MODELING AND ANALYSIS OF MOBILE PUSH NOTIFICATION SERVICES USING PETRI NETS

(Extended version of 7186 at SCC 2014)

Junhua Ding[1, 2]
[1]) Dept. of Computer Science
East Carolina University
Greenville, NC 27587
dingj@ecu.edu

Wei Song
Dept. of Computer Science
Nanjing University of Sci. & Tech.
Nanjing, China
wsong@njust.edu.cn

Dongmei Zhang[2]
[2]) School of Computer Sciences
China University of Geosciences
Wuhan, Hubei, China
jjielee@163.com

Abstract

Mobile push notifications are an important feature in mobile computing services and they have been widely implemented in mobile systems. However, it also brings the vulnerability of security and reliability to the system. Formal specification and verification is an effective approach for understanding the properties of mobile push notification services and ensuring quality of the system development. Due to the dynamic interaction, security and mobility properties in mobile computing services, formally modeling and analyzing them is a grand challenge. In this paper, we proposed an approach to model mobile computing services using a high level Petri nets and analyze them through combining formal verification and testing techniques. The dynamic interaction between users and service providers is modeled with the publisher subscriber architecture. The mobility is modeled with a connector that dynamically connects a user to its service provider based on user's runtime environment, and the security is modeled with a threat model. The effectiveness of the approach is demonstrated with a case study of modeling and analyzing a mobile searching engine that is implemented with mobile push notifications services.

Keywords: mobile computing service; mobile push notification; formal specification; Petri net; testing

1. INTRODUCTION

Mobile push notifications are an important feature of mobile computing services and they have been widely implemented in mobile applications [1]. Mobile push notification services have become an important type of services to deliver contents to users. Many widely used commercial mobile applications, such as social network system Facebook, hangout system Tinder, instant messaging system WeChat, were implemented with mobile push notifications. Applications supporting special services such as weather service, travel service or traffic service were also implemented with mobile push notifications to deliver time sensitive and personalized contents to mobile users. The most important character of mobile push notifications is a service provider pushes real time notifications to subscribers based on their current contexts like their locations, status, and emotions. Mobile push notifications are fairly complex to implement due to several reasons: they have to be implemented based on a third party service such as Apple or Google mobile push notification services, they have to be implemented on secure communication protocols to ensure the security of the mobile communication, and they have to deal with the mobility issue and the dynamic connection between users and service providers. In order to understand mobile push notifications and build high quality mobile computing services, it is necessary to model them with a formal specification and analyze the model rigorously. In this paper, we proposed an approach to model mobile push notification services using a high level Petri net and analyze them through combining formal verification and testing techniques. The dynamic interaction between users and service providers is modeled with the publish-subscribe architecture. The mobility is modeled with a connector that dynamically connects a user to its service provider based on user's runtime environment, and the security is modeled with a threat model to ensure the service is appropriately protected from attacks.

The effectiveness of the approach is demonstrated with a case study of modeling and analyzing a mobile searching system that is implemented with mobile push notifications services. The system was built on Amazon AWS, and the mobile push notification was supported by Amazon notification service and Google mobile push notification service [1][2]. We modeled the mobile push notification using Predicate Transition nets (PrT) – a type of high level Petri nets [13], and rigorously analyzed them using a model based testing tool that supports both formal analysis and software testing. Comparing to other methods, the PrT nets model developed in the approach is executable so that they support simulation and testing easily. More important, the model can be developed step by step based on simulation and analysis results during modeling process, and tests used for testing the model can be directly used for testing the implementation. Therefore, there is no gap between the

analysis technique using for testing models and the analysis technique using for testing the implementation.

The contributions of this research are summarized as follows: We proposed an approach for formally modeling and rigorously analyzing the mobile computing services, particularly the mobile push notification services. We addressed the challenge issues on mobility, security and dynamic interaction in modeling and analysis. The effectiveness of the approach is demonstrated through a case study of modeling and analyzing a mobile searching system. The approach can be used for modeling and analyzing similar mobile computing systems. The modeling process is explained via modeling the mobile searching system in the case study. A PrT net model of the mobile push notification implemented in the system is refined step by step with simulation and analysis results. The case study shows the analysis results-guided modeling approach is effective for understanding and developing formal models of complex systems. The analysis process is explained via analyzing the mobile searching service in the case study. The case study also shows the model based analysis is an easy to use and an effective approach for analyzing complex systems.

The rest of this paper is organized as follows: Section 2 presents the preliminaries for modeling and analyzing mobile push notification services. Section 3 introduces the approach for modeling and analyzing mobile push notification services in general. Section 4 describes a case study of modeling and analyzing a mobile searching system that is implemented with mobile push notification services. Section 5 reviews the related work. Finally, we outline our conclusions as well as future work in Section 6.

2. PRELIMINARIES

In this section, we discuss the preliminaries for modeling and analyzing mobile computing services. The modeling language used in this paper is PrT nets, and the analysis approach is developed based on tool MISTA. We also discuss the architecture of mobile push notifications.

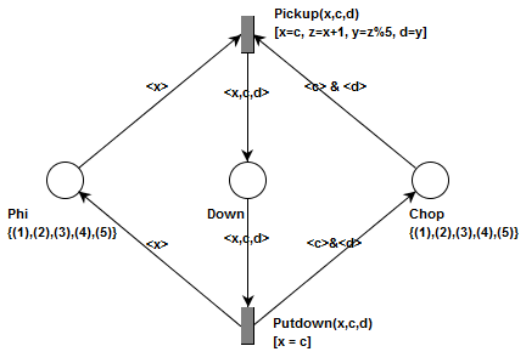


Figure 1. A PrT nets model of dining philosopher problem

2.1 PrT Nets

Predicate/Transition (PrT) nets, a high level Petri net, are used for specifying software systems. The formal definition of PrT nets used in this paper is same as the one defined in [16], which restricted the predicate types as enumerable types

Definition 1 (PrT net). *A PrT net is a tuple* $(P, T, F, \Sigma, L, \varphi, M_0)$, *where*:

1. *P is a finite set of predicates* (*i.e. first order places*), *T is a finite set of transitions and F is a flow relation* (*i.e. normal arcs*). (P, T, F) *forms a directed net.*
2. Σ *is a structure consisting of some sorts of individuals* (*i.e. constants*) *together with operations and relations*.
3. *L is a labeling function on arcs*.
4. φ *is a mapping from a set of inscription formulae to transitions.* $\varphi(t)$ *is built from variables, relations, operations and constants in* Σ.
5. M_0 *is the initial marking, where* $M_0(p)$ *is the set of tokens in place p. Each token is a tuple of constants in* Σ.

Fig. 1 shows a PrT net model for the 5 dining philosophers' problem, which includes transitions *Pickup*, and *Putdown*, places *Phi*, *Chop* and *Down*. Places *Phi* and *Chop* include tokens that are nature numbers representing philosophers or chopsticks, and each token in place *Down* includes represents a philosopher and his/her two chopsticks. Transition *Pickup* has two input places *Phi* and *Chop*, and one output place *Down*. The guard condition in transition *Pickup* is defined based on the relation between the tokens in place *Phi* and *Chop*. The guard condition in transition *Putdown* is defined based on the relation between the tokens in place *Phi* and *Chop*.

2.2 Model-Based Testing and MISTA

MISTA is a model-based testing tool for automated generation of executable test code in model level and program level. It uses function nets (a type of PrT nets extended with inhibitor arcs and reset arcs) [19]for specifying test models so that complete tests can be automatically generated. It also provides a language for mapping the elements in function nets to implementation constructs, which makes it possible to convert the model level tests into program level tests that can be executed against the system under test. MISTA includes several important components: model editor, model parser, simulator, reachability analyzer, test generator, and test code generator. MISTA support the step-by-step execution and random execution of a function net, and the execution sequences and token changing in each place are visualized for inspection. The test generator generates model level tests (*i.e.*, firing sequences of the function net) according to a chosen coverage criterion such as transition coverage or state coverage. The tests are organized and visualized as a transition tree. MISTA supports a number of coverage criteria for test generation from function nets, including

reachability graph coverage, transition coverage, state coverage, depth coverage, and goal coverage. Test code generator generates test code in a chosen target language like Java or C++ from a given transition tree [19].

2.3 Mobile Push Notifications

We explain the mobile push notifications using Amazon Simple Notification Service (SNS) [1] as an example since the system we discussed in the case study was implemented on Amazon SNS. We will discuss a representative mobile push notification service in details in section 3. Amazon SNS is a web service for managing the delivery of messages from publishers to subscribers. A publisher sends a message to a communication channel called a topic in SNS, and a subscriber who subscribed the topic will receive the message or notification via a communication protocol such as email or HTTP/S [1]. Amazon SNS sends the notification message to clients via one of the push notification services: Apple Push Notification Services (APNS), Google Cloud Messaging for Android (GCM), and Amazon Device Messaging (ADM). In order to use a push notification service, a mobile client needs to register itself for the service so that the service can maintain a connection to the client. When a mobile client registered a push notification service, the service returns a device token representing the registered device to Amazon SNS for registering the mobile client in SNS. Using the notification service credentials provided by a notification publisher, Amazon SNS can communicate with the mobile push notification service providers on behalf of the publisher. A publisher sends a message to a topic in Amazon SNS, and then Amazon SNS checks mobile devices according to the tokens it received. The notification message is sent to the push notification services, and each notification service pushes the message to the subscribed mobile clients [1].

One of application examples of mobile push notification is group online game, where game players connect to the online game provider using their mobile devices. While one is playing the game, actions like move and attack from each player appear on the screen right way. When one player switches from the game to another application while other players are still playing, it is necessary to notify the player when an action from others might be of interest to this one. Mobile push notifications can be used for the notification purpose. The publisher of the game application monitors game actions and finds one action may affect the player who just switched from the game to another application (*i.e*, the player is in no longer connected to the game), a push notification message is pushed to the player. First the publisher, who has sent its credentials to Amazon SNS and was authorized to access the SNS, sends the notification message to Amazon SNS. Then Amazon SNS, which has received the device tokens from all players, sends the messages to the push notification services. Finally, one of the push notification services such as APNS the mobile client has registered for pushes the notification message to the registered mobile device that the player is off from the game. A notification could be implemented in different formats like displaying a message, displaying an alert, badging the game app's icon, or even playing a sound. The player may response the notification immediately or later.

3. Modeling and Analysis of Mobile Push Notification Services

In this section, we discuss how to model mobile push notification services using PrT nets, and how to analyze the model using model based testing technique.

3.1. Architecture

The mobile push notification services offered by Apple, Google and Amazon support the publish-subscribe (PS) style of interaction between publishers who produce content and subscribers who subscribe the content. The PS architecture supports the asynchronous interaction between publishers and subscribers, so that message exchange is still possible even one of them is not in active [14]. In order to support the asynchronous interaction, the architecture needs support content queuing and delivering. A GCM implementation includes three components: a GCM-enabled app running on an Android device, an application server that produces notification messages to the app, and a Google GCM server that is responsible for delivering messages from the application server [2]. Each application server has a *Sender ID* to identify itself, each app has an *application ID* for receiving appropriate messages, and a *registration ID* issued by GCM server to the app to build the connection between the application server and the app. The application sever also has a *Sender Auth token* to give it authorized access to GCM services. The basic interaction among the three components of GCM is described as follows:

1. The application server creates a *Sender ID* and *Sender Auth token* with a GCM server, and registers itself to the GCM server.
2. An app registers itself to the GCM sever with its *application ID* and application server's *Sender ID*. The GCM sever creates a *registration ID* and send it to the application server.
3. The application server sends a message to the GCM server; the sever enqueues and stores the message, and then delivers it to the app as soon as the device that is running the app is online.

We model the architecture of GCM services in the publish-subscribe style following the definition in [5]. In the architecture, the publisher is the GCM server and its connected application server that produces notification messages and the subscriber is an app which has subscribed the notification messages. The architecture includes several structural components: an app that registers itself to the GCM server and an application server that sends messages to the GCM server for delivering; a GCM server announces

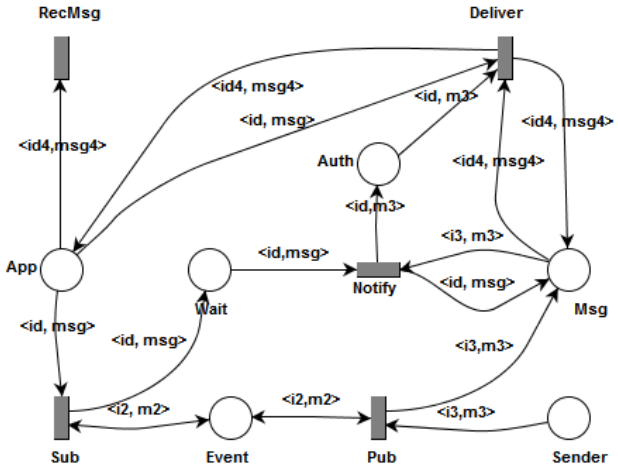


Figure 2. A PrT net model of GCM

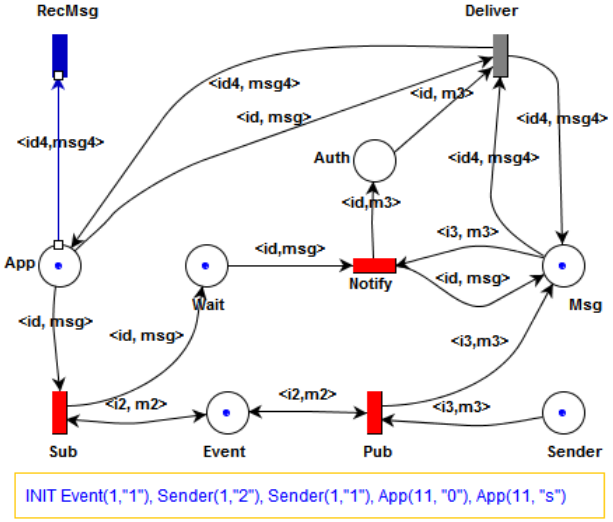


Figure 3. An execution snapshot screen of Figure 2

subscription events and registers each app with its subscribed messages. The publisher and subscriber are bound with the events, and messages are delivered asynchronously by the GCM server. The PrT nets model for modeling the PS architecture of GCM is shown in Fig. 2.

In Fig. 2, each place is a structured data type including two fields: an *ID* field defines identify information, and a *message* field defines a structured message that could be a notification message or an action message. Transition *Sub* defines the subscription behavior, and transition *Pub* defines the publishing behavior. Place *App* represents the app that is going to subscribe messages and receive subscribed messages, and transition *RecMsg* defines the behavior for receiving messages. Place *Sender* represents the application server that produces messages. Place *Event* is used to collaborate between transition *Sub* and *Pub* to ensure that a subscribed message exists in the publisher. Place *Wait* is defined for the subscription messages, and transition *Notify* is defined to register the subscription of messages and send a message of *registration ID* to the subscriber. Place *Auth* is defined for the *registration ID*, and *Msg* is defined for the registered messages, which stores the matrix that maps the each app to its subscribed messages. Transition *Deliver* sends a notification message or *registration ID* to the subscriber. Each transition may include guard conditions to define the firing of the transition in addition to the flow relations that are associated with the transition. Since guard conditions have to be defined with structure details of the messages, they are not defined in Fig. 2 without affection the illustration purposes in this section. Guard conditions will be discussed in detail in the case study section.

We use MISTA to execute the PrT net model for simulating the mobile push notification services and check the interactions among publisher and subscriber. The simulation process is very helpful for developing a correct model. Fig. 3 shows a snapshot screen of the simulation and the corresponding initial marking, which shows an app subscribed an message and will receive a *registration ID*.

3.2. Mobility

Mobile push notification services can deliver messages to both stationary users and mobile users. A stationary user receives subscribed content with a communication device that has a fixed IP address, and a mobile user may access the service from different locations or during moving. Mobile push notification service providers have to collaborate with networking service providers like ISPs and mobile communication carriers to address mobility issues. There are two scenarios of the mobile usage [14]: (1). A mobile user with an app that subscribed the service moves from one location to the other and the app is connected with different IP addresses before and after the moving, and the app is offline during the moving. As soon as the mobile user is connected to the network in the new location, the app communicates with the networking service provider to locate the app, which is transparent to the mobile push notification services like GCM. (2). During the moving period of time, the app is still online. The connection between the app and the network is supported by the networking providers like mobile phone carriers that resolve the location at run time. Mobile push notification services just deliver messages to registered carriers. Therefore, the subscriber and publisher in the model defined in Fig. 2 have to be connected dynamically at run time to support the two mobile scenarios. We add a dynamically configured connector in the PrT net model to bind a mobile user to its service providers at run time.

As defined in [19], a connector is a PrT net that connects with other PrT nets only through its input and output places. A connector serves as a module for defining the connection policies among nets. We split the PrT net

model in Fig. 2 into 2 parts: a publish net and a subscribe net. The two nets are connected with a connector net through its input and output places. The connector net in the PrT net model is shown in Fig. 4, where the dash area is the connector. Transition *Route* in the connector is used to resolve locations at run time and connect a publisher and its subscribers, and it can be expanded into a PrT net depending on routing policies. Places *P10* and *P13* are input places for receiving data, and places *P11* and *P14* are output places for outputting data. Places *P7*, *P8* and *P9*, and transitions *T7*, *T8*, and *T9* are places and transitions added for splitting the publisher from the model as a separated net. Transitions *T10*, *T11*, and *T12* are added for the same purpose for splitting the subscriber from the model as a separated net. The publisher net and the subscriber net are connected with places *P10*, *P11*, *P13*, and *P14*, and all other parts are same as the model in Fig. 2. Through adding a connector into the publish-subscribe PrT net model, the mobility of subscribers and the dynamic connection between a subscriber and its publisher is appropriately modeled. The location of the moving subscriber is resolved by transition *Route*, and it is also responsible for connecting subscriber and its publisher according to its routing policies.

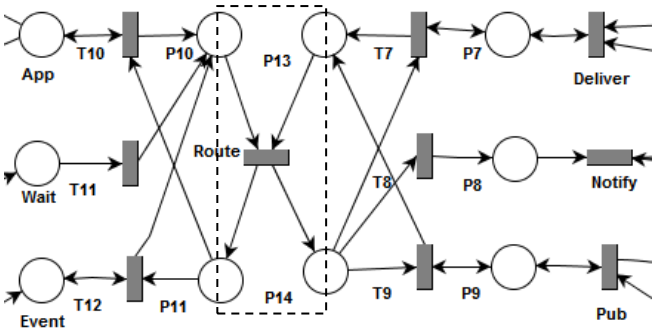


Figure 4. A connector net

3.3. Security

Both GCM and APNS support the communication between mobile users and service providers through secure communication protocols, and the authentication is implemented with secure tokens. In GCM, the *Sender ID*, *application ID*, *registration ID*, *sender auth token*, and *secure communication protocols* are used to ensure the security of the communication. Mobile push notification services such as GCM define the security policies and provide security protections. Therefore, we don't model the security protection policies and process in the model; instead we model the security threats using threat models. A threat model defines whether the service is protected from a security attack. As defined in [22], a threat model is PrT net with one or more transitions that represent a security attack or security vulnerability. Fig. 5 models a possible attack with a threat model for the model defined in Fig. 2. The threat model models a security scenario that an application server (*i.e.* attacker) forges itself as another valid application server (*i.e.* victim) to send notification messages to victim apps, which have subscribed the victim server's messages but not the attacker's. The attacker gets the *Sender ID* of the victim server through reverse engineering of a victim app, and steals the *sender Auth token* from the victim server. The attacker is possible to intercept the *sender Auth token* if the victim server and the GCM server communicates via a non-secure communication protocol. As soon as the attacker gets the *Sender ID* and *sender Auth token* from the victim, it uses them to send messages to GCM server. The GCM sever thinks the messages are sent from the victim server and will deliver the message to the victim apps. In Fig. 5, the dash area is a threat model for modeling the potential attack we just discussed. The threat model is connected to the publish-subscribe model via places *App* and *Sender*, and transition *Pub*, and all others parts are still the same. Transition *ReverseAttack* represents the reverse engineering process for finding the *Sender ID*; *StealToken* defines the action for stealing *Sender Auth tokens* from the victim *Sender*; *MsgAttack* is for fabricating forgery messages; places *SendID*, *FakeMsg* and *Token* represent *Sender ID*s, forgery messages and *Sender Auth tokens*, respectively. For any attack, we can create a threat model using the same idea.

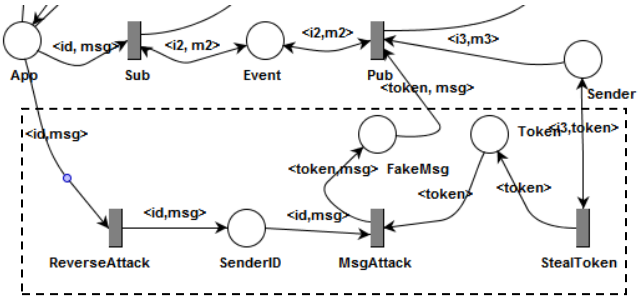


Figure 5. A threat model in PrT net

3.4. Analysis

As soon as the PrT net model of mobile push notification services are built, rigorous analysis of interesting properties of the services can be conducted. The analysis results can help developers to correct and update the model. The PrT models will serves as a design for building systems with the service. In this research, one can conduct simulation and testing of the model during the model development process since PrT nets are executable. Model checking and formal verification also can be conducted on the PrT net model since PrT nets are a formal language. Tool MISTA supports testing and simulation of PrT nets as well as model checking of simple properties. The analysis process of PrT nets using MISTA can be summarized as follows:

1. Compile the model and simulate the execution of the model with initial markings during the model development process to check important execution

sequences and results, look for and correct errors in the model.

2. Define goal states and verify the reachability of the goal states from the giving initial states. States (*i.e.* markings) in PrT nets are defined by the distribution of tokens in places. Verify the reachability of all transitions from the initial states in the model, and carefully check the transitions that are not reachable. Check for deadlock and terminate states in the model, which is a state under which no transition is fireable. Find reasons that cause the deadlock or termination. Define interested assertions for the model, and verify whether or not each assertion holds at any reachable state [21].
3. Select a test coverage criterion, programming language and test engine for generating tests. MISTA tool can generate adequate tests for the selected criterion. Examples of the criterion include transition coverage, state coverage, goal coverage and others. MISTA supports Java, C#, C++ and other programming languages, and test engine like JUni and Window Tester. As soon as the model level tests are generated, one can define the mapping between the objects of the PrT net model and the objects of its corresponding implementation, and then convert the model level tests into the implementation level tests using MISTA.

Simulating the execution of a particular sequence of events in the PrT net model is helpful to understand the model and find and correct defects in the model. Fig. 3 is a screenshot of the execution, where the transitions in red are enabled at the current marking. Users can select the next transition to fire or let system to randomly pick one, and the outputs of the execution and the firing sequence of transitions are useful for checking whether a use case scenario is appropriately implemented in the model. Checking the deadlock and termination states and verification of the reachability of goal states, all transitions, and assertions are important to ensure the quality of the model. They provide a more comprehensive and rigorous checking of the correctness of complex models than the simulation does. Test generation is the most important task in the analysis because generating implementation level tests to adequately test the implementation is the main purpose to create it. The purpose of simulation and verification of the model is to create a correct model, and one of the purposes of building a correct model is to drive the generation of high quality of tests for testing the implementation. This type of testing technique called model based testing greatly improves the effectiveness and performance of software testing.

Each security attack or vulnerability is modeled as a threat model to be integrated into the original model. Simulation and verification of the model is helpful to check whether the attack can occur or not and evaluate the possibility of the occurrence. The threat model is also used to produce tests for testing security issues in the model and the corresponding implementation. Testing with threat models is an effective way for security testing because regular models usually model the desired behaviors, which are not useful for generating tests to test undesired behaviors such as security attacks. Threat models specify undesired behaviors like security attacks explicitly, which are used to generate corresponding tests for testing specific security attacks. Using the security attack discussed in section 3.3 as an example, we can check the possibility of the attack using simulation of the threat model. Under the simulation of a given marking, if an attack sequence is fireable, then it means the attack could occur. Fig. 6 is a screenshot of simulating the threat model defined in Fig. 5. Based on the simulation result, one can find that the attack could occur in the model. However, the attack could only happen based on the assumption that the *sender auth token* could be intercepted.

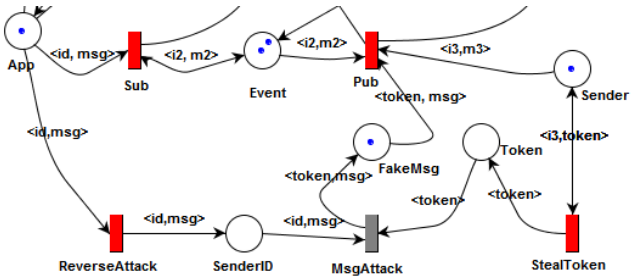


Figure 6. An execution of the threat model in Fig. 5

In order to generate tests for testing the attack, we define an execution sequence of events (*i.e.* transitions) that represent an attack, and then generate tests to cover the given sequence using MISTA. For example, an attack sequence of the threat model in Fig. 5 is: *ReverseAttack*, *StealToken*, *MsgAttack*, *Pub*. The tests generated for covering the sequence is partially shown in Fig. 7. The tests define a set of markings to enable the attack sequence.

```
public void test1() throws Exception {
    System.out.println("Test case 1");
    setUp();
    push-ps-secure-2.ReverseAttack();
    assert push-ps-secure-2.App("11", "s") : "1_1";
    assert push-ps-secure-2.Sender("1", "1") : "1_1";
    assert push-ps-secure-2.Event("1", "1") : "1_1";
    assert push-ps-secure-2.Event("1", "2") : "1_1";
    assert push-ps-secure-2.SenderID("11", "senderID") : "1_1";
    try {
        push-ps-secure-2.StealToken();
        System.out.println("Test failed at test 1_1_1: an expected exception is not thrown!");
        System.exit(1);
    } catch (Exception e) {}
    assert push-ps-secure-2.App("11", "s") : "1_1_1";
    assert push-ps-secure-2.Sender("1", "1") : "1_1_1";
    assert push-ps-secure-2.Event("1", "1") : "1_1_1";
    assert push-ps-secure-2.Event("1", "2") : "1_1_1";
    assert push-ps-secure-2.SenderID("11", "senderID") : "1_1_1";
```

Figure 7. An example of tests for testing attacks

4. Case Study: A Mobile Searching System

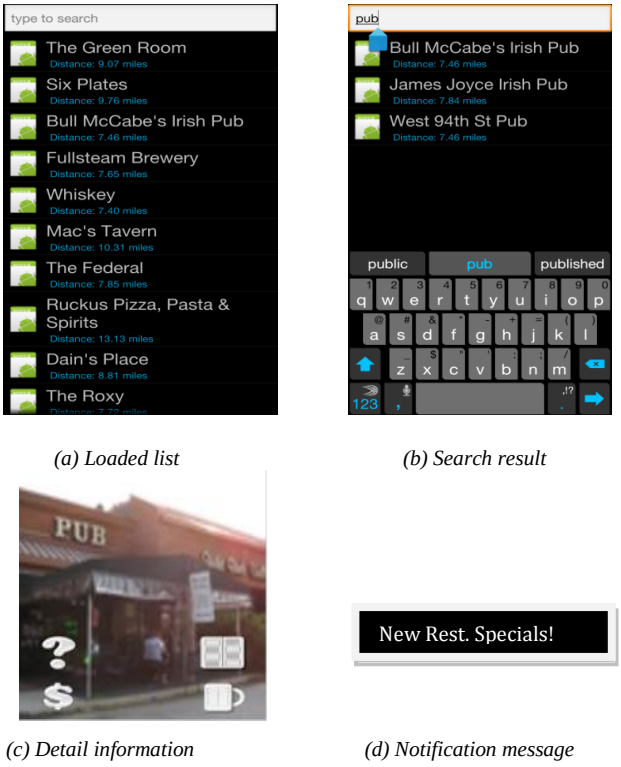
In this section, we conduct a case study of a mobile searching system that was implemented with mobile push notification services based on Amazon AWS and Amazon SNS.

4.1 Basic Functions

The mobile searching system we developed is called Hnite, which was originally designed for college students to search local night events such as specials for restaurants and live music events using their mobile devices. The system was extended later for general users to look for local business specials. The system includes two parts: a mobile application that is installed in Android mobile devices and a server program that was implemented with Amazon web services AWS. Amazon SNS and mobile push notifications were implemented in the system with GCM. Local business owners who have registered for Hnite service can enter business data to the server through the webpages linked to Hnite sever host in Amazon AWS, and they can update the information at any time. A mobile application user can set searching criteria such as favorites of food, sports, music or grocery. Each favorite may include some sub-categories. For example, food might include sub-categories like casual, exotic, and formal. The criteria also include emotion information such as boring, happy, and normal, and status information like busy and free of the user, and distance such as 5 miles, 10 miles for searching related information within the distance range. Other information like location and weather are sent to the application server automatically when a mobile user moves to a new location and is connected to the network. The push notifications are triggered by local events such as when a local business owner updates the business data, or the context of a mobile application user has been changed. For example, a user updated his or her searching criteria, moved to a different place or weather has suddenly changed. The notification messages sent to the application users are fairly brief and informational such as: *lunch specials*, *new pirate music*, *purple devil tickets for sale*. Hnite system limits the length of each message to 100 characters, which is long enough for normal mobile push notification messages. When a user responds to the message via clicking the message, information that is best matched to the notification and user searching criteria is automatically loaded to the mobile device from the server. Then the user can browse or search the loaded information. When a user browses or searches the information, more data might be loaded to the mobile device in the background according to some basic rules such as the search result from the current loaded data is empty, the user expands a top event category but the sub-categories data was not loaded yet or the users browses to the end of the list. Caching data in the mobile device improves searching performance and reliability so that the service is still available when the user is offline. The four screens in Fig. 8 show some basic functions of Hnite: (a) is a list of information pre-loaded in the mobile device according to user's searching criteria; (b) is a dynamically updated searching result, (c) is a detail view of a selected event, and (d) is a notification message.

4.2 Implementation of Mobile Push Notifications

The server program of Hnite is a regular web server that was implemented in Amazon AWS. Local business owners who registered with Hnite's service can enter data to the server via the web pages, and mobile application users of Hnite access the server through the mobile application in their Android devices. Mobile users can set up their favorites and search or browse online information in the server or cached data pre-loaded from the server to the device, and the mobile application can send location related data such as location and weather to the server automatically in the background. Fig. 9 shows the overall structure of system Hnite.



(a) Loaded list *(b) Search result*

(c) Detail information *(d) Notification message*

Figure 8. The searching and notification screens

In the following section, the mobile push notification service we discuss is based on GCM service since Hnite was implemented for Android devices. Before a notification message provider (*i.e.*, Hnite server) sends a notification message to mobile users, it has to send its notification service credentials (*i.e, sender ID*) to Amazon SNS and be

authorized. A mobile application installed in a mobile device needs to register itself in GCM and Amazon SNS before it can receive push notifications. When one registers his or her mobile application in GCM, the GCM server generates a *registration ID* for the device (actually it is for the application, we do not distinguish them as soon as they do not cause confusion). Then the *registration ID* is sent to Amazon SNS for binding the application and its notification provider [1]. The major event sequence for sending a push notification message in Hnite is described as follows:

1. When a business owner updates information in Hnite server, the server checks the updated information and the favorite setting of each mobile application user to decide whether a notification message is needed to be sent out.
2. If a notification message is needed to be sent to one or more mobile users, Hnite server builds a notification message according to the properties of the updated information and a list of *registration IDs* of registered users according to their favorite settings, and sends the message and the user list to Amazon SNS.
3. Amazon SNS checks the message and the list of *registration IDs*, and then sends them to a GCM server on behalf of Hnite server.
4. The GCM server sends the message to all users in the list. It also enqueues and stores the message in case a mobile device is offline. As soon as the mobile device is online, the GCM server sends the message to the device.
5. On the mobile device, the Android system broadcasts the notification to the specified Android application such as Hnite.
6. Users can chose to response the notification or ignore it.

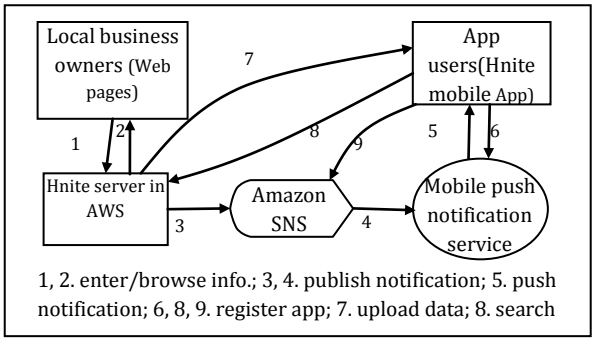


Figure 9. The overall structure of Hnite

When a mobile user of Hnite moves to a new location or updates his or her favorites setting, the updated information is automatically sent to Hnite server directly, which may trigger a push notification message to be sent to the user. The sequence of the events is same as above discussion, but the message is sent to just one user instead of multiple users. In addition, business owners and mobile users can browse contents in Hnite via browsers or Hnite mobile application. Partial data from the server is preloaded or updated into the mobile devices according to favorite setting periodically in the background to improve the searching performance or in case network connection is not available. More data might be automatically loaded when a mobile user browses or searches the preloaded data. Loaded data in the devices can be cleaned or replaced manually or automatically.

4.3 A PrT Nets Model

In this section, we discuss how to model the mobile push notification service in Hnite system using PrT nets. The PrT nets model provides a solid foundation for rigorously analyzing the design and implementation of mobile push notification services. The mobile push notification service includes three parts: a component for connecting Hnite server with Amazon SNS, a component for connecting Amazon SNS and GCM server, and a component for connecting GCM server and mobile devices. Under the guiding of the publish-subscribe model discussed in section 3.1, we model Hnite in publish-subscribe style with more details. Hnite server, Amazon SNS server and GCM server form a publisher, and mobile clients are the subscribers. The registration of mobile clients, the authentication of senders, message generation and delivering are all modelled. Connector discussed in section 3.2 can be easily added to the model to model the mobility of mobile clients, and threat models discussed in section 3.3 can be added to the model for checking security attacks and vulnerabilities.

Fig. 10 is a PrT net model of the mobile push notification service in Hnite. We first consider the case of message generation: as soon as a business owner updated the information in the server, a notification of the change is sent to all registered mobile users. In Fig. 10, place *BusinessOwner* represents the business owner, three parameters of the place are the *ID* of the business owner, the message information, and an action. For example, (*101, "m", "dir*") means a business owner with ID 101 updated information in the server with new information *m*, and a notification about the change is sent to all registered users. A business owner enters (modeled with transition *Enter*) the data into the server via a web browser, and the data is added (represented by transition *AddData*) into Hnite server, which is defined using place *Server*. The server sends its *sender ID* to Amazon SNS server and be authorized. Place *APIKey* has the project key of Hnite server, and it is used to generate *senderID* (represented by place *SenderID*) for the server. The server registers (using transition *RegSender*) the *senderID* in Amazon SNS sever (defined by place *SNS*). Before a notification message is sent out, the *SenderID* has to be approved (by transition *Approve*) and authenticated (by transition *Authenticate*), and the function of sending message is enabled (defined by transition *Enable*). When the information in the server is updated, Hnite server generates a notification message (via transition *GenMsg*) and sends it to SNS. If the message is a topic message and the topic does not exist (handled by transition *Compare*), a new topic

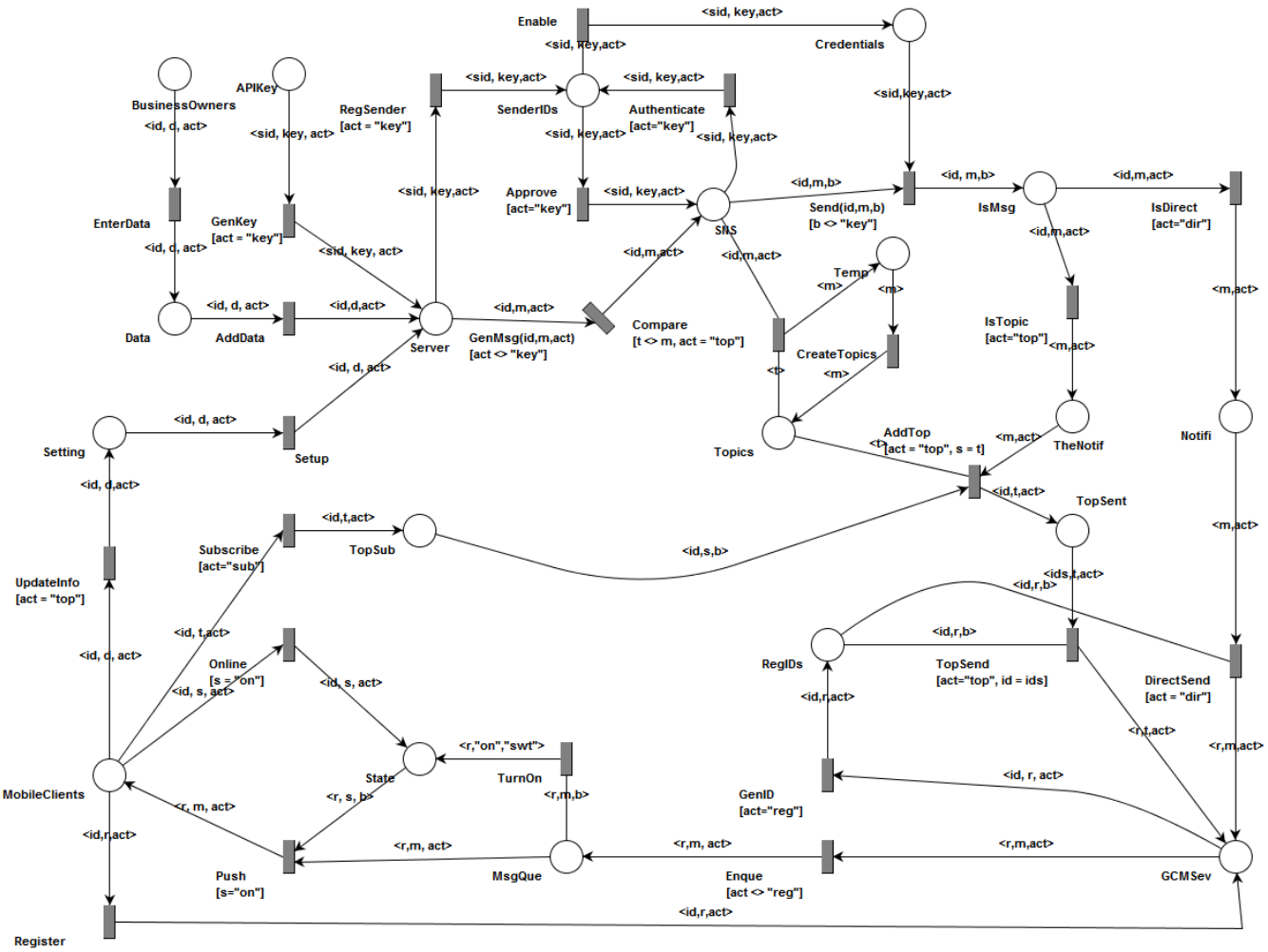


Figure 10. A PrT net model of Hnite

(represented by place *Topics*) is created. If the message is a direct message (represented by transition *IsDirect*), it will be sent to all registered users (via transition *DirectSend*) by GCM server; if it is a topic message (defined by transition *IsTopic*), it is sent to the topic channel (defined using transition *AddTop*), and then it will be only sent to users who subscribed it (defined by transition *TopSend* with guard condition *id = ids* to ensure that only subscribed users will receive the message). GCM server (represented by place *GCM*) sends the notification message it received from SNS to mobile devices based on their *registration IDs* (defined using place *RegIDs*). The notification message is sent to a buffer queue (represented by place *MsgQue*), then if the target mobile device is online, the message is pushed to the device; otherwise, it waits to be sent out until the device is online (represented by transition *TurnOn*). A mobile user (defined by place *MobileClients*) can update (defined by transition *UpdateInfo*) his or her favorites, subscribe (via transition *Subscribe*) a topic, and register itself (via transition *Register*) and get a registration ID with GCM.

We first executed the PrT net model with selected initial markings and checked different execution scenarios until we couldn't find any error. Then we moved ahead to analyze the model and update the model if any error is found during the analysis.

4.4. Analysis

As soon as the PrT net model of the mobile push notification service has been built and successfully compiled, we start to analyze the model for interested properties using tool MISTA. First, the model is executed with random inputs (*i.e.*, initial markings) to help developers understand the model and detect easily found problems. If the simulation result is acceptable, we start the verification of the reachability of goal states, assertions and deadlock states. After that, adequate tests are generated according to selected testing coverage criteria, and finally selected tests are performed to test the model, and the corresponding program level tests generated from the model level tests could be used for testing the implementation of Hnite system. Here

we discuss the analysis process through analyzing the PrT net model in Fig. 10.

First, execute the PrT net model with some valid initial markings using MISTA. For example, we assign some initial markings for the PrT net model defined in Fig. 10. Considering we need test several functions in the model such as the business owners update the information in Hnite server, and it should trigger a notification message be sent to all registered users or subscribed users. Therefore, we give an initial marking for place *BusinessOwner* as (101, "*m*", *dir*), where 101 is the *ID*, "*m*" is the information, and "*dir*" means the information is sent to all registered users. Obviously, "*m*" is only a dummy representation of real data using in the system. For similar consideration, we assigned place *MobileClients* as (1, "*t*", "*top*"), (1, "*t*", "*sub*"), (1, "*x*", "*reg*"), (1, "*on*", "*swt*") to represent an application user updates his or her context information, subscribes a topic, registers the mobile device, and connects it to the network, respectively. Place *APIKey* as (1, 12345, "*key*"), and *Topics* as ("*blk*") mean sending *SenderID* of Hnite server and creating a blank topic, respectively. We ran the model with the initial markings, tracked the execution sequences, and updated the model based on execution results. Finally, we developed the model in Fig. 10.

Second, verify the reachability of goal states, assertions and deadlock states using MISTA. No deadlock or terminate state was found in the PrT net model in Fig. 10, and all transitions are reachable. We defined and checked several important goals as follows:

1. GOAL("*x*", "*m*", "*dir*"), when a business owner updates information in Hnite server, a mobile client will receive a notification message.
2. GOAL("*x*", "*t*", "*top*"), when a mobile client updates his or her favorite, a notification is pushed to the client.

where "*x*" represents the *registration ID* of the mobile client, "*t*" or "*m*" represents a notification message, and "*top*" or "*dir*" means the notification is a subscribed message or a direct message. The analysis found the two goals were all reachable. For checking an undesired goal such as a security attack, one needs to check the negation of the goal to show that the model is safe.

Third, generate adequate tests according to selected test coverage criteria using MISTA. For the PrT net model defined in Fig. 10, we generated model level tests covering all states, all executable paths, and all goal states. We also generated some random tests. Complex execution scenarios can be rigorously tested in model level thanks to the executable capacity of PrT nets. The model level tests are also used for generating program level tests via mapping the model to its corresponding program. The program level tests will be used for testing the implementation of the model. Fig. 11 is a snapshot of the generated tests covering the reachability tree in the PrT net model in Fig. 10, and the number of tests generated for covering the reachability tree is 30849, and they cover total 7712 states in the model. It is not necessary to test all tests for covering reachability tree, so we may manually select some of the tests for the testing. Only 4 tests were needed for adequately testing all transitions, and 1968 tests were generated by MISTA for adequately testing for all states. As soon as tests are available, test engines such as Junit can be used together with MISTA to automate the testing process.

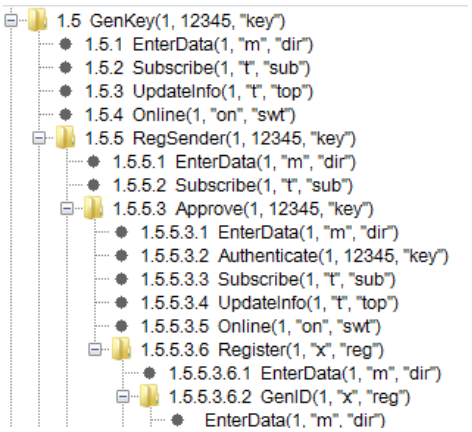


Figure 11. A snapshot of generated tests for the reachability tree of the model defined in Fig. 10

```
public class push-notification-v1Tester_RT{

    private push-notification-v1 push-notification-v1;

    protected void setUp()  throws Exception {
        push-notification-v1 = new push-notification-v1();
    }

    public void test1() throws Exception {
        System.out.println("Test case 1");
        setUp();
        push-notification-v1.EnterData(1, "m", "dir");
        assert push-notification-v1.Topics("blk") : "1_1";
        assert push-notification-v1.MobileClients(1, "on", "swt") : "1_1";
        assert push-notification-v1.MobileClients(1, "t", "sub") : "1_1";
        assert push-notification-v1.MobileClients(1, "t", "top") : "1_1";
        assert push-notification-v1.MobileClients(1, "x", "reg") : "1_1";
        assert push-notification-v1.APIKey(1, 12345, "key") : "1_1";
        assert push-notification-v1.Data(1, "m", "dir") : "1_1";
        push-notification-v1.Subscribe(1, "t", "sub");
        assert push-notification-v1.Topics("blk") : "1_1_1";
```

Figure 12, A snapshot of generated tests in Java

The tests generated in the model level mainly are used for generating program level tests to rigorously test the implementation such as Java programs of the system. The adequate tests generated in the model level are converted to program level tests for testing the implementation to ensure the rigorousness of the test in program level. The tests can be used for testing the programs as soon as the programs have been implemented following the model, but help files are needed to map the differences between the model and its

implementation. Fig. 12 is a Java test generated from the model level tests partially shown in Fig. 11.

Through rigorously analyzing the PrT net model, and adequately testing its corresponding programs, we have the confidence that the quality of the design and implementation of the mobile push notification is ensured.

5. RELATED WORK

Mobile push notifications are an important feature of mobile applications like those installed in smart phones. It has been implemented in many applications like gaming, financing, sports and emergency systems. However, most mobile push notifications have to implemented based on complex third-party notification services such Apple APNS and Google GCM. It also needs implement complex security authentications and authorization among different parties such as the registration of application servers and mobile clients. It is also a grand challenge topic in mobile push notifications to deal with the mobility issue in the system such as a mobile device may move from one location to another, and a mobile device may join or leave a computing task randomly. For example, a group of players are playing online game, but one of them may pause the playing for a while and then join the game later. Research on mobile push notifications was reported in [14], which discussed the service architecture and different mobile scenarios. However, research on formally modeling and analyzing mobile push notification services is rare.

Formal modeling of software systems provides a solid foundation for designing and analyzing a software system to ensure the quality of software development. A variety of work on modeling and analyzing the mobility in mobile computing systems or services has been reported. The logic agent mobility (LAM) discussed in [19]was modeled using two-layer PrT nets, where the two-layer PrT nets introduced connectors to facilitate the communication between nets. The two-layer PrT nets have the same semantics of PrT nets for each net. Although it is fine to model mobile push notifications using LAM, analyzing LAM model is a grand challenge due to the static definition of the connectors in the two-layer PrT nets, and the complexity of composing several PrT nets in different layers using connectors. We used the connector in this research same as the one in LAM, but PrT nets used in this research is regular PrT nets so that the model can be easily analyzed using MISTA. Reference nets [11] have been used for modeling different types of mobility in [9] through using "nets within nets" style [10]. The 'nets within nets" reference nets are natural for modeling mobile push notifications since a mobile computing system can be modelled as a system net and the mobile push notification can be modeled as a token net within the system net. However, analyzing the net needs a new tool that is not available and the semantics of reference nets are different to traditional Petri nets. Therefore, the complexity of formal analysis of reference nets comparing to regular Petri nets is great increased. CPrT nets are PrT nets extended with dynamic channels for synchronous communications and the "nets within nets" style that was introduced in [3]. CPrT nets can model mobile computing systems with concise and easily understandable models. It is able to nicely model the dynamic configuration of software architecture of mobile computing systems. However, an automated analysis tool for CPrT nets does not exist. π-calculus [12] is the first language offering features for formally modeling mobile computing systems via specifying process movement across channels. "π-Calculus is a way of describing and analyzing systems consisting of agents which interacts among each other, and whose configuration or neighborhood is continually changing" [12]. Comparing to π-Calculus, PrT nets are more intuitive for modeling mobile computing systems thanks for its graphic notation. It is also easy to simulate the execution of the net model due to its executable ability. In this paper, PrT nets are used for modeling the mobility in mobile push notifications with a dynamically configured connector.

Many research results on modeling and analyzing software security have been reported. Xu et al. [20] proposed a threat-driven modeling and verification of secure software using Aspect-oriented Petri nets, and they also proposed an approach for automated security test generation with formal models [22]. We adopted the threat model proposed by Xu et al. [22] to model security attacks and vulnerabilities in mobile push notification services, and used MISTA to generate security tests following the same idea discussed in [22].

One of the purposes of formally modeling software systems is to build a foundation for formal analysis. Although formal analysis provides a rigorous way for verifying correctness of software systems, it is infeasible for analyzing large software systems due to the state explosion issue. In [5], He and *et. al*. reported a method for formally analyzing Petri nets using model checking and other formal proof techniques. Ding and He discussed an approach for model checking a mobile computing system in an extended version of PrT nets [3]. Several other researchers also explored the technique for testing models and designs of software systems [15]. Test generation is the most important task in software testing. The main task of the analysis discussed in this paper is generating adequate tests based on PrT net models. There are mainly four types of test generation: program-based, specification-based, model-based and random test [16]. Program-based test generation falls into two categories: static methods and dynamic methods. Static methods generate test by analyzing the code without running the program, such as symbolic execution [8] and static single assignment form [6]. Dynamic methods produce test using heuristic analysis of runtime behaviors, such as dynamic symbolic execution [17]. Specification-based methods generate abstract test from formal specifications (*e.g.*, in UML [18] and Petri nets [21][23]) and then transform the model level test into implementation

level test. Radom test generation selects an arbitrary subset of all possible input values over the input space according to certain probabilistic distribution [4]. The test generation discussed in this paper belongs to specification-based test generation. In order to achieve the rigorous as well as practical of modeling and analysis, we model a system using formal language PrT nets and analyze the PrT models using a model based testing technique that tests models with supplementary of simulation and model checking. The analysis approach has both advantages of informal analysis like testing and formal analysis like model checking. Tests generated from the model not only are used for testing the model, but also are used for generating implementation level tests so that the consistency between a model and its implementation is well ensured, which is absent in most existing approaches [21]. The similarity of the techniques used in modeling and implementation phases is important for sharing analysis results in both phases to improve analysis effectiveness and efficiency.

6. SUMMARY

Mobile push notifications are an important feature of mobile applications. Due to its complexity, formally modeling the mobile push notification is necessary to understand its design and implementation, and rigorously analyzing it is required for building a correct mobile system with mobile push notifications. In this paper, we introduced an approach for modeling and analyzing mobile push notification services in PrT nets and discussed how to model the mobility with a dynamically configured connector and security with threat models. We conducted a case study of modeling the mobile searching system particularly the mobile push notification services using PrT nets, and analyzed the PrT net model using tool MISTA. The case study has shown the effectiveness of the modeling and analysis approach we proposed in this research, and the model we developed is also useful for others who are interested in mobile push notifications. In the future, our research focus is on modeling and analyzing more security attacks and vulnerabilities in the mobile push notification services.

7. ACKNOWLEDGMENT

We thank Zachary Dain for the implementation of partial of the mobile searching system, and Dr. Dianxiang Xu for offering tool MISTA. This research is supported in part by award #CNS-1262933 from the National Science Foundation. Junhua Ding's research was also partially supported by the guest professorship grant from school of computer sciences at China University of Geosciences.

8. REFERENCES

[1] Amazon.com, Amazon Simple Notification Service, *Developer Guide*, API Version 2010-03-31, URL: http://docs.aws.amazon.com/sns/latest/dg/welcome.html, last accessed on Dec.. 8th, 2014.

[2] Android.com, Google Cloud Messaging, http://developer.android.com/google/gcm/gcm.html, last accessed on Dec. 8th, 2014.

[3] Ding, J., He, X., Formal Specification and Analysis of an Agent-Based Medical Image Processing System, *Intl. Journal of Soft. Eng. and Knowledge Eng.*, Vol. 20, No. 3, pp. 1 – 35, 2010.

[4] Duran, J. W., Ntafos, S. C., An Evaluation of Random Testing, *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 4, pp438-443, July 1984.

[5] Garlan, D., Khersonsky, S., Kim, J.S.: Model checking Publish-Subscribe systems. *Model Checking Software*, *LNCS* vol. 2648, pp. 166-180, 166–180, 2003.

[6] Gotlieb, A., Botella, B., and Rueher, M., Automatic Test Data Generation Using Constraint Solving Techniques. In *ACM Int. Symp. on Soft. Testing and Analysis (ISSTA). Soft. Eng. Notes*,23(2):53-62, 1998.

[7] He, X., Yu, H., Shi, T., Ding, J., and Deng, Y., Formally Specifying and Analyzing Software Architectural Specifications Using SAM. *Journal of Systems and Software*, vol.71, no.1-2, pp.11-29, 2004.

[8] King, J. C., Symbolic execution and program testing, *Comm. of the ACM*, v.19 n.7, p.385-394, July 1976.

[9] Köhler-Bußmeier M., Moldt, D., and Rolke, H., Modeling mobility and mobile agents using nets within nets. In W.M.P. van der Aslst and E. Best, eds, *LNCS*, vol. 2769 (2003) pp.121-139.

[10] Köhler-Bußmeier M., A Survey of Elementary Object Systems, *3rd Intl. Workshop on Logics, Agents, and Mobility (LAM'10)*, vol. 7, pp. 19-36, 2012.

[11] Kummer, O., and Wienberg, F., Renew – the reference net workshop. http://www.renew.de, (1999), last accessed on March 2012.

[12] Milner, R., *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, (1999) New York.

[13] Murata, T., Petri Nets: Properties, Analysis and Applications. In *Proceedings of the IEEE*, vol.77, no.4, (1989) pp. 541-580.

[14] Podnar, I.; Hauswirth, M.; Jazayeri, M., Mobile push: delivering content to mobile users, *Proc. of 22nd International Conference on Distributed Computing Systems Workshops*, pp.563 -568, 2002.

[15] Richardson, D, Wolf, A.: Software testing at the architectural level. In: *Proc. of the 2nd Intl. Soft. Architecture Workshop*. pp. 68–71, 1996.

[16] Shan L., and Zhu, H., Generating Structually Complex Test Cases By Data Mutation: A Case Study of Testing An Automated Modelling Tool. *The Computer Journal*, Vol. 52, No. 5, 2009.

[17] Xie, T., Tillmann, N., Halleux, J. de, Schulte, W., Fitness-guided path exploration in dynamic symbolic execution. *IEEE/IFIP International Conference on Dependable Systems & Networks*, vol., no., pp.359-368, 2009.

[18] Xu, D., and Ding, J., Prioritizing State-Based Aspect Tests. *Proc. of ICST 2010*, pp. 265-274, Paris, France, 2010.

[19] Xu, D., Yin, J., Deng, Y. and Ding, J., A Formal Architecture Model for Logical Agent Mobility. *IEEE Trans. on Software Engineering*. vol. 29, no. 1 (2003), pp. 31-45.

[20] Xu, D., and Nygard, K. E., Threat-Driven modeling and verification of secure software using Aspect-oriented Petri nets, IEEE Trans. of software engineering, vol. 32., No. 4, pp. 265 -278, April 2006.

[21] Xu, D, A Tool for Automated Test Code Generation from High-Level Petri Nets. *32nd Int. Conf. on Apps. and Theory of Petri Nets* , Newcastle, UK, June 20-24, 2011.

[22] Xu, D.; Tu, M; Sanford, M.; Thomas, L.; Woodraska, D.; Xu, W., Automated Security Test Generation with Formal Threat Models, *IEEE Transactions on Dependable and Secure Computing*, vol.9, no.4, pp.526 - 540, July-Aug. 2012.

[23] Zhu, H., and He, X., A methodology of testing high-level petri nets. *Journal of Information and Software Technology*. v44, pp. 473-489, 2002.

Authors

Junhua Ding is an associate professor of computer science with East Carolina University (ECU). He received BS, MS and PhD, all in computer science in 1994, 1997, and 2004, respectively. Prior he joined ECU at 2007, he had worked as a software engineer and project manager with medical companies for 8 years. His research interests are software design and analysis, software testing, Petri nets, and cytometry. He has published 60 peer reviewed conference proceeding and journal papers. He is a member of ACM and IEEE.

Wei Song received the PhD degree in Computer Software and Theory from Nanjing University, China, in 2010. Currently, he is an associate professor at Nanjing University of Science and Technology, China. His main research interests are focused on services computing and software engineering, including software evolution, service composition, business process management, and formal methods like Petri nets. He has published more than 20 research papers in journals and international conferences such as ICWS, SCC, APSCC, etc. He is a member of ACM and a senior member of CCF.

Dongmei Zhang is a professor and chair of department of computer applications at China University of Geosciences (Wuhan). She received BS, MS and PhD. in 1994, 1999 and 2007, respectively. Her research interests are data mining, machine learning, and evolutionary computation.