the implementation stages both in time and complexity. This feature also provides an additional flexibility degree, because it is easy to change the implementation on the fly.
- **Floating-point versus fixed-point precision**. As a consequence of the off-line processing and the usage of high-level programming environments, the operations are carried out in the floating-point domain instead of using fixed-point operations available for real-time devices. This permits the researcher to focus on the implementation rather than considering some other problems like arithmetic precision of the operations.

Testbed hardware components can be classified into three groups according to their functionality, as shown in Fig. 1. The first one is the host system, usually a PC and consequently referred to as host PC. It is the equipment that allocates one or more boards containing the digital section of the hardware testbed (including D/A and A/D converters). The second group is constituted by the digital hardware components and, finally, the third one is formed by the RF front-ends. The D/A and A/D converters generally constitute the frontier between

the digital and analogue hardware. In the digital section, a bus termed main bus allows transferring the data from the host to the testbed hardware and the other way around. Next, a set or just one digital bus interconnects the digital hardware (DSPs, FPGA, memory buffers, etc.) with the D/A and A/D. Finally, for the interconnection among the D/A and A/D converters and the RF front-ends coaxial cables are used.
With this basic configuration, it is possible to send samples directly coming from the main bus, convert them into the analog domain using the D/A converters and up convert them to the desired carrier RF using the front-ends. At the receiver side, the signals are down converted by the RF front-ends, digitally converted by the A/D converters and then sent to the host through the main bus. Note that the main bottleneck in this scheme is the maximum data rate provided by the main bus, especially if there are no digital up and down converters available. In the most recent boards, by using PCI express or similar solutions is possible to use the host memory as a buffer while samples are transferred in real-time through the main bus.
A first improvement of this basic scheme consists in incorporating a digital up converter (DUC) before each D/A converter at the transmitter and a digital down converter (DDC) after each A/D converter at the receiver. These devices can be dedicated elements or can be implemented in a FPGA. Incorporating DUC and DDC into the MIMO testbed design allows transferring complex signals from/to the host, reducing both the transfer rate at the main bus and the software complexity. For MIMO operation, DDCs and DUCs must be fully synchronized. Additionally, another improvement consists in implementing some of the time-consuming sample-level tasks in FPGAs (e.g. time and frequency synchronization).

2.1 Baseband Components

Baseband components are the hardware elements necessary to deal with baseband and/or Intermediate Frequency (IF) signals. Frequently, such baseband components are allocated on carrier boards installed on conventional PCs. Current testbeds use carrier boards equipped with standardized buses to access testbed hardware such as USB, PCI, cPCI, PCI express or similar. A manufacturer compliant with the PXI alliance (PXI, 2009) produces carrier elements that are compatible with the most typical buses present in host equipments. When available, important hardware components in a testbed are the storage buffers, especially when the main bus does not support the necessary rate demanded by the D/A and A/D converters. Such buffers allow performing signal operations off-line while data is sent and acquired in real-time.
The interconnection among the previously described elements is carried out using buses that must be capable of transmitting the data fast enough. Finally, external circuitry such as clock distribution and/or triggering is needed. Sometimes, the most advanced hardware manufacturers include such circuitry as part of the commercial boards, simplifying the later setup. When MIMO processing is required, fully synchronization among the different devices is mandatory. Sometimes manufacturers announce a MIMO system but just a scaled SISO solution is offered.

2.2 RF Front-Ends

RF front-ends constitute one of the major hindrances in the testbed building process. They are responsible of up converting IF or baseband signals to RF. The most common RF bands are the unlicensed ISM located at 2.4 and 5.8 GHz. It is easier to find components for such bands but also unpredictable interference is present in such spectrum portions. High linearity and flexibility in terms of supported carried frequencies and bandwidths are desirable features for

the RF front-ends. However, the most advanced front-ends commercially available are only dual-band and have fixed maximum bandwidth. A fundamental feature needed to be able to carry out experiments is the possibility of modifying the transmit power. Also, the majority of the front-ends are designed for SISO operation, making difficult to adapt them to a MIMO system. Moreover, once the front-end has been acquired, extensive test on it is needed and expensive measurement equipment is required.
It is important to emphasize that RF hardware is expensive compared to the cost of the digital hardware and does not follow Moore's law (Moore, 1998).

3. Basic Testbed Software

Methodologies that cover the entire development process, from source code suitable for simulations and executed off-line, to real-time implementation in the testbed, as well as software tools according to these methodologies, are extremely scarce. The most popular methodology for testbed development is rapid prototyping (Kaiser et al., 2004; Rupp et al., 2003; 2006). A typical approach used to develop real-time algorithms to be run in a testbed consists in starting with a simulation implementation. Next, the simulation is migrated to the testbed and, finally, a real-time implementation is obtained. It is also convenient to split the real-time implementation into several steps: firstly, a fixed-point code is produced; next a DSP implementation is obtained; and, afterwards, the software modules that do not meet time requirements are migrated to FPGA, making use of high level tools when possible. Nowadays, except for the Mathworks tool suite (Mathworks, 2009) combined with some VHDL code generators, there is a lack of high level tools and development environments suitable to fit the previous steps.
Also, some standardisation efforts have just started in order to make compatible hardware components and software modules from different manufactures and developers. The PXI alliance (PXI, 2009) plus the Software Defined Radio (SDR) Forum (SDR Forum, 2009) and initiatives such as the Software Communications Architecture (SCA) (SCA, 2009) constitute the starting point of a new generation of modular, flexible and standard radio interfaces suitable for research. Nowadays, however, the previously mentioned initiatives have not already produced as a result the availability of high level tools allowing final users and researchers not involved in the hardware development to access the testbed hardware at a reasonable abstraction level.
All above reasons serve as a motivation for our work, which aims at bridging the existing gap among the hardware, the software elements provided by the manufacturers and the abstraction level required by the final users. Moreover, the proposed solution allows embedding real-time modules in the processing chain, as if they were part of the testbed hardware. For example, real-time frequency and time synchronization algorithms can be run at the receiver side and integrated into the digital hardware. Additionally, given that the proposed solution is designed to be easily and quickly integrated in any other kind of system, it becomes a very useful tool to help in testing real-time applications, especially during the first stages of the development. A good example is the ability to develop in parallel different parts of the transmitter and the receiver in real-time. While the real-time transmitter is still under development, the team developing the real-time receiver can make use of the testbed to generate the transmit signals and feed the real-time receiver with them.

4. Software Technology Behind the Distributed Multilayer Architecture

The field of computer science has come across problems associated with complexity since its constitution. The software architecture discipline is centred on the idea of reducing complexity through abstraction and separation. There are different kinds of software architectures. Among them, the most interesting for our work are: the client-server architecture, which serves as the basis for most of the available architectures; the three-tier model; and finally, the most recent and advanced model-view-controller architecture.

4.1 Client-Server Software Architecture (Two-Tier Software Architecture)

Client-server describes the relationship between two computer processes in which one process (the *client*) makes a service request to another process (the *server*). Applications following the client-server architecture represent an evolution with respect to those called "monolithic".
The basic operation mode for client-server applications consists in that each instance of the client process sends requests to one or more connected servers. In turn, servers accept these requests, process them, and return the requested information to the client. The basic client-server architecture considers only two types of hosts: clients and servers. Consequently, it is also known as two-tier model. A special case of the two-tier software architecture arises when an instance simultaneously acts as a client or as a server, resulting in the peer-to-peer architecture.
The two-tier software architecture presents the following advantages with respect to monolithic systems:

- The responsibilities of the whole system are now split between the client and the server, which brings the opportunity to decouple them.
- Servers can control the access to the resources to guarantee that only those clients with appropriate permissions access and change the data.
- Different client/server types can work with different kinds of servers/clients, which helps in the integration of heterogeneous systems.

As a main disadvantage, the mechanism used to interconnect clients and servers (frequently standard socket connections) may become both a bottleneck and a weak point of the system, because a failure in the interconnection mechanism will stop the whole system.

4.2 Multi-Tier Software Architecture

The multi-tier software architecture represents a natural evolution of the client-server model towards a higher number of levels. That is the reason why it is also known as the *n-tier software architecture*. Basically, the multi-tier software architecture is a client-server architecture consisting of more than two tiers, and where the inner tiers act simultaneously as client for one tier and as server for the other. The main difference with respect to the peer-to-peer model is that the tier order (its position or level in the multi-tier structure) does matter. An inner layer is also termed middleware because acts as an intermediary between two adjacent tiers.
Note that the multi-tier software architecture, as well as the two-tier one, is a mechanism to design the physical structure of the system. Given one of the models, several degrees of freedom are still possible to get the logic structure of the system.

4.3 Tier Interconnection Mechanisms

Up to now, we were using a certain abstraction level to define the previous architectures. Usually, each tier is mapped to a set of processes obtained from the same master process. This

can be easily understood with an example. Imagine a two-tier system. One tier is the client while the other is the server and they are mapped into two different processes interconnected by means of sockets. It does not matter if both processes are in the same physical machine or not. What is important is how they interact with each other. The server will be waiting for a request from one of the clients. As soon as the request arrives, the server starts to serve the petition and, simultaneously, starts waiting for another request. When the server process is going to serve a new request, it clones itself to serve the petition. Frequently, this process cloning mechanism is omitted and it is considered as an implementation detail. Therefore, to describe the interaction between different tiers, it is assumed that each tier is mapped into a process that interchanges messages with other processes (tiers). Although this approach is not completely rigorous, it is enough for our purposes. In the literature such interactions are described my means of sequence diagrams as we will do.
But there is still one open question: How are messages sent from one tier to another? In principle, data transfer between two tiers is part of the architecture design, but in practice only the message types and the data are defined by using sequence diagrams, which drives to the definition of a part of the system termed *protocol*. A protocol defines the interaction among all tiers, specifying the type of messages to be sent from one tier to another or to several other tiers, as well as the type of data sent in each message. Messages are transferred using plain socket connections or more elaborated mechanisms (e.g. SNMP, CORBA, Java RMI or even Web Services). The different approaches have their own advantages and disadvantages and there is still no perfect system for message interchange. However, what is really important is to use standardized mechanisms for message interchanging, thus guaranteeing independence among tiers.

4.4 The Model-View-Controller Architectural Pattern

The Model-View-Controller (MVC) is a pattern used in software engineering derived from the multi-tier software architecture. Successful use of the pattern drives to the isolation of the business logic from the user interface, allowing one to be freely modified without affecting the other. The controller collects user inputs, the model manipulates (and usually stores) application data, and the view presents results to the user. The MVC was first described in (Trygve, 1978) and can be used as an architectural or design pattern. As an architectural pattern, it splits an application into three independent layers that can be run in different computers: the presentation or user interface layer, termed the view; the business logic, called the model; and the controller, an intermediate layer adapting the view to the model and vice versa.

4.5 Applying Software Engineering to MIMO Testbeds

In the first sections of this chapter we present the typical hardware architecture available in MIMO testbeds as well as the typical abstraction level offered by the software included with the hardware. We stressed that such abstraction level is by far not enough and, consequently, a final user or a researcher not involved in the testbed development and later setup cannot directly access it. Along this Section, we introduced architectural software models, starting with the well-known client-server model and ending with the recently proposed model-view-controller, widely used in web applications as well as distributed applications. These architectural models serve as an inspiration to propose a novel software architecture useful to bridge the existing gap between the abstraction level offered by testbeds and the one demanded by researchers and final users. This new architecture is explained in the following Section.

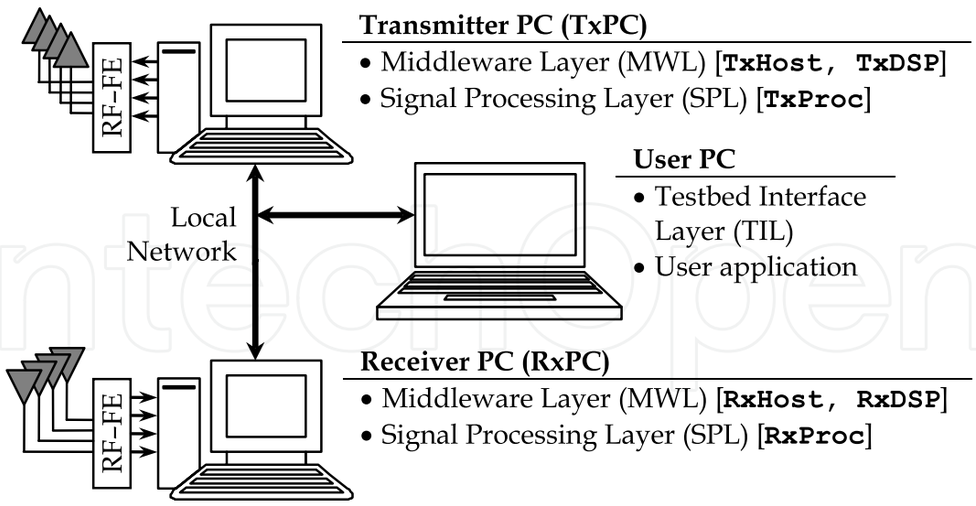


Fig. 2. General scheme of the MIMO Testbed showing the three different layers: middleware (MWL), signal processing (SPL) and testbed interface (TIL). The corresponding process name for each layer and each PC is shown between brackets.

5. Distributed Multilayer Software Architecture for MIMO Testbeds

Fig. 2 shows the proposed testbed hardware organization, which can be extrapolated to most of the testbeds. It is constituted by two ordinary PCs hosting the testbed hardware, one for the transmitter (referred to as TxPC) and other for the receiver (named RxPC). The digital section of the testbed hardware is installed inside the PCs while the RF front-ends (RF-FE) remain outside the PC case. The two PCs are attached to the network, something that is very useful because this way the PC desktops can be remotely accessed. The remote user PC is also attached to the network. In the rest of the Section we are assuming the following:

1. The testbed consists of the transmitter and the receiver, i.e., two nodes. Although our designs can be easily extended to an arbitrary number of nodes, it is better to start with the simplest case of two nodes to keep things simple.
2. It does not matter whether the testbed operates outdoor, indoor, outdoor to indoor or vice versa. We assume that a network connection can always be established among the testbed nodes and the user PCs.
3. The testbed PCs will use a standard operating system supporting remote desktop or remote operation from a PC attached to the network. This is a fundamental feature because it enormously simplifies software deployment as well as day-to-day maintenance of the systems. Otherwise, a replacement for such tools should be provided.

5.1 From the MIMO Testbed to the Multilayer Software Architecture

Fig. 2 shows, close to the PC drawings, the names of the corresponding three layers of the proposed architecture that are, from the lowest to the highest level: the middleware layer (MWL), the signal processing layer (SPL) and the testbed interface layer (TIL). The MWL and the SPL are split into transmit and receive parts, while the TIL has just one instance running on the user PC. Actually, the TIL is just an Application Program Interface (API) that has to be

included with the user application. In our standard deployment, the MWL and the SPL are allocated in the same PCs containing the rest of the testbed hardware. Although the MWL is the only layer required to be installed in the hardware PCs, the rest of the software can be deployed in any machine attached to the network.
This is not an arbitrary proposal but a design inspired on the multi-tier and the model-view-controller software architectures. First of all, let us identify the use cases of our application. Basically, there is just one use case or action carried out by the final user: transmitting a set of discrete-time sequences through the testbed and, consequently, through the wireless channel. Therefore, the layer corresponding to the view is the interface that allows accessing the testbed, sending the sequences and getting back the acquired signals. This is what we term testbed interface layer, and is the topmost layer of our software architecture.
The controller plays a very important role in a system. It is responsible of getting the service requests from the view, adapting and sending them to the model where the business logic resides. The gathered results are then sent back to the view to be presented to the user. The controller plays the role of the middleware concept presented in the multi-tier software architecture. In the testbed system the controller is called middleware and its functionality is clear: to configure and control the hardware in order to be able to take the requests (discrete-time sequences) from the TIL; to adapt and send them to the hardware to be transmitted by the antennas; and, finally, to carry out the reciprocal operations at the receiver side. At the end of the process the TIL returns the acquired discrete-time signals.
However, our architecture presents three strong differences with respect to the model-view-controller and the multi-tier architecture:

- Adaptation of the discrete-time sequences provided by the TIL in the requests, both at the transmitter and the receiver, consists in performing different signal processing operations that sometimes can even be executed using different kind of processors. For instance, a general-purpose processor (GPP), a graphic processor unit (GPU) or real-time devices like an FPGA or a DSP. It is thus necessary to put all these operations together and, consequently, the SPL concept comes out. Therefore, the SPL is in charge of carrying out most of the signal processing operations needed between the MWL and the TIL.
- One of the reasons to use a multi-tier architecture jointly with the model-view-controller is the ability of the resulting application to be distributed among different machines. Thus, different instances of the tiers run on different machines. However, in the testbed system there are two kinds of nodes: the transmitter and the receiver. If a multi-node testbed is available, different transmitter and receiver pairs will be distributed among different nodes. As a result, the MWL (and consequently the SPL) is split into two sides: the transmitter side and the receiver side. They will be referred to as Tx MWL, Rx MWL, Tx SPL and Rx SPL, respectively. Additionally, they are also identified because the processes mapping the architecture design are also split into two sets, one for the transmitter and the other for the receiver.
- Finally, another particularity present when adapting the model-view-controller and the multi-tier software architectures to testbed software architecture is that the hardware is required to be attached to the host system by means of a bus (e.g. PCI, PCIX, PCI Express, etc.). Consequently, it is not possible to sustain standard network connections through such buses, and the standard tools and techniques available for interconnecting layers are not applicable (except when a custom implementation is provided or when

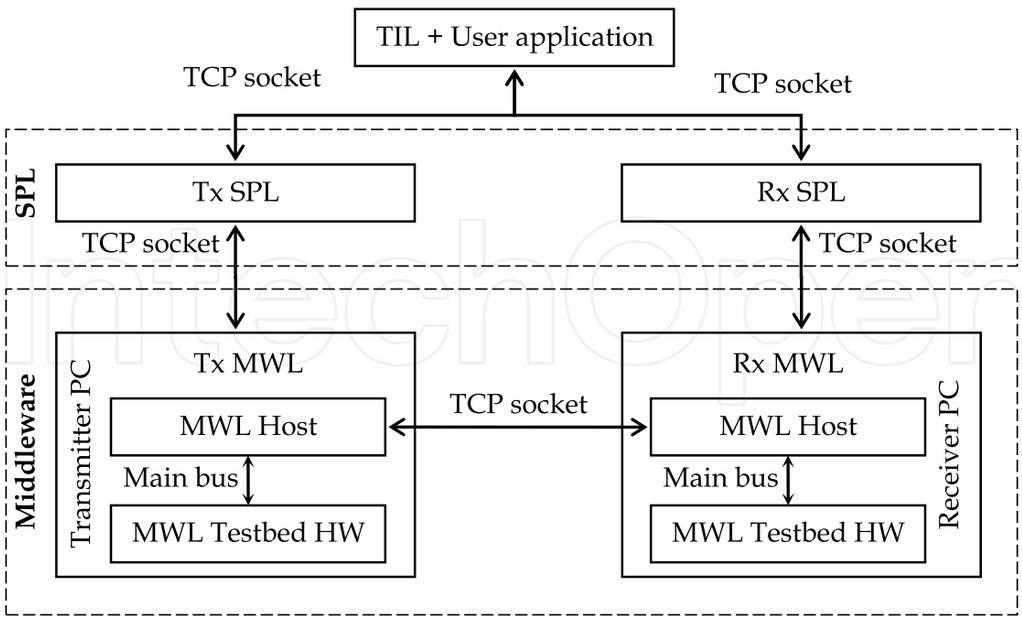


Fig. 3. Basic structure of the distributed multilayer software architecture for MIMO testbeds. The three layers are shown: MWL, SPL and TIL. Additionally, the testbed-hardware sub-layer in the middleware and the different interconnection mechanisms are included.

the hardware is attached through a network connection). These reasons motivate another division in the MWL, generating a *Testbed-hardware* sub-layer (with its respective parts for the transmitter and the receiver sides) responsible of dealing with hardware aspects as well as solving the bus connection issues with the host part of the MWL. Having in mind that in our testbed this sub-layer runs on the DSPs available at the transmitter and the receiver, the corresponding processes are referred to as TxDSP and RxDSP.

5.2 Logical and Physical Designs of the Distributed Software Architecture

In Fig. 3, the basic structure of the proposed architecture is shown. There are two sides, transmitter and receiver, joined at the TIL, which has to hold connections with both sides of the SPL. All links between the different layer elements are implemented by using standard socket connections. The exceptions are the links between the sub-layers of the MWL that use a proprietary protocol over the existing main bus interconnecting the hardware and the host. There are strong reasons for using standard socket connections instead of any other higher-level mechanism like, for example, web services. However, this does not imply that in some situations other connection types can better solve some problems. It is also possible to use different link types among different layers. For example, the TIL and the SPL can be connected using web services, thus allowing total independence of the platform and full remote access. However, the ability of such high level techniques to sustain high data rates and low latency connections is not the best. For these reasons, sockets offer enough flexibility while providing fast connections. In the case of bus connections the bus type obviously limits the latency and the data rates.

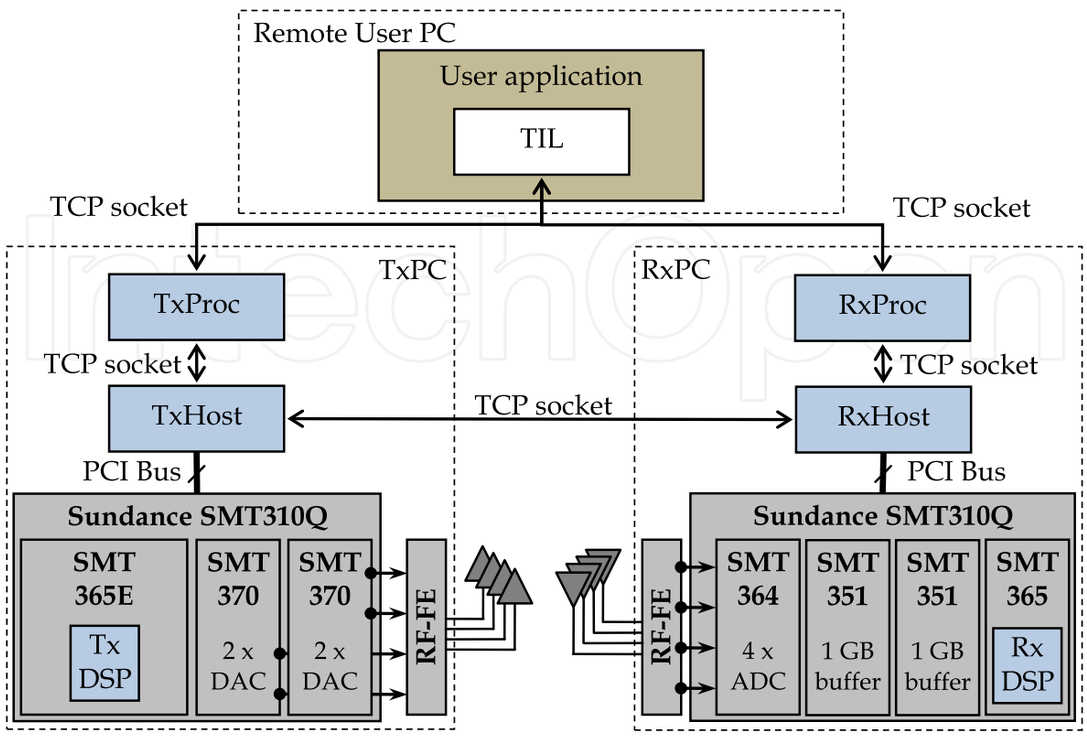


Fig. 4. Testbed scheme containing the hardware and the software architecture deployment as well as the links among the components. The different processes are shown in blue, and are located in the usual place for a typical architecture deployment. The digital hardware sections of the testbed, as well as the RF front-ends, are shown in gray. Finally, the user application is in dark green, containing the TIL in white. Note that TIL appears in white and not in blue because it is not a process but an user application.

While in Fig. 2 the general structure of the testbed is depicted, showing the correspondence between the hardware elements and the software layers, Fig. 4 illustrates the block diagram of the entire system. Three main parts can be distinguished: the testbed hardware that allows us to transmit discrete-time signals over multiple antennas; the multilayer software architecture that makes the hardware accessible to end researchers at a high abstraction level; and, finally, the user application implemented using the testbed capabilities and the architecture facilities. The lowest software level (i.e. the MWL) is required to be installed in the same PCs as the testbed hardware because it uses the system buses to communicate with the hardware. Otherwise, this restriction would not be applicable. The other two layers can be installed in any other available PC. However, in the standard deployment, the SPL is included with the MWL in the testbed PCs. Finally, the TIL is installed in the remote user PC.

5.3 Short Description of the MIMO Testbed Hardware

At the bottom of Fig. 4 a very basic diagram of our testbed hardware is shown. A first release of the testbed hardware was presented in (Ramírez et al., 2008), where the baseband modules were from Sundance Multiprocessor Ltd. and the RF front-ends were developed at the Uni-

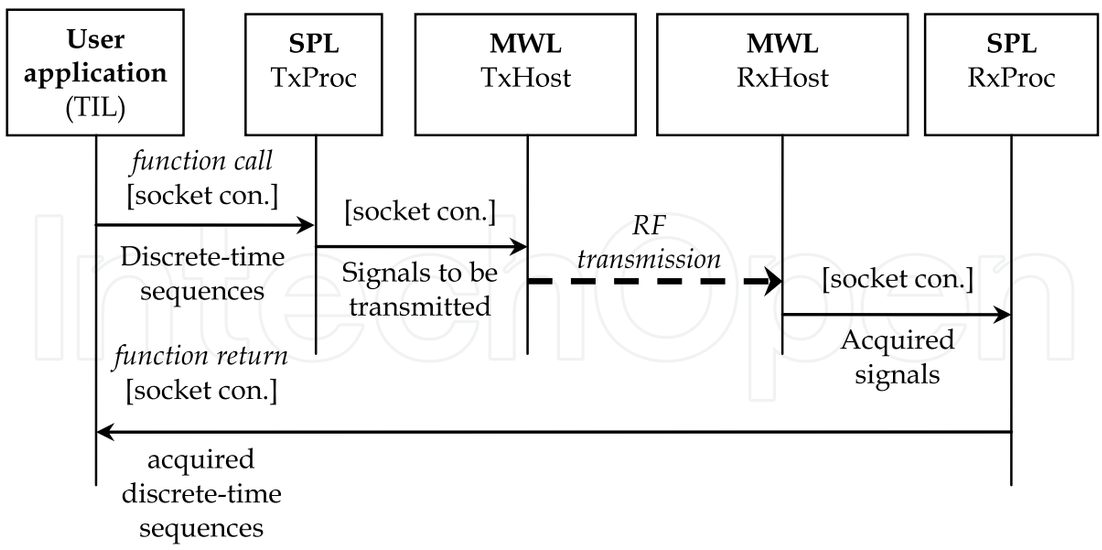


Fig. 5. Frame transmission example. It shows how a single frame transmission is carried out from the discrete-time sequences. For the sake of simplicity, the MWL is considered as a whole, even when it is actually split in two sub-layers.

versity of Cantabria. This hardware has been updated and new RF front-end modules from Lyrtech have been included. Also, minor module reorganisations have been done.
Currently, the hardware of the testbed is based on a Sundance Multiprocessor (Sundance, 2009) SMT310Q PCI carrier board and a basic processing module: the Sundance SMT365 equipped with a Xilinx Virtex-II FPGA and a Texas Instruments C6416 DSP at 600 MHz. The processing module has two buses that can transfer 32-bit words up to 400 MB/s, allowing the connection with the Sundance SMT370 module, which contains a dual AD9777 D/A converter and two AD6645 A/D converters. The SMT370 module also has a 2 Msamples per-channel memory that is used to load the frames to be transmitted. At the receiver side, the data acquired by the A/D converters is stored in real-time in two SMT351G memory modules offering 256 Msamples per A/D converter. Finally, in an off-line task, data is passed to the middleware through the PCI bus. At the transmitter side the Sundance SMT365E is used instead of the SMT365. It contains a larger FPGA as well as a larger amount of memory but the remaining features are the same.

5.4 Simple Transmission Example

In order to shed more light into the system architecture, let us explain, step by step, the transmission of a single frame (see Fig. 5). After the discrete-time sequences to be transmitted have been generated at the user application, a function from the TIL is called passing to it the corresponding symbol vectors (one vector per transmitting antenna). These symbols are then sent to the SPL (TxProc) where they are converted (if necessary) to discrete-time signals ready to be transmitted by the hardware. Finally, such signals are subsequently sent to the middleware (TxHost). When both the Tx and Rx PCs are ready to complete a transmission, the signals are passed to the testbed hardware through the corresponding TxDSP process to be transmitted by the antennas. At the receiver side, the MWL (RxHost) acquire the signals stored into the hardware buffers by RxDSP. Next, they are forwarded to the receiver SPL (RxProc). At this

moment, the Tx-Rx PCs are ready to process another frame while the acquired signals are converted to discrete-time sequences in the same format required by the end user. Finally, the discrete-time sequences are forwarded to the user application through the TIL, completing the entire process.

The previous example clearly shows the advantages derived from the use of a multilayer architecture instead of a monolithic system: the hardware is better exploited and each layer can be customized or extended according to specific capabilities of the hardware or particular needs demanded by users, all without developing a completely new monolithic system each time the specifications change. The last advantage turns especially valuable in a research environment where specifications change very fast.

5.5 Testbed Interface Layer

After describing the complete system architecture in the previous sections, this and the following sub-sections focus on the layers that compound the architecture designed for our MIMO testbed (Ramírez et al., 2008). In this subsection we deal with the topmost testbed interface layer (TIL), leaving the description of the other two lower-level layers for the next sub-sections.

The TIL interacts with the researcher or the final user by calling a simple function implemented and specifically adapted to the development environment under consideration, with the only requirement of supporting standard TCP socket connections. At the transmitter side, its main task consists in sending to the SPL the discrete-time sequences to be transmitted plus the necessary parameters. In the same way, the TIL receives the acquired signals, being noticed if any error occurs.

The main goal of the TIL is making the rest of the layers accessible to the final users by taking into account the type of development environment they use. For this reason, the user layer is jointly executed with the user application (see Fig. 3 and Fig. 4), being also integrated in the same process (or set of processes) forming the user application (i.e. a simulation or a demonstration software showing the rest of the testbed capabilities). Therefore, a different TIL implementation satisfying the particular user development environment requirements can be made available (Matlab, Simulink, C/C++, Java, etc).

5.6 Signal Processing Layer

The signal processing layer (SPL) is network-connected to both the TIL and the MWL (see Fig. 3 and Fig. 4). It provides remote access and makes the other two layers platform-independent with respect to each other. This fact permits the SPL to be run in a PC cluster, allowing intensive computing to generate the data to be transmitted and to process the acquired sequences. This layer consists of two different processes that carry out the signal processing operations needed to link the TIL and the MWL. The first process (TxProc) receives the discrete sequences to be transmitted from the TIL and performs all necessary operations such as up sampling, pulse-shape filtering, I/Q modulation and frame assembly, in order to generate the discrete-time signals that will be sent to the middleware (TxHost). Similarly, the second process (RxProc) waits for the acquired signals from the MWL and performs the time and frequency synchronization operations followed by the demodulation, filtering and down sampling. The resulting vectors are sent to the TIL. It is important to emphasize that the SPL is designed and implemented by clearly separating its two main tasks: interconnection with the adjacent layers (TIL and MWL) and execution of the required signal processing tasks. Depending on the

type of sequences given to TxProc, not all operations will be required either at the transmitter or the receiver side.
The SPL can also incorporate advanced features such as a limited feedback channel for the evaluation of precoded MIMO systems. This feedback channel is easily implemented using the network connection available between the transmitter and the receiver.
A SPL detached from the other architecture layers is fundamental to deploy multiuser MIMO scenarios. After extending the current TxProc and RxProc implementations in order to support multiuser MIMO techniques, the set of TxProc processes run at the transmitter users while the RxProc processes do at the receiver users.
In some cases, the researchers may wish to implement most or even all operations present in the SPL. It is still possible to configure the layer as a bypass, making only use of the interconnection capabilities and not performing any signal processing operation at all.

5.7 Middleware Layer

The middleware layer (MWL) fills the gap between the testbed hardware and the signal processing layer, allowing discrete-time signals to be transferred through the system bus and making possible the synchronization between the TxPC and the RxPC using standard TCP network connections. The MWL is split into two different sub-layers (see Fig. 3 and Fig. 4). The topmost sub-layer is responsible of establishing the network connections between the transmitter and the receiver, and with the higher layer (the SPL). The bottom sub-layer corresponds to the testbed hardware configuration and control software. Fig. 3 and Fig. 4 shows the MWL with its two sub-layers plus the connections with adjacent layers.
Four different processes constitute the middleware. The first two, termed TxHost and RxHost, run respectively on the TxPC and RxPC. They are implemented in standard C++ language and use sockets to establish the necessary network connections:

- One connection between the TxHost and RxHost processes. It is used to synchronize the transmitter and the receiver, so the receiver knows when the signal acquisition process has to start.
- Another connection is established between the TxHost and the TxProc processes and it is used to link the transmitter side of both the middleware and the signal processing layers.
- Finally, there is also a connection between the RxHost and the RxProc processes.

The remaining two processes are the transmitter and the receiver processes that run on their respective Digital Signal Processors (DSPs) available in the testbed hardware. Thus, the transmitter DSP process (TxDSP) performs data transfers through the PCI bus jointly with the TxHost process and configures and controls the hardware components at the TxPC. In the same way, the RxHost process and the DSP receiver process (RxDSP) are responsible of transferring the data through the PCI bus and, from the DSP side, controlling and configuring the testbed hardware components.
The MWL serves the maximum number of requests that the hardware supports. It serves a request while the data for the next frame is still being generated by any computer in the network and, when the first frame is passed to the SPL, the MWL is ready to accept a new frame. With this architecture scheme, the MWL simultaneously serves requests from other SPL instances running in different PCs.
For the specific case of our MIMO testbed, in order to release the MWL implementation from dealing with the lowest level hardware details, the Sundance SMT6025 software development

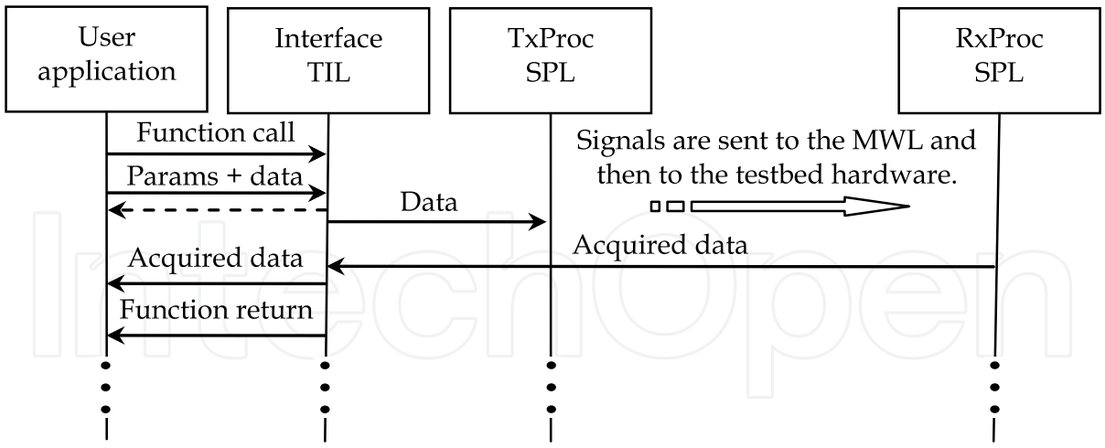


Fig. 6. Sequence diagram describing the interaction among the user application, the testbed interface layer (TIL) and the signal processing layer (SPL).

kit and the Sundance SMT6300 operating system driver are used in the TxHost and the RxHost implementations. The Texas Instruments Code Composer (Texas Instruments, 2009), as well as the 3L Diamond DSP+FPGA (3L Ltd., 2009), are utilized for the TxDSP and RxDSP implementations. If a different hardware type is considered, there should exist different tools avoiding dealing with the lowest-level hardware details.
The middleware concept constitutes a great leap forward in MIMO testbed technology, allowing access to hardware using ordinary network connections and allowing synchronization between the transmitter and the receiver. It also permits to incorporate a wired feedback channel. The MWL design supports extensibility in many senses. For instance, many transmitter and receivers can be used through broadcast network connections. Also, although the DSP processes are hardware dependent, the control logic between the transmitter and the receiver, as well as with the higher layers, is valid for different hardware testbeds. Finally, since the hardware interfaces from the host side and the software related to DSPs are developed as separated software modules, they can be replaced by implementations suitable for any other testbed.

6. Interaction between TIL, SPL and the User Application

Fig. 6 shows a simplified version of the interaction among the user application, the TIL and the SPL. When the user application has generated the discrete-time sequences to be transmitted, the testbed calls some function in the TIL. Both sequences, as well as their type, are included as input parameters. Then, depending on the specified additional input parameters, the TIL offers the possibility to return the control to the user application in order to carry on doing other tasks while the signals are being transmitted and acquired. The TIL establishes a connection with the TxProc at the SPL and sends the transmit data. Next, the interaction described in Fig. 7 takes place and, as a result, the signals are properly transmitted and afterwards acquired by the MWL and sent to the RxProc at the SPL. Finally, the acquired data is transferred to the TIL and sent to the user application. If the user application got the control after calling the TIL, then the acquired sequences are not transferred to the TIL until the user application calls again the TIL.

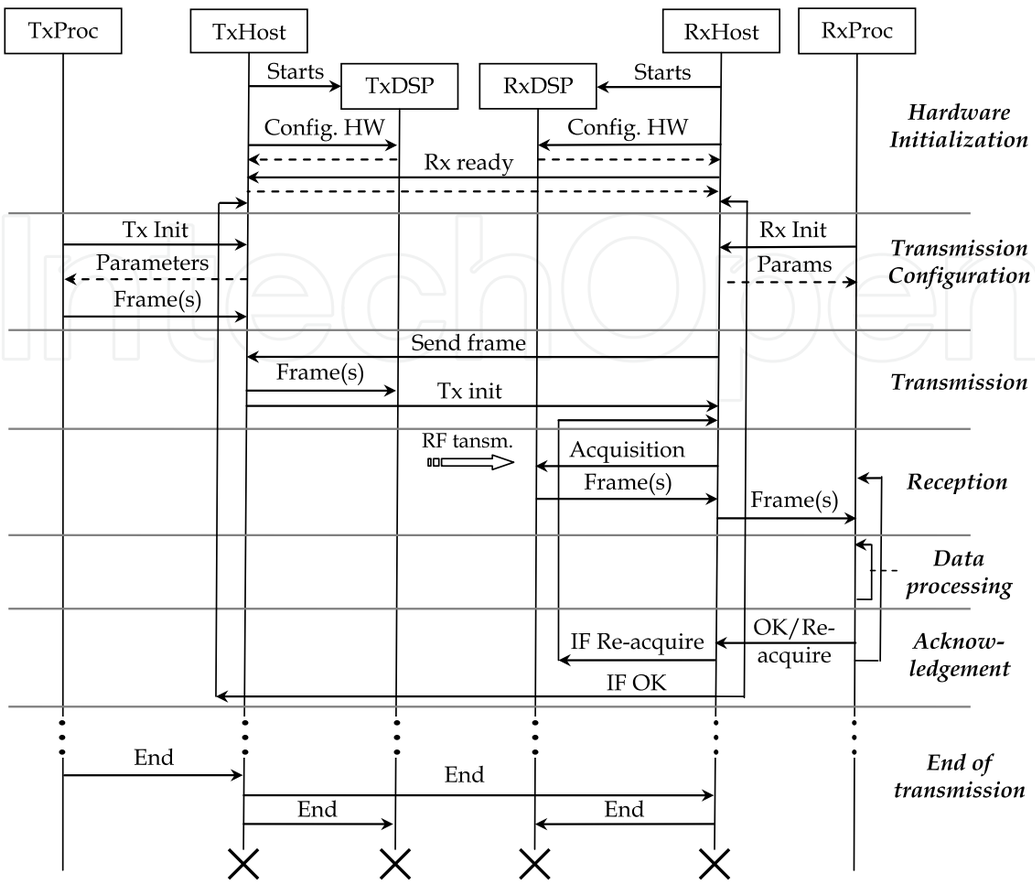


Fig. 7. Basic sequence diagram of one of the variety of protocols implemented in our testbed. It is termed basic operation mode. Dashed arrows represent acknowledgements (ACK). Regular arrows represent messages sent from a source to a destination.

7. Interaction between SPL, MWL and the Hardware

In this section we will explain some functioning modes of the testbed architecture. All of them have been implemented in our MIMO testbed and they allow us different operational modes.

7.1 Basic Mode Operation

Fig. 7 shows the sequence diagram of the "basic operation mode protocol". The protocol fully specifies the behaviour of the six processes of the software architecture. Thus, different protocols allow using the testbed and the software architecture in different scenarios and for distinct purposes. The basic operation protocol consists of the following seven stages:

1. **Hardware initialization**. Both transmitter and receiver hardware must be set up before beginning the actual operation. At the operating system level, two ordinary processes are started: TxHost and RxHost, which then load the TxDSP and RxDSP binaries into the respective DSPs. Once loaded, these processes initialize the baseband hardware modules. Additionally, they allocate the memory buffer needed for communication through the PCI bus. In this stage, the "Starts" message ensures that the communication

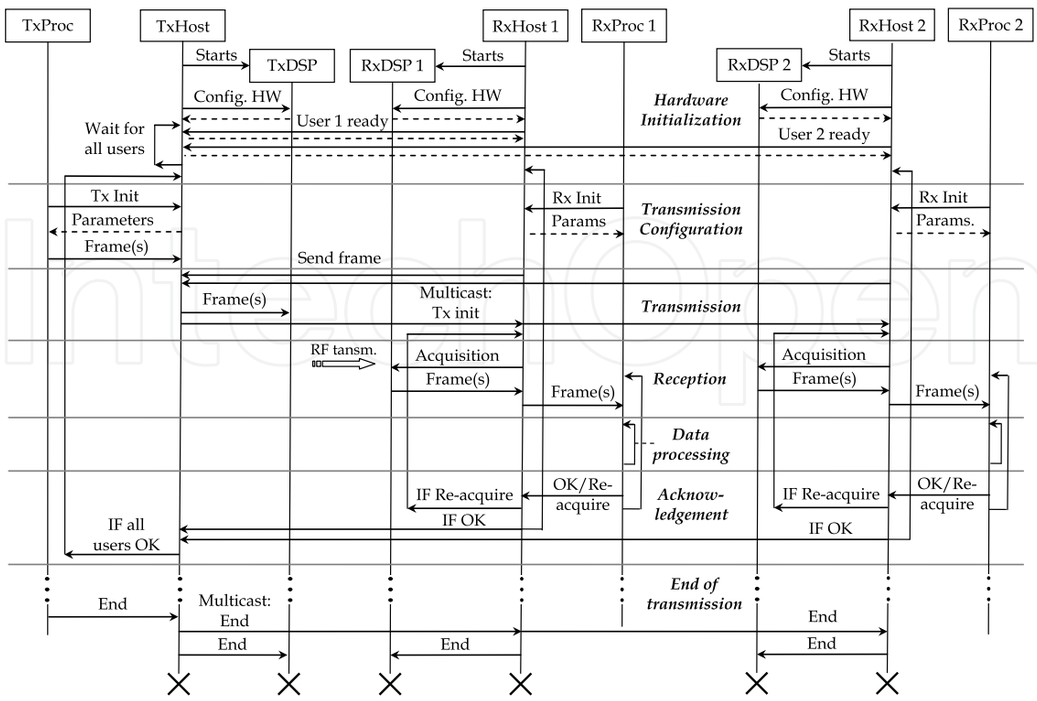


Fig. 8. Sequence diagram of the multiuser testbed protocol particularized for the case when just two receive users are considered.

between the host and the DSP can be successfully established. Afterwards, the ”Config. HW” messages are sent to the DSP and the TxDSP and RxDSP proceed to properly configure the hardware. They reply with an acknowledgement (ACK) represented in Fig. 7 by a dashed arrow.

2. **Transmission configuration**. At this stage an instance of the SPL accesses the MWL to configure a transmission. The process starts when a user application that makes use of the TIL transmits a set of discrete-time sequences. After receiving the sequences to be sent, TxProc sends the ”Tx Init.” message to the TxHost, which returns the parameters needed by the TxProc. Control parameters such as the frame size or the number of frames to be transmitted are configured. Moreover, other parameters such as the signal power strength, can be sent to TxHost during this stage. Also, the frame or set of frames to be transmitted is sent to TxHost. Finally, at the receiver side, a communication between RxHost and RxProc is established to ensure that the transmission can be carried out.

3. **Transmission**. Once TxHost has received all parameters, the transmission process can begin. First, RxHost must notify that it is able to receive frames by using the message ”Send Frame”. Then, TxHost sends to TxDSP the parameters and the frames acquired in the previous step and TxDSP communicates the inminent cyclic transmission of the current frame. Finally, TxHost sends the message ”Tx init” to RxHost notifying the availability of the frame; TxHost sends an ”Acquisition”message to TxDSP; and the frame is acquired and transferred to the TxHost through the PCI bus. Later on, the signals are sent to the RxProc and, finally, they are transferred to the user application

through the TIL. The transmitter side will continuously send the frame until a new message "Send Frame" is received from RxHost.

4. **Reception**. After receiving the "Tx init" message from TxHost, RxHost sends the "Acquisition" message to RxDSP. Then, RxDSP captures the signals and sends them back to RxHost through the PCI bus. Then, signals are sent to RxProc, properly processed in this layer, and finally they are sent to the TIL, making them available to the user application.
5. **Data processing**. In our testbed data processing is completely carried out off-line, even though hardware permits real-time processing by means of, for instance, available FPGAs. Basically, during this stage the received data are properly adapted and processed.
6. **Acknowledgement**. After processing the data, the receiver must tell the transmitter what to do next. The receiver can ask for retransmission of the last frame or for a new frame. In the first case, the receiver moves on the transmission stage. In the latter, the receiver needs to prepare a new transmission, so it must go back to the transmission configuration stage.
7. **End of transmission**. TxProc receives a finalization message, which is properly propagated to the rest of the testbed processes with the objective of adequately closing all opened resources and to finish all testbed processes.

7.2 Burst Transmission

This case involves the transmission of a number of data frames with the goal of testing a given transmission system. The sequence diagram is the same as the one depicted in Fig. 7 with the following two exceptions:

- At the beginning of the transmission configuration stage, TxProc notifies the number of frames to be transmitted and sends them to TxHost.
- After finishing each single-frame transmission (probably with a certain number of reacquisitions carried out by the receiver side) TxHost automatically moves on the next frame to be transmitted.

This mode can be exploited to emulate a real wireless communication system. For instance, a video can be sent in real-time and the users can watch it at the receiver side.

7.3 Performance Measurements

Another case of use is the automation of performance measurements corresponding to a particular algorithm or a specific transmission system. In this case, the protocol works in a quite similar way as in burst transmission: during the data processing stage the measurements are performed and the receiver decides whether it has to keep on doing measurements or it has already collected enough data. This functioning mode was used while doing performance measurements in (Pérez-Iglesias et al., 2008; Ramírez et al., 2008), where different STBC techniques were tested in real environments. The data processing stage can be modified in order to just store the acquired signals at the receiver side to be later on processed off-line.

7.4 Using a Feedback Channel

MIMO systems performance can be improved if the receiver feeds back Channel State Information (CSI) from the receiver to the transmitter, so it can adapt its transmssion scheme to the actual channel characteristics. It is straightforward to implement a feedback channel within

the protocol: the CSI information must be sent during the acknowledgement stage to notify the transmitter whether it has to re-configure the transmission parameters or not. To do so, in the sequence diagram shown in Fig. 7 there should be added two ”Send CSI” messages: one from the RxHost to the TxHost and another from the TxHost to the TxProc.
If the signal adaptation at the transmitter side takes place at the user application, then no feedback messages are needed in the protocol because everything is available at the user application. It is necessary to call the testbed interface layer twice: one to be able to estimate the channel and the following with the adapted signals.

7.5 Deployment of a Multiuser Environment

The last given case of study describes the functioning of the designed protocol when working in a multiuser environment where several nodes receive information from a central node (broadcast channel). The protocol just needs to be generalized to support the synchronization of several receiver nodes.
Fig. 8 presents the use case for two receiver users. In order to implement this scenario it is necessary to incorporate some modifications:

- After configuring the testbed hardware, TxHost waits for the ”ready” message from all receive users.
- TxHost waits for the ”Send Frame” message from all users before starting the transmission.
- The frame is cyclically transmitted and each individual user can carry out as many acquisitions as desired. Once all users have acquired all necessary data, they send an ”OK” message to the TxHost. When TxHost has received all ”OK” messages from all users, the current frame transmission is finished.
- When TxProc receives an end message, it passes it to TxHost which launches a multicast message to all receive users in order to properly finish.

7.6 Using a Simpler Protocol

In some cases it is interesting to provide a simpler protocol in order to delegate most of the control mechanisms to the user application. Now the TIL has two different functions allowing the control of the transmitter and the receiver separately. After the hardware initialization step (see Fig. 7), the user application decides to send a set of discrete-time sequences and then uses the TIL to send them to the SPL. Afterwards, the SPL sends them to the MWL to be transmitted, but RxHost is waiting for RxProc in order to carry out the acquisition. When the user application decides to acquire the data, it calls again the TIL which asks RxProc to acquire the data. Immediately, RxProc asks RxHost to acquire the data and returns it to the user application through the RxProc and the TIL.
The main difference of this functioning mode with respect to the basic operation mode is the lack of messages between TxHost and RxHost, which are now completely autonomous. This presents the advantage of easier integration of the testbed nodes with other nodes and, at the same time, all nodes can be directly controlled from the user application. However, as a disadvantage, the testbed control is moved to the user application. Consequently, the user application has now to deal with some hardware details (e.g. the size of buffers) but properly encapsulated in the TIL. In addition, performance losses arise as a consequence of the testbed being controlled from the user application.

8. Conclusion

In this chapter we propose a new distributed multilayer software architecture for MIMO testbed user access. The architecture fills the gap that currently exists between commercial hardware components and the most common abstraction level used by researchers. The architecture overcomes the limitations of implementing new algorithms directly from the testbed. Instead of using the low-level interfaces typically provided by manufacturers, the architecture supplies a high-level interface access for testbeds. It releases researchers from the necessity of knowing the details of the testbed hardware. For instance, they can easily test new algorithms without developing a completely new source code release specifically for the testbed, thus speeding up the implementation and test tasks.

The key point to the multilayer architecture design is the use of ordinary network connections to link both software modules and hardware components. These connections are also useful for decoupling the software layers. Several advantages are obtained as a consequence of the multilayer software architecture. Also, multiuser scenarios and feedback channels can be constructed extending the proposed architecture. Finally, it is also possible to simplify the presented architecture protocols in order to ease the integration with heterogeneous nodes. The testbed control is moved to the user application instead of keeping it in the inner layers.

9. References

3L Ltd (2009), `http://www.3l.com/`

Borkowski, D.; Brhl, L.; Degen, C.; Keusgen, W.; Alirezaei, G.; Geschewski, F.; Oikonomopoulos, C. & Rembold, B. (2006). SABA: A testbed for real-time MIMO systems. *EURASIP Journal on Applied Signal Processing*, Vol. 2006, 2006.

Caban, S.; Mehlführer, C.; Langwieser, R.; Scholtz, A.L. & Rupp, M. (2005), Vienna MIMO Testbed, *EURASIP Journal on Applied Signal Processing*, Vol. 2006, 2006.

Fabregas, A.G.; Guillaud, M.; Slock, D.T.M.; Caire, G.; Gosse, G.; Rouquette, S.; Ribeiro Dias, A.; Bernardin, P.; Miet, X.; Conrat, J.M; Toutain, Y.; Peden, A. & Li, Z. (2005), A MIMO-OFDM Testbed for Wireless Local Area Networks, *EURASIP Journal on Applied Signal Processing*, Vol. 2006, 2006.

Foschini, G. & Gans, M. (1998), On Limits of Wireless Communications in a Fading Environment when Using Multiple Antennas, *Wireless Personal Communications*, Vol. 6, 1998, pp. 311-335.

García-Naya, José A.; González-López, M. & Castedo, L. (2008), An Overview of MIMO Testbed Technology, *Proceedings of 4th International Symposium on Image/Video Communications over Fixed and Mobile Networks (ISIVC 2008)*, July, 2008, Bilbao

García-Naya, José A.; Pérez-Iglesias, Héctor J.; Fernández-Caramés, T. M.; González-López, M. & Castedo, L. (2008), A distributed multilayer architecture enabling end-user access to MIMO testbeds, *Proceedings of IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications, 2008 (PIMRC 2008)*, September, 2008, Cannes.

Haustein, T.; Forck, A.; Gäber, H.; Jungnickel, V. & Schiffermüller, S. (2005), Real-Time Signal Processing for Multiantenna Systems: Algorithms, Optimization, and Implementation on an Experimental Test-Bed, *EURASIP Journal on Applied Signal Processing*, Vol. 2006, 2006.

Kaiser, T.; Wilzeck, A.; Berentsen, M. & Rupp, M. (2004). Prototyping for MIMO Systems An Overview, *Proceedings of the XII European Signal Processing Conference (EUSIPCO 2004)*, September, 2004, Vienna.

The Mathworks (2009), http://www.mathworks.com/

Moore, G. (1998), Cramming more components onto integrated circuits, *Proceedings of the IEEE*, Vol. 86, No. 1, 1998, pp. 82–85.

Nieto, X.; Ventura, L.M. & Mollfulleda, A. (2006), GEDOMIS: a broadband wireless MIMO-OFDM testbed, design and implementation, *Proceedings of 2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM 2006)*, 2006, Barcelona.

PXI: PCI eXtensions for Instrumentation (2009), http://www.pxisa.org

Pérez-Iglesias, H.J.; García-Naya, J.A.; Dapena, A.; Castedo, L. & Zarzoso, V. (2008), Blind channel identification in Alamouti coded systems: a comparative study of eigendecomposition methods in indoor transmissions at 2.4 GHz, *European Transactions on Telecomunications*, Vol. 19, No. 7, November 2008, pp. 751–759.

Ramírez, D.; Santamaría, I.; Pérez, J.; Vía, J.; García-Naya, J.A.; Fernández-Caramés, T.M.; Pérez-Iglesias, H.J.; González López, M.; Castedo, L. & Torres-Royo, J.M. (2008), A comparative study of STBC transmissions at 2.4 GHz over indoor channels using a 2 x 2 MIMO testbed, *Wireless Communications and Mobile Computing*, Vol. 8, No. 9, November 2008.

Rao, R.M.; Wiejun Zhu Lang, S.; Oberli, C.; Browne, D.; Bhatia, J.; Frigon, J.-F.; Jingming Wang Gupta, P.; Heechoon Lee; Liu, D.N.; Wong, S.G.; Fitz, M.; Daneshrad, B. & Takeshita, O. (2004), Multi-antenna testbeds for research and education in wireless communications, *Communications Magazine, IEEE*, Vol. 42, No. 12, December 2004, pp. 72–81, ISSN: 0163-6804.

Rupp, M.; Burg, A. & Beck, E. (2003). Rapid Prototyping for Wireless Designs: the Five-Ones Approach, *Signal Processing Europe*, Vol. 83, 2003, pp. 1427-1444.

Rupp, M.; Mehlfhrer, C. & Caban, S. (2006), Testbeds and Rapid Prototyping in Wireless System Design, *EURASIP Newsletter*, Vol. 17, No. 3, 2006, pp. 32-50.

Rupp, M.; Caban, S. & Mehlfhrer, C. (2007), Challenges in Building MIMO Testbeds, *Proceedings of European Signal Processing Conference (EUSIPCO 2007)*, 2006, Poznan.

SDR Forum: Software Defined Radio Forum (2009), http://www.sdrforum.org

SCA: Software Communication Architecture (2009), http://sca.jpeojtrs.mil

Sundance Multiprocessor, Ltd. (2009), http://www.sundance.com

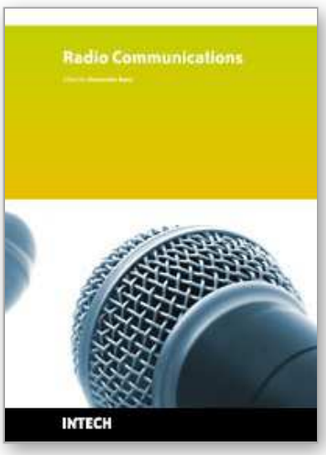
Telatar, I. E. (1999), Capacity of Multi-Antenna Gaussian Channels, *European Transactions on Telecommunications*, Vol. 10, No. 6, 1999, pp. 585-595.

Texas Instruments (2009), http://www.ti.com

Trygve, M. H. (1978), MVC, XEROX PARC, http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

Wilzeck, A.; El-Hadidy, M.; Cai, Q.; Amelingmeyer, M. & Kaiser, T. (2006). MIMO Prototyping Testbed with off-the-shelf plug-in RF Hardware, *IEEE Workshop on Smart Antennas (WSA 2006)*, 2006 Ulm.

Zhu, W.; Browne, D. & Fitz, M. (2005). An Open Access Wideband Multi-Antenna Wireless Testbed with Remote Control Capability, *Proceedings of 2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM 2005)*, Trento.



Radio Communications

Edited by Alessandro Bazzi

ISBN 978-953-307-091-9
Hard cover, 712 pages
Publisher InTech
Published online 01, April, 2010
Published in print edition April, 2010

In the last decades the restless evolution of information and communication technologies (ICT) brought to a deep transformation of our habits. The growth of the Internet and the advances in hardware and software implementations modified our way to communicate and to share information. In this book, an overview of the major issues faced today by researchers in the field of radio communications is given through 35 high quality chapters written by specialists working in universities and research centers all over the world. Various aspects will be deeply discussed: channel modeling, beamforming, multiple antennas, cooperative networks, opportunistic scheduling, advanced admission control, handover management, systems performance assessment, routing issues in mobility conditions, localization, web security. Advanced techniques for the radio resource management will be discussed both in single and multiple radio technologies; either in infrastructure, mesh or ad hoc networks.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Jose A. Garcia-Naya, M. Gonzalez-Lopez and L. Castedo (2010). A Distributed Multilayer Software Architecture for MIMO Testbeds, Radio Communications, Alessandro Bazzi (Ed.), ISBN: 978-953-307-091-9, InTech, Available from: http://www.intechopen.com/books/radio-communications/a-distributed-multilayer-software-architecture-for-mimo-testbeds

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen