Analyzing Event Chains of Private Branch Exchange Systems Modeled with UML 2.0

Saehwa Kim
Department of Information and Communications Engineering
Hankuk University of Foreign Studies

This paper analyzes event chains of private branch exchange (PBX) systems modeled with UML 2.0. UML 2.0 is a de-facto industrial standard as a real-time object-oriented modeling language. Our PBX UML 2.0 model adopts proxy design pattern and the event chains are derived from a bottom proxy layer that accepts external inputs. It is critical to correctly analyze event chains for a system to meet any real-time requirements such as deadlines. This case study is to guide for designing real-time object-oriented models for any other kinds of complex real-time communications systems.

1. Introduction

Real-time embedded systems are becoming increasingly more sophisticated and complex, while at the same time experiencing a shorter time-to-market with greater demands on reliability. As a result, the need for systematic software development methods and tools for real-time embedded systems is ever increasing. Object Management Group (OMG) [1] have initiated Model Driven Architecture (MDA) [2] as an approach to support model-to-code bridge, which clearly shows the high demand for the ability to generate executable applications directly from object-oriented models. Model Driven Architecture uses the Unified Modeling Language (UML), UML 2.0 [3], to allow modeling of executable architectures. Using UML 2.0, the industry standard modeling language, designers can raise the abstraction level and stop worrying about implementation level concepts like tasks and mutexes and instead focus on the desired behavior of their systems.

However, many real-time applications still are not modeled in real-time object-oriented models due to many reasons including the steep learning curve for real-time object-oriented models. To overcome this problem, we argue that various case studies should be performed for

---

1 This work was supported by Hankuk University of Foreign Studies Research Fund of 2010.
modeling real-world real-time systems. Specifically, we argue that a reference real-time object-oriented model should be provided that has a reasonable degree of complexity.

This paper analyzes event chains of private branch exchange (PBX) systems modeled with UML 2.0. Our PBX UML 2.0 model adopts proxy design pattern and the event chains are derived from a bottom proxy layer that accepts external inputs. We have presented the structural and behavior models for each component in [4]. This paper describes event chains for the PBX UML 2.0 model. Event chains that start from the external input events and finish with any external outputs or internal data manipulations are directly related with the end-to-end timing constraints. Thus, it is critical to correctly analyze events chains to make the system to meet any real-time requirements, such as deadlines. We believe that such kinds of case studies essentially helpful in guiding for designing real-time object-oriented models for any other kinds of complex real-time communications systems.

The remainder of the paper is organized as follows. Section 2 describes UML 2.0 modeling language. Section 3 describes the top-level structured model of our PBX system. Section 4 explains detailed event chains for each external event. The final section concludes the paper.

2. UML 2.0

UML 2.0 is a general purpose modeling language developed by the OMG, and contains corrections and new content based on user feedback on the industry standard UML 1.x modeling language. One of the important additions in UML 2.0 is the concept of structured classes. This concept makes it possible to define the run-time structure of a class as the composition of multiple structured classes connected together. It has been developed to properly represent complex, event-driven, potentially distributed real-time and embedded systems. The additions to UML 2.0 are inspired by ROOM [5]; another object-oriented modeling technique for real-time systems.

The basic element of model construction in UML 2.0 is a structured class. A structured class represents an object within the system that communicates with other structured classes exclusively through interfaces called ports. As shown in Figure 1 (a), structured classes connected together define the run-time structure and communication channels of an application. A finite state machine, represented by a state diagram, represents the behavior of a structured class. An example of a state diagram is depicted in Figure 1 (b). Receiving messages via ports causes the state machine to make transitions, executing the logic contained in the structured
The full behavior of a system is defined by the composition of all structured classes, their connections and their state machines. The structure of a structured class is defined in a structure diagram. In this diagram other classes can be used as parts of the composition. These are referred to as structured-class-parts. A structured-class-part can be either fixed, optional or plug-in. All fixed structured-class-parts contained in a system are instantiated when the system is initialized. Alternatively, a structured-class-part can be marked as optional or plug-in and such a structured-class-part is instantiated dynamically according to the needs of designers. They are not instantiated at initialization but must be explicitly created and destroyed by a state transition. A plug-in structured-class-part is not an actual instance, but is a reference to an existing structured class instance in the model. Plug-in structured-class-parts are created by importing a reference an instance that is created in an incarnated optional or a fixed structured-class-part. The reference can also be dynamically deported.

Other concepts in UML are replication of structured-class-parts and ports. A replicated structured-class-part has multiple instances, which is referred to as having multiple cardinality. Each individual instance of a replicated structured-class-part can be accessed by using the replication index.

3. PBX System Modeled in UML 2.0
Figure 2 shows our PBX system modeled in UML 2.0 briefly. The model actually exploits 29 structured-classes and consists of four top level structured-classes: ProxyManager, DeviceManager, OAMSbsubsystem, and CallController. We presented more detailed descriptions for the structure and behavior models in [4].

You can see examples of dynamic structures found in our PBX model. The hashed structured-class-parts are optional, and the solid structured-class-parts are plug-ins. In the figure you can see that the PhoneProxy instance, which was explicitly incarnated by the ProxyManager structured class, is later imported into the DeviceManager structured-class-part. The four top level structured-class-parts are examples of fixed structured-class-parts.

The PhoneProxy is a replicated structured-class-part; there are multiple instances of PhoneProxy in ProxyManager, but it is modeled as one structured-class-part. Replicated ports can be understood in much the same way. A structured class may require multiple instances of one port and so the port is replicated. Messages may be sent from all of the port instances at once or they may be sent from one particular instance by specifying the port index. In Figure 2, you can see that the port connecting the ProxyManager and OAMSbsubsystem structured-class-parts is replicated so that each instance of PhoneProxy has a discrete connection to OAMSbsubsystem.

4. Event Chains
Event chains in the PBX system are, for the most part, initiated by input from a physical cell phone device such as powering the cell phone on or pressing one of the keypad buttons. The only digression from this is found in the connection observation mechanism which intermittently pings the physical cell phone device to detect if the connection is dropped. In the real world the cell phone would communicate with the PBX over a radio connection. This connection is not modeled in the PBX model used. Instead the button presses are directly forwarded to the PBX system. This is a simplification to make the model easier to understand. We summarize major possible events chains in the following subsections.

4.1 Powering On/Off (bPwr)

The power button, as would be expected, signals to the system that the cell phone device is being turned on or being turned off. In the former case, upon receiving the power signal the PhoneProxy returns a PwrOnAck signal to the cell phone device and proceeds to inform OAMSubsystem that the cell phone has been powered on. The port index that the signal is received on is the extension index associated with the PhoneProxy, with which OAMSubsystem can match a telephone number. If a number exists for that extension, OAMSubsystem signals the DeviceManager to incarnate a Phone instance and import the PhoneProxy associated with this Phone instance. Now that there is a complete abstraction of the cell phone, OAMSubsystem returns the cell phone number to the PhoneProxy which in turn sends the number to the cell phone.

The case for powering off reverses the above. The PhoneProxy informs OAMSubsystem that the cell phone is being turned off. OAMSubsystem informs the DeviceManager of this, the Phone instance is destroyed, and OAMSubsystem acknowledges that the cell phone has been powered off. This acknowledgment is passed on to the PhoneProxy, and then to the cell phone, as a PwrOffAck signal.

4.2 Pressing Number Buttons (button0-button9)

When a number button is pushed, the digit signal does not propagate deeply into the PBX system. Instead, up to 7 digits are queued in the PhoneProxy. Upon receiving the send button (bSnd) signal, the digits are delivered into the system for processing.

4.3 Requesting and Receiving Calls (bSnd)
When a PhoneProxy receives the bSnd signal and has a full digit queue, the PhoneProxy sends an Offhook signal, along with the digits, to its associated Phone instance. The Phone then incarnates an OrigSession and signals CallController to create a Call instance. When the Call is incarnated, the newly instantiated OrigSession is imported into this Call instance. The Call instance validates the dialed number with OAMSubsystem, and upon validation requests a TermSession associated with the called Phone extension.

At this point, one of three things may happen. The Phone associated with the number may be (1) offline, (2) busy, or (3) available. In the first two cases the appropriate error message is sent back to the cell phone and the system reverts to the state it was in before the attempted call. In the latter case, the called Phone incarnates a TermSession which is then imported into Call. The Call instance now has imported an OrigSession from the originating Phone and a TermSession from the terminal Phone, and a Ringing signal is propagated back to the respective cell phones. If the terminal cell phone responds with a bSnd signal, the call is completed and a Connected signal is propagated to the cell phones.

4.4 Terminating Calls (bEnd)

The bEnd signal from the cell phone is ignored unless the cell phone has just dialed a number that is ringing but has not yet answered, or the cell phone is currently connected in a call. In both cases bEnd results in the OrigSession, the TermSession, and the Call instance being destroyed, and the involved Phones become ready to receive or place a new call.

4.5 Observing Connections (ping)

After a cell phone powers on, the PhoneProxy is responsible for assuring that the connection to the cell phone device is available. This is achieved by using an intermittent ping signal, which in turn is echoed by an ack from the phone. If the cell phone fails to respond for a determined length of time it is treated as if a bPwr signal had been received, turning off the cell phone.

5. Conclusion

The paper presented the event chains of private branch exchange (PBX) systems modeled with UML 2.0. The presented PBX system is a real-world complex and real-time communication
Analyzing Event Chains of Private Branch Exchange Systems Modeled with UML 2.0

system servicing cell phones. UML 2.0 is a de-facto industrial standard real-time object-oriented modeling language. One of the major hurdles for adopting real-time object-oriented models is a steep learning curve for any developers. As an effort to overcome it, we presented a reference real-time object-oriented model for in [4]. As a continuing activity, this paper presented an analysis of event chains for the same system in [4]. It is critical to correctly analyze event chains for a system to meet any real-time requirements such as deadlines. Our PBX UML 20 model adopts proxy design pattern and the event chains are derived from a bottom proxy layer that accepts external inputs. This case study is to guide for designing real-time object-oriented models for any other kinds of complex real-time communications systems.

References


UML 2.0 으로 모델링된 사설 교환기의 이벤트 사슬 분석

김세화
정보통신공학과
한국외국어대학교

요약: 본 논문에서는 휴대폰을 UML 2.0 으로 모델링된 사설 교환기(PBX)의 이벤트 사슬을 분석한다. UML 2.0 은 실질적인 산업계 표준 실시간 객체지향 모델링 언어이다. 이 PBX 모델은 프록시 설계 패턴을 채택하며 이벤트 사슬은 외부 입력을 받는 하부 프록시 계층으로부터 유도된다. 종료시한과 같은 실시간 요구사항을 만족시키기 위해서는 이벤트 사슬을 정확하게 분석하는 것이 필수적이다. 개발자들은 임의의 다른 종류의 복잡한 실시간 통신 시스템을 실시간 객체지향 모델로 설계할 때 이 사례 연구를 활용할 수 있다.

---

2 본 논문은 2010년도 한국외국어대학교 학술연구비 지원에 의해서 연구되었음