SOFTWARE—PRACTICE AND
Softw. Pract. Exper. 2007; **37**:1087–1132
Published online 1 February 2007 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe.800

Implementing a generic component-based framework for telecontrol applications

Avraam N. Chimaris and George A. Papadopoulos*,†

Department of Computer Science, University of Cyprus, 75 Kallipoleos Street, P.O. Box 20537, CY-1678, Nicosia, Cyprus

SUMMARY

The rapid growth of telecontrol systems is one of the major trends in today's network-oriented community. The implementation of generic frameworks, consisting of reusable components that can form the basis for the development of such systems, is a necessity. There is a plethora of associated applications that can be developed in a distributed environment, such as audio/video teleconferencing, groupware and collaborative computing environments, remote controlled services, etc. In this paper we design and implement a generic framework of components that can be used for the realization of telecontrol applications. This category of applications focuses primarily on the issues of managing distributed units on remote end-systems. Such applications contain remote units and administrators that are connected and exchange data and control messages. We analyse the outlined architecture of our framework and the most important system operations. We also describe the communication protocol used in message exchanges between the constituent components. Finally, we illustrate the usefulness of our framework by presenting two applications that were created by extending the basic software infrastructure. Copyright © 2007 John Wiley & Sons, Ltd.

Received 6 January 2005; Revised 13 July 2006; Accepted 5 November 2006

KEY WORDS: telecontrol systems; distributed control system (DCS); supervisory control and data acquisition system (SCADA); component-based systems; reusable software components, patterns and frameworks

1. INTRODUCTION

Distributed computing technologies are extensively used in many parts of the industrial sector. There exist many categories of distributed applications, most of them with different types of requirements and operational functions. The degree of complexity of such systems has increased to

*Correspondence to: George A. Papadopoulos, Department of Computer Science, University of Cyprus, 75 Kallipoleos Street, P.O. Box 20537, CY-1678, Nicosia, Cyprus.
†E-mail: george@cs.ucy.ac.cy

the extent that dependencies between system components cannot be neglected. This necessitates the presence of frameworks to provide a methodology for integrating different components and assisting in the rapid development of so-called telecontrol applications. Although it is impossible in general to implement a generic framework for all types of applications, we would like to have available more focused frameworks for specific domains of distributed applications, comprising reusable components, which can then be used to 'instantiate' specific applications. In this paper we analyse the basic concepts of telecontrol systems; most of the applications in this domain focus on the notion of 'controlling and monitoring units'.

A typical telecontrol system provides complete and remote automatic control of the devices comprising it. It collects data, manages alarms and signals from the involved devices and, often, reconfigures remotely the latter. Such systems are very useful in areas which are difficult to access or in cases where it is not possible or desirable to have local staff. The architecture of a telecontrol system is simpler than that of a general purpose distributed control system (DCS) and can be seen as comprising a set of administrator units managing a set of remote devices. Typical telecontrol environments can be seen in holiday homes, caravans, boats, private homes and real estates. Typical telecontrol functionality includes the monitoring of various machines, passage control, tele-measuring, reading of gas, electricity and water consumption, etc. Currently, these applications are becoming increasingly popular, covering a range of domains beyond the house automation installations. In agriculture, for example, telecontrol applications are used for handling temperature and humidity alarms in corn fields, feeding machines in fish farms, frost danger alarms in weather stations, etc. The police are currently using telecontrol systems for traffic monitoring; remotely controllable traffic devices are instructed to self-configure their operations. Also, trailers and rented cars are using telecontrol for billing purposes; such systems are using the evolving global positioning system (GPS) technology combining monitoring with location awareness. One of the most important areas where telecontrol functionality is quite important is the industrial area. The catalogue of related applications in this area is quite long, spreading from passage control to fuel station tank control, machine alarms, automatic stock checking, etc.

In contrast with the main functionality of a telecontrol system, a DCS is typically composed of a heterogeneous collection of hardware and software entities scattered over a collection of heterogeneous platforms (operator stations, remote units, process computers, programmable controllers, intelligent devices) and communication systems (analogue, cabling, serial lines, field buses, LANs, satellite). A fully-fledged DCS may have to support mobility, hard real-time constraints, high fault tolerance or run-time performance. Such systems are typically used in applications such as process control, environmental control, mechatronics or power systems.

Consequently, a telecontrol system can be seen as a subcategory of DCS. Whereas in the case of the former category of systems, the involved devices are of a rather certain and focused functionality (typically monitoring), in the case of the latter systems, a broader set of distributed subsystems must be handled. In our framework, we combine characteristics from both types of systems in order to implement a framework that is capable of handling distributed devices such as a telecontrol system but that also supports centralized control such as a DCS. We view a typical telecontrol application as being one comprising a set of distributed heterogeneous devices with programmable application interfaces that are controlled and monitored by a distributed set of administrators.

In order to give a rough idea of what constitutes a telecontrol application, we will now highlight the major features of such applications. We have already mentioned that in such applications the user can manage and monitor a set of remote units or devices. This means that a set of instructions can be applied to the remote device or the device can notify by means of special alarms. Communication channels are implemented through various sets of communication technologies, such as the global system for mobile communications (GSM) and the associated general packet radio service (GPRS) which are well-known standards in wireless communication, HTTP, LAN, WLAN, Bluetooth, etc. These communication channels are used for exchanging the predefined control and alarm messages. However, there are many different ways not only to connect but also to extend the architecture of such systems. Initially, telecontrol applications were restricted to one-to-one interactions and the monitoring aspect was quite poor with regards to user functionality. Nowadays, telecontrol applications provide autonomous notification and monitoring mechanisms that simplify the administration costs of large systems.

The rest of the paper is organized as follows. In the next section, we briefly present the enabling technologies that can be used in developing telecontrol applications. In Section 3 we analyse the requirements that a telecontrol framework should address; based on this analysis, such a framework is presented in Section 4, along with the components and scenarios that were implemented to support telecontrol functionality. In Section 5 we present two applications that were implemented using our framework. The paper ends with reference to related and future work and conclusions.

2. ENABLING TECHNOLOGIES FOR TELECONTROL SYSTEMS

Contemporary control systems should pay particular emphasis to the software that is required to implement and run them. This software focuses on both issues of computation (i.e. the code required to run particular control units) and issues of coordination (i.e. the code required to realize communication between different control units). The latter is particular true for distributed control or telecontrol systems. It has been argued [1] that software technologies have often played a rather limited role in developing control systems. The purpose of this paper is to show that software technologies can in fact play an important role in the development of such systems.

With regards to the facilitation of software development for telecontrol systems, a number of software technologies can be used. Object technology, component-based software engineering along with frameworks and patterns can be used to create reusable components. Distributed computing and network technologies can be used to provide the required communication infrastructure. Middleware technology can be used to abstract away the details of the underlying hardware infrastructure. It has been argued that there is a historical separation between the control engineers who design controls and the software engineers who implement them [2]. Consequently, being able to provide generic and reusable frameworks that can be used to realize a range of control applications serves towards bridging this gap.

Software components can be used as a means of building reusable software. Such components may be built from scratch with the intention of being reused in the future; alternatively, they may already exist and be available for selection from software component repositories (the so-called commercial-off-the-shelf (COTS) components). We are particularly interested here in distributed components, namely those that interconnect and communicate through a network.

Component-based control systems allow the reusability of code across a number of similar applications, thus reducing development and testing time. In order to develop software components with a high degree of reusability, a particular domain of applications must first be analysed in order to derive a common basic framework of sharable functionality which is exhibited by all the applications in the domain in question. In this paper, this domain is the one for telecontrol applications and it is examined for common features in the next section. The components comprising a basic and extendable software architecture can be specialized further within individual applications that are built using this generic framework. In Section 4 we present such a generic framework for telecontrol applications, whereas in Section 5 we illustrate its reusability by means of developing two similar but distinct applications. Using component-based development of control software has some additional benefits, over and above that of reusability. It is easier to build fast a prototype of the system for simulation purposes (rapid prototyping) by using software components that behave like hardware devices. Also, interfacing between different control units can be realized in a standardized way, thus facilitating the integration of possibly heterogeneous devices. Finally, a component-based approach separates component development from system development. Consequently, control system integrators need not develop control programs; instead, they configure and logically couple components together using coordination principles and communication channels through which the components monitor each other and exchange information.

The framework we present in Section 4 makes heavy use of object technology and UML [3]. However, whereas object technology supports reusability at the lower level of code generation, patterns and frameworks allow reusability at the level of software architectures. The idea here is that applications over a sufficiently specific domain exhibit similar functionality, independently of what each particular application does. For instance, applications in the same domain may share common controller configurations or dynamic reconfiguration patterns. Thus, by combining object technology with frameworks, it is possible to design generic 'skeletons' that describe the basic functionality of a specific type of application; such skeletons are implemented as a group of basic objects with associated classes. Then, specific 'instances' of these skeletons comprising extended objects with subclasses inheriting the basic functionality from the superclasses of the generic skeletons themselves can be used to create specific applications. In Section 4 we present such a generic framework of telecontrol applications using the analysis performed in the following section.

So far we have focused the discussion on how to develop reusable components. However, another major factor in contemporary control systems is the communication between these components and their coordination. This is particularly important in the case of distributed or telecontrol systems with the constituent components executing in a remote fashion from each other. The topologies that can be formed in such scenarios range from typical client–server models to fully distributed ones. In the former case, one can differentiate between the case of 'thin' clients and 'fat' servers (where most of the processing is done at the servers' side and the clients are responsible only for presenting the results) and the case of 'fat' clients and 'thin' servers where clients have enhanced autonomy in processing capabilities and the servers are mostly used for storing shared data and possibly also for inter-client communication. It is also possible to have a sort of mixed setup with the clients exhibiting a somewhat autonomous behaviour but under the discrete supervision of servers. This is the setup we employ in our telecontrol framework that is described in this paper. In all these cases, distributed control exhibits some beneficial characteristics, such as concurrent execution, replication of functionality, robustness and scalability. However, these benefits come at the cost of needing a more elaborate communication scheme between the interacting components. In Section 4 we present a communication protocol that is used to provide distributed coordination and communication between the various control units.

The involvement of distributed control often necessitates the use of a middleware level of abstraction, located between the operating system and the software applications. Its role is to handle communication issues between the components executing in a distributed fashion and do so in such a way that the user is unaware of the components' physical location or the peculiarities of the underlying hardware architecture. Such middleware platforms provide the ability to automate communication tasks, integrate components in a standardized way and enhance interoperability across different machine architectures. There exist numerous such middleware models. One such model is Microsoft's component object model (COM) and its associated distributed component object model (DCOM) which allows COM objects to run and communicate in a distributed fashion [4]. Recently, Microsoft has introduced the .NET environment [4], supporting Internet-based connectivity between components. Another environment is the CORBA model [5], a software standard for component-based development, proposed by the Object Management Group (OMG). Yet another model is the Java-based proposal by Sun Microsystems, which encompasses basic infrastructure such as Java Beans and Enterprise Java Beans and remote method invocation (RMI) but also more holistic environments such as Jini [6].

On another front, the development of Internet technologies and service-oriented computing has prompted a few researchers to view the development of a DCS as a ensemble of Web services. Here, a typical such environment would use XML and SOAP for communication and coordination among the control system's constituents [7]. The XML technology is offering a simple and efficient way to exchange well-formed data between connected nodes [8]. The recent message-oriented middleware (MOM) servers are extending the standard XML technology into a powerful set of data structures, instructions and procedure calls, executing on distributed nodes [9].

As already mentioned, the use of middleware platforms provides useful abstractions for the easy integration of components executing in a distributed fashion. However, their use is not without its problems and limitations. Many of these middleware models are proprietary either in the underlying platform (e.g. Windows for Microsoft models) or in the language used (Java for the case of Sun Microsystems). CORBA is the only middleware that is truly independent of platform or language but it has often been criticized owing to the heavy overhead resulting from the rather elaborate way objects need to adopt in order to call each other; this problem arises from CORBA's attempt to address all possible functionality or requirements [1]. Furthermore, middleware platforms often do not deal with such issues as real-time or quality-of-service requirements and, instead, require large overheads on physical resources [10]. Finally, the learning curve for familiarization with a middleware platform in order to be able to use it effectively is often rather steep. As a result, we have decided to go for a rather 'light' framework that features simple development and flexibility; consequently, we do not make use of any middleware platform.

3. REQUIREMENTS FOR TELECONTROL APPLICATIONS

Figure 1 presents the infrastructure of a typical telecontrol system. This particular instance is built on GSM and provides a set of control and alarm messages.

In our framework, the telecontrol application is defined to be a set of distributed heterogeneous devices with programmable application interfaces that are controlled and monitored by a set of distributed administrators. We have mentioned in the introduction that DCSs offer the advantage of

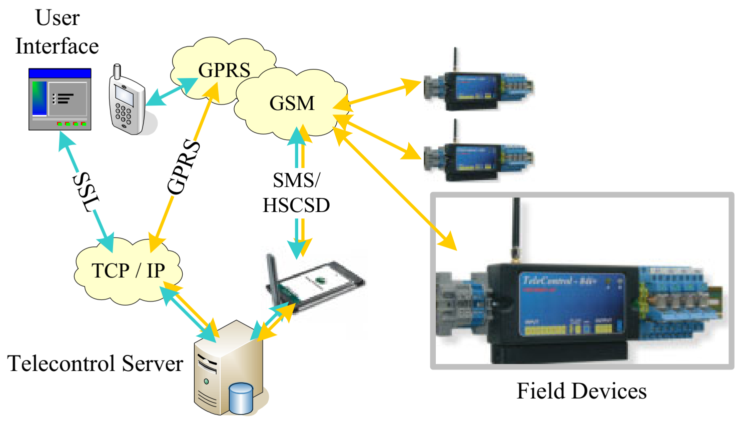


Figure 1. A typical telecontrol system.

centralized control, usually by an administrator located at a central location. Our framework distributes the centralized control over a set of controllers that are located on the administrators' sites by supporting more administration units. We envisage the DCS performing both monitoring and control of a process or device, thus acting more like supervisory control and data acquisition (SCADA) systems. Initial analysis of typical DCSs reveals that most of them consist of a remote control panel, some communication medium and the central control situated on the operator's site. The remote control panel is often referred to as a digital communication unit (DCU) containing terminal blocks, input/output modules (both analogue and digital), a computer or proprietary processor, and a communications interface. A DCU, depending upon where it is located, can perform both monitoring and control of processes. In a SCADA system these units provide both input and output modules for monitoring and control. The communication medium in a DCS is a cable or wireless link that serves to connect the DCU to the central control facility. The central control is used by a supervisor or operator that monitors and controls the processes. These units are supported by some human–machine interface providing a user-friendly environment for handling the distributed subsystems. Consequently, a DCS is comprised of a supervisory controller that runs on a central server and communicates with subordinate controllers via some form of peer-to-peer network that is running on the distributed subsystems. The supervisor sends set points to and requests data from the distributed controllers. The distributed controllers control their process actuators (switches, valves, flow controllers, etc.), based on requests from the supervisor like a simple telecontrol system. The SCADA systems that were previously mentioned consist of some central monitoring system (CMS) and one or more remote stations. The communications network is the medium for transmitting information between remote stations and the CMS.

The previous analysis of the DCS and SCADA systems reveals important architectural issues that could be used in our framework to enhance the distribution of administrators. Telecontrol systems can be seen as a subcategory of DCS but they are mainly focused on the 'handling of devices' rather than on the centralization of monitoring and control functionalities. DCSs and SCADA systems support more complex mechanisms that enhance fault tolerance, efficient communication and process

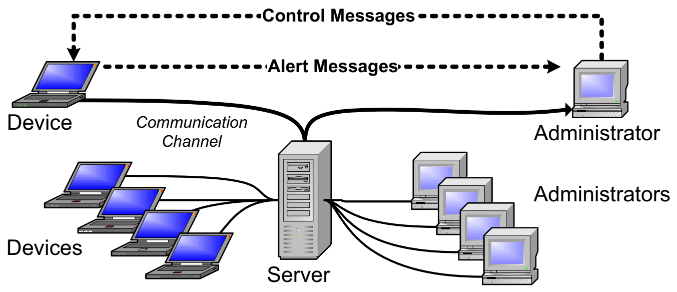


Figure 2. Generic architecture of a telecontrol system.

handling rather than handling devices interfaces, etc. The infrastructure of telecontrol systems follows a generic modelling approach similar to the DCSs. We can view such a system as a federation of agencies with each agency pursuing some objective through the execution of an operational cycle that is based on three interrelated activities: sensing, reasoning and acting [1]. In order to understand the most important constituents of a telecontrol system, we analyse the functionality issues related to the connected units and the communication between them. These aspects are critical in the implementation of any telecontrol system as they are the most important building blocks.

- *Connected units.* A telecontrol system comprises a set of distributed, autonomous units, connected through defined communication channels. The most important tasks of such systems are the ability to 'control' and 'monitor'. These tasks are implemented within a group of different participants. These participants are separated into the units that are monitored (administered devices) and the users that manage them (administrators). By generalizing this scenario, we can state that a telecontrol system contains a set of devices and administrators. These entities can communicate with each other directly or, as indicated in Figure 2, through a centralized server. The connected units are monitored and handled by the administrators through well-defined interfaces.
- *Communication.* The autonomous units in a DCS should implement a communication mechanism to send and receive messages in order to support their contributing task within the distributed system functionality. In a telecontrol system these messages are related to the 'monitoring' and 'controlling' functionality mentioned previously. For example, we can expect control messages to be transmitted from an administrator to a device. The device can notify the administrator of the occurrence of a specific monitoring event by using alert messages. This suggests the existence of a 'communication' infrastructure that is common among typical telecontrol applications. Even though in some cases this functionality can be extended to support more specific features, the 'monitoring' and the 'controlling' tasks are still supported within a common signalling approach. The communication infrastructure is implemented by means of channels that are not restricted to a specific communication technology. Such kinds of communication can be wired or wireless. This does not affect the functionality or the role of telecontrol communication channels as the latter should still transmit and receive messages from one unit to another.

We have already stated that telecontrol applications are a subcategory of DCS. A set of distributed units (devices) is controlled by a number of administrators through communication channels. In a telecontrol system, each administrator can manage a connected device independently of the work done by the other administrators. An administrator cannot affect the execution of the whole system, even if there are failures on a specific device. The communication channels can be effectively used by the distributed users in order to access the active distributed devices, even though some of them may be offline. These features characterize the major advantages of a well-formed distributed system. Figure 2 presents the generic architecture of a typical telecontrol system.

The main constituents of such a system are a set of *Devices*, a set of *Administrators* and a *Server* for handling communications. Note that there is a cross-cutting relationship between these three architectural entities and the two main aspects of a telecontrol system (connected units and communication), as these entities are units which also communicate with each other.

The devices must provide the required mechanisms for the following tasks.

- *Handling electronic devices connected to them.* These devices are typically sensors that can provide special information to the administrators (such as temperature, fire alarms, light status, etc.) but they also act as controllable units capable of receiving specific messages which can alter their behaviour.
- *Triggering data changes and alerting signals to connected units*. This feature is related to the required role of the devices to notify administrators of their status by using specific signals. These status alerts are handled by the connected administrators and when there is need they act by giving new instructions to the devices by means of control signals.
- *Receiving control signals*. This is the ‘passive’ role of the devices. They are receiving ‘control’ signals that are executed in order to change their behaviour.

The administrators have to provide the basic functionality for monitoring and controlling the devices (there is a one-to-one correspondence between an administrator and a device).

- *Monitoring device information*. This is feasible by ‘listening’ to notification alerts on their communication channel. When there are changes on the status of the connected devices, they will be notified in order to act by providing new directions to the devices.
- *Giving directions and commands signals to the connected devices*. They can send special control signals to the connected devices in order to force them to alter their behaviour (e.g. a light switch could be turned off).

The existence of a server is not a necessary feature in telecontrol applications. In some cases such systems use a shared dataspace within which messages are exchanged. In order to reduce the overhead and the traffic in such systems, these servers can be used to coordinate both connections and signals. This is more efficient system and the major functional requirements that must be provided by them are as follows.

- *Binding devices and administrators*. The server is responsible for supporting a communication medium that will assist devices and administrators in exchanging signals.
- *Coordinating data and control signals*. All messages are directed through the servers. They should have information for the sender–recipient pair to forward control and alert signals.

- *Supporting other important functionality issues*. Security issues, historical data archiving, communication channels handling, etc. must also be supported.

The above features highlight the major functional requirements of a typical telecontrol system. There are also some non-functional requirements that guarantee the proper functioning of the system.

- *User interface*. A telecontrol system is required to provide a user-friendly interface with which the administrator will be able to handle the connected devices quickly and easily. The existence of such interfaces is burdened by the fact that these devices often vary in type. 'Generalizing' the user interface often produces difficult to use communication patterns.
- *Performance characteristics*. Timing is often important in a telecontrol system. The alert signals may be transmitted from the devices to the administrators under timing constraints. Delays in signal reception could cause a delayed reaction from the administrators that could have an impact on the functionality of the system. Similarly, the control signals that are transmitted from the administrators to the devices should also arrive within bounded time.
- *Error handling and extreme conditions*. Most of these situations are related to signalling and connectivity losses. In telecontrol applications the reliability of the transmitted signals is important. The loss of signals could reduce the performance of the system because retransmitted scenarios will produce retransmissions that could affect the traffic within the communication channels. The connections should be monitored in order to avoid loss of signals.
- *Quality issues*. We can detect various factors that will improve the provided service of a telecontrol system: reliability, interoperability and maintainability are only some of the quality issues that should be provided by the system.
- *Physical environment*. The physical environment is related to the communication medium that is supporting the connected units.

The above analysis is on a par with other assessments made for similar categories of control systems. In [11], the authors list some functional aspects that must be addressed in the building of control systems for building automation. Most of the aspects discussed, namely access control, intrusion and safety alarms, fault detection and monitoring, are also addressed by our analysis. A similar analysis has been presented in [10] for the development of a component-based DCS for assembly automation, in [12] for the development of a middleware-platform-based system for robotics and in [13] within the general context of DCSs. Also, in [14] the authors present a model for component-based control systems, featuring a number of layers, such as communication and coordination, device control, etc. These layers are similar in nature to our dedicated connected units and the mediator. A similar layer-based approach, where the particular layers proposed (communication layer, control layer, application layer) are represented in our analysis by the dedicated connected units and the mediator, is presented in [15] in the context of next-generation industrial automation environments. Consequently, the issues addressed in this section are the major characteristics of a typical telecontrol system. The proposed framework, presented in the next section, builds on the analysis of this section.

4. PROPOSED TELECONTROL FRAMEWORK

Following the analysis given in the previous section we now correlate the previous constituent system elements to our three types of required units and design a structure that is closely related to the generic architecture shown in Figure 2. The proposed units are the *remote units* (handling a set of controllable devices), the *control centre* (the server) and the *administration units* (administrators). The devices in Figure 2 are not represented in our framework as single controllable units, but rather we group them under the control of a remote unit. Remote units are small 'intelligent' subsystems that have the ability to communicate with a predefined set of devices. Consequently, every remote unit handles a number of devices. Such devices have been mentioned in the introduction (alarms, heating, lights, etc.). In our approach we also use a server unit for binding various sets of remote units and administration units. This server unit is defined as a control centre, as it is the central mechanism for controlling a telecontrol application. The control centre implements the central communication unit to which both remote units and administration units can get connected and exchange data and control messages. For example, by administrating the message flows, it can notify administration units about changes to a remote unit's status and specific control signals from the administration units can trigger predefined changes on a remote unit. It is quite clear that the communication channels are initiated through the control centre which is responsible for redirecting and handling communication scenarios. The administration units, on the other hand, are used to monitor and control the connected remote units (implicitly, an administrator unit handles a number of devices through their supervisor remote unit). Owing to their communication capabilities and implementation they are separated into the remote administrators and their Web-based instances, the mobile administrators. Although the roles of these two types of administrator are similar to one another, their implementation is based on different communication techniques. The following section gives more information on this issue. Figure 3 shows the basic structure of our generic telecontrol framework, based on the analysis of Figure 2.

All the previously described units were implemented by reusable components that provide the required functionality to support a telecontrol application. Although a complete listing of all the possible communicating scenarios involving these units cannot be compiled, it is evident that the most important of them involve the notion of 'controlling' and 'monitoring' remote devices. In this section we describe the generic building blocks that can easily be used to instantiate a telecontrol application.

By using the previous functional and non-functional requirements of a generic telecontrol system we identify the following critical issues that were implemented in our framework. The remote units must provide the required mechanisms for the following tasks.

- *Handling remote devices connected to them*. Each remote unit is responsible for handling one or more remote devices. For example, in a telesecurity system a remote unit could handle the lockers and sensors of a specific apartment. For functionality purposes, we implement a grouping mechanism that groups remote units under logical relationships. In this case, a number of remote units that handle an apartment's security are grouped into logical sets representing buildings and neighbourhoods. These 'groupings' are handled by the control centre that organizes the presented information in a 'tree-like' fashion, highlighting relationships between remote units. The grouping is helpful because related units are gathered into related sets and thus can easily be found. In Figure 4 we present this functionality by using a telesecurity example.
- *Triggering data changes and message alerts to connected units*. They monitor remote devices and when there is a change, the new data are transmitted to the connected units through the control centre.
- *Receiving update and control messages in order to change their behaviour*. They receive and execute the instructions sent by the administration units.

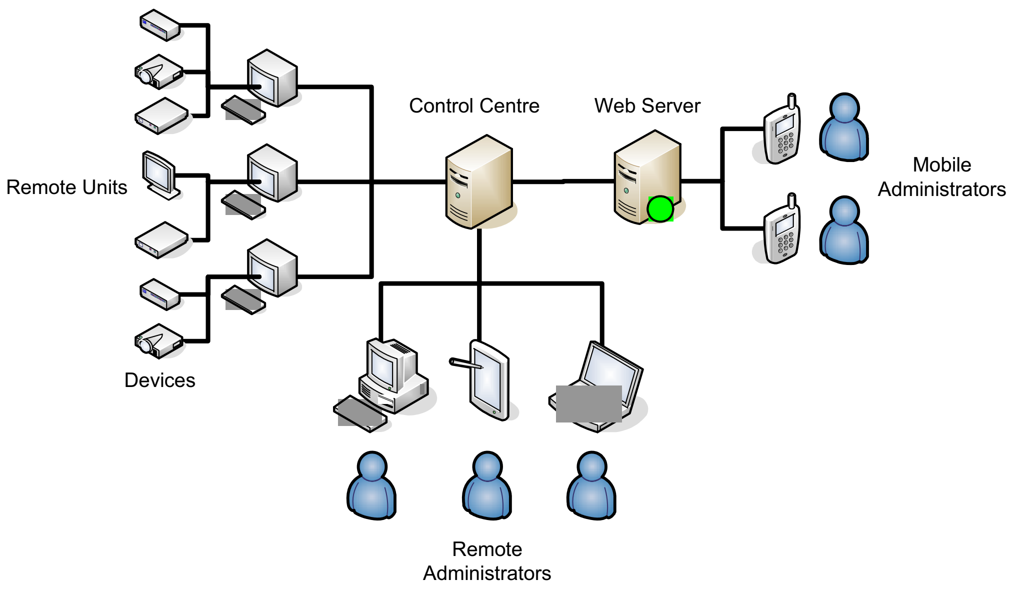


Figure 3. A hierarchical generic framework of a telecontrol system.

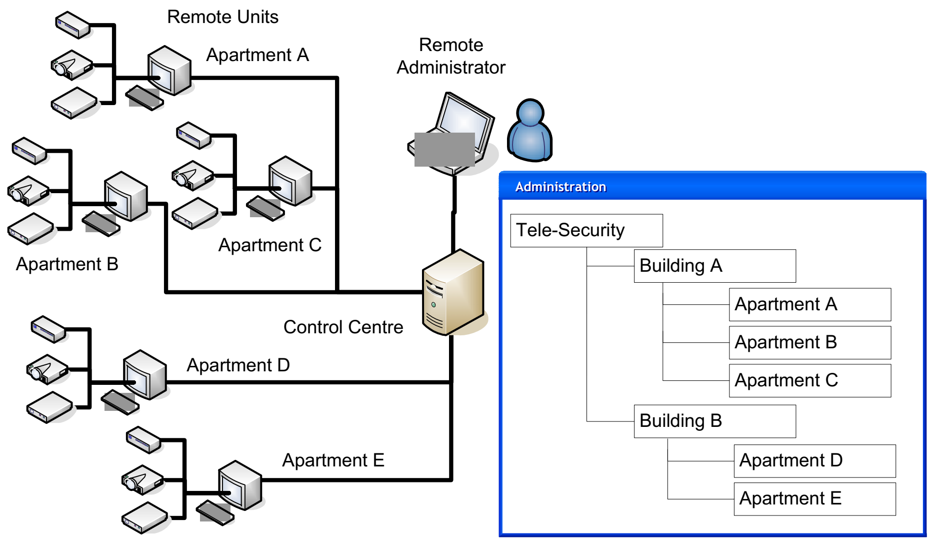


Figure 4. ‘Grouping’ remote devices according to functionality.

- *Supporting secure and consistent communication*. This non-functional requirement is crucial for the proper execution of telecontrol message flows. These mechanisms guarantee message transmissions in a secure and reliable manner.
- *Ensuring the availability and efficiency of the communication channel*. By using this function, a remote unit can detect a disconnected session. The message history can be used to reproduce any disrupted communication scenario in order to drive the application into a consistent state.

The administration units have to provide the required functionality of monitoring and controlling remote units.

- *Monitoring remote units information*. The available remote units should be presented into the 'grouping' form that is described below.
- *Giving directions and commands to the remote units*. This is the controlling functionality that supports the flow of direction to the requested remote units.
- *Supporting secure and consistent communication* and *ensuring the availability and the efficiency of the communication channel.*

The control centre is used as a 'server'. This unit supports all the required functionality for coordinating connected units (remote units, administration units) and messages (control, alert). In order to be more specific, we summarize the following major functionalities of this type of unit.

- *Binding remote units and administration units*. The administration units are requesting control of a specific remote unit. The control centre is responsible for 'locking' the requested remote unit and taking part as a server in the communication scenarios.
- *Coordinating data and control messages*. All given messages have a sender and a receiver. The control centre analyses these messages and forwards them into the proper communication channels.
- *Ensuring the availability and efficiency of the communication channels with remote units and administrators*. When there are disconnected sessions, the control centre 'pauses' the disrupted communication scenarios and gives some extra time for reconnection before a timeout rollbacks the executed actions.
- *Supporting mechanisms to ensure the consistency of message flows in the communication channels*. This non-functional requirement deals with the correct flow of messages, based on the occurrence and steps of communication scenarios.
- *Supporting the 'grouping' mechanism*. This is a helpful functionality that presents remote units in a friendlier manner to the connected administrators.

We will show in the following sections how these issues help us to separate the functionality into specialized components. Figure 5 presents the major packages of our architecture that groups the constituent components of each role.

In the following section we present the major sets of classes and components for each telecontrol unit. These sets express the fundamental functionality that was derived from the analysis in Section 3. The distributed sets of components communicate through channels and exchange data based on their role in the system. Initially, we will discuss some communication issues (message structure, communication scenarios). Then, we will extensively analyse the internal structure of each telecontrol unit by presenting the communication scenarios in which the unit is participating.

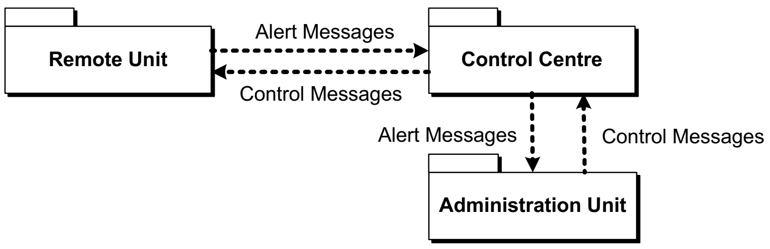


Figure 5. The architecture of our telecontrol framework.

Although the framework components could be named after their architectural counterparts, we prefer to use slightly different names, so as to be clear at any time whether we are talking about the software architecture of the system or the derived implementation framework.

4.1. Communication issues

Any distributed system must necessarily have an infrastructure of communication channels between its remote parts that exchange messages in a well-formed communication protocol [16]. In order to implement the communication aspect, it was necessary to create a telecontrol protocol which would cover the needs related to communication issues of such systems. We did not implement the low-level communication layers but we structured a communication protocol based on the logic and the functionality of our units. Similarly to other DCSs [17], we implement specific types of messages that are transmitted through the underlying communication channels. In our framework we use generic communication components that are responsible for initializing the appropriate mechanisms for handling the supporting communication (e.g. Bluetooth, Wifi, GPRS, etc.). Therefore, our communication infrastructure is generic and various types of communication combinations can be supported. For example, we could use the proposed framework by implementing a control centre that will handle WiFi-enabled remote units and Web-enabled remote administrators. Message transmission and parsing is automatically supported in well-formed classes that are implemented on top of these technologies (e.g. Web services to support Web-enabled applications, classes that implement a Bluetooth serial communication, etc.). This communication strategy was extended by a set of communication scenarios to guarantee the proper execution of the required communication tasks. At a higher level of communication, such scenarios can support the synchronization mechanisms of the distributed information, the alert signals handling, etc. In the following sections we analyse the implemented communication scenarios and the types of messages involved.

4.1.1. Message types and communication protocol

The analysis of telecontrol systems showed that in communication we should implement the basic mechanisms of sending/receiving ‘monitoring’ and ‘control’ messages. These messages are encapsulated into predefined sets of communication scenarios that involve all types of connected units. The separation of monitoring and control messages somehow underlines the ‘direction’ of the signals.

Owing to the fact that monitoring is information provided by the remote devices, we expect to have defined monitoring messages that are sent from the remote units to the administration units, in contrast to the control that is supported by a set of signals that are directed from the administrators to the remote units. In order to support a 'protocol' between the telecontrol units, we separate the communication needs into the following groups of communication tasks for each type of unit.

- A remote unit is required to handle the following functions:
    - notify an administration unit that there are changes to its status;
    - accept and execute control instructions to change behaviour;
    - provide its initial status as soon as it connects to the telecontrol system.
- An administration unit is required to handle the following functions:
    - obtain information concerning any connected remote units in order to be able to select one of them to monitor and control (this information should be up-to-date and be refreshed continuously);
    - be notified of any changes on the monitored remote unit;
    - provide its initial information as soon as it connects to the telecontrol system;
    - give directions (control signals) to the monitored remote unit to change behaviour.

The control centre acts as a server so we have to expect a behaviour that ensembles its role in the system. It should decide, redirect and coordinate all communication messages in order to ensure a smooth flow of information in the telecontrol channels. This central unit is responsible for providing and triggering the 'notifying' mechanisms that are required in the adopted 'observer pattern' [18].

- The control centre is required to handle the following functions:
    - trigger the 'notifying' mechanism with regards to any changes to the remote unit (the administrators that are monitoring the specified remote units should be informed when an event occurs, i.e. a change or alert);
    - coordinate and redirect the arriving communication messages;
    - store and update the local information that contains the data of all communicating units.

The above needs that can be easily realized from the analysis of telecontrol systems have to be coordinated into communication chains as a transactional approach. Before we proceed into the analysis of these chains (communicating scenarios), we have to specify the types of messages that are required for implementing the communication aspect. Initially, we analysed the required information that was necessary to be exchanged between the communicating units in each task. We identified the following set of information types, based on which the format of the communication messages was designed.

- Local and distributed information, stored in units. This type of information contains the data that store the status of remote units and administration units within their local spaces. This distributed information has to be merged into a global repository in order to implement a telecontrol database which the communicating units can access. The monitoring mechanism uses such information that is usually provided by the remote units to the administration units. The data repository is situated on the control centre that is responsible for coordinating the system's data. Based on this type of information, we implemented the following types of messages that are dealing with data handling.

  - *Data message*. This type of message can hold the information that characterizes a connected unit (both remote units and administration units). For example, a remote unit uses this type of message in order to send its structured data to a connected peer. This type of message could be implemented, for example, by Web service calls that will serialize the units' information as arguments. Using Bluetooth communication, this information can be sent through structured XML packages [19] that hold the required information. In our implemented applications we use a Data Model that is designed over XML.
  - *Presentation style message*. This type of message is used to send a presentation style preference to a the connected peer. This style is used to present the received data in the desired form on the connected peer unit. In our implemented applications we used XSLT [20,21] transformations in order to present information as a Web form. There is also the possibility of providing a 'customizable' style that can be used to dynamically draw the required control for presenting the received information. This type of message is handled only by the administration.
  - *Update message*. This type of message is used in updating the structural data at the connected peer. The 'observer pattern' implementation uses such messages in order to inform the connected units of changes and events. This type of message is important to keep the monitored information up-to-date.
  - *Alert message*. This type of message is used when notifying administrators of certain incidences on remote units. This type of information is important in the monitoring scenario as the alert should activate the administrator to take some actions on the remote unit (its local device). The alert message uses a name for the alert and a priority flag to declare the importance.

- *Control signals*. This type of information (instruction) is directed from the administrators to the remote devices. The control signals are quite similar to the updating messages but move in the opposite direction. The final target is the referred remote unit that will have to apply the requested instructions.
- *General communication signals*. In most cases of implementing a custom communication 'protocol' between some communicating units, generic messaging proves to be an important method for dealing with information crucial to the system. The coordination between the connected units is made feasible by means of a set of generic signals that are implemented to support data synchronization, and updated lists on the connected units.

The above messages can be used to implement communication scenarios to support the major functionality of a telecontrol system. These scenarios deal with all the predefined information and messages that were previously mentioned. These scenarios are reusable and are encapsulated into the building blocks of any other instantiation that uses our framework. The most important such scenarios are presented below.

- *Scenario 1: Registering a remote unit or an administration unit to the control centre.* This scenario occurs when a new unit is connected to the control centre. During the registration, a data message (DATA_MSG) is passed to the control centre with the required stylesheet in a presentation style message (ST_MSG). The information that arrives at the control centre is stored in the local repository. During the registration of a remote unit a mechanism is triggered that notifies all the connected administrators that a new remote unit has been connected (Figure 6).

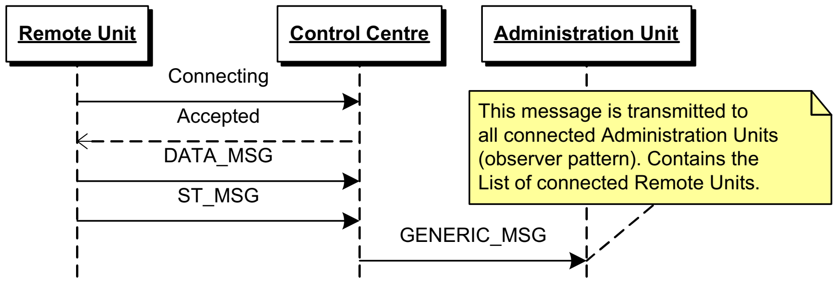

Remote Unit
Control Centre
Administration Unit
Connecting
Accepted
DATA_MSG
ST_MSG
GENERIC_MSG
This message is transmitted to all connected Administration Units (observer pattern). Contains the List of connected Remote Units.

Figure 6. Scenario 1: Registering a remote unit or an administration unit to the control centre.

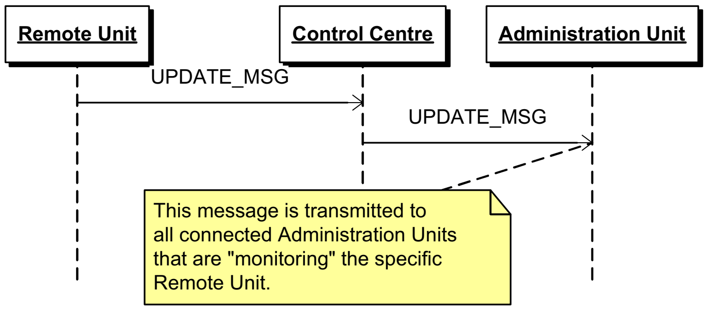

Remote Unit
Control Centre
Administration Unit
UPDATE_MSG
UPDATE_MSG
This message is transmitted to all connected Administration Units that are "monitoring" the specific Remote Unit.

Figure 7. Scenario 2: Notify all administration units of status changes/alerts on remote units.

- *Scenario 2: Notify all administration units of status changes/alerts on remote units.* This scenario is related to the 'monitoring' issue of a telecontrol application. In this scenario the remote unit transmits an update (UPDATE_MSG) or alert (ALERT) message to the connected administration units that are currently monitoring it. Figure 7 shows the messages in this scenario.
- *Scenario 3: Give controlling directions to a specific remote unit.* This scenario is directed from an administration unit to a monitored remote unit. The communication is quite simple because it uses control signals (CONTROL_MSG) that are passed through the control centre to the proper remote unit.
- *Scenario 4: Unregister remote units and administration units from the control centre.* When a failure occurs, the telecontrol system should be informed of any 'disconnected' units. In this case the lists of connected units are refreshed and the monitoring of currently unavailable units is rejected for consistency reasons. Figure 8 shows which messages are transmitted when a monitored remote unit is disconnected and becomes unavailable to the administration unit. These messages are general communication signals (GENERIC_MSG).

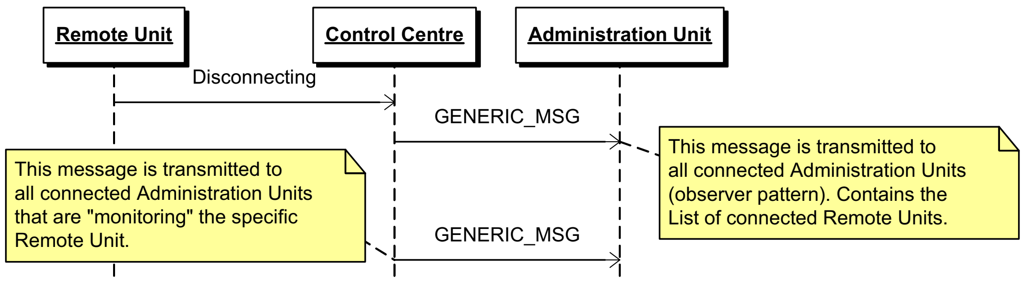

Remote Unit
Control Centre
Administration Unit
Disconnecting
GENERIC_MSG
This message is transmitted to all connected Administration Units (observer pattern). Contains the List of connected Remote Units.
This message is transmitted to all connected Administration Units that are "monitoring" the specific Remote Unit.
GENERIC_MSG

Figure 8. Scenario 4: Unregister remote units and administration units from the control unit.

The above communication scenarios represent some of the most characteristic chains of telecontrol messages. There are some additional scenarios that are related to the functionality of our framework, but for the purposes of this paper it is not necessary to provide details on how these work. The above scenarios cover the major issues of communication. All the above communication mechanisms are encapsulated into our reusable components that can be used to implement telecontrol applications. The developer uses the above scenarios as soon as they start using the core communication classes of our framework.

4.1.2. *Communication classes*

As has previously been mentioned, units are connected by means of communication channels. The following observations can be made:

- a remote unit is connected to the control centre in a one-to-one communication setup;
- various sets of remote units and administration units are connected to the control centre (the control centre uses a large set of communication channels, one for each connected unit);
- an administration unit is connected, through a one-to-one communication channel, with the control centre.

The above observations refer to the networking of these systems. The ‘logical’ communication is slightly broader because the communication scenarios implement ‘logical communication channels’ from an administration unit to a selected set of remote units. In our approach, each administration unit can monitor and handle more than one remote unit at some specific time. In this section we do not deal with the logical layer of the communication but rather with how this communication is implemented in the physical layer. The major classes that are used in communication are the *communicators* and the *communicator switches* that are related to the number of channels that can be handled. For communication that is implemented on a one-to-one fashion, a *communicator* class is used. If there is a need to handle more than one connected unit (the control centre), then a *communicator switch* is used. Both remote units and administrators are required to implement a single communication channel with which they can be linked to the control centre. In this case a *communicator* component is used to support communication by providing an interface to initialize, send and receive notification of messages. On the other hand, the control centre is required to have a set of channels supporting a variable number of remote units and administrators. The control centre uses two *communicator switches* to handle remote units and remote administrators. The *communicator switches* use an identification mechanism in order to identify which is the connected unit. In the following sections we give more details on how these classes cooperate in a unit's architecture.

4.2. Telecontrol units

In this section we describe how the telecontrol units are implemented in order to support the generic telecontrol system functionality. Although the functionality of such systems varies based on different sets of requirements, there is a core set of the latter that is common to the whole family of these systems, as our previous analysis has shown. In the following sections we analyse these core components as they were implemented to support the required functionality.

4.2.1. Remote units

Some of the activities included in this unit have been described in Section 3 and the beginning of Section 4. The remote units are monitored by the remote administrators. For simplicity, our implemented framework provides a one-to-one relation between the remote units and remote administrators during the update process. In this way we avoid any inconsistency scenarios when different updated copies are situated on remote administrators. The remote administrators are not restricted to requesting a remote unit but this unit should be 'locked' to disable any update attempts by other administrators. The remote units manage and provide the data for their connected devices and support the mechanisms for notifying and alerting with regards to specific changes. They have to be active for receiving instructions and ready for immediate response. They should control their local devices and manage instructions by forwarding them to the proper recipients. By separating these concerns we detect the major classes that provide the main features of this component. First, we note that most of the communication scenarios supported, as described in Section 4.1.1, involve the administration of local data (data of connected devices). A class was necessary to store the data structures and to support functionalities for loading and updating information. Our remote units use a data class, called *cUnit*, for storing and handling local data. All the communication messages for this class have monitoring or updating directions. Second, for the communication issue, a *cCommunicator* class was used to support a messaging mechanism through the current connection channels (Wifi, Bluetooth, etc.). Figure 9 presents a class diagram for the remote unit (white coloured boxes in this and other similar figures denote reusable components, whereas grey coloured boxes denote application specific components; this will be further explained in Section 5). In this diagram both *cUnit* and *cCommunicator* are shown. However, it was important to somehow coordinate the previous classes into a class that was able to utilize the concept of the telecontrol communication protocol. The responsible class is the *cRU_ControlUnit* that manages both communication and local data. Similarly, in all the other types of units, there are classes similar to *cRU_ControlUnit*. Their role is crucial to the system as they process the communication messages in order to execute the defined local processes. For example, the *cRU_ControlUnit* can receive an update message and use it to update local data. Some of the major activities the *cRU_ControlUnit* class can perform are summarized as follows.

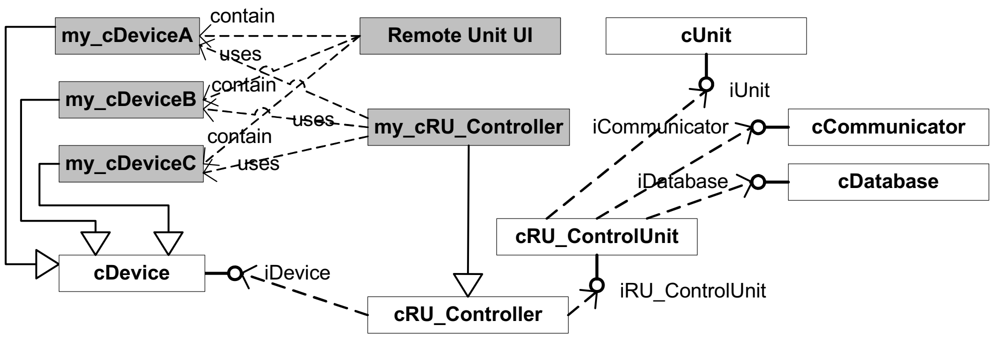


Figure 9. Remote unit structure.

- To handle the *cCommunicator* for connections to the control centre.
- To perform the activities on communication messages arriving from *cCommunicator*. These messages are mainly updating and teleconferencing messages that are directed from the administrations units to the specified remote unit.
- To deal with further functionality deriving from the communication scenarios that are mentioned in Section 4.1.1. As shown in Figure 6 (Scenario 1), the remote units have to send their initial data to the control centre after they are connected. During this task, the *cRU_ControlUnit* receives information from the *cUnit* and sends it to the administration units via the communication channel in *cCommunicator*.

Another responsibility for this type of unit is to retrieve and update data on certain devices. These devices are typically electronic devices that are connected to the remote units and handled by a defined set of commands. In this paper we will avoid defining any external signals, as we pay emphasis primarily to how these telecontrol units cooperate. In the two applications that we implemented by means of this framework, we used ‘simulators’ that play the role of connected devices. These classes that support ‘device components’ are called *cDevice* and they encapsulate the basic functionality for handling a device. In the instantiations of a new application that uses our framework, the *cDevice* class should be extended by each supporting device in order to be linked into the application infrastructure. In order to support the customization and use of a remote unit, we implement a new class (*cRU_Controller*) that manages and binds both the *cDevice* and *cRU_ControlUnit*. Similar to the extending form of *cDevice*, *cRU_Controller* is extended with the type of the instantiated application. Each application can extend this class in order to provide a user-friendly environment that can support the tasks of the selected instantiated application.

Before we proceed with the analysis of the other telecontrol units we will present the major data and control flows in the remote unit implementation. These flows highlight all the required activities that are expected by a typical telecontrol system.

- *Initializing a remote unit*. In this process, the remote unit instantiates its internal objects (*cRU_ControlUnit*, the *cDevice* and the *cDatabase*). The *cUnit* receives information from *cDevice* in order to correspond to the current status of the connected device.

- *Connecting to the control centre*. In this process, the *cRU_Controller* uses the *cRU_ControlUnit* to accomplish a connection with the control centre. The *cRU_ControlUnit* handles the contained communication class (c*Communicator* class), which is capable of initializing the required communication channel. After a successful connection, a 'copy' of the remote unit's data (the status of the connected devices) is transmitted to the control centre which is responsible for storing these data in its local collection.
- *Updating device status*. This process is triggered when a change occurs on *cDevice*. For example, in one instance of the generic framework, namely the telemedicine application, *cDevice* triggers the process when a patient receives a pill from a stack. This occurrence generates an event that notifies the *cRU_Controller* to inform the control centre of the new status change.
- *Sending alert messages*. The alerts are a category of messages that are triggered when some situation occurs. In the telemedicine application, alerts are sent when a pill stack is empty or when the patient did not take their medication within a certain time period after being notified to do so. In these situations an appropriate structured message is transmitted to the control centre in order to inform the administrators about this incident.
- *Retrieving update messages*. This process is triggered after the c*RU_Communicator* receives an update message. A control centre or an administrator sends the update message after they change their 'local' copy. We have mentioned that each remote unit sends its data as a 'copy' to the administrators. With this process the original data that is stored in the remote unit (*cUnit* Class) is changed, owing to the update command.
- *Closing the connection with the control centre*. This process involves the c*RU_Controller*, the *cRU_ControlUnit* and the *cRU_Communicator*. The *cRU_Controller* sends a disconnect message to the *Control Unit*, which notifies the *cCommunicator* to close the communication channel (LAN, WLAN, Bluetooth, etc.).

At this point we highlight the basic steps that must be followed in the implementation of a new instance of a remote unit. For example, in the telemedicine application, the remote units are handling devices that manage a set of stacks holding a number of pills. A medication is submitted to the device and only when the medication time is reached is the patient notified in order to take their pills. In this application 'instantiation', the *my_cDevice* component extends the generic *cDevice* component in order to provide a graphical user interface (GUI) that supports a specific set of functions that correspond to telemedication functionality. In a similar way, the same package can be used to implement a new telecontrol application. The application developer must take care of the following issues while implementing a new remote unit instantiation.

- To code all events that the c*RU_ControlUnit* triggers, through the binding of the extended *cRU_Controller*. These events are important because they represent control signals that are received from the device through the connection of the control centre.
- Some of these events must have a 'linked' functionality to the *cDevice* components. When specific events occur, the implemented coding of the developer will help the *cDevice* component to receive automatic notification and change status accordingly.
- *cDevice* must be extended for sending the alert messages through an already extended *cRU_Controller* (e.g. quantity of pills in a stack has changed, etc.).

By using the above steps, the developer can implement a remote unit that will contain the required functionality based on the proposed framework and restrict the developer's tasks to defining the extra coding of designing and creating communication with the devices.

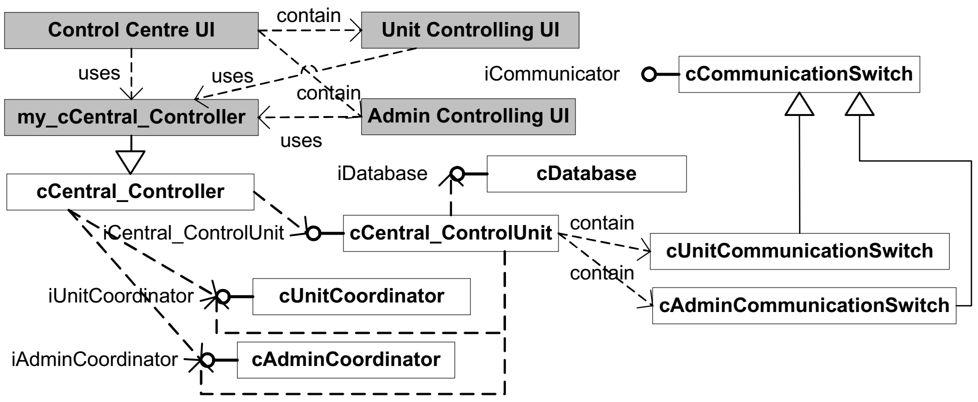


Figure 10. Control centre.

4.2.2. Control centre

The control centre is a major entity in our framework. It is the middleware unit, the coordinator, the message handler and the mediator of the telecontrol framework. The control centre should normally run on a powerful device and use a Web server in order to implement the mobile administrator role (Section 4.2.3). The control centre device uses predefined ports through which units are connected and structured messages are sent and received. It is quite obvious that the control centre must separate the remote unit and administrator connections in order to split the functionality into different parsers and activity handlers. To that end, a class that handles two communication classes (*cCommunicatorSwitches*) and two coordinating classes (*cUnitCoordinator* and *cAdminCoordinator*) to serve the remote units and the administrators, respectively (see Figure 10) has been used. The two coordinating classes coordinate the data classes of remote units and administrators respectively. The received messages in the *cCommunicatorSwitches* are parsed in the controlling class (*cCentral_ControlUnit*), triggering associated activities to be executed by the coordinating classes. The major components that comprise the control centre unit are the c*Central_ControlUnit*, *cCommunicatorSwitches*, *cUnitCoordinator* and *cAdminCoordinator* components. In the implementation of a control centre it is important to implement a GUI in which both remote units and administrators can be handled. This GUI should be implemented by a set of controlling classes, similar to the c*RU_Controller* that was used in the remote unit. The units' controlling class is used for presenting and handling the remote unit's data. During communication with the connected unit a data structure (received with a *Data_Message*) and a *Presentation_Style_Message* are passed by the remote unit in order to create an identical copy of the remote unit data on the control centre unit and a proper presentation. These structured data are received and used together in order to present a remote unit's data structure in the preferred stylesheet. On this form a group of controls is also added in order to offer a set of functionalities that can change the remote unit's data.

The structure of the control centre is shown in Figure 10. The major functionality of the control centre is contained in the communication of the *cCommunicatorSwitches* and the *cCentral_ControlUnit*. The *cCentral_ControlUnit* receives a defined set of structured messages with which it updates local copies and forwards messages to remote units and administrators. In order to define the sender of the structured messages of the switches, an identification mechanism is used that uniquely determines the correct source. In our framework we have implemented all updating scenarios between the telecontrol units; these are briefly described below. In the following case, *Update_Messages* are exchanged between the communicating units in order to achieve a consistent state of the distributed replicated objects of the whole system.

- *A remote unit changes its local data*. The control centre is notified and it changes its local copy by receiving an update message. If the copy is already requested by an administrator unit then the same update message is forwarded to that specific administrator. A set of identifiers is used to denote which remote administrators are requested to handle that specific remote unit.
- *The control centre changes a remote unit's data on its local 'copy'*. The control centre notifies both the remote unit and the administrator (if someone is currently handling that specific remote unit) to update the latter.
- *An administrator changes a local remote unit's data*. The control centre is notified by an update message and it changes its local copy accordingly. Then the control centre forwards the same update message to the remote unit in question.

Other scenarios that are not related with the 'server' role of the control centre are described below. These scenarios deal with the initialization phase and when there are connections and disconnections on the control centre.

- *The initialization of the control centre*. In the initialization process, the control centre prepares the *cCommunicatorSwitches* to accept connections from remote units and remote administrators. These switches are used as 'servers' that listen for requests from 'clients' (client–server communication). This process also instantiates the two coordinating classes which manage the two sets of connected remote units and administrators.
- *A remote unit is connected/disconnected from the control centre*. In this case the control centre updates the local list of connected remote units and then updates the list of connected units in all connected remote administrators by using a generic message. If an administrator uses a disconnected remote unit, then it is notified in order to release the copy of the remote unit's data. During the connection of a new remote unit, a *cUnit* class is created to handle the remote unit's data and a synchronization message is sent back to the remote unit.
- *A remote administrator is connected/disconnected to/from the control centre*. It updates the local list of connected administrators and if the remote unit that previously requested it is not released, the control centre automatically releases it. During the remote administrator's connection a *cAdmin* class (see Section 4.2.3) is created to hold administrators' data and a general message is sent in order to inform the administrator of the remote units that are currently connected to the control centre.

The major functionality that needs to be implemented by a programmer to build the control centre is associated with the 'linking' functionality of the control unit triggers and the controlling classes. This unit focuses on the presentation and controlling manner. The 'coordination' code is included in the core components and a 'programmer-defined' *cCentral_Controller* container must be used to handle not only the core components but also the controlling classes.

4.2.3. Administration units

The administration units are separated into two main categories. In the first category we have the remote administrators that are simple 'channel'-oriented applications that instantiate a communication channel through a wireless or wired networking infrastructure. In the second category are the mobile administrators that are 'Web'-oriented and they use Web communication in order to access the control centre and its connected remote units. In this case we use a Web server which is responsible for instantiating a defined set of classes to handle communication. These units are supporting a one-to-one relation during the update procedure but they can handle any remote unit that is available for administration.

First we will describe the remote administrators, because the structure of the mobile administrators is an enhanced model of the former with many similarities. The remote administrator structure is quite similar to that of the remote unit. It uses a local copy of a remote unit's data that is updated and informed by *Update_Messages* that are sent initially to the control centre and are subsequently forwarded to the original remote unit. The major activities that can be performed by this unit are as follows.

- *Connect/disconnect to/from the control centre.* This action is performed during the initialization of the remote administrator.
- *Request a selected remote unit by using a list that informs the administrator about the available units.* Each remote administrator is informed by the control centre of the currently connected remote units. In this list all the identifications are packed in order to help the remote administrator to send messages with a specified recipient.
- *Change the local copy of a remote unit that is retrieved from the control centre.* Changes are sent by using an update message to the control centre. The control centre afterwards notifies the remote unit in order to update the original source.

The remote administrator uses a *cCommunicator* to connect to the control centre, like a remote unit. The c*Admin_Controller* exists in the structure of this unit in order to be able to perform changes on copies of remote units. The administrator uses two data classes. The first is a c*Admin* class that contains local information and the other is a *cUnit* class that contains the copy of the selected remote unit that is currently handled by the administrator. The structure of the remote administrator is presented in Figure 11. The functionality of the remote administrator can be analysed as follows.

- *Initialization of the remote administrator.* This process uses execution steps similar to those for the remote unit but in this case the unit uses a *cAdmin* class in order to store the local information. This information is used to identify the administrator and cannot be altered.
- *The remote administrator connects to the control centre.* The administrator unit uses its *cCommunicator* to open a communication channel with the control centre. This unit sends its structured data to the control centre for identification reasons. These data messages are used only for an administrator's representation and not for changing any data elements.
- *Requesting data for a remote unit.* The remote administrators use a request message in order to trigger the control centre to transmit a copy of a remote unit's data. This message is simple,

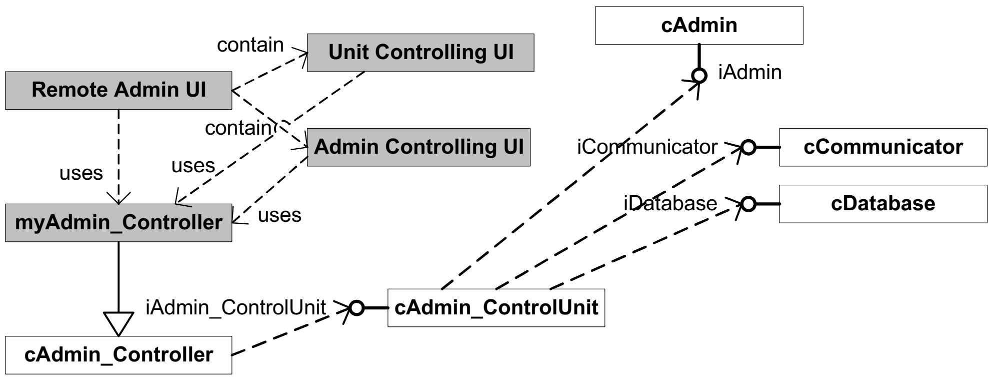


Figure 11. Remote administrator.

using only a data element that uniquely identifies the requested remote unit. In the execution of this process, the remote administrator receives a data and a stylesheet message to construct an identical presentation of the original unit.

- *A remote administrator updates a remote unit.* The remote administrators use a similar controlling form (GUI) with the control centre in order to update a remote unit. The remote unit that is used is the unit that was previously received by the use of the request message. The administrator can update data by using the update messages. These messages initially change the copy of the remote unit that is stored in the control centre and afterwards the original information that is kept in the actual remote unit.
- *Retrieving update messages for a remote unit.* In case of changing data at the remote unit or at the control centre, an update message is sent to the remote administrator to update its local copy.
- *A remote administrator retrieves the list of remote units.* This process is executed every time changes occur in the list of connected remote units in the control centre. When a remote unit is connected to the control centre, list messages are transmitted to all the connected administrators in order to inform them of the new status of the list.
- *A remote administrator closes the connection to the control centre.* This process is used to close the communication with the control centre. In this process the *cAdmin_Controller* sends a disconnect message to the *nAdmin_ControlUnit*, which notifies the *cCommunicator* to close the communication channel.

The *mobile administrators* are mobile devices that can retrieve and update data from a selected remote unit. The data in the mobile devices are dynamically built by requests to the Web server from the control centre device. When dynamic content is requested from the mobile administrator, the Web server initializes a specific component that contains the main core classes of the remote administrator's connectivity. A *cCommunicator* is required to initialize a communication channel to the control centre, a *cMobile_ControlUnit* to parse the messages of the control centre and a *cUnit* to store a copy of a remote unit's data in order to administer the unit in question. In general, this type of Administrator

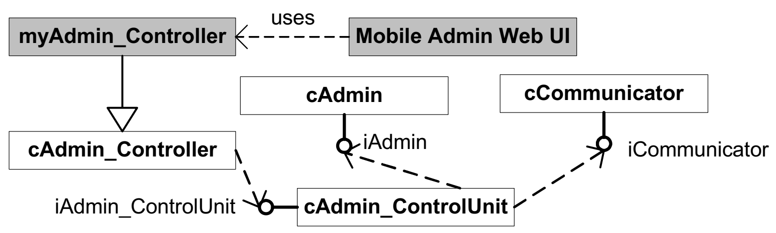


Figure 12. Mobile administrator.

is called a 'virtual' remote administrator that communicates with the control centre by using the Web server as a middleware environment. The major functionality of this telecontrol unit can be analysed as follows (the structure of the mobile administration unit is presented in Figure 12).

- *Initialization of the mobile administrator.* This process is used to initialize the contained classes of the mobile administrator to be used in the Web pages. The most important class is the *cMobile_ControlUnit* class that handles the communication and the messaging procedure.
- *A mobile administrator connects to the control centre.* This process is used in every Web page in order to connect to the control centre. In every request this is the first process that is executed as none of the subsequent processes can be realized without a connection to the control centre.
- *A mobile administrator requests a remote unit.* When remote unit data are requested from a mobile administrator, the control centre is triggered to transmit the structured data of the remote unit. The structured messages are used to construct a local copy of the remote unit, which is stored in a *cUnit* class like that of the administrator units. The wireless device receives the content of the remote unit, which is constructed by parsing the local *cUnit* class to the appropriate Web format.
- *The mobile administrator updates a remote unit.* The mobile administrator submits the changes to the control centre by transmitting an update message.
- *A mobile administrator retrieves the list of remote units.* The list of remote units is received when the mobile administrator connects to the control centre. This process is executed after initialization when a connection link is present. The Web content contains a list of remote unit references as links that can be used for the request procedure.
- *The mobile administrator closes the connection to the control centre.* This process is automatically invoked every time a request, list or an update procedure is executed. The initialized structure of the mobile administrator is unloaded and the *cCommunicator* automatically closes the communication connection. The main functionality is hosted in the control centre, which also unloads the temporary class (*cAdmin*) that was initiated to hold the mobile administrator's information.

The framework presented in this section is available to developers accompanied by the required documentation. This documentation includes the source code, a set of examples (derived for the framework's two 'instantiations' described in the following section) and also a general description of the framework that includes its purpose, how it can be used and how it works (derived from the contents of this section). We highlight here the fact that the description of the framework we are adopting does not deal merely with listing the classes and methods comprising the code but goes further into explaining the inner workings of the framework and how the entities comprising it interact with each other. In that respect, the proposed framework is useful for beginners as well as for experienced programmers.

Our framework is implemented in C#, it contains approximately 30 classes and 5000 lines of code. These classes are reusable, and they are dynamically instantiated within the applications when the developer includes the controlling classes (*cRU_Controller*, *cCentral_Controller*, *cAdmin_Controller*) in their telecontrol applications. The developer should create at least one instantiation of these controlling classes in order to implement the telecontrol units of our framework. The developer should also provide the user interface (UI) that is related only to the application's requirements, rather than to the general functionality of the framework.

5. EXAMPLE APPLICATIONS

As a proof of the concept of how our framework can be used, we apply it to implement two different example applications. On the one hand, these applications are similar because their mission is essentially the same, namely to 'handle remote units' like any other typical telecontrol application. On the other hand, however, they have distinct characteristics and requirements while at the same time acting as an umbrella for a plethora of similar applications. The first example application is a telemedicine system that handles pill dispenser devices and the second is a telesecurity system that monitors lights, doors and alarms in a set of buildings. We illustrate how our framework can be used by presenting a step-by-step construction of the two applications, based on the individual elements of the approach as discussed in the previous section. Before we proceed with the presentation of these example applications, we will analyse their major features by which they are classified as telecontrol systems.

- Both applications are correlated to the generic infrastructure of the proposed telecontrol framework (presented in Section 4). Specifically, they can be separated into the proposed generic roles of units (remote units, control centre and administration units) according to the supported functionality.
- Both can be implemented by the provided functionality that is embedded into our generic framework. They should use and extend the implemented functionality, for updating, synchronization and notification purposes. We have already mentioned that the code of the generic units is compiled and packaged into different libraries that contain the core of the required units. By using these libraries, the example applications illustrate the easy and efficient use of the ready-made coding. If the required additional functionality is rather limited and straightforward to support, this then is a good measure of the framework's reusability.

We begin the analysis of the two applications by identifying the major roles required of the respective tasks; consequently, we define their tasks and functionality. First, considering the telemedicine application:

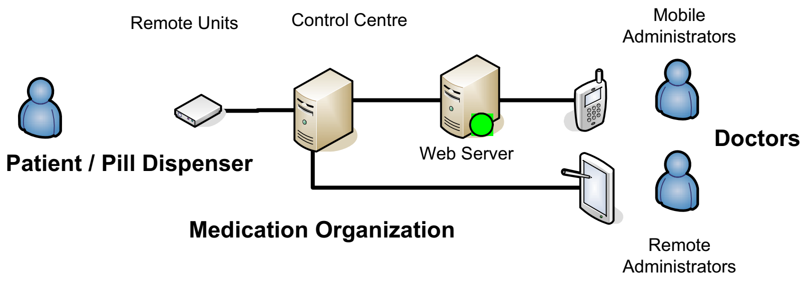


Figure 13. The telemedicine architecture.

The telemedicine system handles pill dispenser devices at remote clients. These clients are located in patients' homes and release pills based on a defined schedule. This schedule may dynamically be altered through instructions provided by doctors who are based at remote geographical locations.

The 'handling of pill dispenser devices' is the equivalent of 'handing remote devices', which is the fundamental task of telecontrol applications. The pill dispensers are remote units that are based in patients' homes and monitored by administrators. In this case, the doctors are the administrators for the pill dispensers and act as administration units. The control centre is required in this case for reliability and security reasons. The control centre can be easily implemented under the responsibility of a medication organization that supplies the required pills. The overall scenario can be represented by the following architecture (Figure 13) that is similar to the generic architecture of Figure 3.

We now correlate the major supporting features of our framework (as described in Section 4) to those required by the telemedicine application. First, we consider the role of the pill dispensers (the remote units):

- handling the electronic devices connected to them and releasing pills on a monitored schedule;
- triggering data changes and message alerts, notifying both the medical organization and the doctors in the event of the dispenser running short (or running out) of a certain pill type;
- receiving update and control messages in order to change their local medication schedule (doctors use the communication infrastructure to influence the behaviour of the pill dispensers in order to provide a suitable medication schedule to the patient).

The medication organization acts as a control centre and has the following responsibilities:

- providing a connection between the pill dispensers and doctors;
- coordinating data and controlling message exchanges between them;
- ensuring the availability, efficiency and consistency of message flows within the communication channels (a reliable exchange of information between the different stakeholders involved in this application is crucial to the health of the patients);
- using the 'grouping' mechanism to gather patients with similar medication needs.

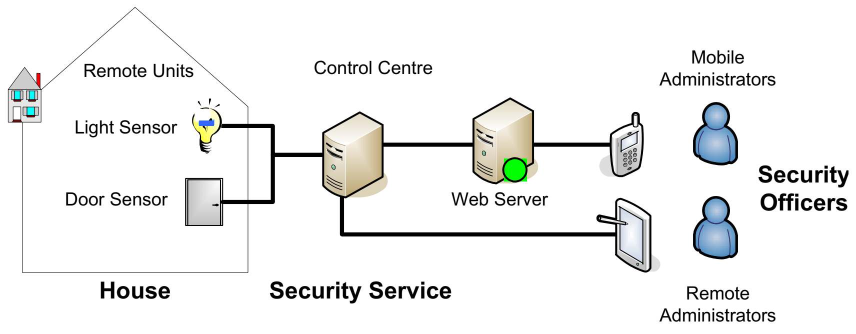


Figure 14. The telesecurity architecture.

Doctors fulfil the role of the administration units and are connected through a reliable communication channel (like the remote administrators) or wirelessly (like the mobile administrators). By using the relevant features of our framework we can state that doctors are responsible for the following tasks:

- doctors should monitor remotely the pill dispenser devices;
- they should also provide directions and commands for changing the medication schedule on selected pill dispenser devices, as required.

The presented outline analysis integrates the telemedicine application with the generic telecontrol framework. The analysis of the second application is similar to that of the first. The telesecurity application can be outlined as follows:

> *The telesecurity system handles a set of sensors and lockers in connected houses. These houses are grouped into defined blocks (neighbourhoods) for better administration. There is a security service responsible for coordinating and assigning houses to specific security officers in different geographical locations.*

The 'handling of sensors and lockers' is again the fundamental task of telecontrol applications. Each house is monitored by a security officer that receives alerts from the security service. The similarity with our proposed generic architecture is obvious. Each house should be a remote unit, the officers are the administration units and the security service acts as the control centre. By using this pairing we constructed the following architecture (Figure 14) to match the generic architecture of Figure 3.

Each house should support mechanisms for the following tasks:

- handling the electronic devices connected to them, for light sensing, door locking and motion detection;
- triggering changes and message alerts, notifying both security service and officers when the light or door status changes and when the motion detectors are alerted.
- receiving update and control messages in order to switch off alarms or lights, close doors, etc.

The security service, acting as a Control Centre, is responsible for the following tasks:

- assigning houses to specific security officers;
- coordinating data and control messages between them;
- ensuring the availability, efficiency and consistency of message flows within the communication channels (the information flowing within the system is crucial for ensuring its security);
- using the 'grouping' mechanism to assemble houses within the same neighbourhood.

The security officers communicate through wired or wireless communication channels to monitor and change the status of the houses:

- officers should monitor remotely the houses status (e.g. lights, doors, motion, etc.);
- they should give instructions and commands for changing the status on selected devices (e.g. closing a specific door).

The similarity between both applications and the proposed telecontrol framework is rather obvious. The major features are supported but in order to illustrate the framework's reusability we should show how the generic libraries are used during the implementation phase. In the following sections we present each instantiation and provide detailed information on how the generic core is customized according to the application needs. First, however, we give some general remarks as to how the framework is used. In Figure 5 we show the major packages that contain the coding that implements the telecontrol framework. In Figure 9 we show the contained components of the remote units. In the same figure, the grey coloured components are the instantiated components that link each example application to the provided functionality. Similarly, in Figure 10 we separate the components of the control centre into the reusable and extended ones. Figures 11 and 12 deal with the internal structure of the administration units, providing a similar separation. In the implementation phase, each application should provide the required extended components in order to take advantage of the underlying coding that supports the distributed communication and coordination in the telecontrol framework. The developer in either example application should provide their own UI for the telecontrol units. All the scenarios that were mentioned in the previous sections are encapsulated into the core of the framework, ready to be invoked in order to instantiate connections, exchange structured messages, etc.

In the following sections we describe in detail how these building blocks are linked together to implement telecontrol systems. Both implementations are based on the same development steps that can be outlined as follows.

- All the telecontrol units (remote units, control centre and administration units) will use the related library (package) as presented in Figure 5.
- The white coloured components in the architecture figures (see Figures 9–12) remain unaltered in each implementation. They provide the basic infrastructure that is fully reusable in each instantiation of the framework and implement, among other things, the communication aspects, signalling mechanism, synchronization scenarios, alerting and coordination, etc.
- The grey coloured components are those which need to be extended or implemented in each application in order to provide the customizable features of the particular instantiation. By analysing each telecontrol unit we can make the following observations.

- ◦ In the remote units the customizable components deal with device interaction (like *my_cDeviceA*), a main controlling component (*my_cRU_Controller*) and a UI component (*Remote Unit UI*) (see Figure 9). The *my_cDevice* components use the API of the electronic device and by using a binding with the customizable controlling component they can be monitored and managed. The UI of the whole instance is contained in a customizable UI.
- ◦ The control centre uses an extended controlling component (*my_cCentral_Controller*) and three types of UI for each role. These are the *control centre UI*, the *unit controlling UI* and the *admin controlling UI*. All these components are bound to the controlling component which provides the required information.
- ◦ The remote administrators have their own extended controlling unit (*cAdmin_Controller*), but they require a *unit controlling UI* and an *admin controlling UI*. The first is needed to present information from the controllable remote unit and the second to present its local information. The above two UIs are bound to a general UI (*remote admin UI*).
- ◦ The mobile administrators use a Web interface, and thus they require a controller (*myAdmin_Controller*) and a Web UI generator (*mobile administrator Web UI*).

Both the telemedicine and the telesecurity applications follow the above guidelines during their implementation. The grey coloured components in the class diagrams are required to be implemented by the developer. The core components which handle messaging, communication, coordination, etc. are encapsulated into the provided set of the framework's component packages.

5.1. The telemedicine application

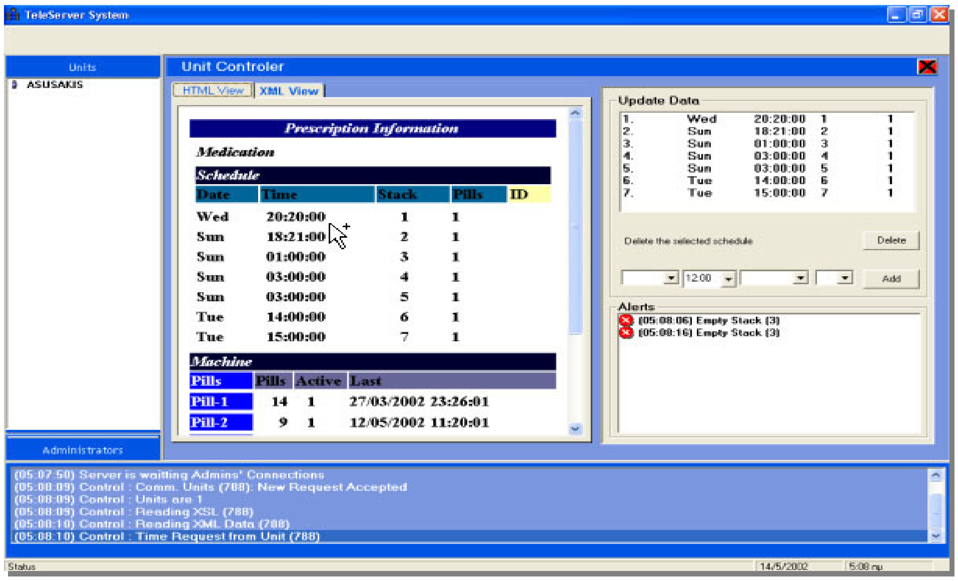
As mentioned before, pill dispensers are analogous to the remote units of the generic architecture. They are used for handling stacks of pills and a certain medication schedule which is stored on the remote devices. This schedule can be changed when new instructions are sent from remotely located doctors through the telecontrol communication infrastructure.

At this point we describe in detail the steps the programmer needs to take to implement the telemedicine system by using our generic framework.

- *Step 1: Include our telecontrol core classes in the new application.* All the telecontrol units (remote units, control centre and administration units) will use the related library (package) as presented in Figure 5. As mentioned in the previous section, all the white coloured components in the architecture figures (see Figures 9–12) remain unaltered in each implementation and they are the core components supporting our proposed telecontrol architecture.
- *Step 2: Design a data model to store required information.* In a telemedicine application we need to store information of the pill dispensers (the number and type of pills in stacks) and medication schedule. Our internal structure is based on an XML schema. The structured data that is used in the telemedicine application is shown below. We use `MedSchedule` elements to store information of a certain medication that is linked to a specific stack of pills. This element uses date and time tags to determine the medication date, and tags to determine which stack of pills will be used and how many pills will be taken. In this data structure, tags are used to determine the state of the stack, the number of pills that are currently stored in the stack, the colour that is used in the UI to determine the specific pill type and, finally, the date that the stack was used to retrieve medication.

```
<TTP>
   <PrescriptionID>
     <MedicationID>
       <MedSchedules>
         <MedSchedule ID="1">
           <MedDay>Mon</MedDay>
           <MedTime>01:00:00</MedTime>
           <MedPill>1</MedPill>
           <MedNoPills>1</MedNoPills>
         </MedSchedule>
         <MedSchedule ID="2">
           . . .
         </MedSchedule>
         . . .
       </MedSchedules>
       <MedExpiray>10</MedExpiray>
     </MedicationID>
   </PrescriptionID>
   <Device>
     <Pill1>
       <Active>1</Active>
       <Colour>16711680</Colour>
       <Count>13</Count>
       <Last>09/11/2002 01:10:00</Last>
     </Pill1>
     <Pill2>
       . . .
     </Pill2>
     . . .
   </Device>
</TTP>
```

- *Step 3: Implement the requited classes/components that should be bounded on the controlling classes of the framework.* Each telemedicine unit should be designed using the provided components and a set of components that will use the provided interfaces as shown in the individual class diagrams (see Figure 9). The 'device' is initialized and during the initialization process the above XML data is built. This functionality, as mentioned previously, is supported by the developer. In the functionality of the pill dispenser it is required to implement events that should be triggered when the patient receives a pill, a pill stack is empty or the patient fails to take their medication after being notified, etc. In order for doctors to be informed of any changes, an *Updated* method must be implemented by the developer and called.
  The *Updated* method is used to update a piece of defined structured data of the remote unit. The local copy changes and the update messages are forwarded through the core components (*cCommunicator*) to the control centre. At this point we have to define what changes must be implemented in the control centre. The extended control centre implements the required UI for



Prescription Information

Medication

Schedule

Date	Time	Stack	Pills	ID
Wed	20:20:00	1	1	
Sun	18:21:00	2	1	
Sun	01:00:00	3	1	
Sun	03:00:00	4	1	
Sun	03:00:00	5	1	
Tue	14:00:00	6	1	
Tue	15:00:00	7	1	

Machine

Pills	Pills	Active	Last
Pill-1	14	1	27/03/2002 23:26:01
Pill-2	9	1	12/05/2002 11:20:01

Update Data

```
1.   Wed   20:20:00   1   1
2.   Sun   18:21:00   2   1
3.   Sun   01:00:00   3   1
4.   Sun   03:00:00   4   1
5.   Sun   03:00:00   5   1
6.   Tue   14:00:00   6   1
7.   Tue   15:00:00   7   1
```

Delete the selected schedule

12:00

Alerts

```
[05:08:06] Empty Stack (3)
[05:08:16] Empty Stack (3)
```

```
(05:07:50) Server is waitting Admins' Connections
(05:08:09) Control : Comm. Units (788): New Request Accepted
(05:08:09) Control : Units are 1
(05:08:09) Control : Reading XSL (788)
(05:08:10) Control : Reading XML Data (788)
(05:08:10) Control : Time Request from Unit (788)
```

14/5/2002 5:08 πμ

Figure 15. The telemedicine control centre.

presenting information. Each UI should be built on top of the structured information stored in the extended controller and the associated XSL stylesheet that is provided from both the remote and administration units. In some cases (as in the telemedicine application) 'grouping' instructions should be followed to present the patients' list (see Figures 10–12).

- The patients of the telemedicine applications are handled as remote units. A GUI should be provided for presenting and handling the pill dispensers. Each pill dispenser will implement the *cDevice* class in order to be monitored by the *cRU_Controller*. A customizable controller that extends the *cRU_Controller* will be designed to enhance the provided functionality and implement the controlling coding on the pill dispensers. In this way, the monitoring binding will be provided by the *cDevice* classes and the extended *cRU_Controller* will handle the arrived control signals by effecting the referred pill dispensers. Note that the medication schedule is handled by the extended *cRU_Controller* and it is updated when the controller receives *Update_Messages* that deal with the medication.
- The control centre of the telemedicine application should be implemented by designing an extended instance of the *cCentral_Controller*. This controller does not handle the low-level communication (e.g. updating, synchronizing, etc.) but it deals with the GUI. The required UI are as follows: *control centre UI* (to present the lists of patient's and doctors), the *unit controlling UI* (to show patients information—pill stacks and medication schedule) and the *admin controlling UI* (to show information about the connected doctors). In Figure 15 we present the structure of the control centre UI. All information is retrieved through the

interface of the extended controller (*cCentral_Controller*) and in this UI we can observe the following features.

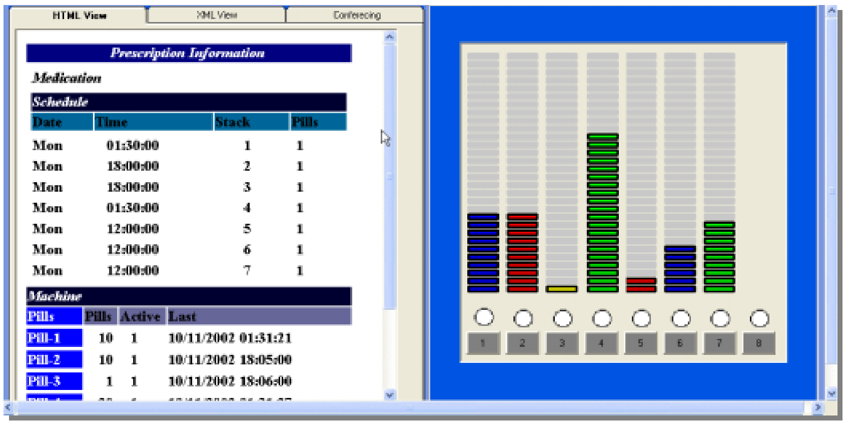
- On the left-hand side of the figure is a list of the patients and doctors along with the ‘grouping’ directions of patients.
- The messages that arrive from the connected units (both patients and doctors) are displayed.
- The required presentation (use of stylesheet) for the selected patient or doctor is also shown (in this figure a remote unit UI (patient) is used). A set of controls are automatically generated on the UI in order to provide the mechanisms of notification of update messages from the control centre.

◦ Doctors are implemented by the remote administrator units that are supported by an extended administration unit (*cAdmin_Controller*). This controller handles a *unit controlling UI* and an *admin controlling UI* to show selected patients’ information or local data.

- *Step 4: Provide the required ‘binding’ coding to support the communication scenarios of the telemedicine application.*

  ◦ In patients’ units (the remote units of the telemedicine application) the following activities take place during the communication scenarios.

    - *Application’s initialization*. In this phase the application starts by binding the UI of the pill dispenser component to the remote unit UI. The serialization mechanism of the ‘device’ is extended from the initial UI, and the information is passed on to the controller through its interface. The customizable controller retrieves local information that is packed into a *Data_Message* and a *Presentation_Style_Message* and passed to the control centre; it also retrieves the ‘grouping’ information in order to relate these data with others. In the ‘grouping’ instructions, patients with the same medication are grouped together. The communication information should be applied to the patient’s controller in order to instantiate the communication channel based on the developer’s preferences. Also the doctors should inform the control centre of their initialization in order to be able to receive automatically the list of patients in their local repository.
    - *Local information changes*. If the local information stored in pill dispensers is altered, a function is called that triggers a ‘comparing’ method to identify the changes. The developer does not need to implement any synchronization or send method. The underlying core will automatically detect and direct the identified changes to the connected control centre. The alerts are implemented in a similar fashion, but they are associated with provided methods in the extended controller. In the doctor’s units the developer should provide the coding that will trigger the *Update_Messages* as soon as the doctor interacts with the local *remote unit UI* controller.
    - *Update directions*. The controlling component of the patients, upon retrieving the control signals, triggers predefined methods on the pill dispensers.



Prescription Information

Medication

Schedule

Date	Time	Stack	Pills
Mon	01:30:00	1	1
Mon	18:00:00	2	1
Mon	18:00:00	3	1
Mon	01:30:00	4	1
Mon	12:00:00	5	1
Mon	12:00:00	6	1
Mon	12:00:00	7	1

Machine

Pills	Pills	Active	Last
Pill-1	10	1	10/11/2002 01:31:21
Pill-2	10	1	10/11/2002 18:05:00
Pill-3	1	1	10/11/2002 18:06:00

Figure 16. The telemedicine ‘device’.

- The extended control centre implements the required UI for presenting information. Each UI should be built on top of the structured information stored in the extended controller and the associated stylesheet that is provided from both the remote and administration units. In some cases (as in the telemedicine application) ‘grouping’ instructions should be followed to present the patient’s list. These UI enable a set of controls that should be used in order to pass the appropriate control message on the selected telemedicine unit (patients or doctors). An update of the medication schedule will automatically generate an *Update_Message* that is transmitted to the referred patient.
- The same remote unit UI is used by the administration units, specifically by the remote administrator. In this case, in the remote admin UI the list of available patients is presented. In the remote administrator the required information is organized and handled by the extended controller (*myAdmin_Controller*). During the selection of a specific remote unit from the list, the developer should bind the selection with a *Request* method provided in our framework. This method will pass the appropriate message to the control centre and the requested information will arrive immediately.

Figure 16 presents the UI of the telemedicine pill dispenser. We can see the ‘device’ with the pill stacks and the presentation of the structured data based on the provided stylesheet. In order to provide some more detailed information we provide the following example to demonstrate the reusability of our framework. In the functionality of the pill dispenser, it is required to implement events that should be triggered when the patient gets a pill, a pill stack is empty or the patient fails to take their medication after being notified, etc. In order for doctors to be informed of any changes, an *Updated* method must be implemented by the developer and called. Special attention is needed only during the alerts, as the developer should bind each special scenario with the alerting method of the initialized controller. During changes in any data, the following flow (in pseudo code) is used.

```
Local Updating ...
If Alert_Scenario_1 then Controller.Alert(....)
If Alert_Scenario_2 then Controller.Alert(....)
...
Controller.Update
```

For example, when a patient gets a pill from a stack and the pill stack is reduced, there are some checks that generate alerts, and finally an *Update* method is triggered. This method will automatically generate an appropriate *Update_Message* to synchronize the information stored in the control centre. This update message that will be transmitted for this change could, like the following example, be based on the internal XML schema.

```
<TTP>
  <TTP_Header>
    <MsgType>UPDATE_MSG</MsgType>
    <Host>
        <Name>Pill Dispenser</Name>
        <IP>192.168.2.8</IP>
        <Description>Personal Medical Device</Description>
    </Host>
    <CreatedDate>10/05/2005 13:31</CreatedDate>
  <TTP_Header>
  <TTP_Body>
    <XML_UPDATE>
        <XML_PATH>Device/Pill2/Count</XML_PATH>
        <XML_DATA>12</XML_DATA>
    </XML_UPDATE>
    <XML_UPDATE>
        <XML_PATH>Device/Pill3/Count</XML_PATH>
        <XML_DATA>7</XML_DATA>
    </XML_UPDATE>
  </TTP_Body>
</TTP>
```

The above message notifies the control centre of the pill reduction in stacks 2 and 3. If there are no pills in a specific pill stack, an *Alert* message is constructed and forwarded to the control centre. The developer is not needed to provide the data for the alert. By using a provided method (*SendAlert*) in the interface of the controller the following alert message will be transmitted to the control centre and to the supervising remote administrator.

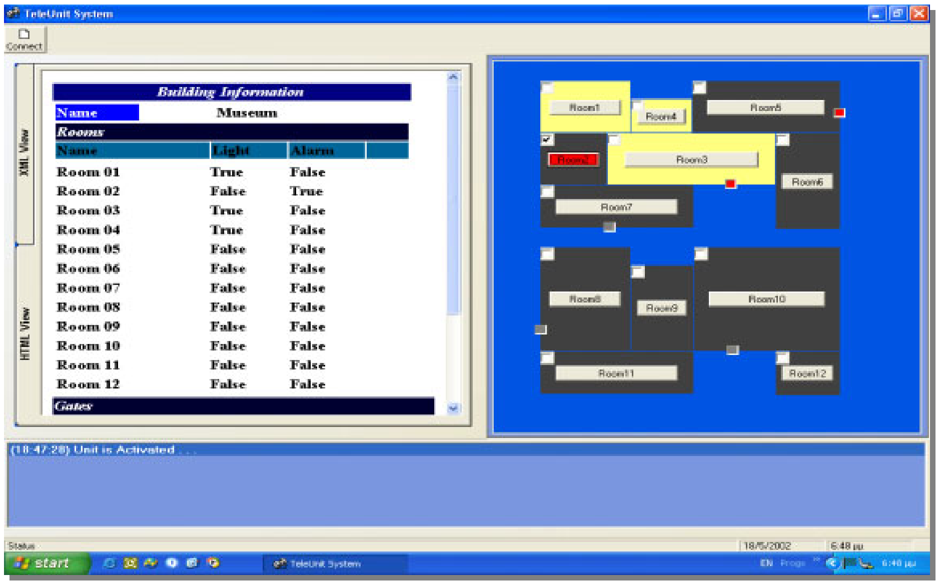
```
<TTP>
  <TTP_Header>
     <MsgType>ALERT</MsgType>
     <Host>
         <Name>Pill Dispenser</Name>
         <IP>192.168.2.8</IP>
```

```
        <Description>Personal Medical Device</Description>
    </Host>
    <CreatedDate>10/05/2005 13:31</CreatedDate>
<TTP_Header>
<TTP_Body>
    <Alert>
        <Name>NO_PILLS</Name>
        <Priority>5</Priority >
        <Description>No pills in STACK 2</Description >
        <Time>10/05/2005 13:00</Time >
    </Alert>
</TTP_Body>
</TTP>
```

In conclusion, each telecontrol unit uses an extendable controller that organizes all the required information into present data, messages and alerts. During implementation, these customizable components are extended in order to bind the 'custom tasks' (like the pill release or the empty stack) with a sequence of 'alert checks' and 'updates'. These scenarios for synchronizing and alerting the distributed telecontrol units are underlying core components that are reusable from the instantiation. The implementation of the telemedicine system proved to be quite simple as a result of using the provided core classes. We did not need to concern ourselves with issues related to communication, synchronization, requests or updates. All this functionality is encapsulated into the provided controllers. Our effort concentrated on the binding of the custom user interfaces (forms, controls, etc.) with the provided framework interfaces. The complexity of this task is closely related to the efficient use of these interfaces. For example, some critical interfaces could be ignored, such as the communication interface which is important to the initialization of the communication channels between the telecontrol units. In order to avoid such scenarios we provided a complete documentation through which the developer will add all the required coding to make their application concrete. In general, the time needed for providing the required coding for the framework core classes is much shorter than the time needed to implement such applications from scratch. By using C# and the framework's provided classes (45 core classes) we implemented this application by adding just 10 additional classes. These classes deal with the required functionality needed for the handling of the specialized pill dispenser device (stacks, medications, etc.). The coding was quite simple because most of the telecontrol coding was already provided by the framework. By instantiating our framework and adding about 1500 lines of coding we managed to implement a functional telemedicine application. This statistical information illustrates the reusability of our framework while the telecontrol functionality is encapsulated in the provided set of core classes.

5.2. The telesecurity application

The second example application is a telesecurity system that handles lights, doors and alarms in a set of buildings. The implementation of this system is quite similar to the previous application. We used a data structure to contain information for a building and a 'device' component that handles the external devices and sensors. The structured data are generated from a function provided by the developer at



Building Information

Name	Museum

Rooms

Name	Light	Alarm
Room 01	True	False
Room 02	False	True
Room 03	True	False
Room 04	True	False
Room 05	False	False
Room 06	False	False
Room 07	False	False
Room 08	False	False
Room 09	False	False
Room 10	False	False
Room 11	False	False
Room 12	False	False

Gates

```
(18:47:28) Unit is Activated . . .
```

18/5/2002 6:48 μμ

Figure 17. Telesecurity remote unit.

the initialization of the application. Figure 17 shows the remote unit's UI. The right-hand side shows rooms and doors (by using the 'device' UI). The structured data are presented on the left-hand side by using the preferred format (stylesheet). In the same way the messages of the telesecurity system are forwarded to the communicating units through the underlying core components that support the telecontrol communication.

The implementation for the telesecurity application was constructed using the same steps as used for the telemedicine example.

- *Step 1: Include our telecontrol core classes in the new application.* These classes have been mentioned several times in our development steps.
- *Step 2: Design a data model to store required information.* The data that holds the information of the building is shown below. *Room* elements are used to store information about the rooms of the building (lights status, alarm status) and *Gates* to support external doors status.

```
<TTP>
    <Building>
        <Name>Museum</Name>
        <Rooms>
            <Room ID="1">
                <Name>Room 01</Name>
                <Light>True</Light>
                <Alarm>False</Alarm>
            </Room>
            <Room ID="2">
                <Name>Room 02</Name>
                <Light>False</Light>
                <Alarm>True</Alarm>
            </Room>
            ...
        </Rooms>
        <Gates>
            <Gate ID="1">
                <Name>Gate 01</Name>
                <Open>True</Open>
            </Gate>
            <Gate ID="2">
                <Name>Gate 02</Name>
                <Open>True</Open>
            </Gate>
            ...
        </Gates>
    </Building>
</TTP>
```

- *Step 3: Implement the required classes/components that should be bounded on the controlling classes of the framework.* In this step, in common with the previous example application, we design the UI and extended controlling classes for supporting the telesecurity application. The building components can be seen in Figure 17. All of these components use the provided API from the electronic devices. Small boxes represent the external doors of the building. The 'device' component is self-contained with all methods and events in order to be handled as a 'virtual' electronic circuit, connected to the building's electronic panel. The UI of the administrating controlling forms is shown in Figure 18. The control signals are passed from the administrators through control interactions (e.g. buttons to turn on/off the light in a selected room of the building, buttons to open/close a selected gate, etc.)
- *Step 4: Provide the required 'binding' coding to support the communication scenarios of the telesecurity application.* In this step the developer should provide all the required coding for the executing, monitoring and controlling scenarios over the distributed set of remote units (building) and administration units (Security officers). So, the developer should be responsible for the following implementations.
    - Implement the coding that will bind the devices (building presentation—lights, gates, alarms) on the controller coding. This coding will be automatically triggered when a change occurs. These changes are followed by a set of *Update_Messages*.
    - Implement the coding for the Administration UI when the previous update messages arrived. The supported controls should be referred to a specific coding that will trigger the *Control_Messages* when the administrator interacts.

In order to provide some more detailed information we use the following example in which the telesecurity 'device' in a remote unit (building) is initialized. The previously described structured data are retrieved from the 'device' and presented through its UI. The following pseudo code contains the initialization process of the telesecurity remote unit. In this coding we retrieve information about the rooms and gates, which is passed to the controlling component.

```
For Each ROOM In BUILDING
   Get Light Status
   Get Door Status
   Get Alarm Status

   Controller.Initialise(ROOM DATA)
Next
```

In the implementation of the telesecurity application we used a different set of scenarios to prove that our framework covers the major needs of such a system. We used the simple scenarios of opening and closing doors, switching lights on/off and also setting the alarm sensors on/off. For extended security reasons we used procedures to immediately close all doors and lights in case of a general alarm. In such cases, the *remote unit UI* of the remote administrator uses a looping signalling procedure to notify all available items (doors, lights). This customization has nothing to do with the core functionality. The UI uses the provided interface of the controlling component that supports the low signalling process of the update message. The following *Update_Message* is automatically transmitted from the administration units when an update method is called. After this message is sent, the remote unit will execute the contained instructions by closing all the doors and switching off all the lights of the building.

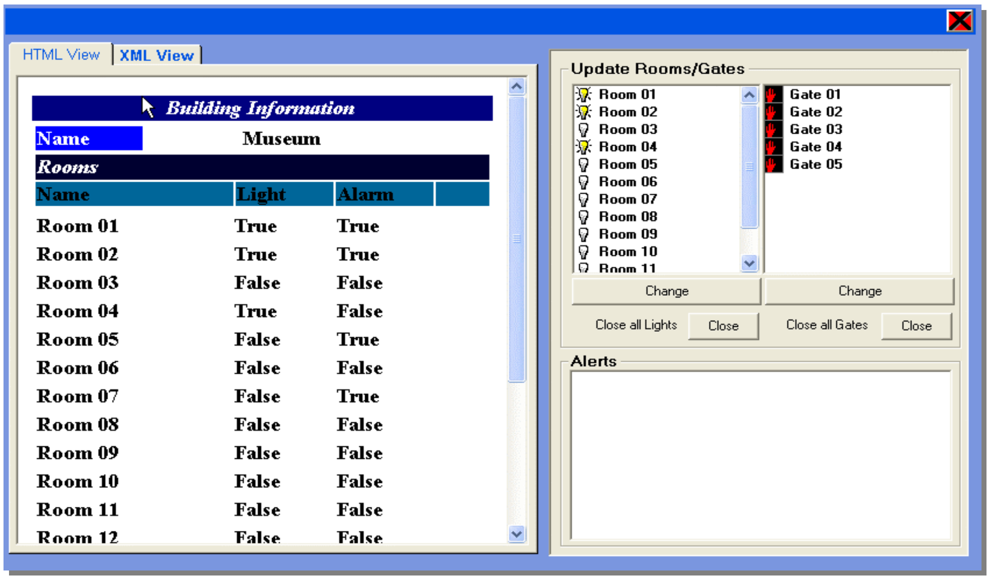
```
<TTP>
  <TTP_Header>
     <MsgType>UPDATE_MSG</MsgType>
     <Host>
        <Name>APPARTMENT 12A</Name>
        <IP>192.168.3.2</IP>
        <Description>"ATHENA" Apartment 12A</Description>
     </Host>
     <CreatedDate>10/05/2005 13:31</CreatedDate>
  <TTP_Header>
  <TTP_Body>
     <XML_UPDATE>
        <XML_PATH>Building/Gates/Gate: 1/Open</XML_PATH>
        <XML_DATA>False</XML_DATA>
     </XML_UPDATE>
     <XML_UPDATE>
        <XML_PATH>Building/Gates/Gate: 2/Open</XML_PATH>
        <XML_DATA>False</XML_DATA>
     </XML_UPDATE>
     . . .
```

```
      <XML_UPDATE>
         <XML_PATH>Building/Rooms/Room: 1/Lights</XML_PATH>
         <XML_DATA>False</XML_DATA>
      </XML_UPDATE>
      <XML_UPDATE>
         <XML_PATH> Building/Rooms/Room: 2/Lights</XML_PATH>
         <XML_DATA>False</XML_DATA>
      </XML_UPDATE>
   </TTP_Body>
</TTP>
```

The alert message that is generated and transmitted by the telesecurity remote unit in case of a robbery or illegal entry is presented below. The controller will be used by the motion detectors ('device') in order to call the alert method (*SendAlert*) to transmit the following message to the administrator units.

```
<TTP>
   <TTP_Header>
      <MsgType>ALERT</MsgType>
      <Host>
         <Name> APPARTMENT 12A </Name>
         <IP>192.168.3.2</IP>
         <Description>'ATHENA' Apartment 12A</Description>
      </Host>
      <CreatedDate>10/05/2005 13:31</CreatedDate>
   <TTP_Header>
   <TTP_Body>
      <Alert>
         <Name>ROOM INTRUSION</Name>
         <Priority>5</Name>
         <Description>Illegal intrusion in the "Kitchen"</Name>
         <Time>10/05/2005 13:00</ Time >
      </Alert>
   </TTP_Body>
</TTP>
```

The controlling component is the only extended element that needs to be developed to support the administrating procedures in the administration units. It is similar to that presented in Figure 18 which shows (a) the selected remote unit's information, (b) the controls that are used for opening/closing doors, switching on/off lights and handling alarms and gates. Consequently, the generic telecontrol framework has been successfully instantiated into a fully functional telesecurity system that monitors buildings. Similarly to the telemedicine application, in the construction of the telesecurity application we did not need to concern ourselves with issues related to communication, synchronization, requesting or updating. All these functionalities are encapsulated into the provided core components and



Building Information

Name	Museum	
Rooms		
Name	Light	Alarm
Room 01	True	True
Room 02	True	True
Room 03	False	False
Room 04	True	False
Room 05	False	True
Room 06	False	False
Room 07	False	True
Room 08	False	False
Room 09	False	False
Room 10	False	False
Room 11	False	False
Room 12	False	False

Room 01
Room 02
Room 03
Room 04
Room 05
Room 06
Room 07
Room 08
Room 09
Room 10
Room 11

Gate 01
Gate 02
Gate 03
Gate 04
Gate 05

Figure 18. The telesecurity control centre.

used in our custom user interfaces (forms, controls) to support the required tasks. By using the provided documentation the development of this application proved to be simple and less time consuming. The provided framework was extended by 12 more classes which deal with security and graphical presentation. *Rooms*, *Buildings*, *Sensors* are some of these classes that are implemented and adjusted within the provided framework. An additional 2000 lines of code were enough to make the implementation of this application feasible. As in the case of the telemedicine application, the telesecurity application is implemented by extending the core classes of the generic framework.

6. RELATED WORK AND DISCUSSION

The area of DCSs has received considerable attention over the last few years. Most of such systems use dynamic components, connectors and well-defined interfaces to set up their ensemble of components [22]. Particular attention has been paid to the role that contemporary software engineering techniques can play in the development of such systems. In [2] the authors cite figures which show that 60–80% of development time for a control system is spent on software development. They then go on to present a number of software technologies that can be used in the development of reusable DCSs, such as object technology and component-based software development, middleware platforms and real-time issues. Similar surveys are reported in [15] and [23]. Despite the important role that software technology is playing in the development of (distributed) control systems, it has been reported that the former is often not sufficiently taken into account by the designers of the latter. Heck *et al.* [2] states that historically there has been a separation between the control engineers who design controls and the software engineers who implement them. Similarly, Sanz [1] expresses surprise as to the rather limited role that software technology is playing in control engineering journals and symposia. Consequently, there appears to be a need to further promote the use of software technologies in (distributed) control applications, and in this paper the authors have attempted to promote this concept.

However, there are a number of proposals that promote the use of middleware platforms (notably CORBA) in the development of distributed control applications. Holloway [24] presents a general analysis on the use of CORBA for developing distributed control applications. In [1] Sanz proposes the use of CORBA for applications related to robot teleoperation and remote operation of hydraulic power plants. Along similar lines, Kuo and MacDonald [12] presents a distributed robotics framework based on real-time CORBA, and in [25] Canella and Bassato combines CORBA and Java in the development of a DCS for a nuclear establishment. In contrast, [17] uses the COM/DCOM technology for developing an environment for open control systems. Also, Wills *et al.* [26] develop their own middleware environment (Open Control Platform) for supporting distributed and reconfigurable control systems. As mentioned previously, the use of middleware platforms is not without problems, most of them stemming from reasons of efficiency (for the case of CORBA) or the use of proprietary technology (for the case of COM/DCOM or Java/RMI). In this work, we have chosen to adopt a 'lighter' component-based model that does not make use of a middleware environment and, consequently, tends to be more efficient than the aforementioned models.

In Section 2 we mentioned the use of Internet technologies and service-oriented computing in developing distributed control applications. To that end, Dy-Liacco [27] proposes the use of XML and SOAP for real-time control systems, Lee *et al.* [28] present a SOAP-based framework for DCSs and in [29] Kapsalis *et al.* develop an architecture for industrial automation systems where the constituent components are viewed as Web services. However, using SOAP requires defining basic control services [28]; also, SOAP has been criticized for being connection oriented and not suitable for multicasting, which many have negative effects on bandwidth time [30]. Furthermore, recent studies [31] suggest that it often suffers from performance degradation compared with the use of remote-procedure-call-based protocols, such as DCOM. Initially we developed our own protocol, using XML-based messages delivered over TCP/IP, as has been done by other researchers [17]. In retrospect, however, we adopted a more flexible approach, as presented in Section 4.1, which is independent of any particular protocol; indeed, SOAP can be used to realize the communication scenarios described in Section 4.1.1.

Our work is a specific framework that uses the above range of techniques to specify a well-defined pattern of telecontrol systems. Connections, XML messages, synchronization, object serialization, coordination and design patterns are just some of the technologies we use to derive a generic telecontrol framework. It is interesting to note that the framework presented in this paper makes use of a number of design patterns that have been proposed in the relevant literature. In addition to the observer and mediator patterns [18], our framework makes use of patterns specifically developed for designing and implementing DCSs [32], namely visibility and communication between control modules (control modules request, perform services or signal with regards to state changes), objects and concurrency, client–server-based services, distribution of control modules and remote control. To summarize, our framework enjoys high reusability (most of the core classes are used as they are and a few need to be extended) and requires only limited effort in 'spawning example application' out of it.

In contrast, [33] reports on a framework targeting similar applications as our own does but based on CORBA. Harinath *et al.* [34] describes an object-oriented framework, also for control applications, based on the use of COTS components. In a more general setting, [35] describes an open framework for multimedia services where they use the notion of coordination in terms of connectors which encapsulate the coordination aspects and components dealing with the computational aspects (in contrast, we also highlight the coordination and computation separation but in terms of a specialized component along with specific sub-components within the major component units of our framework). Also, [22] describes an environment for real-time control systems, where legacy code is integrated with minimal effort. One should also mention here the methodologies and frameworks for developing control systems in general. In particular, we mention the function block concept [36], the Attitude and Orbit Control Systems (AOCS) framework for satellite systems [37], the real-time framework for robotics and automation [13], NEXUS for robotics [38] and OSACA for automation [39]. Other approaches include the work reported in [40,41] on context aware applications, [42,43] on supervision and control, [44] for Web-based environments, [45] for extensible middleware platforms and [46] for real-time data collection and analysis. Our work differs from these previous works in the types of applications that are targeted in each case or in the fact that some of them do not adhere to component-based principles and thus do not offer the reusability found in our framework.

As a final comment, we note that any performance penalties associated with the use of our framework over a direct implementation are expected to limit themselves to the issue of communication between the constituent components, owing to the generic nature of the messages' structure.

7. CONCLUSIONS AND FUTURE WORK

Generic frameworks comprising reusable components are currently becoming more popular because they can assist in the rapid development of large-scale and complicated applications. It is difficult to implement a complex distributed application from scratch, as one needs to know how these units will be connected and coordinated with each other. Even if standard middleware covers the major aspects of distributed applications, it is still a complicated task and advanced programming knowledge is needed.

In this paper we have developed a generic component-based framework for telecontrol applications. Our framework offers a simple way to implement a class of distributed applications using a set of components that support the communicating and messaging aspects. The developer can use this framework without any knowledge of communication channels and messages. We believe that the structured messages that we use cover most major communication scenarios in telecontrol applications. The reusability of the framework has been illustrated with its dual 'instantiation' into two specific but non-trivial application domains: a telemedicine environment and a telesecurity application. The core of our framework uses simple messages and can be extended to support other processes. Structured data transfers and teleconferencing are only two of the processes that can take place in a distributed communication.

Looking back at all the steps comprising this piece of work, we can say that two issues proved to be of special importance. The first issue is the analysis of existing telecontrol systems in order to derive a common denominator of requirements that our framework should support. If such an analysis is not done properly, then the derived framework will unavoidably be incomplete. At the same time, of course, one must first define the kind of applications one will develop a framework for, otherwise one runs the danger of having to mutually satisfy a diverging set of needs and requirements.

The second issue is the clear separation of communication from computational concerns, as is advocated by the notion of coordination [47,48], which leads to enhanced reusability of a framework's components.

As for future work, we aim to study the extension of the core infrastructure to support file transfers, (uploading, downloading) messages of remote execution and enhanced security. Furthermore, we will investigate its potential to be transformed into a fully-fledged Web services Internet application, where the major components will form specialized federations separating component deployment from platform configuration and application composition [49] and where SOAP will be used along with XML for communication [28]. Finally, we will attempt to enrich the documentation dossier for our framework, including some additional case studies illustrating how it can be used.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous referees for a scrupulous review and a plethora of constructive comments that have considerably improved the final version of the paper.

REFERENCES

1. Sanz R, Alonso M. CORBA for control systems. *Annual Reviews in Control* 2001; **25**:169–181.
2. Heck BS, Wills LM, Vachtsevanos GJ. Software technology for implementing reusable, distributed control systems. *IEEE Control Systems Magazine* 2003; **23**(1):21–35.
3. Booch B, Rumbaugh J, Jacobson I. *The Unified Modeling Language User Guide*. Addison-Wesley: Reading, MA, 1999.
4. Szyperski C, Gruntz D, Murer S. *Component Software* (2nd edn). Addison-Wesley: Reading, MA, 2002.
5. Orfali R, Harkey D, Edwards J. *Instant CORBA*. Wiley: New York, 1998.
6. Edwards KW. Core JINI. *The Sun Microsystems Press*. Prentice-Hall: Englewood Cliffs, NJ, 1999.
7. Lowe W, Noga ML. A lightweight XML-based middleware architecture. *Proceedings of Applied Informatics 2002*, Innsbruck, Austria, 18–21 February 2002. ACTA Press: Calgary, AB, 2002; 125–130.
8. Raatikainen K. Middleware for future mobile networks. *Proceedings of the IEEE International Conference on 3G Wireless and Beyond*, San Francisco, CA, 30 May–1 June 2001. IEEE Press: Piscataway, NJ, 2001; 722–727.
9. Szyperski C. *Software, Beyond Object-Oriented Programming*. Addison-Wesley: Reading, MA, 1998.
10. Lee SM, Harrison R, West AA. A component-based distributed control system for assembly automation. *Proceeding of the 2nd IEEE International Conference on Industrial Informatics (INDIN'04)*, Berlin, Germany, 23–26 June 2004. IEEE Press: Piscataway, NJ, 2004; 33–38.
11. Kastner W, Neugschwandtner G, Soucek S, Newman HM. Communication systems for building automation and control. *Proceedings of the IEEE* 2005; **93**(6):1178–1203.
12. Kuo YH, MacDonald BA. Designing a distributed real-time software framework for robotics. *Proceedings of the 2004 Australasian Conference on Robotics and Automation*, Canberra, Australia, 6–8 December 2004. Australian Robotics and Automation Association: Sydney, 2004; 1964–1969.
13. Traub A, Schraft R. An object-oriented real-time framework for distributed control systems. *Proceedings of the IEEE International Conference on Robotics and Automation*, Detroit, MI, 5–15 May 1999, vol. 4. IEEE Press: Piscataway, NJ, 1999; 3115–3121.
14. Speck A. Component-based control system. *Proceedings of the 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2000)*, Edinburgh, Scotland, 3–7 April 2000. IEEE Press: Piscataway, NJ, 2000; 176–184.
15. Tommila T, Ventä O, Koskinen K. Next generation industrial automation—needs and opportunities. *Automation Technology Review* 2001; 34–41.
16. Halsall F. Data communications. *Computer Networks and Open Systems*. Addison-Wesley: Reading, MA, 1996.
17. Lee W, Park J. Middleware architecture for open control systems in the distributed computing environment. *Transactions on Control, Automation and Systems Engineers* 2001; **3**(3):190–195.
18. Gamma E, Helm R, Johnson R, Vlissides J. *Design Patterns: Elements of Re-usable Object-oriented Software*. Addison-Wesley: Reading, MA, 1995.

19. W3C. Extensible Markup Language (XML). http://www.w3.org/TR/2004/REC-xml-20040204 [October 2004].
20. W3C. HTTP—Hypertext Transfer Protocol. http://www.w3.org/Protocols [October 2004].
21. W3C. XSL Transformations (XSLT). http://www.w3.org/TR/xslt [October 2004].
22. Younis MF, Aboutabl M, Kim D. Software environment for integrating critical real-time control systems. *Journal of Systems Architecture* 2004; **50**(11):649–674.
23. de Lucena VF Jr. Flexible Web-based management of components for industrial automation. *PhD Thesis*, University of Stuttgart, 2002.
24. Holloway FW. Evaluation of CORBA for use in distributed control systems. *Internal Report UCRL-ID-133254*, Lawrence Livermore National Laboratory, February 1999.
25. Canella S, Bassato G. Design of a distributed control system based on CORBA and Java for a new rib facility at LNL. *Proceedings of the 8th International Conference on Accelerator and Large Experimental Physics Control Systems (SLAC-R-592, Stanford LAC Electronic Conference Proceedings)*, San Jose, CA, 27–30 November 2001; 493–495.
26. Wills L, Sander S, Kannan S, Kahn A, Prasad JVR, Schrage D. An open control platform for reconfigurable, distributed, hierarchical control system. *Proceedings of the 19th Digital Avionics Systems Conference (DASC-2000)*, Philadelphia, PA, 7–13 October 2000. IEEE Press: Piscataway, NJ, 2000; 4D2/1-8.
27. Dy-Liacco TE. Enabling technologies for operation and real-time control in a liberalized environment. *Proceedings of the EPRI—Second European Conference on EMS*, Budapest, Hungary, 2–4 November 1999; 1–6.
28. Lee C, Park J, Kim Y. A SOAP-based framework for the internetworked distributed control systems. *Proceedings of the 1st International Workshop on Advanced Internet Services and Applications (AISA 2002)*, Seoul, Korea, 1–2 August 2002 (*Lecture Notes in Computer Science*, vol. 2402). Springer: Berlin, 2002; 195–204.
29. Kapsalis V, Charatsis K, Georgoudakis M, Papadopoulos G. Architecture for Web-based services integration. *Proceedings of the 29th Annual Conference of the IEEE Industrial Electronics Society (IECON'03)*, Roanoke, Virginia, 2–6 November 2003, vol. 1. IEEE Press: Piscataway, NJ, 2003; 866–871.
30. Chilingarian S, Eppler W. Universal data exchange protocol based on OPC XML. *Proceeding of the 2nd Workshop on Information Technology and its Disciplines (WITID 2004)*, Kish Island, Iran, 24–26 February 2004. Iran Telecom Research Centre, 2004.
31. Davis A, Zhang D. A comparative study of SOAP and DCOM. *The Journal of Systems and Software* 2005; **76**(2):157–169.
32. Aarsten A, Brugali D, Menga G. Designing concurrent and distributed control systems. *Communications of the ACM* **39**(10):50–58.
33. Sashima A, Kurumatani K. MAEP: A standard communication protocol for artificial economy an instance of X-SS. *Proceedings of the 6th Joint Conference on Information Sciences (JCIS 2002)*, Research Triangle Park, NC, 8–13 March 2002. JCIS/Association for Intelligent Machinery: Durham, NC, 2002; 1085–1088.
34. Paal S, Kammüler R, Freisleben B. Separating the concerns in distributed deployment and dynamic composition in Internet application systems. *OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003*, Catania, Sicily, Italy, 3–7 November 2003 (*Lecture Notes in Computer Science*, vol. 2888). Springer: Berlin, 2003; 1292–1311.
35. Fuentes L, Troya JM. Towards an open multimedia service framework. *ACM Computing Surveys* 2000; **32**(1).
36. Lewis R. *Modelling Control Systems Using IEC 61499*. The Institution of Electrical Engineers: London, 2001.
37. Brown T, Pesetti A, Pree W, Henzinger T, Kirsch C. A reusable and platform-independent framework for distributed control systems. *Proceedings of the IEEE/AIAA Digital Avionics Systems*, Dayton, GL, 14–18 October 2001, vol. 2. IEEE Press: Piscataway, NJ, 2001; 1–11.
38. Fernandez J, Gonzalez J. NEXUS: A flexible, efficient and robust framework for integrating software components of a robotic system. *Proceedings of the IEEE International Conference on Robotics and Automation*, Leuven, Belgium, 16–20 May 1998, vol. 1. IEEE Press: Piscataway, NJ, 1998; 524–529.
39. Lutz P, Sperling W, Fichter D, Mackay R. OSACA—the vendro neutral control architecture. *Proceedings of the European Conference on Integration in Manufacturing (IiM'97)*, Dresden, Germany, 24–26 September 1997. ESPRIT, 1997; 247–256.
40. Power R, O'Sullivan D, Conlan O, Lewis D, Wade V. Utilizing context in adaptive information service for pervasive computing environments. *Proceedings of the Workshop on Applying Adaptive Hypermedia Techniques to Service Oriented Environments: Pervasive Adaptive Web Services and Context Aware Computing*, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 23–26 August 2004, *CS-Report 04-19*; 499–507.
41. Brandley NA, Dunlop MD. Towards a user centric and multidisciplinary framework for designing context-aware applications. *Proceedings of the Workshop on Advanced Context Modelling, Reasoning and Management, 6th International Conference on Ubiquitous Computing (Ubicomp 2004)*, Nottingham, England, 7–10 September 2004.
42. Preuveneers D *et al.* Towards an extensible context ontology for ambient intelligence. *Proceedings of the 1st European Symposium on Ambient Intelligence (EUSAI 2004)*, Eindhoven, The Netherlands, 3–4 November 2004 (*Lecture Notes in Computer Science*, vol. 3295). Springer: Berlin, 2004; 149–159.
43. Capobianchi R, Coen-Porisini A, Mandrioli D, Morzenti A. A framework architecture for supervision and control systems. *ACM Computing Surveys* 2000; **32**(1).

44. Erickson T, Kellogg WA, Laff M, Sussman J, Wolf TV, Halverson CA, Edwards D. A persistent chat space for Work Groups: The design, evaluation and deployment of Loops. *Proceedings of the Symposium on Designing Interactive Systems: Processes, Practices, Methods and Techniques (IDS 2006)*, Penn State, PA, 26–28 June 2006. ACM Press: New York, 2006; 331–340.
45. Bruneton E, Riveill M. An architecture for extensible middleware platforms. *Software: Practice and Experience* 2001; **31**(13):1237–1264.
46. Bauer T, Leake DB. A research agent architecture for real time data collection and analysis. *Working Notes of the 2nd International Workshop on Infrastructure for Agents, Multi-Agent Systems and Scalable Multi-Agent Systems, 5th International Conference on Intelligent Agents*, Montreal, Canada, 29 May 2001; 61–66.
47. Ciancarini P, Tolksdorf R. Coordination middleware for XML-centric applications. *Proceedings of the ACM Symposium on Applied Computing (SAC 2002)*, Madrid, Spain, 10–14 March 2002. ACM Press: New York, 2002; 336–343.
48. Cabri G, Leonardi L, Zambonelli F. XML dataspaces for mobile agent coordination. *ACM Symposium on Applied computing (SAC 2000)*, Como, Italy, 19–21 March 2000, vol. 1. ACM Press: New York, 2000; 181–188.
49. Harinath R, Srivastava J, Richardson J, Foresti M. Experiences with an object-oriented framework for distributed control applications. *ACM Computing Surveys* 2000; **32**(1).