Using Design Patterns in a HSDPA System Simulator

Gaspar Pedreño, Juan J. Alcaraz, Fernando Cerdán
Department of Information Technologies and Communications
Polytechnic University of Cartagena, Plaza del Hospital, 1, 30202
Cartagena, Spain
{gaspar.pedreno, juan.alcaraz, fernando.cerdan}@upct.es

***Abstract*— Wireless network simulators have become a fundamental tool to study, evaluate and improve wireless networks. The quick evolution of wireless technology involves the necessity of continuous changes and updates in these simulators which makes them more complicated and time demanding. Regarding software design, wireless network simulators are generally based on the object-oriented paradigm. However, the use of more advanced programming techniques (such as UML and design patterns) is still an issue for further research. This paper discusses the application of design patterns in a HSDPA system simulator, pointing the suitable pattern and its application place in the wireless system. We focus on the strategy pattern whose implementation is showed in two points of the simulator: the scheduler and the radio channel. Strategy pattern allows us to change the scheduling mechanism or the radio channel even in execution time and simplifies the inclusion of new scheduling schemes or new radio channels.**

Keywords- HSDPA, design patterns,wireless system simulation

I. Introduction

In order to propose, develop and validate protocols and specifications in wireless networks, researchers tend to resort to software simulators instead of hardware solutions. Simulation is particularly advantageous due to its quickness, reliability and, especially, low cost. Simulators are also appropriate to evaluate system performance. Several wireless network simulators and emulators have already been developed [1] in order to evaluate 3G system performance and propose new enhancements. As far as software design is concerned, wireless network simulators are generally based on the object-oriented paradigm. However, the use of more advanced programming techniques (such as UML and design patterns) on wireless network simulators is still an issue for further research. In the field of fixed networks simulators, some works have already dealt with the application of these techniques. In [2], authors make an extensive use of software design techniques, especially design patterns, for the development of a protocol simulation framework applied to ATM networks. Design patterns are standard solutions to common problems in software design [3]. Used in a simulator, design patterns help to make it readable, expandable and reliable since they have been proven effective.

We describe a wireless network simulator that makes use of object-oriented paradigm, UML and design patterns. It was implemented in C++ using OMNET++ libraries [4] and consists of an UMTS system including High Speed Downlink Packet Access (HSDPA) functionalities. It has been already used to evaluate TCP over RLC performance [5] and also to propose the inclusion of a buffer management system in the RLC layer [6].

In this paper, we identify several points of a wireless system simulator where design patterns can be used. We mention these points, the pattern or patterns suitable for them, why the selected patterns are suitable and what advantages they provide. In particular, we show and describe the implementation of the scheduler and radio channel of the HSDPA system based on the strategy pattern. The application of this pattern is appropriate when different variants of an algorithm are needed. It avoids complex, algorithm-specific data structures. UML notation has been used to describe the relationship between objects throughout this article.

This paper is structured as follows. In section 2, design patterns are presented, focusing on those which can be applied in wireless system simulators. Next, we identify where these design patterns are applicable to a wireless system simulator. We show how to implement both a scheduler and a radio channel based on the strategy pattern. Finally, we conclude our paper in section 5.

II. P Design Patterns

Design patterns are effective and reusable solutions to recurring non-trivial problems in software design. They are effective since those problems have already been solved in previous occasions by experienced designers. They are reusable as they can be applied to several design problems in different circumstances. Therefore, design patterns must be customized to solve a general design problem in a particular context. The main purposes of using software design patterns is to support reuse in design and boost confidence in software systems since these solutions have been proven effective.

The pioneering work in software design patterns was done in [3]. It includes 23 of the most commonly used general-purpose design patterns that are application domain independent.

A. Strategy Pattern

The behavioral pattern strategy (also known as Policy) is applied when different behaviors or different variants of an algorithm are needed. With this pattern you obtain a simple code structure where each variant of the algorithm is implemented in its own class. This scheme lets one change the algorithm independently from clients that use it, even in execution time. Such decoupling makes the architecture robust with respect to unforeseen changes. This way, new algorithms can be added by only creating a new class, avoiding modifying previous code. Moreover, encapsulating the behavior in separate classes eliminates conditional statements.

As can be seen in the examples of Figures 2 and 4,in Section IV the strategy pattern consists of three elements:

- *Strategy, it* is an interface common to all supported algorithms.
- *ConcreteStrategy, it* is the algorithm implementation deriving from the *Strategy* interface (subclassing).
- *Context,* it is configured with a *ConcreteStrategy* object. It maintains a reference to a *Strategy* object and may define an interface that lets *Strategy* access its data.

Strategy and *Context* interact to implement the chosen algorithm. A *Context* may pass all data required by the algorithm to the *Strategy* when the algorithm is called. A *Context* forwards requests from its clients to its strategy. Clients usually create and pass a *ConcreteStrategy* object to the *Context*; therefore, clients interact with the *Context* exclusively. There is often a family of *ConcreteStrategy* classes for a client to choose from.

B. *Singleton Pattern*

This pattern ensures that a class only has one instance, and provides a global point of access to it. Some classes, such as a scheduler, require having exactly one instance. This aim can be got by a global variable which makes the object accessible, but does not keep the designer from instantiating multiple objects. However, a better solution is to make the class itself responsible for keeping track of its sole instance. The class ensures that no other instances can be created and it can provide a way to access the instance. This is the singleton pattern.

The Singleton class can be subclassed to refine operations and representation. Then, you can configure the application with an instance of the class you need at run-time. Besides, singleton pattern can be configured to permit a variable number of instances depending on the application requirements.

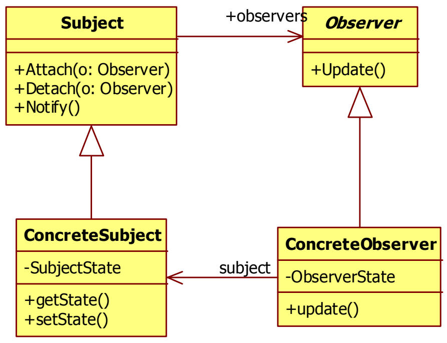

Figure 1. General observer pattern scheme

C. *Builder Pattern*

Builder pattern is employed to encapsulate the construction of a product and allow it to be constructed in steps. It is especially useful when this product has a large number of configurable parameters or characteristics since it allows to encapsulate the way the object is constructed. Builder pattern separates the construction of the object from its representation so that the same construction process can create different representations.

Builder pattern hides the internal representation of the product from the client. Product implementations can be swapped in and out since the client only sees an abstract interface.

D. *Observer Pattern*

The observer pattern (also known as publish/subscribe) defines a one-to-many dependency between objects so that when one object changes its state, all its dependents are notified and updated automatically. As can be seen in Figure 1, the observer pattern consists of four elements:

- *Subject* knows its observers. Any number of *Observer* objects may observe a subject. It provides an interface for attaching and detaching Observer objects.
- *Observer* defines an updating interface for objects that should be notified of changes in a subject.
- *ConcreteSubject* stores state of interest to *ConcreteObserver* objects.
- *ConcreteObserver* maintains a reference to a *ConcreteSubject* object. It stores state that should stay consistent with the subject's. It implements the *Observer* updating interface to keep its state consistent with the subject's.

ConcreteSubject notifies its observers whenever a change occurs that could make its observers' state inconsistent with its own. After being informed of a change in the concrete subject, a *ConcreteObserver* object may query the subject for information.

The coupling between subjects and observers is abstract and minimal. Because *Subject* and *Observer* are not tightly coupled, they can belong to different layers of abstraction in a system. A lower-level subject can communicate and inform a higher-level observer, thereby keeping the system's layering intact

III. SUITABLE POINTS FOR THE APPLICATION OF DESIGN PATTERNS IN AN UMTS-HSDPA SIMULATOR

In this section we describe where and which design patterns can be applied in the development of an UMTS simulator with HSDPA functionalities.

In order to evaluate performance, flexibility and expandability are desirable in several points of the system. These points are characterized by having different possible implementations (algorithms or models) which can be interchangeable. This characteristic is found in the HSDPA scheduler, the flow control (between RNC and Node B) and the buffer management in RLC (Radio Link Control). In this sense, it is completely necessary an efficient and effective software design to make algorithms interchangeable (even in execution time), add new ones and modify the existed ones. This solution is provided by the strategy pattern as we have seen in the previous section.

The scheduler implementation is completely described in Section 4. With regard to flow control in HSDPA, it works between the RNC and the Node B in order to maximize the offered physical layer resources, while avoiding MAC-hs buffer overflow. The HS-DSCH flow control mechanism option stated in the 3GPP specifications [7] is the same as that proposed for the DSCH in Release ´99 and is known as a *credit-based* system. However, the flow control implementation is an open issue and optimizations and new schemes can be used [8]. Therefore, strategy pattern could be applied to set a flexible architecture.

It has been reported [5,6] that overbuffering and buffer overflow at the link layer of radio access networks has negative effects on TCP performance. Several works have dealt with this problem, proposing the use of Active Queue Management Techniques (AQM) at these buffers to improve end-to-end performance. Some classical schemes, like Random Early Detection (RED), have been evaluated [5], showing good results. Some additional new algorithms have also been proposed, e.g. in [6], providing even higher improvement. The use of strategy pattern for the implementation of the buffer management in a 3G simulator will ease the evaluation of different AQM proposals.

Besides, there are other two parts of the UMTS system where strategy pattern can be applied. They are not real algorithms or mechanism but they can be seen like this in a simulator: radio channel implementation and RLC operation modes. Basically, the implementation of the radio channel can be based in an overall packet loss probabilities model or a detailed radio propagation one. This will be discussed in section 5. The Radio Link Control (RLC) protocol provides segmentation and retransmission services for both user and control data. Each RLC instance is configured by the RRC (Radio Resource Control) to operate in one of three modes: transparent mode (TM), unacknowledged mode (UM) or acknowledged mode (AM). Each one of these modes can be considered as a different algorithm and implemented as an object (*ConcreteStrategy*) subclassed from a common interface (*Strategy*). In this way, the RLC entity (*Context class*) could change its operation mode selecting the appropriate "strategy" in anytime.

Some objects should only have one instance as for example the scheduler object. In order to make sure a class has only one instance, the singleton pattern is used.

In a wireless simulator, it must be possible to create different types of users according to their category, radio channel type, timers' duration, window sizes, etc. Therefore, there are plenty of parameters to configure an object which always behave in the same way. This is exactly the job of the builder pattern. It would allow to encapsulate the construction of the users and created them in steps, hiding the construction process to the user itself.

In UMTS, the RRC protocol is in charge of the signalling (mobility management, call control, session management, and so on) between the mobile users (UE) and the UTRAN. It is a clear example of a cross-layer system. The control interfaces between the RRC and all the lower layer protocols are used by the RRC layer to configure characteristics of the lower layer protocol entities, including parameters for the physical, transport and logical channels. The same control interfaces are used by the RRC layer, for example to command the lower layers to perform certain types of measurement and by the lower layers to report measurement results and errors to the RRC. According to these measures, RRC layer must take decisions and send messages to other layer if it is necessary. This scheme fits perfectly in the observer pattern. This pattern can get full decoupling of the communicating entities in time space and synchronization.

IV. IMPLEMENTATION EXAMPLES

In this section we show how to apply the strategy pattern to the scheduler and the radio channel of an UMTS simulator with HSDPA functionalities. Such pattern requires implementing an interface but C++ does not have that possibility. Instead of an interface, an abstract class has been used.

A. Scheduler

The HSDPA system of UMTS is based on a shared channel, HS-DSCH (High Speed Downlink Shared Channel), where a packet scheduler decides what users are served in each Transmission Time Interval (TTI). We consider multiple users since code multiplexation can be employed in the HS-DSCH. This shared channel makes use of a fast link adaptation system to compensate the variations in the downlink radio-channel conditions. Therefore, base stations (Nodes B) know the radio-channel characteristics immediately and this information can be used in the scheduling mechanism. Although no particular scheduling algorithm is included in the 3GPP specifications, there are several well-known algorithms and others recently developed by several research groups [9].

Our simulator implements three types of scheduling algorithms:

- Round Robin. Resources are allocated on a sequential way and consequently it does not utilize any knowledge of the radio channel's condition.
- Maximum C/I. Each TTI, users with the best instantaneous channel characteristics are served. With this scheme, the cell throughput is maximized but it results unfair with users located on a cell border.
- Fair Scheduling. It is an intermediate option between both above. It takes into account variations in the radio channel conditions while applying a certain degree of fairness.

We have created a simple architecture using the strategy pattern that enables the interchange of the scheduling mechanism even in execution time. Figure 2 shows an UML diagram of the implemented structure. The *Mac_HS class* corresponds to the *Context class* seen above and performs

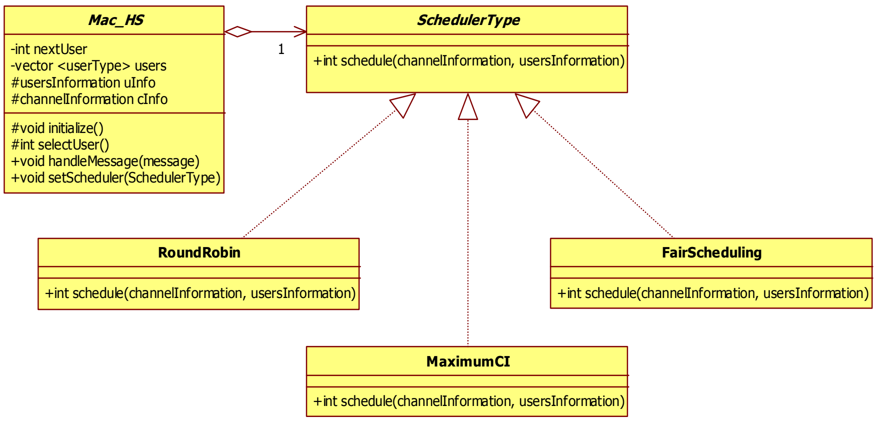
Mac_HS
-int nextUser
-vector <userType> users
#usersInformation uInfo
#channelInformation cInfo
#void initialize()
#int selectUser()
+void handleMessage(message)
+void setScheduler(SchedulerType)
1
SchedulerType
+int schedule(channelInformation, usersInformation)
RoundRobin
+int schedule(channelInformation, usersInformation)
FairScheduling
+int schedule(channelInformation, usersInformation)
MaximumCI
+int schedule(channelInformation, usersInformation)

Figure 2. UML class diagram used in the scheduler implementation

Mac-HS layer functions [10], including scheduling. It would have been possible to define the scheduling mechanism as a method inside this class but it would have supposed to change all the method code every time a new algorithm is implemented. Other possibility would be to use inheritance and define different *Mac_HS* subclasses, each one with its own scheduling algorithm. This alternative compromises reusability as it couples *Mac-HS* implementation with a particular scheduling mechanism. Besides, changing the scheduling algorithm is not allowed in execution time.

The strategy pattern converts each scheduling algorithm into a class. The relation between *Mac_HS class* (the *Context* class) and scheduling method is the *has-a* type (composition). Thus, the *Mac_HS class* has a *SchedulerType* object which is set up with a specific scheduler (*ConcreteStrategy*) in the *initialize* method and can be dynamically modified by the *setScheduler* method. We can see these methods in Figure 3 where we include some code extracted from the Mac_Hs class (*Context*).

In summary, the architecture implemented here allows an easy study of the behavior and efficiency of different scheduling algorithms. Besides, its design keeps the door open for introducing new algorithms. Although we show here an example over HSDPA, this scheme could be extended to any scheduling mechanism.

B. *Radio Channel Implementation*

The radio channel implementation plays an important role in a wireless network simulator. Some models are simply based in overall packet loss probabilities, while others try to characterize a real channel using a detailed radio propagation model. According to this, radio channel implementation could be thought as an algorithm which has different versions. Therefore, strategy pattern can be also applied to model its structure increasing flexibility.

Figure 4 shows the implemented channel structure. We have defined a *Context class*, called *Channel*, which corresponds to the physical layer. It has several objects representing the specific radio channels which are separated from their implementations (strategies). These channels are three [11]:

- High Speed Downlink Shared Channel (HS-DSCH).

```
class SchedulerType;
// Class declaration for Mac_Hs
class Mac_Hs : public cSimpleModule{
    // Method and data members declarations
    private:
      int nextUser;//next user to be served
      vector <userType> users;//connected users
      …
    protected:
      void initialize();//give value to variables
      int selectUser();//choose user to be served
      SchedulerType* scheduler;
      usersInformation* uInfo;//c++ struct
      channelInformation* cInfo; //c++ struct
      …
    public:
       Module_Class_Members(Mac_Hs,
       cSimpleModule,0);
       void handleMessage(cMessage *msg);
       void setScheduler(SchedulerType* sch);
       …
};
// Mac_Hs - Implementation
void Mac_Hs::initialize(){
       scheduler = new RoundRobin();
       …
}
void Mac_Hs::handleMessage(cMessage* msg){
       nextUser = selectUser();
       //with this index the user is choosen in
       //the vector called "users"
       …
}
int Mac_Hs::selectUser(){
      //Some initialization code before
      //forwarding the request to the scheduler
      /Information about channel and user is sent
      int Nuser=scheduler->schedule(uInfo,cInfo);
      …
      return Nuser;
}
void Mac_Hs ::setScheduler(SchedulerType *sch)
{scheduler = sch;}
```

Figure 3. Code extracted from Mac_Hs class.

It is in charge of transporting users' data in downlink direction.

- High Speed Shared Control Channel (HS-SCCH). It deals with downlink signaling information. It allows users to decode data from the HS-DSCH.
- High Speed Dedicated Physical Control Channel (HS-DPCCH). It carries the uplink control information, mainly acknowledgements and channel feedback information.

This architecture, based on strategy pattern, allows us to choose among different channel models for each preceding radio channel. At this moment, we have implemented two types of models:

- Markov Doppler Channel. The wireless channel generates error bursts in a per-frame basis according to a two-state Markov chain as described in [12]. The Doppler frequency, f_d, of the UE determines the average burst length. Lower f_d causes longer bursts of errors.
- Real radio propagation channel model, similar to this one used in [1]. This more realistic scheme is completely necessary for the HS-DSCH in order to model *Fast Link Adaptation* and *Fast Scheduling* [11]. It takes into account typical long- and short-term variations in propagation conditions. Moreover, this scheme allows to choose among several radio environments (indoor office, urban, rural, pedestrian, vehicular, etc) and to configure some parameters (distance, mobile speed, etc).

As a final result, we can evaluate HSDPA performance in different propagation environments with only changing the radio channel object implementation. The implementation of the channels can also be selected dynamically and, consequently, the effect of sharp channel variations (e.g. caused by a user who entries into a building or a tunnel from the street) can be studied. In addition, this structure enables the integration of both new radio channels and new channel implementations in the simulator effortlessly. The code of the channel implementation is not included in this paper due to extension constraints.

V. Conclusion

In this article we have disclosed in which points of an UMTS simulator with HSDPA functionalities design patterns can be applied. Not only that, but also what is the suitable pattern for each point and what advantages it provides. We have pointed out the benefits of using design patterns in simulation. We have used UML diagrams to describe our designs in a rigorous and widely accepted way. In particular, we have shown the application of the strategy pattern to the radio channel and scheduler implementations. The result is a flexible architecture which allows to change radio channel and scheduling configurations even in execution time. In this way, different models of radio channels and schedulers can be tested in order to evaluate performance. Besides, the architecture is expandable, thus, new channel implementations and new scheduling algorithms can be added at any time.

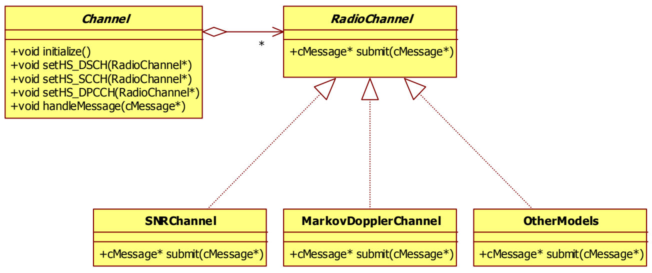
Channel
+void initialize()
+void setHS_DSCH(RadioChannel*)
+void setHS_SCCH(RadioChannel*)
+void setHS_DPCCH(RadioChannel*)
+void handleMessage(cMessage*)
*
RadioChannel
+cMessage* submit(cMessage*)
SNRChannel
+cMessage* submit(cMessage*)
MarkovDopplerChannel
+cMessage* submit(cMessage*)
OtherModels
+cMessage* submit(cMessage*)

Figure 4. UML class diagram used in the radio channel implementation.

Acknowledgment

This work was supported by the Spanish Inter-Ministerial Sciece and Technology Commission under project TEC2005-08068-C04-01/TCM.

References

[1] Neil Whillans (Editor). *End-to-End Network Model for Enhanced UMTS*. IST SEACORN Project Deliverable D3.2v2, October 2003

[2] H. Hüni, R. Johnson, and R.Engel. *A Framework for Network Protocol Software*. Annual ACM Conference on Object-Oriented Programming Systems, 1995.

[3] E. Gamma, R.Helm, R.Johnson, and J.Vlissides. *Design Patterns, Elements of Object-Oriented Software*. Addison-Wesley, 1995.

[4] András Varga, *OMNET++ 3.2 User Manual and API Reference*, www.omnetpp.org

[5] J. J. Alcaraz, F. Cerdán and J. García-Haro, *Optimizing TCP and RLC Interaction in the UMTS Radio Access Network*, IEEE Network, vol 20, no. 2, Mar. 2006 pp. 56 - 64.

[6] J. J. Alcaraz and F. Cerdan, *Using Buffer Management in 3G Radio Bearers to Enhance End-to-End TCP Performance*, in *Proc. IEEE 20th AINA*, vol. 2, Apr. 2006, pp. 437-441.

[7] 3GPP TS25.877, *High Speed Downlink Packet Access: Iub/Iur Protocol Aspects*, version 5.1.0, June 2002

[8] P. J. Legg, *Optimised Iub flow control for UMTS HSDPA*, in Pro. IEEE Vehicular Technology Conference (VTC 2005-Spring), Stockholm,Sweden, June 2005.

[9] Wha Sook Jeon, Dong Geun Jeong, and Bonghoe Kim. *Packet Scheduler for Mobile Internet Services Using High Speed Downlink Packet Access*. IEEE Transactions on Wireless Communications, Vol. 3, No. 5, September 2004.

[10] Harri Holma , Antti Toskala, *WCDMA for UMTS: Radio Access for Third Generation Mobile Communications*, Third Edition, John Wiley & Sons, Inc., New York, NY, 2004

[11] 3GPP TS25.308, "UTRA High Speed Downlink Packet Access (HSDPA); Overall description; Stage 2", version 6.3.0, December 2004

[12] A. Chockalingam and M. Zorzi, 1999. *Wireless TCP Performance with Link Layer FEC/ARQ*, in *Proc. IEEE ICC'99*, pp. 1212-16.