Exploiting an Object-Oriented Simulation Tool to Model a PCS Network

Brian A. Malloy and Bin Chen
malloy, bchen@cs.clemson.edu
Department of Computer Science
Clemson University
Clemson, SC 29634

Abstract

In this paper, we describe the design and implementation of a simulator for a personal communication service network (PCS) using SIMPLE++, a special purpose system for simulation modeling. The SIMPLE++ system includes a special purpose language, similar to C++, that provides facilities for object-oriented programming. By including a programming language as part of the system, the SIMPLE++ user can construct a model with the detail required to capture the semantics of the system under study. SIMPLE++ also includes facilities for model construction using iconic based blocks; this facility provides ease of use and can obviate the need for the end user to be an accomplished programmer. Thus, the SIMPLE++ system incorporates the advantages of using a special purpose language together with the advantages of using an icon based tool.

keywords: Methodology, object technology, object-oriented, object-model, general purpose language, icon, simulation tool

1 Introduction

The recent growth and development of languages and tools to facilitate simulation modeling has had a tremendous impact on the field of simulation. This growth includes the development of general purpose *simulation languages*, such as GPSS/H[16], and MODSIM III[14], together with the development of *simulation tools* such as Extend[5] and ProModel[1]. Simulation languages provide expressivity, permitting the programmer to control the details of the model. The advantage of using a simulation language is the flexibility of design together with the speed of execution of a compiled language. The disadvantage of using a simulation language is the overhead required of the programmer to become proficient in using the language. Simulation tools avoid the overhead of learning a language, permitting the simulationist to construct a model using library-based iconic blocks, where each block describes a step in the simulation model[9]. The advantage of using a simulation tool is the ease of use, providing simulation capabilities for the end user rather than exacting the overhead required to learn a programming language. The disadvantage of using simulation tools is that they are sometimes prohibitively inefficient for large simulation models and they lack expressivity for detailed design.

In this paper, we describe the design and implementation of a simulator for a personal communication service network (PCS) using SIMPLE++[1], a special purpose system for simulation modeling[7]. The SIMPLE++ system includes a special purpose language, similar to C++, that provides facilities for object-oriented programming. By including a programming language as part of the system, the SIMPLE++ user can construct a model with the detail required to capture the semantics of the system under study. SIMPLE++ also includes facilities for model construction using iconic based blocks; this facility provides ease of use and can obviate the need for the end user to be an accomplished programmer. Thus, the SIMPLE++ system incorporates the advantages of using a special purpose language together with the advantages of using an icon based tool.

A PCS network is a wireless communication network that provides service for mobile phone users[4]. The communication area covered by a PCS is partitioned into *cells* where each cell has a broadcast station with a set of assigned radio channels. A mobile phone, or *portable*, resides in the signal range of a particular cell

[1] SIMPLE++ stands for **SIM**ulation in **P**roduction, **L**ogistics, **E**ngineering, and its implementation in C++[8].

for a period of time and then moves to another cell. If a portable is involved in a call when it is moving out of signal range of the current cell, it frees the channel that was allocated to the call and requests a channel from the cell into which it is moving. This action of passing a call-in-progress from one cell to another is called a *call handoff*. If no channel is available in the destination cell then the call is *blocked*. If a channel is available in the destination cell then it is allocated and the call continues with no perceivable interruption.

To provide good service, blocked calls must be minimized. Simulation studies have proven invaluable for evaluating channel assignment schemes to facilitate good service. Three channel assignment schemes that have been used are fixed channel assignment (FCA), dynamic channel assignment (DCA) and a hybrid channel assignment (HCA) [10, 13, 17, 18]. In FCA, a broadcast station is assigned a fixed number of channels; if a call handoff is attempted when all channels are in use, then a blocked call results. In DCA, the station is assigned a fixed number of channels but when the channels are in use, the station may borrow an unused channel from a neighboring station to service a call handoff. HCA is a combination of FCA and DCA.

The cells in a network are typically square shaped, but hexagonal shaped cells better capture the range of a broadcast station. For square shaped cells, a square shaped network is formed but for hexagonal shaped cells the shape of the network is not square since the boundaries of the network are jagged. Two important shapes for networks have been studied: S-mesh and H-mesh[12]. Studies have shown that for square shaped networks more than one thousand cells are required to avoid perturbations to the simulation due to the effect of portables leaving at the boundaries[2, 12]. However, because simulations of large networks are computationally intensive, some PCS network simulations have modeled less than fifty cells[12].

Our SIMPLE++ model for a PCS network can be configured as a square, an H-mesh or an S-mesh. The network contains cells that can be shaped as squares or as hexagonals and the stations in the cells can use either of the three channel assignment schemes: FCA, DCA or HCA. The remarkable feature of our model is that by exploiting inheritance together with the advantages of using a simulation tool, the SIMPLE++ design and implemention of all of these features required *less time* than the C^{++} design and implemention of a PCS network that could only be configured as a square and contained square cells with an FCA scheme[6]. Moreover, we are able to monitor the SIMPLE++ PCS network simulation much more closely than the C^{++}

simulation; for example, we can monitor the blocking factor for calls in the PCS network as the simulation is executing. While monitoring the PCS simulation, we observed that the blocking factor stabilized much sooner than the 250,000 cycles that were executed in previous simulations[2, 6]. Using this observation, we were able to construct a large PCS network containing 1024 cells. This network contained almost two hundred thousand objects during each cycle of the simulation and, during an execution of five thousand cycles, generated almost one and a half million events. However, using the graphic-based tools included in the SIMPLE++ system, we were able to obtain results similar to experiments acquired using a simulation system compiled with C++[2, 6].

The SIMPLE++ system provides control over simulation time so that we can temporarily halt the simulation during execution, print statistics or debugging information, and then continue the simulation. Also, we can easily extract statistics from the SIMPLE++ simulation including automatically generating graphs and charts for statistics gathered during the simulation. In this paper, the graphs and most of the figures that we present were garnered during the execution of our SIMPLE++ simulation.

A validated comparison of SIMPLE++ with other tools is beyond the scope of this paper; however, an overview of simulation tools can be found in reference [15], included in this journal issue. An overview of techniques and systems for simulating communication networks can be found in reference [11].

In the next section we provide background about the SIMPLE++ modeling system and about PCS networks. In Section 3, we describe our SIMPLE++ model and in Section 4 we present some results from our experiments. Finally, in Section 5 we draw conclusions.

2 Background

The focus of this paper is our use of object orientation, together with the simulation tools incorporated into the SIMPLE++ system, to model a PCS network. Thus, we begin this section with an overview of SIMPLE++, followed by background about PCS networks.

2.1 Overview of SIMPLE++

Users of traditional simulation software are typically required to build a complete model to implement a simulation system. This model is usually *inflexible* to extension and modification. This is not the case with

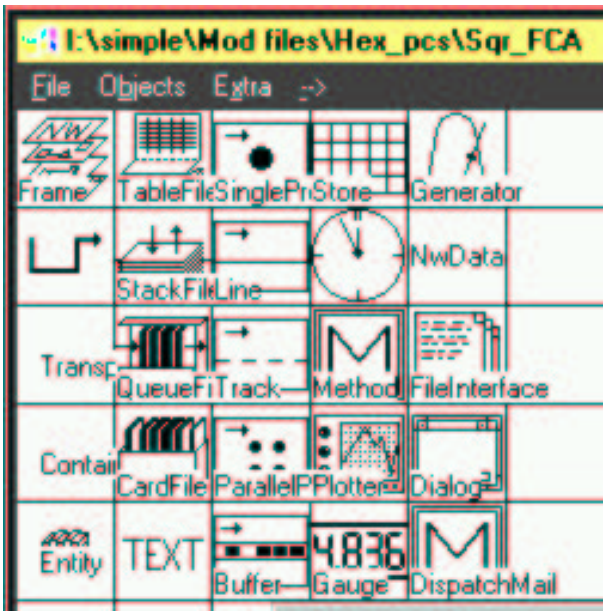


Figure 1: *The default building block library.* This figure illustrates the default building block library in the initial icon that is opened by the user of a SIMPLE++ system. This library contains the objects that will be used to construct the simulation model.

SIMPLE++, which provides an integrated user environment with features that include object technology, graphical and animation capabilities, and capabilities to monitor the simulation dynamically. To give a flavor for SIMPLE++, we now describe the *default building block library.*

Figure 1 illustrates a *default building block library*, which is analogous to a class library in C^{++}. A complete description of the default building block library can be found in reference [7]; to give a flavor of SIMPLE++ usage, we now describe four objects in the default building block library: the *frame*, *TableFile*, *EventController* and *Method.*

The *frame* object is the basis of all user defined models. A simulation model is created by first copying the frame building block. The copy can then be opened to allow objects or models from the building block library to be inserted into the model. The *TableFile* represents an array of records where each record contains fields that can be accessed directly. Data values can be entered into the records in the table and the SIMPLE++ package contains methods to facilitate reading and writing tables. The *EventController* is the SIMPLE++ counterpart to the clock in traditional simulation. The EventController controls simulation time and orders all events in the simulation by time. The *Method* is the counterpart to the C^{++} member function and can contain code written in SimTalk. SimTalk is a language included in the SIMPLE++ package.

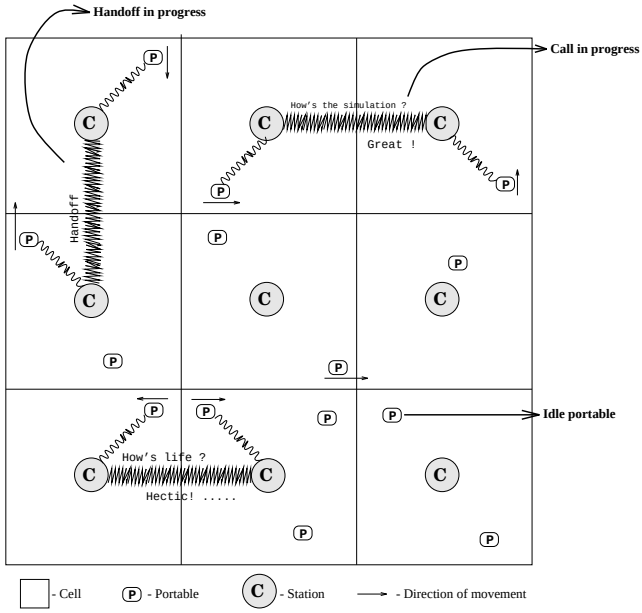


Figure 2: *Sample PCS network.* This figure contains a sample PCS network, configured as a square, containing 9 cells where each cell is also square shaped. Each cell contains a broadcast station and some portables, where some portables have a call in progress and others are simply in the cell. In our simulation, we track all portables through the network.

2.2 Personal Communication Service (PCS)

A personal communication service (PCS) network [4] is a wireless communication network that provides service for mobile phone users or PCS subscribers. The communication area covered by a PCS is partitioned into areas called **cells** with a set of radio channels assigned to each cell. A mobile phone, or **portable**, resides in the signal range of a particular cell for a period of time and then moves to another cell.

When a subscriber makes a phone call, the status of the destination portable is determined. If the portable is currently involved in another call then the line is busy and the call is **dropped**; the cell does not proceed past this point. If the line is not busy then the cell in which the portable resides attempts to allocate a radio channel to connect the call. If the cell is unable to find a free channel to connect the call then the call is **blocked**. The arrival of a new call is not the only way that a channel may be requested. If a portable is involved in a call when it is moving out of signal range of the current cell, it frees the channel that was allocated to the call and requests a channel from the cell into which it is moving. This action of passing a call-in-progress from one cell to another is called a **call handoff**. If no channel is available in the destination cell then a **handoff block** occurs and the call is terminated. If a channel is available in the destination cell

then the channel is allocated and the call continues with no perceivable interruption. Channels are released only when a portable with a call-in-progress moves out of the current cells signal range or the call completes.

An important criteria used to judge the quality of a PCS network is the **blocking probability** or the ratio of the number of blocked calls to the number of attempted calls. Intuitively, the blocking probability is the probability that a call will not be connected due to channel availability. To provide quality service to subscribers, it is important to engineer the PCS network so that the blocking probability is low, typically less than 1 percent [2].

Blocking probability can be controlled in a PCS network simulation by adjusting several of the parameters that define the network. These parameters are **average call length**, **average call interarrival time**, **number of channels per cell**, and **number of portables per cell**. As the ratio of portables per cell to channels per cell decreases, so does the blocking probability. Likewise, as the average call length decreases and the average call inter-arrival time increases, the blocking probability decreases.

3 The Model

In this section, we exploit object orientation to construct a PCS network simulator. We begin our presentation from the user point-of-view, showing how to input parameters to the SIMPLE++ simulator, how to monitor the system dynamically and how to gather statistics. We provide some insight into the construction of the simulator and conclude the section by showing the underlying object model that facilitates extending the simulator.

3.1 Using SIMPLE++ to construct the PCS Model

Figure 3 illustrates a SIMPLE++ view after opening the PCS network application model. The right side of the figure contains the `model icon` that includes the default building block library (top five rows), and our *PCS network model* (bottom five rows; i.e., rows 7 through 12). We described the default building block library in section 2.1; we now present the bottom five rows of the `model icon`; these rows describe our PCS model.

Row seven of the `model icon` contains the icon *Sqr_FCA*, a *cell* object, a *portable* object and an OpenDialogWindow. The *Sqr_FCA* icon represents our main PCS model, the *cell* icon represents a cell in the PCS

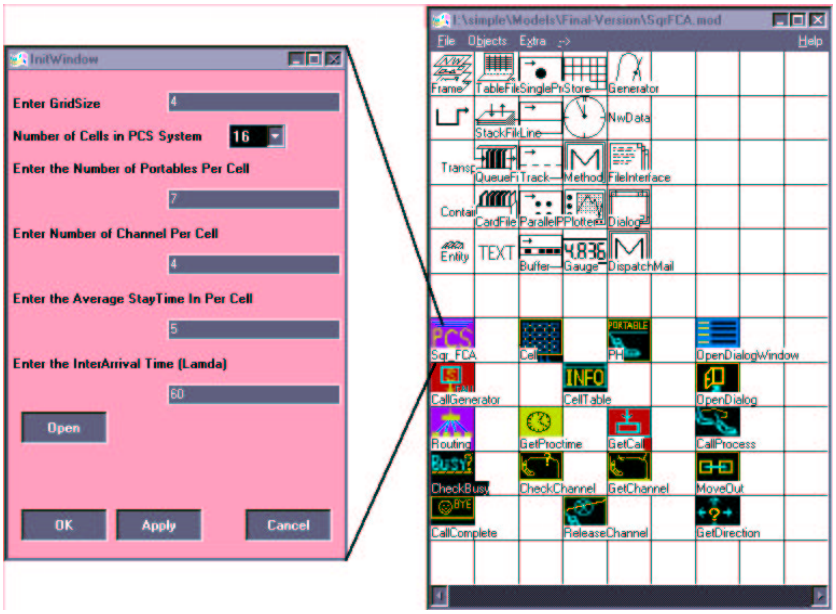


Figure 3: *Initial view.* The right side of this figure contains the default building block library and our PCS network model: the top 5 rows are the library and the bottom 5 rows are our PCS model. The left side of the figure contains an interface to the PCS model that permits the user to enter parameters to describe the simulation. This model uses a fixed channel assignment scheme to assign channels to portables in a cell.

network containing one or more portable objects, the *portable* icon represents a portable residing in a cell, and the OpenDialogWindow is used to abstract details of the model. Each of these objects contain methods to implement actions of the objects in the simulation. For example, the cell object contains a call generator object, *CallGenerator*, that invokes a method to route a call to one of the portables in the cell according to the specified distribution.

By double clicking on the *Sqr_FCA* icon, a window is opened, the *InitWindow*, to permit the user to input parameters to define the PCS network simulation. *InitWindow* is illustrated on the left side of Figure 3, where the *GridSize* is 4, the *Number of Cells in the PCS System* is 16, the *Number of Portables per Cell* is 7, the *Number of Channels Per Cell* is 4, the *Average StayTime in Each Cell* is 5, and the *InterArrival Time* is 60. Parameters such as these are supplied each time the simulation is executed. By clicking on the *Apply* button of *InitWindow*, the parameters are entered into the system. The user must then click on the *Open* button, to open the *Sqr_FCA.EventController* window, shown in Figure 4.

Figure 4 shows the *Sqr_FCA.EventController* window that contains a window with buttons to permit the user to manipulate the simulator: an *init* button to initialize the simulation, a *start* button to start the

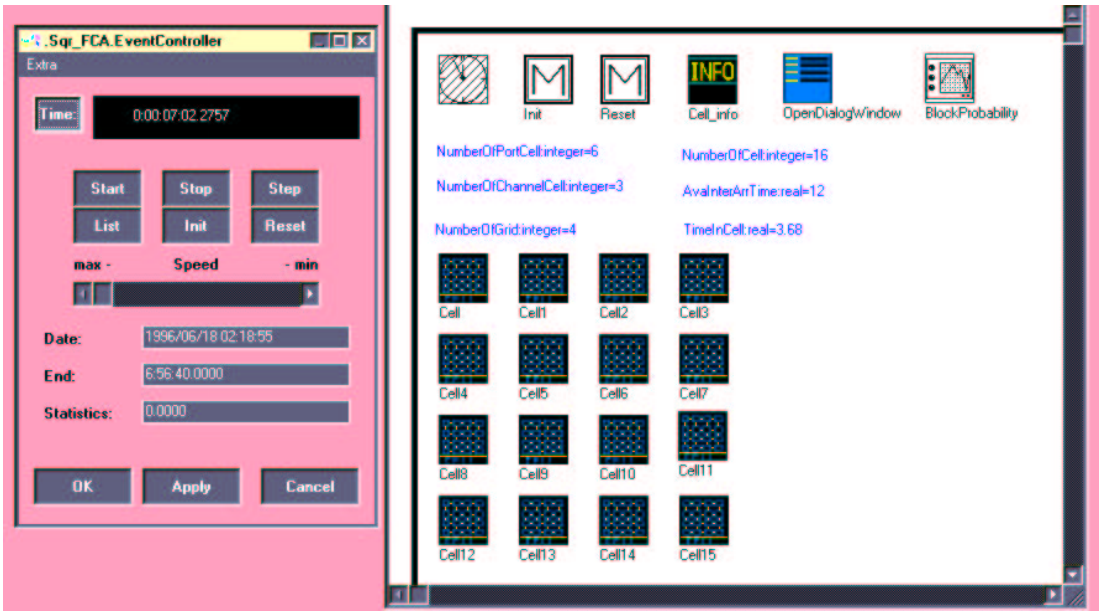


Figure 4: *Event Controller window.* This figure contains the *Sqr_FCA.EventController* window including a window on the left that contains buttons and a window on the right that contains icons. The buttons are used to manipulate the simulator; for example, the *start* button begins execution of the simulator, and the *step* button is used for debugging. The icons on the right side of the figure represent objects in the currently executing simulation.

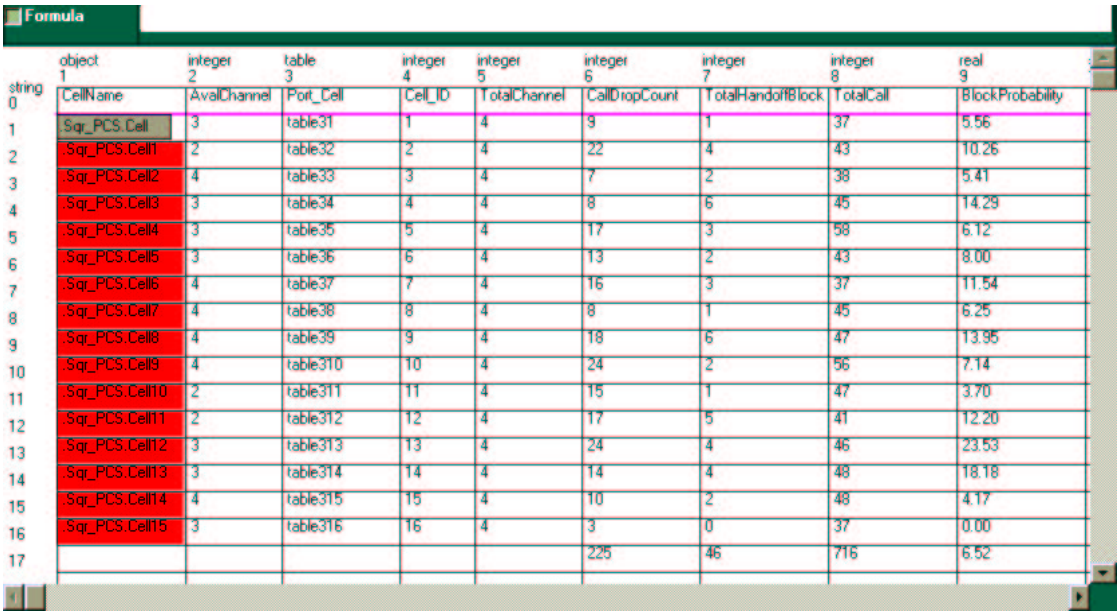
---

simulation, and a *stop* button to stop the simulation. There are two buttons in the *Sqr_FCA.EventController* window that can be used for debugging: the *list* button can be used to list the events that are currently in the event queue (future events), and a *step* button to permit the user to simulate events one at a time. Finally, the *reset* button destroys objects that have been created so that the user can re-initialize the system and restart the simulation.

The right side of figure 4 shows additional icons to represent objects in the currently executing simulation. One important icon is shaped like a clock, representing the event controller. The two square icons to the right of the event controller icon contain upper case M's; these icons represent methods for initializing and resetting the system. the init and reset methods are invoked through the *init* and *reset* buttons shown on the left side of Figure 4. Figure 4 also shows 16 icons representing the cells in the PCS network.

At this point, the user can simply click on the *start* button to begin the simulation. However, to further illustrate the model, we now discuss facilities in the model to permit further monitoring of the system. By double clicking on the *Cell_info* icon of Figure 4, the *cell info table* window, illustrated in Figure 5, is opened.

The *table window* contains information about each cell in the simulation; this information includes the



Formula

string 0	object 1 CellName	integer 2 AvalChannel	table 3 Port_Cell	integer 4 Cell_ID	integer 5 TotalChannel	integer 6 CallDropCount	integer 7 TotalHandoffBlock	integer 8 TotalCall	real 9 BlockProbability
1	Sqr_PCS.Cell	3	table31	1	4	9	1	37	5.56
2	Sqr_PCS.Cell1	2	table32	2	4	22	4	43	10.26
3	Sqr_PCS.Cell2	4	table33	3	4	7	2	38	5.41
4	Sqr_PCS.Cell3	3	table34	4	4	8	6	45	14.29
5	Sqr_PCS.Cell4	3	table35	5	4	17	3	58	6.12
6	Sqr_PCS.Cell5	3	table36	6	4	13	2	43	8.00
7	Sqr_PCS.Cell6	4	table37	7	4	16	3	37	11.54
8	Sqr_PCS.Cell7	4	table38	8	4	8	1	45	6.25
9	Sqr_PCS.Cell8	4	table39	9	4	18	6	47	13.95
10	Sqr_PCS.Cell9	4	table310	10	4	24	2	56	7.14
11	Sqr_PCS.Cell10	2	table311	11	4	15	1	47	3.70
12	Sqr_PCS.Cell11	2	table312	12	4	17	5	41	12.20
13	Sqr_PCS.Cell12	3	table313	13	4	24	4	46	23.53
14	Sqr_PCS.Cell13	3	table314	14	4	14	4	48	18.18
15	Sqr_PCS.Cell14	4	table315	15	4	10	2	48	4.17
16	Sqr_PCS.Cell15	3	table316	16	4	3	0	37	0.00
17						225	46	716	6.52

Figure 5: *Cell info table.* This figure contains information about each cell in the PCS network. Column 1 lists the cell name, column 2 lists the number of channels available for this cell, and column 3 points to a table containing information about the portables in this cell. Column 4 is a unique id for the cell, column 5 is the total number of channels available in this cell, column 6 maintains the number of calls dropped, column 7 maintains the number of handoffs that were blocked because no channels were available, column 8 is the total number of calls arriving at this cell and column 9 is the blocking probability. This table can be monitored during execution of the PCS simulation.

---

cell name, *CellName* in column one, and the number of available channels, *AvalChannel* in column two. Column three of the *table window* in Figure 5 lists the name of a table that stores further information about portables in the current cell. Note that our simulator uses nested tables. Column six through column nine contain information about the current cell; this information changes dynamically as the simulation is executing. These columns can be used to monitor the progress of the simulation.

To implement the fixed channel assignment scheme, methods in the portable object consult column two *table window* to determine if there is an available channel. If there is an available channel then the portable can receive the call; if there are no available channels, then there is a *handoff block* and the portable cannot receive the call.

3.2 Using Inheritance to Extend the Basic Model

To illustrate the extensibility of the SIMPLE++ model of the PCS network, we now describe the changes required to extend the model to use a dynamic channel assignment scheme rather than the fixed channel

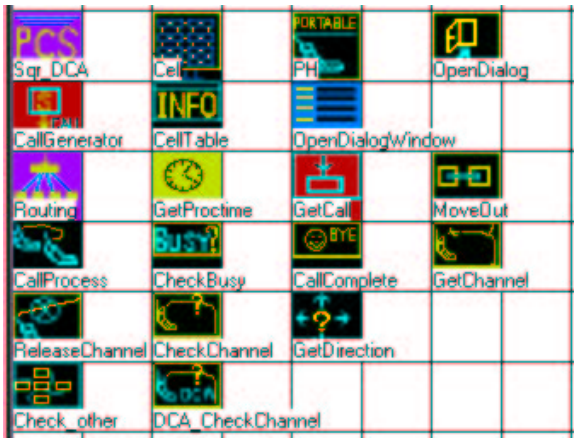


Figure 6: *Extending the model.* The top row of the figure contains an icon for the square network using the dynamic channel allocation scheme(*Sqr_DCA*). The icon *DCA_CheckChannel*, shown in the bottom row, is a method derived from *Check_Channel*; *DCA_CheckChannel* extends the functionality of *Check_Channel*.

---

assignment scheme described in the previous section.

Figure 3 illustrates the model that uses a fixed channel assignment scheme; this model is illustrated in rows five through seven on the right side of the figure. To extend the model to include dynamic channel assignment, The icon *DCA_CheckChannel*, shown in the bottom row of Figure 6, is added to the library; this icon represents a method, derived from method *Check_Channel*, to extend the functionality of *CheckChannel*. Method *CheckChannel* is listed in Figure 3 and, together with *DCA_CheckChannel*, also appears here in Figure 6. Also, an icon for method *Check_other* is added to the model. Note that the Sqr_FCA icon has been replaced by the icon Sqr_DCA in row one of Figure 3.

The *CheckChannel* method consults column two in the *cell info table* (Figure 5) to determine if a channel is available in the current cell. *DCA_CheckChannel* is composed in CheckChannel; if there are no channels available in the current cell, the *Check_other* is called by *DCA_CheckChannel* to determine if a channel can be borrowed from a neighboring cell. In the network with hex cells, each cell has six neighbors and *Check_other* will consult each one in an attempt to find an available channel. If an available channel is found, it will be temporarily owned by the borrowing cell until the call is completed. This "extra" channel is maintained in the borrowing cell until another cell requires an additional channel and this channel is available. This extension to DCA is facilitated by the *Method* construct in SIMPLE++.

Figure 7 summarizes the class hierarchy that extends our original square network using the fixed channel

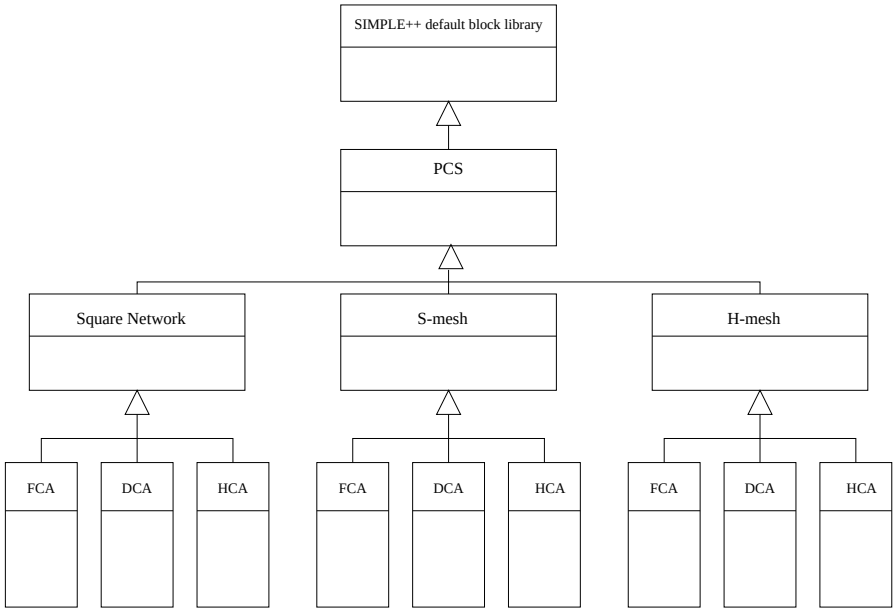


Figure 7: *Object Model.* This figure summarizes the object model that captures the design of our PCS simulator. Our original network was square shaped and used the fixed channel assignment scheme (FCA). We later extended the network to include the S-mesh and the H-mesh and incorporated two additional channel assignment schemes: dynamic channel assignment (DCA) and hybrid channel assignment (HCA). The base class for our class hierarchy is the SIMPLE++ default building block library.

---

assignment scheme (FCA). The boxes in the figure represent classes and lines represent inheritance; a line connects a base class with its derived classes and the arrow points to the base class. Note that the base class in our hierarchy is the SIMPLE++ default building block library, illustrated in Figure 1.

4 Performance

The experiments reported in this section summarize our results on a large network containing 1024 cells. Each cell initially contained 15 portables with 8 channels allocated to each cell using the fixed channel assignment scheme (FCA). The network was square shaped and contained cells that were square shaped. All experiments ran for five thousand clock cycles. Each experiment was executed twelve times, the high and low values were discarded and the results reported in this section are averages over ten executions. The host architecture for the simulator contained a Super Sparc processor running at 50 MHz with one megabyte of secondary cache and 128 megabytes of memory. The operating system was Solaris 2.5. For a comparison of the three channel allocation schemes simulated on a smaller network using hexagonal shaped cells on both an S-mesh and H-mesh, see reference[3].

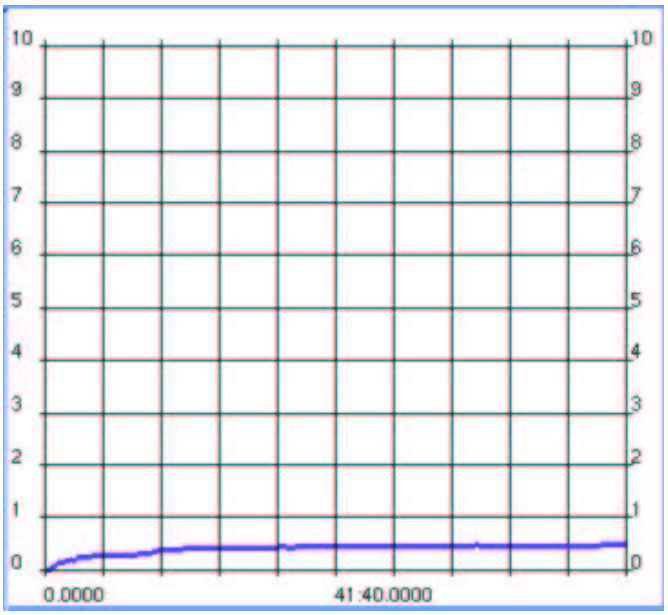


Figure 8: *Blocking percentage summary.* The line graph in this figure summarizes the blocking percentage for a square PCS network with square cells using the FCA scheme. There were 1024 cells in the network and each cell initially contained 15 portables with an average move out rate of 1/75 time clicks.

Figure 8 illustrates the change in blocking probability for the square shaped PCS network reported here. In the graph, the horizontal axis plots time and the vertical axis plots the blocking probability. The time on the horizontal axis ranges from 0 to 5000 clock ticks so that each vertical line represents 500 clock ticks. The blocking probability plotted on the vertical axis ranges from zero to ten percent. The graph was produced by the SIMPLE++ tool, which reports time in hours and minutes. Since the simulation was executed for five thousand clock cycles, the midpoint of the x axis is marked as 41:40.0000, representing a simulation time of 41 hours and 40 minutes, which is 2500 clock cycles.

Figure 8 shows that the blocking probability has stabilized well below one percent at the midpoint of the graph and remains stable over the final 2500 ticks of the clock. Thus, this figure shows that 5000 clock ticks is sufficient to produce stable simulation results.

Table 1 summarizes the results of the simulation. The first column describes the number of portables, N, initially assigned to each cell, and the average time before a portable *move out* occurs, M. The second column describes the blocking percentage, the third column describes the wall clock time required to run the simulation for 5000 simulation clock ticks and the fourth column describes the total number of calls that arrived at portables (*Total Calls*). Note that simulation terminates when time reaches 5000 cycles so that

S-mesh (FCA) (N, M)	*blocking percentage*	*time (Hours)*	*Total Calls*	*Total Events*	*Total Objects*
N=15, M=1/75	0.52	4.72	466,945	1,400,831	1,571,791
N=15, M=1/45	0.61	5.48	466,957	1,400,812	1,571,776
N=15, M=1/15	0.74	6.68	466,969	1,400,889	1,571,849

Table 1: *Statistics summary.* This table contains statistics to describe the results for a network configured as an S-mesh using the fixed channel allocation scheme (FCA). There are 15 portables initially in each cell and each portable has an average move out time of one every 75 clock ticks.

---

events are likely to remain in the event list. The fifth column in Table 1 summarizes the total number of events processed during the simulation and the final column summarizes the total number of objects in the simulation.

There are three rows of data in Table 1, where the first row describes a model with 15 portables initially in each cell and each portable moves out of a cell every 75 clock ticks, on average. We used a Poisson distribution to define the average move out. The model in the second row allowed portables to move out every 45 clock ticks and the final row allowed portables to move out every 15 seconds, on average. The table illustrates that as portables move out more frequently, more calls are blocked when using fixed channel assignment. For example, when portables moved out every 75 clock cycles, the blocking percentage is 0.52 percent; however, when portables moved out every 15 clock cycles, the blocking percentage rises to 0.74 per cent. Also, as portables move out more frequently, more execution time is required for simulating 5,000 clock cycles. For example, when portables moved out every 75 clock cycles, the simulation only required 4.72 hours to model 5,000 clock cycles; however when portables moved out every 15 clock cycles, 6.68 hours were required to model 5,000 clock cycles. Note that the *Total Calls*, *Total Events* and the *Total Objects* in the simulation remained relatively constant across the simulations.

5 Concluding Remarks

We have described our design and implementation of a PCS network simulation using the SIMPLE++ system. SIMPLE++ includes a *simulation language* together with an integrated *user environment* with features that include object orientation, graphical and animation capabilities, and dynamic monitoring of the simulation. We have used SIMPLE++ to quickly implement a network whose communication area can

be configured as a square, an S-mesh or an H-mesh. Cells in the network can be square or hex shaped and we have implemented three channel assignment schemes. The extensions and modifications that we have incorporated into our model show the advantages of using object technology together with a special purpose simulation system.

References

[1] D. Benson. Simulation modeling and optimization using promodel. *Proceedings of Winter Simulation Conference*, pages 447–452, 1996.

[2] C. D. Carothers, R. M. Fujimoto, Y. B. Lin, and P. England. Distributed simulation of large scale PCS networks. *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 2–6, January 1994.

[3] B. Chen, B. A. Malloy, and J. C. Peck. An extensible simple++ simulator for pcs network simulation. *Proceedings of the 1997 Conference on Object-Oriented Simulation*, pages 87–92, 1997.

[4] D. C. Cox. Personal communications a viewpoint. *IEEE Comunications Magazine*, 128(11), 1990.

[5] B. Diamond. Extend: A library-based, hierarchical, multi-domain modeling system. *Proceedings of Winter Simulation Conference*, pages 240–248, 1993.

[6] J. T. Douglass, D. A. Gupta, B. A. Malloy, and David A. Sykes. An efficient, extensible design of a PCS network simulation. *Proceedings of Object-Oriented Simulation Conference*, pages 109–114, January 1996.

[7] AESOP GambH. SIMPLE++ reference manual. 1995.

[8] AESOP GambH. Internet home page: http://www.aesop.de/simple.html. 1996.

[9] D. Krahl. Modeling with extend. *Proceedings of the 1996 Winter Simulation Conference*, pages 578–583, 1996.

[10] S.S. Kuek and W. C. Wong. Ordered dynamic channel assignment scheme with reassignment in highway microcells. *IEEE Transactions on Vehicle Technology*, 41(3):271–277, 1992.

[11] A. M. Law and M. G. McComas. Simulation of communications networks. *Proceedings of Winter Simulation Conference*, pages 73–77, 1996.

[12] Y. B. Lin and V. W. Mak. Eliminating boundary effect of a large scale personal communication service network. *ACM Transactions on Modeling and Computer Simulation*, 4(2):165–190, April 1994.

[13] A. M. Montroy and B. A. Malloy. A parallel distributed simulation of a large-scale PCS network: Keeping secrets. Technical Report 95-106, Clemson University, May 1995.

[14] A. Mullarney. MODSIM III - a tutorial. *Proceedings of Winter Simulation Conference*, pages 542–546, 1996.

[15] C. Roberts and Y. Dessouky. An overview of object-oriented simulation. *Journal of Simulation*, 1998.

[16] D. S. Smith and R. C. Crain. Industrial strength simulation using GPSS/H. *Proceedings of Winter Simulation Conference*, pages 165–171, December 1993.

[17] W. C. Wong. Packet reservation multiple access in a metropolitan microcellular radio environment. *IEEE Jounal on Selected Areas in Communications*, 10(6):918–925, August 1993.

[18] M. Zhang and T. S. Yum. Comparisons of channel-assignment strategies in cellular mobile telephone systems. *IEEE Transactions on Vehicle Technology*, 38(4):211–215, 1989.