Seamless Implementation of a Telephone Switching System Based on Formal Specifications in RTPA

Yingxu Wang, University of Calgary, Canada
Yousheng Tian, University of Calgary, Canada

ABSTRACT

The Telephone Switching Systems (TSS) is a typical real-time system that is highly complicated in design and implementation. In order to deal with the extreme complexity in real-world settings, a suitable and efficient mathematical means is required beyond any programming language. To this purpose, an efficient and precise denotational mathematics known as the Real-Time Process Algebra (RTPA) and the RTPA methodology for system modeling are introduced. Empirical experimental results are reported in this paper on the implementation of TSS based on formal models of the system in RTPA. Three phases of experiments are designed on TSS conceptual modeling, system interface design, and programming implementation and testing. All groups in the experiments with 7 to 8 members have been able to efficiently understood, design, and implement the TSS system in a simplified version in four weeks, which has been estimated as a 10+ person-year project in the industry. The efficiency and expressiveness of RTPA are empirically demonstrated base on the case studies in the experiments.


1. INTRODUCTION

A Telephone Switching System (TSS) is a complex real-time system (Thompson, 2000; McDermid, 1991; Wang, 2007, 2009). The functional structure of the TSS system can be described by a conceptual model as illustrated in Figure 1, which consists of four subsystems known as the call processing, subscribers, routes, and signaling subsystems.

In the conceptual model of the TSS system, its configuration encompasses 1 call processor and 16 subscribers. There are 5 internal switching routes and a set of 5 signaling trunks provid-
ing the dial, busy, ringing, ring-back, and special tones. The call processor modeled by a set of functional processes operates on the line scanners, call records, digits receivers, signaling trunks, system clock, and routes in order to implement a coherent program-controlled switching system.

An important finding in the basic research in software engineering is that any software system, including the hybrid hardware and software system can be rigorously modeled by a set of structural models, a set of process models, and their interactions (Hoare, 1978; Wang, 2002, 2007; Wang & King, 2000). The structure model (SM) is a rigorous abstraction and formal representation of the system’s architecture and components. The process model (PM) is a rigorous description of the system’s behaviors and operations onto the abstract SMs.

**Definition 1.** An abstract Structure Model (SM) is a generic architectural model for a software system, its internal control structures, and its interfaces with hardware components and external input/output, which can be rigorously modeled and refined as an $n$-tuple, i.e.:

$$SM \triangleq \mathbf{R} \{ (S_i | \forall e \in S_i, p_i(e) >) \}$$

(1)

where $S_i$, $1 \leq i \leq n$, is a set and also a type of elements $e$ that share the property $p_i$.

**Definition 2.** An abstract Process Model (PM) of a program $\wp$ is a composition of a finite set of $m$ embedded processes according to the time-, event-, and interrupt-based process dispatching rules, $\uparrow e_k S \rightarrow P_k$, i.e.:

$$UPM \triangleq \wp \mathbf{R} \{ \uparrow e_k S \rightarrow P_k \}$$

$$= \mathbf{R} \{ \uparrow e_k S \rightarrow P_k \}$$

$$= \mathbf{R} \{ \uparrow e_k S \rightarrow \mathbf{R} \{ s_i(k), r_j(k) s_j(k) \} \}, j = i + 1$$

(2)

*Figure 1. Functional structure of the TSS system*
where $s_i$ and $s_j$ are one of the 17 RTPA meta-processes, $r_{ij}$ is one of the 17 RTPA algebraic process operations, and $e_k$ is a general, timing, or interrupt event.

This paper reports an empirical experiment on the implementation of the TSS system based on formal models of the system in RTPA for conceptual modeling, system interface design, and programming implementation. In the remainder of this paper, the architectural designs of TSS in term of the structure models (SMs) in both RTPA and UML are described in Section 2. The functional designs of TSS in term of the process models (PMs) in both RTPA and UML are elaborated in Section 3. A set of comparative experiments in the design and implementation of the TSS system by three groups is demonstrated in Section 4.

2. THE ARCHITECTURAL DESIGN OF TSS BASED ON THE FORMAL MODELS IN RTPA

As described in Definition 1, the structure model is a rigorous abstraction and formal representation of the architecture of a system and the layout of a component. This section formally describes the structure models of TSS at the hierarchical levels of system and components from the top down, before the process models of TSS can be rigorously elaborated in Section 3.

2.1. The System Architectural Model of TSS

Based on the conceptual model as described in Figure 1, the TSS system encompasses four subsystems known as the CallProcessingSubsysSM, SubscriberSubsysSM, RouteSubsysSM, and SignalingSubsysSM. A formal model of the TSS system, TSS$\_$.ArchitectureSM, has been modeled in Wang (2009). The relationship between these subsystems of TSS can be illustrated by a package diagram in UML (Mishra, 1997; ISO, 2005; Wang & Huang, 2008) as shown in Figure 2. In the package diagram, CallProcessingSubsysSM is further refined by seven functional processes such as CallOriginationPM, DialingPM, CheckCalledStatusPM, ConnectionPM, TalkingPM, CallTerminationPM, and ExceptionalTerminationPM.

The TSS system encompasses 6 SMs, such as the line scanners, digit receivers, signaling trunks, system clock, switching routes, and internal call records, for modeling the system hardware interfaces and internal control structures. A rigorous RTPA model of the TSS architecture is given in Equation 1.

$$\text{TSS$\_$.ArchitectureSM} \triangleq \text{<LineScanners : SM | [16]>}$$
$$\| \text{<DgitsReceivers : SM | [16]>}$$
$$\| \text{<SignalTrunks : SM | [5]>}$$
$$\| \text{<SysClock : SM | [1]>}$$
$$\| \text{<Routes : SM | [5]>}$$
$$\| \text{<CallRecords : SM | [16]>}$$

(3)

where LineScannersSM, DgitsReceiversSM, SignalTrunksSM, and SysClockSM are SMs for hardware interfaces, while RoutesSM and CallRecordsSM are SMs for internal control structures.

2.2. The Structural Model of TSS at the Component Level

The six components of TSS denoted in Equation 3 can be formally refined as a detailed SM. For instance, the LineScannersSM is an abstract structure model of the interface device of a telephone switching system that connects a subscriber or a pair of telephone lines to the switching system.
Each telephone or subscriber is assigned a line scanner in a switching system. The SM of the line scanners, LineScannersSM, is designed as shown in Figure 3, where all the 16 concrete line scanners in TSS share the same SM structure.

LineScannersSM encompasses five fields known as the StatusN, PortAddressH, ScanInputB, CurrentScanBL, and LastScanBL. In the LineScannersSM model, the StatusN denotes the operating states of the line scanner with the type of natural number (integer); The PortAddressH denotes the designated addresses of a set of port interfaces in hexadecimal type; The ScanInputB denotes the information input in byte type from scanning the lines as specified by the PortAddressH; and the CurrentScanBL and the LastScanBL denote the logical line scan status in Boolean type. For each scan period, the following operations are conducted: LastScanBL := CurrentScanBL, and

Figure 2. The package diagram of TSS in UML

Figure 3. The structure model of the line scanners in TSS

\[
\text{LineScannersSM} \triangleq \bigcup_{i=0}^{15} \text{LineScanner}(\mathbb{N}, \mathbb{SM}); \\
\langle \text{Status} : \mathbb{N} | \text{StatusN} = \{0, \text{Idle}, 1, \text{HookOn}, 2, \text{HookOff}, 3, \text{Busy}, 4, \text{Invalid}\}\rangle, \\
\langle \text{PortAddress} : \mathbb{H} | \text{FF00H} \leq \text{PortAddressH} \leq \text{FF0FH}\rangle, \\
\langle \text{ScanInput} : \mathbb{B} | \text{ScanInputB} = <xxxx xxxxB>\rangle, \\
\langle \text{CurrentScan} : \mathbb{BL} | T = \text{HookOff} \land F = \text{HookOn}\rangle, \\
\langle \text{LastScan} : \mathbb{BL} | T = \text{HookOff} \land F = \text{HookOn}\rangle
\]
CurrentScan_{BL} := (ScanInput_{B, b_0})_{BL}, i.e., (ScanInput_{B, b_0})_{BL} := T when ScanInput_{B, b_0} = 1, otherwise (ScanInput_{B, b_0})_{BL} := F.

The 16 concrete objects of the line scanners in TSS share the same structural model LineScanners_{SM}. The concrete line scanner objects can be derived on the basis of the abstract schema as given in Figure 3. Each concrete model, LineScanner(i_{N})ST, obtains its refined physical or logical parameters according to its index number i_{N}.

As a counterpart, the formal component models of TSS in RTPA can be illustrated by informal means in UML known as the class diagram as shown in Figure 4. In the class diagram, only a set of data attributes and their types are modeled. However, more important information about the constraints of the attributes for a given component cannot be specified. In addition, further refinement of a certain attribute into more detailed structure models is impossible in UML.

The architectural model of a system not only creates an abstract representation of the system and its component as well as related attributes, but also provides a foundation for embodying its functions. On the basis of the architectural models of TSS, its functional or behavioral models will be able to be rigorously developed (Dromey, 2006; Wang, 2002, 2007) as elaborated in the next section.

3. THE FUNCTIONAL DESIGN OF TSS BASED ON THE FORMAL MODELS IN RTPA

According to Definition 2, the process model is a rigorous description of the system’s functional behaviors embodied by a set of process operations onto the SMs of the system. On the basis of the SMs of TSS as derived in Section 2, this section formally describes the process models of TSS, particularly its call processing processes such as CallOrigination_{PM}, DialingImpact, CheckCalledStatus_{PM}, Connecting_{PM}, Talking_{PM}, and CallTermination_{PM}, and ExceptionalTermination_{PM}.

Figure 4. A class diagram of TSS in UML
3.1. The Call Origination Process

Call origination is the first process of TSS call processing that identifies new call requests of subscribers and creates associated internal control structures for each new call. The call origination process of TSS, CallOriginationPM, is modeled in Figure 5 (left). The conceptual model of CallOriginationPM is illustrated in a UML activity diagram as shown in Figure 5 (right) (Mishra, 1997; ISO, 2005; Wang & Huang, 2008). Details may refer to the formal models of TSS (Wang, 2009).

3.2. The Process of Dialing

Dialing is the second process of TSS call processing that receives digits dialed by the calling subscriber on a specific line and registers them in the associated call record. The dialing process of TSS, DialingPM, is modeled in Figure 6 (left). The conceptual model of DialingPM is illustrated in a UML activity diagram as shown in Figure 6 (right). Details may refer to the formal models of TSS (Wang, 2009).

3.3. The Process of Check Called Status

Check called status is the third process of TSS call processing that looks into the current status of a given called subscriber, finds an available internal switching route between the calling and called parties, and sends busy tone to calling subscriber when called is busy or no route is free in the system. The check called status process of TSS, CheckCalledStatusPM, is modeled in Figure 7 (left). The conceptual model of CheckCalledStatusPM is illustrated in a UML activity diagram as shown in Figure 7 (right). Details may refer to the formal models of TSS (Wang, 2009).

Figure 5. The design models of Call Origination in RTPA and UML
3.4. The Process of Connection

Connecting is the fourth process of TSS call processing that informs the called subscriber with the ringing tone, and at the same time, sends the ring back tone to the calling subscriber that is waiting for the answer of the call. The connecting process of TSS, ConnectingPM, is modeled in Figure 8 (left). The conceptual model of ConnectionPM is illustrated in a UML activity diagram as shown in Figure 8 (right). Details may refer to the formal models of TSS (Wang, 2009).

3.5. The Process of Talking

Talking is the fifth process of TSS call processing that physically connects both parties using pre-seized route in the dialing process when the called subscriber answered, and monitors terminations by either party. The talking process of TSS, TalkingPM, is modeled in Figure 9 (left). The conceptual model of TalkingPM is illustrated in a UML activity diagram as shown in Figure 9 (right). Details may refer to the formal models of TSS (Wang, 2009).

3.6. The Process of Call Termination

Call termination is the final process of TSS call processing that handles call ending by either party, releases the occupied route, and immediately sends busy tone to the other party that has not hooked-on. The call termination process of TSS, CallTerminationPM, is modeled in Figure 10 (left). The conceptual model of CallTerminationPM is illustrated in a UML activity diagram as shown in Figure 10 (right). Details may refer to the formal models of TSS (Wang, 2009).
Figure 7. The design models of Check Called Status in RTPA and UML

Figure 8. The design models of Connection in RTPA and UML
4. EXPERIMENTS ON COMPARATIVE IMPLEMENTATIONS OF THE TSS SYSTEM BY MULTIPLE GROUPS

Based on the RTPA methodology and models (Wang, 2007, 2008a, 2008b, 2008c), software code can be seamlessly generated by automatic or manual implementation as shown in Figure 11 (Wang et al., 2010b). According to the scheme of RTPA-based code generation, the RTPA architectural model for the TSS system is used to generate the structural framework and global/local variables of classes or objects; while the RTPA behavioral process models are then transferred into object methods in a target programming language such as Java, C++, and C#.

Three groups of 23 final-year undergraduate students have been chosen to take part in an experiment for implementing the TSS system in a programming language based on the formal models of TSS as described in Sections 2 and 3. The experiment was divided into three phases: (a) conceptual modeling of TSS; (b) system interface design; and (c) programming and testing. The following subsections describe the experimental outcomes and findings from the three groups in the implementation of the TSS system.

4.1. Conceptual Modeling of the TSS System

In software engineering, system analysts and developers (programmers) may focus on different aspects of the system in design and implementation. System analysts put more efforts on system analysis and modeling in order to formally specify the system based on given requirements. On the other hand, software developers transfer a given specification into executable programs and verify its functionality against the specifications.

In the experiment, students were required to demonstrate their understanding of the given system by developing the conceptual models of TSS in UML. This results in three versions of...
Figure 10. The design models of Call Termination in RTPA and UML

Figure 11. Program code generation based on the formal models
the conceptual models of TSS systems as shown in Figures 12 through 14, which represents the perceptions and interpretations of individual groups on the formal models of the system.

These UML models submitted show that students are able to understand the system functions and formal specifications correctly. Each diagram expresses right behaviors according to the RTPA specifications for TSS. The experiment shows that the rigorous RTPA specifications are helpful to improve conceptual modeling of complex systems with less ambiguity as in the informal means such as in UML and programming languages.

4.2. Design of the TSS System Interfaces

The second part of the experiment is the development of the system interface of TSS independently carried out by the three groups. This is an important phase in system design and composition that involves certain level of creativity, vision, and art skills. Developers are required to design a Graphic User Interfaces (GUI) according to the system specifications, clients’ expectation, and their own understanding about the target system before it is fully implemented.

The results of the experiment on the design of the TSS’s GUI turn to be very interesting as shown in Figures 15 through 17. Three colorful and expressive system interfaces have been developed that represent the architecture (configurations) and functions (usages) of the TSS system. It is noteworthy that the same set of given specifications for the TSS system has resulted in various system interfaces as the external models and appearance of the system, which represent the interpretations, abstraction skills, experiences, and styles of different groups.

4.3. Programming Implementation and Testing

The third phase of the experiment is to develop the code and to implement the functions of the TSS system based on the conceptual and formal models of the system as well as its GUI. This phase seemed to be the easiest part in system implementation on the basis of all preparations in the earlier phases, as well as the availability of the formal models of TSS in RTPA. The results show that component-level programming based on well developed system models is a straightforward transformation of the SMs and PMs into a certain programming language as illustrated in Figure 11.

In the experiment, all the three groups adopted an object-oriented programming language, such as Java, C#, and .Net. All SMs of TSS have been implemented in the target programming

Figure 12. The conceptual model of TSS using activity diagram (Group B)
Figure 13. The conceptual model of TSS using sequence diagram (Group B)

Figure 14. The conceptual model of TSS using activity diagram (Group C)
Figure 15. The GUI interface of TSS by Group A

Figure 16. The GUI interface of TSS by Group B
languages based on the formal models. For example, the implementations of the CallRecordSM as a structure model of TSS result in three classes from each group as shown in Figures 18 through 20.

Accordingly, all PMs of TSS have been implemented in the target programming languages based on the formal models in the three groups. For example, DigitReceivingPM, as formally modeled in Figure 21, is a special real-time support process of TSS that receives the called subscriber number sent by the calling subscriber in high frequency periodical interrupt cycles in order to meet its timing constraints (Wang, 2009; Wang et al., 2010a).

The listings of class DigitsReceiver developed by the three groups are provided in Appendix A through Appendix C. This experiment shows that all functions can be captured and implemented in programs on the basis of the formal models in RTPA, after the conceptual model is well understood, the SMs are uniquely defined, and the GUI is created.

Testing and verification of the TSS system implementation are conducted by all groups. Each group has designed a set of test cases based on the expectations and requirements of the formal models of TSS in RTPA. The testing results as well as group presentations and inspections show that the system requirements and specifications of the complex TSS system have been properly understood and implemented. Though, in a real-world industrial setting, more development effort and through testing processes will still be needed for designing and implementing such a complex TS system.

5. CONCLUSION

A Telephone Switching System (TSS) has been recognized as a highly complicated real-time system. This paper has reported a comparative experiment on the implementation of such systems by three groups of students based on formal specifications of the system in RTPA. Three phases of experiments have been conducted on conceptual modeling of TSS, system interface design
Figure 18. The Call Recorder class designed by Group A

![Call Recorder class designed by Group A](image1.png)

Figure 19. The Call Recorder class designed by Group B

![Call Recorder class designed by Group B](image2.png)
Figure 20. The Call Recorder class designed by Group C

```

```

Figure 21. The process model of the digits receiving process in TSS

```

```

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
for TSS, and programming implementation and testing. All groups with 7 to 8 members have been able to efficiently understand, design, and implement the TSS system in a simplified version in four weeks, which has been estimated as a 10+ person-year project in the industry. The efficiency achieved in the development project has been partially due to the RTPA methodology which recognizes that any software system can be rigorously modeled by a set of interacting structure models (SMs) and process models (PMs).

The practical formal engineering methodology of RTPA for system modeling and specification provides a coherent notation system and systematical approach for large-scale software and hybrid system design and implementation. A series of formal design models of real-world and real-time applications in RTPA have been developed using RTPA notations and methodologies (Wang, 2002, 2007, 2008a, 2008b, 2008c, 2008d; Wang & Huang, 2008; Wang et al., 2010b) in the formal design-engineering approach, such as the telephone switching system (TSS) (Wang, 2009), the lift dispatching system (LDS) (Wang et al., 2009), the automated teller machine (ATM) (Wang et al., 2010c), the real-time operating system (RTOS+) (Wang et al., 2010a, 2010b), the autonomic code generator (RTPA-CG) (Wang et al., 2010b), the ADTs (Wang, Ngolah, Tan, Tian, & Sheu, 2010), the file management system (FMS) (Wang, Ngolah, Tan, Tian, & Sheu, 2011), the doubly-linked-circular list (DLC-List) (Wang, Ngolah, Tan, & Sheu, 2011), the universal arrays (UA) (Wang, Huang, & Lie, 2011), trees (Wang & Tan, 2011), digraphs (Wang & Adewumi, 2012), and the air traffic control system (Wang et al., in press). Further studies have demonstrated that RTPA is not only useful as a generic notation and methodology for software engineering, but also good at modeling human cognitive processes in cognitive computing and computational intelligence as reported (Wang, 2003, 2010; Wang & Ruhe, 2007; Wang & Chiew, 2010; Tain et al., 2011).

ACKNOWLEDGMENT

The authors would like to acknowledge Natural Science and Engineering Council of Canada (NSERC) for its partial support to this work. The authors thank the students who took part in the experimental case studies. The authors would also like to thank the anonymous reviewers for their invaluable comments that have greatly improved the latest version of this article.

REFERENCES


Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.


Yingxu Wang is professor of denotational mathematics, cognitive informatics, software science, and brain science. He is President of International Institute of Cognitive Informatics and Cognitive Computing (ICIC), Director of Laboratory for Cognitive Informatics and Cognitive Computing, and Director of Laboratory for Denotational Mathematics and Software Science at the University of Calgary. He is a Fellow of WIF (UK), a founding Fellow of ICIC, a P.Eng of Canada, a Senior Member of IEEE and ACM. He received a PhD in Software Engineering from the Nottingham Trent University, UK, and a BSc in Electrical Engineering from Shanghai Tiedao University. He has industrial experience since 1972 and has been a full professor since 1994. He was a visiting professor on sabbatical leave in the Computing Laboratory at Oxford University in 1995, Dept. of Computer Science at Stanford University in 2008, the Berkeley Initiative in Soft Computing (BISC) Lab at University of California, Berkeley in 2008, and CSAIL at MIT (2012), respectively. He is the founder and steering committee chair of the annual IEEE International Conference on Cognitive Informatics and Cognitive Computing (ICCI*CC). He is Editor-in-Chief of Journal of Advanced Mathematics and Applications (JAMA), Associate Editor of IEEE Transactions on System, Man, and Cybernetics (Part A), founding Editor-in-Chief of International Journal of Cognitive Informatics and Natural Intelligence (IJCINI), and founding Editor-in-Chief of International Journal of Software Science and Computational Intelligence (IJSSCI). Dr. Wang is the initiator of a few cutting-edge research fields or subject areas such as Denotational Mathematics (i.e., concept algebra, inference algebra, semantic algebra, real-time process algebra, system algebra, granular algebra, and visual semantic algebra); Cognitive Informatics (CI, the theoretical framework of CI, neuroinformatics, the logical model of the brain (LMB), the layered reference model of the brain (LRMB), the cognitive model of brain informatics (CMBI), the mathematical model of consciousness, and the cognitive learning engine); Abstract Intelligence (AI, mathematical models of intelligence, theoretical foundations of brain science); Cognitive Computing (such as cognitive computers, cognitive robots, cognitive agents, and cognitive Internet); Software Science (on unified mathematical models and laws of software, cognitive complexity of software, and automatic code generators, the coordinative work organization theory, and built-in tests (BITs)); basic studies in Cognitive Linguistics (such as the cognitive linguistic framework, mathematical models of abstract languages, the deductive semantics of languages, deductive grammar of English, and the cognitive complexity of online text comprehension). He has published over 130 peer reviewed journal papers, 230+ peer reviewed conference papers, and 25 books in denotational mathematics, cognitive informatics, cognitive computing, software science, and computational intelligence. He is the recipient of dozens international awards on academic leadership, outstanding contributions, research achievement, best papers, and teaching in the last three decades.

Yousheng Tian is a PhD candidate in cognitive computing and software engineering with the International Institute of Cognitive Informatics and Cognitive Computing (ICIC) as well as Lab for Software Science and Denotational Mathematics (LSSDM) in Dept. of Electrical and Computer Engineering at the University of Calgary, Canada. He received a PhD from Xian Jiantong University, China, in Computer Science in 2002 and was a Post Doctoral Fellow at LSSDM during 2006 to 2007. His research interests are in cognitive informatics, cognitive computing, software engineering, machine learning, and denotational mathematics.
APPENDIX A

Listing of DigitsReceiverPM in JAVA (Group A)

DigitsReceiver.java
package model;
public class DigitsReceivers{
    private int Status;
    private int DigitPort;
    private int DigitInput;
    private int DigitInput2;
    private int StatusPort;
    private int StatusInput;
    private int Digit1;
    private int Digit2;
    private int NumOfDigitsReceived;
    public DigitsReceivers(int receiverNum){
        Status = 0;
        DigitPort = 0;
        DigitInput = 0;
        StatusPort = 0;
        StatusInput = 0;
        Digit1 = 0;
        Digit2 = 0;
        NumOfDigitsReceived = 0;
    }
    public int getDigitInput2() {
        return DigitInput2;
    }
    public void setDigitInput2(int digitInput2) {
        DigitInput2 = digitInput2;
    }
    public int getNumOfDigitsReceived() {
        return NumOfDigitsReceived;
    }
    public void setNumOfDigitsReceived(int numOfDigitsReceived) {
        NumOfDigitsReceived = numOfDigitsReceived;
    }
    public void setDigitPort(int newPort){
        DigitPort = newPort;
    }
    public int getDigitPort(){
        return DigitPort;
    }
    public int getStatus() {
        return Status;
    }
    public void setStatus(int status) {
        Status = status;
    }
    public void setDigitInput(int newInput){
        DigitInput = newInput;
    }
    public int getDigitInput(){
        return DigitInput;
    }
}
public void setStatusPort(int newPort){
    StatusPort = newPort;
}
public int getStatusPort(){
    return StatusPort;
}
public void setStatusInput(int newInput){
    StatusInput = newInput;
}
public int getStatusInput(){
    return StatusInput;
}
public void setDigit1(int digit1){
    Digit1 = digit1;
}
public int getDigit1(){
    return Digit1;
}
public void setDigit2(int digit2){
    Digit2 = digit2;
}
public int getDigit2(){
    return Digit2;
}
public int getDialedNumber(){
    return Digit1*10 + Digit2;
}
public void setNumDigitsReceived(int numReceived){
    NumOfDigitsReceived = numReceived;
}
public int getNumDigitsReceived(){
    return NumOfDigitsReceived;
}
public void reset(){
    this.setNumDigitsReceived(0);
    this.setStatus(0);
    this.setStatusPort(0);
}

APPENDIX B
Listing of DigitsReceiverPM in JAVA (Group B)

public class DigitsReceiver {
    public Port DigitPort = null;
    public Port StatusPort = null;
    public DigitsStatus Status = DigitsStatus.NoDial;
    public int Digit1 = 0;
    public int Digit2 = 0;
    public int DigitsReceived = 0;
    public byte StatusInput = (byte) 0x00;
    public byte DigitInput = (byte) 0x00;

    public DigitsReceiver(int DigitPort, int StatusPort) {
    }
class DigitsRecievers
{
    public List<DigitsReceiver> digitsReceivers;
    public DigitsReceiver getDR(int i)
    {
        return digitsReceivers.ElementAt(i);
    }
    public void incommingDigit(int line, int digit)
    {
        digitsReceivers[line].incommingDigit(digit);
    }
    public void receive(CallRecords crs)
    {
        List<CallRecord> records = crs.callRecords;
        //Find any CallRecord in Records that Has a Call Process of 2 or “Dialing”
        foreach (int i in Enumerable.Range(0, records.Count))
        {
            if (records.ElementAt(i).getCallProcess() == (int)CallProcessStates.Dialing) //If we are at the dialing status
            {
                DigitsReceiver dr = digitsReceivers.ElementAt(i);
                //Read data from memory into the statusInput
                dr.statusInput = dr.statusPort.getNumericData();
                if (dr.statusInput == 1) //If the status port has been set....
                {
                    //Set status to dialing
                    dr.status = (int)DialStatus.DialStarted;
                    //Read data from memory int digitInput
                    dr.digitInput = dr.digitPort.getNumericData();
                    if (dr.numOfDigitsReceived == 0)
                    {
                        dr.digit1 = dr.digitInput;
                        dr.status = (int)DialStatus.Dialing;
                        dr.numOfDigitsReceived++;
                        dr.statusPort.setBooleanData(false);
                    }
                    else
                    {
                        dr.digit2 = dr.digitInput;
                        dr.status = (int)DialStatus.DialCompleted;
                        dr.numOfDigitsReceived = 0;
                        dr.statusPort.setBooleanData(false);
                    }
                }
                else
                {
                    dr.digit2 = dr.digitInput;
                    dr.status = (int)DialStatus.DialCompleted;
                    dr.numOfDigitsReceived = 0;
                    dr.statusPort.setBooleanData(false);
                }
            }
        }
    }
}

APPENDIX C

Listing of DigitsReceiverPM in JAVA (Group C)
digitsReceivers = new List<DigitsReceiver>();
foreach (int i in Enumerable.Range(0, numberOfLines))
{
    Port p1 = new Port(i * 2);
    Port p2 = new Port((i * 2) + 1);
    ports.Add(p1);
    ports.Add(p2);
    digitsReceivers.Add(new DigitsReceiver(0, p1, p2, 0, 0, 0));
}
}