UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Aljaž Čukajne

MOBILNA APLIKACIJA ZA SKRBNIKE ODPRTOKODNEGA PRODUKTA ZA NADZOR OMREŽIJ

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Rok Rupnik

Ljubljana, 2011

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00132/2011
Datum: 02.09.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ALJAŽ ČUKAJNE**

Naslov: **MOBILNA APLIKACIJA ZA SKRBNIKE ODPRTOKODNEGA PRODUKTA ZA NADZOR OMREŽIJ**

MOBILE APPLICATION FOR THE SUPPORT OF OPENSOURCE PRODUCT FOR NETWORK MANAGEMENT

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Za računalniška omrežja, kjer skrbniki omrežij za upravljanje omrežja uporabljajo odprtokodni produkt Zabbix zasnujte mobilno aplikacijo za skrbnike omrežij, ki jim bo omogočala podporo upravljanju omrežja v stanju mobilnosti. Pri tem uporabite aplikacijski vmesnik produkta Zabbix in poskrbite za najvišji možen nivo varnosti. Mobilna aplikacija naj bo razvita na platformi Android.

Mentor:

doc. dr. Rok Rupnik

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani **Aljaž Čukajne**,
z vpisno številko **63070064**,

sem avtor diplomskega dela z naslovom:

Mobilna aplikacija za skrbnike odprtokodnega produkta za nadzor omrežij

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  doc. dr. Roka Rupnika

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 16.9.2011 Podpis avtorja: _____

Zahvala

Za pomoč in nasvete pri izdelavi diplomske naloge se zahvaljujem mentorju
doc. dr. Roku Rupniku.

Zahvaljujem se svojim staršem, ki so mi omogočili študij in mi vedno stali ob strani.

Zahvala gre tudi vsem sošolcem in prijateljem, za vso pomoč in moralno podporo.

KAZALO

POVZETEK

Diplomsko delo predstavlja razvoj aplikacije za mobilno platformo Android, ki je namenjena upravljanju omrežja s podporo nadzorovanja različnih mrežnih storitev, strežnikov ter ostale mrežne strojne opreme. Za dostop do podatkov aplikacija uporablja obstoječi programski vmesnik odprtokodne programske rešitve Zabbix, ki je v delu opisana in predstavljena kot ena izmed mnogih programskih rešitev za nadzorovanje. Pri opisu aplikacije smo razdelali tudi razvojno ogrodje Android SDK in njegove osnovne elemente, ki v kombinaciji z razvijalskim orodjem Eclipse tvorijo močno okolje za razvoj aplikacij.

Kot končni izdelek se aplikacija uporablja za spremljanje stanja računalnikov v računskem centru ali omrežju, zbiranju kazalnikov delovanja računskih virov v različnih časovnih okvirih ter opozarjanju na izpade ali morebitne težave pri njihovem delovanju. Pri načrtovanju aplikacije smo se osredotočili za prikaz tistih informacij, ki so za uporabnika mobilne ali druge naprave najbolj uporabne. Aplikacija tako omogoča pregled trenutnega stanja odjemalcev skupaj s morebitnimi napakami, ki se prožijo ob izrednih dogodkih, pregled detajlnih lastnosti odjemalcev ter prikaz generiranih grafov za različne kategorije v izbranem časovnem obdobju.

Ključne besede

Android, Zabbix, programski vmesnik, Eclipse.

ABSTRACT

The thesis describes the development of an application designed for the Android platform, whose purpose is to support network management and monitoring of various network services, servers and other network hardware. The application is using the existing Zabbix application programmable interface for accessing data, which is described and presented in this thesis as one of the many software solutions for network managing. Supporting the description of the application is a representation of the Android SDK and its basic features, which, in combination with the Eclipse development tool, forms a powerful environment for application development.

As a final product, the application is used for monitoring computers in a computer center or network, collecting indicators of computational resources in various time frames, and pointing out failures or shortfalls of their performance. When designing the application, we focused on presenting the information, which is most relevant for users of mobile phones and other devices. The main features of the application are that they provide an overview of the current state of hosts along with any possible errors, which are triggered in case of extraordinary, a detailed overview of host properties, and a display of generated graphs for various categories within a selected time period.

Keywords

Android, Zabbix, application programming interface, Eclipse.

1. UVOD

1.1. Ideja

V zadnjih 50 letih je internet spremenil način medsebojne komunikacije bolj kot katerikoli medij poprej. Danes je tako že povsem samoumevno, da imamo internetno povezavo na voljo kjerkoli in kadarkoli, govorimo lahko celo o vseprisotnem računalništvu (angl. *ubiquitous computing*). To dejstvo se poruši šele, ko zaradi neznanega razloga ne moremo dostopati do naše spletne lokacije ali strežnika ali celo do naše internetne povezave. Takrat naenkrat postanemo nedostopni in izolirani od informacij, ki so v našem interesu. Vedeti moramo, da je takšen scenarij dandanes še vedno zelo pogost in tako za uporabnika, kot tudi za administratorja oz. vzdrževalca omrežja ali spletnega strežnika, zelo stresen.

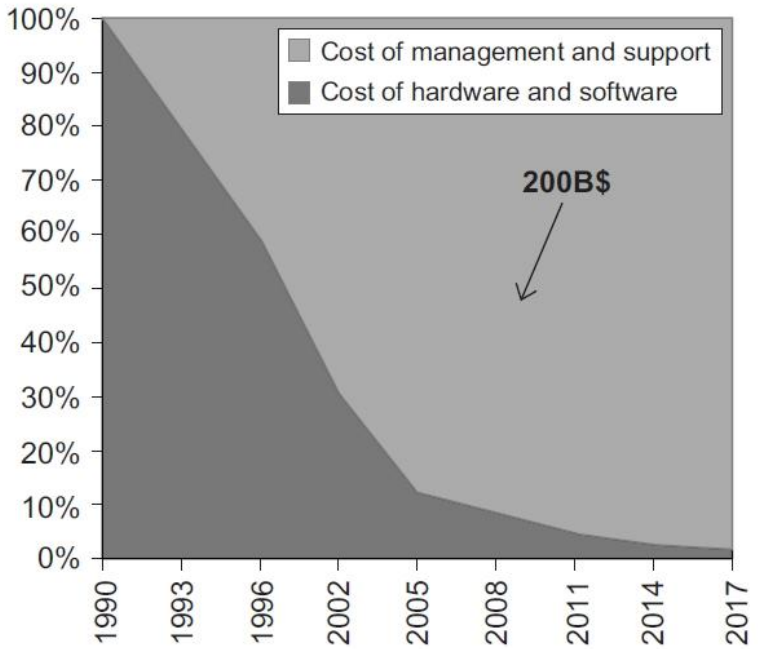
Da bi se izognili podobnim situacijam, imamo danes na voljo številna napredna orodja, ki nadzorujejo in upravljajo z omrežji. Poznamo veliko komercialnih in odprtokodnih orodij za nadzorovanje računalniškega omrežja, katerih izbira je pogojena z željami in potrebami posameznika, podjetja ali druge institucije. V kolikor lahko sploh upravičimo vse stroške za nabavo in postavitev sistema komercialnih orodij, v nasprotnem primeru se je potrebno ozreti po odprtokodnih rešitvah, ki ponujajo primerljive lastnosti in zanesljivost delovanja ter tehnično podporo.

1.2. Cilj diplomske naloge

Cilj diplomske naloge je bil ustvariti enostavno, pregledno in učinkovito aplikacijo za podporo programski rešitvi Zabbix, ki bi delovala na mobilnih telefonih in drugih napravah z operacijskim sistemom Android. Takšne naprave so dovolj majhne, a hkrati dovolj zmogljive, da ponudijo večino funkcionalnosti, ki smo jih vajeni pri klasičnih računalnikih. Pri tem ima glavno vlogo operacijski sistem, ki ponuja dovolj zmogljivo ogrodje za razvijanje aplikacij.

2. NADZOR OMREŽJA IN OMREŽNIH NAPRAV

Računalniško omrežje lahko definiramo kot sistem med seboj neodvisnih računalnikov, ki so v omrežje povezani zaradi izmenjave podatkov in delitev perifernih enot. Prva taka omrežja so se pojavila v sedemdesetih letih prejšnjega stoletja, bila so omejena s številom priključenih postaj in večinoma centralizirana, prenos podatkov je potekal asinhrono in relativno počasi. Vseeno je združitev računalnika in komunikacij imela ogromen vpliv na razvoj računalniških sistemov. S pomočjo hitrega razvoja na teh dveh področjih so se začele podirati stene velikih računalniških centrov, omrežja so se začela širiti in povezovati ter so postajala vsa kompleksnejša in hitrejša. Tudi napovedi za prihodnost predvidevajo še intenzivnejši napredek na področjih radijskih in podatkovnih komunikacij, protokolov, programskega inženiringa, aplikacij in vseh ostalih področij, ki se ukvarjajo s podatki in njihovo izmenjavo med omrežji. Govorimo lahko celo o preobilju tehnologij in standardov, zaradi česar imajo administratorji ter drugi operatorji, ki se s temi tehnologijami ukvarjajo, vse več dela z nepredvidenimi kompleksnimi zapleti in drugimi napakami na njihovih omrežjih. V primeru, da takšne napake povzročijo izpad omrežja za daljše obdobje, ima to lahko pogubne posledice za ugled podjetja ter njegove potencialne prihodke. Skozi proces evolucije računalniških omrežij od začetka do danes, je vse bolj na pomembnosti pridobivalo njihovo učinkovito upravljanje, kar se jasno vidi tudi na razmerju med stroški nabave strojne in programske opreme ter ceno njenega vzdrževanja tekom življenjskega cikla[1] (Slika 1).



Slika 1: Razmerje med stroški nabave strojne in programske opreme ter ceno vzdrževanja le-te po letih in v prihodnosti.

2.1. Definicija mrežnega upravljanja in nadzora

Statistike kažejo, da zelo redko prihaja do izpadov ali napak na neki omrežni napravi, ki prej ni kazala simptomov slabega delovanja ali sledi nekih izrednih dogodkov. In ker se napake ponavadi dogajajo takrat, ko nismo ustrezno pripravljeni na njihovo reševanje, se moramo obrniti na posebna programska orodja za nadzorovanje omrežja in omrežnih naprav. Takšna orodja omogočajo konstantno nadzorovanje dogodkov v omrežju ter priključenih napravah z uporabo različnih metod za obveščanje odgovorne osebe, če katera od meritev prekorači definirano pragovno vrednost.

Da bi razumeli aktivnosti mrežnega nadzora, si je najprej potrebno pogledati vse specifikacije upravljanja z omrežji. Po definiciji so to dejavnosti, metode in orodja, ki se nanašajo na delovanje, vzdrževanje in administracijo omrežnih sistemov. V želji, da bi standardizirali to področje, je bil leta 1980 ustvarjen standardni model za upravljanje telekomunikacijskih omrežij, ki služi tudi kot ISO standard FCAPS[2]. Termin FCAPS je akronim za angleške besede Fault, Configuration, Accouting (ponekod jo nadomešča beseda Administration), Performance, Security.

Standard za upravljanje omrežja je tako razdeljen na pet delov:

- **upravljanje napak** (angl. *Fault managment*) – obravnava detekcijo, lociranje, beleženje in obveščanje uporabnikov o napakah. Ker takšne napake lahko povzročijo izpad celotnega ali degradacijo dela omrežja, je v obvladovanje napak potrebno vložiti veliko truda;

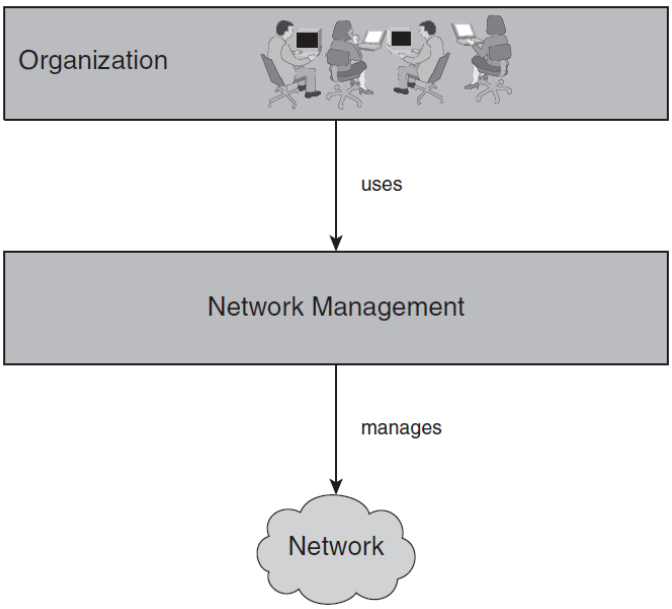
- **upravljanje nastavitev** (angl. *Configuration managment*) – obsega načrtovanje, inženiring in instalacijo omrežja z uporabo ustreznih nastavitev, ki omogočajo, da omrežje deluje kot utečena celota. Pomembno je tudi nadzorovanje omrežnih in sistemskih storitev ter vodenje evidence o opravljenih nastavitvah, da se ob morebitnem izpadu točno ve, kateri del nastavitev je potrebno ponovno opraviti;

- **upravljanje kalkulacij** (angl. *Accouting managment*) – združuje vse aktivnosti povezane s merjenjem in ovrednotenjem uporabe omrežja ali določenih omrežnih storitev, s strani posameznika ali skupine uporabnikov. Uporabljajo se za namene relugacije omrežja, zbiranja podatkov ter morebitnega zaračunavanjem rabe omrežja ali storitev;

- **upravljanje zmogljivosti** (angl. *Performance managment*) – omogoča upravljavcu, da s pomočjo zbiranja in analiziranja podatkov ugotovi učinkovitost trenutnega sistema in hkrati poda oceno njegove zmogljivosti v prihodnosti. Pri tem si pomaga z različnimi kazalci v omrežju, kot so prepustnost, odstotek izkoriščenosti, pogostost napak in odzivni časi posameznih področij;

- **upravljanje varnosti** (angl. Security managment) – proces kontrole dostopa do posameznih sredstev v omrežju. Dostop ureja ustrezna politika omrežja, ki določa skupine ali posameznike, ki imajo omogočen dostop do podatkov in sredstev, kar omogočajo postopki avtorizacije uporabnikov ter šifriranja podatkov. Hkrati obravnava tudi področja detekcije in preprečevanja vdorov v omrežja ter beleženja podatkov o morebitnih poskusih vdora.

Ob preučitvi zgornjega modela lahko hitro opazimo, da vsi vidiki upravljanja omrežja za svoje delovanje potrebujejo določene podatke o omrežju. Ker ima vsak vidik upravljanja različne zahteve po osveževanju podatkov, je zajem podatkov potrebno prilagoditi potrebam najzahtevnejšega. Da bi ugodili takim potrebam, uporabljamo različna orodja v podporo upravljavcem omrežja, ki jih imenujemo programska orodja za nadzor omrežja. Večina programskih paketov se sicer ne osredotoča samo na en referenčni model, saj so strukturirani na različne načine in zaradi različnih razlogov, vseeno pa pri večini velja dejstvo, da jih umestimo med uporabnika, ki je lahko del neke večje združbe ali organizacije, ter omrežje, ki ga uporabnik/organizacija uporablja (Slika 2).



Slika 2: Vloga in umestitev sistema za nadzor omrežja.

Takšna umestitev sicer ni nujno obvezujoča. Tudi ob nenadnemu izpadu sistema omrežnega nadzornika ali ene od njegovih storitev bi, odvisno od nastavitev, omrežje še vedno lahko nemoteno delovalo. To je le znak upravljavcu omrežja, da je sam odgovoren za opravljanje vseh nalog nadzorovanja in zbiranja podatkov, ki so bile prej dodeljene programu za nadzor.

Orodja za mrežni nadzor tako ponujajo veliko različnih funkcij, ki jih lahko razdelimo na:

- **zbiranje podatkov:** osnova vsakega orodja za nadzorovanje. Podatki se zajemajo in zbirajo preko agentov, namestniških strežnikov, s pomočjo protokolov za nadzorovanje ter ostalih virov;
- **obveščanje:** zbrani podatki se primerjajo z definiranimi pragovnimi vrednostmi. Če pride do odstopanja, so uporabniku poslana opozorila prek različnih medijev, kot so vizualna opozorila na zaslonu, elektronska pošta, SMS itd;
- **shranjevanje podatkov:** pridobljene podatke je treba redno shranjevati v dnevniške datoteke, ki se uporabljajo za morebitno kasnejšo obdelavo in arhiviranje;
- **vizualizacijo podatkov:** ljudje hitreje dojamejo in si lažje predstavljajo podatke v grafični obliki. Zbrane podatke se obdela in prikaže v obliki grafov, strukturnih krogov, zemljevidov, stolpičnih diagramov itd.

2.2. Predstavitev orodij za mrežni nadzor

Pri vpeljavi orodij za nadzor omrežja, mrežnih naprav in strežnikov ter orodij za merjenje zmogljivosti omrežja je treba izvesti obširno analizo trenutnega stanja našega omrežja. Analizirati je treba obstoječ promet omrežja ter predvideti njegovo maksimalno prepustnost, izpostaviti kritične elemente v omrežju, ki so najbolj izpostavljene izpadom, ter tiste, ki ob morebitnem izpadu lahko povzročijo največ škode v delovanju omrežja. Ker so orodja za nadzor omrežja v prvi vrsti namenjena sistemskim administratorjem in vzdrževalcem mrežne opreme, je pri načrtovanju vpeljave orodja potrebno upoštevati tudi njihovo mnenje in pretekle izkušnje z različnimi orodji za mrežni nadzor.

A še preden začnemo analizirati zbrane podatke o omrežju in njegovih napravah, se je potrebno odločiti, ali bomo poleg odprtokodnih testirali tudi komercialne rešitve. H končni odločitvi največ pripomorejo razpoložljiva sredstva, obstoječa znanja sistemskih administratorjev ter kompleksnost sistema. Razlike med komercialnimi in odprtokodnimi programskimi rešitvami so od izdelka do izdelka različne, a vseeno lahko izpostavimo nekaj osnovnih razlik.

Komercialno programje so vsa orodja in programi, ki jih podjetje razvije, z namenom ustvarjanja zaslužka. Večina komercialnih rešitev sicer nudi tudi časovno ali funkcionalno omejeno programje (angl. *Software*), ki je namenjeno predstavitvi programske rešitve in njenemu testiranju. Z njimi podjetje lažje primerja različne programske rešitve, preden se odloči vložiti sredstva v zahtevam najprimernejšo. Poleg tega komercialne programske rešitve, zaradi dodelane dokumentacije in dobre prodajne podpore, omogočajo hitrejšo namestitev in vzpostavitev okolja. Hkrati nudijo izobraževanja zaposlenih, kjer predstavijo vse funkcionalnosti aplikacije za nadzor ter udeležence seznanijo z njeno učinkovito rabo. Nekatera podjetja tudi sama dobavijo strojno opremo ali celo ponudijo gostovanja na njihovi stojni opremi. Iz tega lahko sklepamo, da so takšne rešitve primernejše za srednja velika in velika podjetja, ki želijo uvesti programsko rešitev za nadzorovanje, ki omogoča lastno neodvisno podporo za vzdrževanje in posodabljanje nadzornega sistema.

Največja prednost odprtokodnih orodij za nadzorovanje pa je ta, da se lahko uporabljajo popolnoma brezplačno in brez kakršnihkoli omejitev. Praviloma so dosti bolj odprta za modifikacije in omogočajo različne nestandardne razširitve preko vtičnikov (angl. *plugin*). Primerna so manjša in avtonomnejša podjetja, ki lahko sama poskrbijo za namestitev in vzdrževanje omrežnega nadzora. Pri nekaterih rešitvah dokumentacija temelji predvsem na prispevku skupnosti v obliki forumov in spletnih strani tipa »Wiki«, poleg teh pa nekatere ponujajo dobro dodelane dokumentacije in poleg pomoči uporabnikov nudijo tudi (plačljive) tečaje za zaposlene in seminarje.

Programski paketi za nadzor se tako formalno ločijo glede na naslednje funkcije:

- **samodejno odkrivanje** (angl. *Auto Discovery*) – aplikacija ima možnost samodejnega iskanja in odkrivanja novih odjemalcev ali omrežij, s katerimi se lahko poveže;
- **agenti** (angl. *Agent*) – produkt za nadzorovanje omogoča uporabo lastnih programskih agentov, ki tečejo na nadzorovani napravi in ob zahtevi vračajo rezultate poizvedb strežniku. Agenti olajšajo konfiguracijo nadzorovane naprave, a niso obvezni za uporabo. Nadzor preko procesov SNMP, ki tečejo v ozadju, ne šteje kot uporaba agenta;
- **uporaba protokola SNMP –** definira, ali aplikacija lahko prejema in procesira zahteve preko protokola SNMP;
- **spletna aplikacija za dostop** (angl. *WebApp*) – dostop do začelnega dela sistema je možen preko spletne aplikacije, ki omogoča različne nivoje interakcije z zalednim delom sistema:
    - ogled podatkov (angl. *Viewing*) – nadzorovani podatki in poročila o nadzoru so na voljo za ogled preko spletnega grafičnega vmesnika,
    - potrjevanje odločitev (angl. *Acknowledging*) – uporabniki lahko prek spletne aplikacije potrjujejo alarme in druga opozorila,
    - podpora poročanju (angl. *Reporting*) – uporabniki lahko uredijo določena poročila o ciljnem delovanju sistema in jih preko čelnega dela sistema izvedejo,
    - polni nadzor (angl. *Full Control*) – vsi aspekti aplikacije se lahko urejajo preko spletne aplikacije. To vključuje tudi vzdrževanje na najnižji stopnji, kot so temeljne konfiguracije aplikacije ter posodobitve sistema;
- **urejanje dostopa** (angl. *Access Control*) – urejanje pravic uporabnikov ter omejevanje dostopa do določenih segmentov aplikacije in njenih sistemov za določene uporabnike ali skupine uporabnikov;
- **uporaba vtičnikov** (angl. *Plugins*) – zmožnost aplikacije, da s pomočjo namestitve uradnih ali neuradnih vtičnikov omogoča posebne ali dodatne funkcije, ki jih aplikacija v osnovi drugače ne ponuja;
- **mreže** (angl. *Maps*) – grafično prikazovanje strežnikov in nadzorovanih naprav ter njihovega trenutnega stanja skupaj s povezavami med njimi;
- **porazdeljen nadzor** (angl. *Distributed Monitoring*) – zmožnost porazdelitve nadzorovanja na več strežnikov, razbremenitev centralnega nadzornega strežnika;
- **trendiranje** (angl. *Trending*) – aplikacija ponuja zmožnost analize trendov o podatkih, ki se pretakajo skozi omrežje v določenem preteklem časovnem obdobju;

- **predikcija trendiranja** (angl. *Trend Prediction*) – s pomočjo posebnih algoritmov lahko aplikacija, na osnovi preteklih trendov, napove gibanje trenda podatkov skozi omrežje;

- **metoda shranjevanja podatkov** (angl. *Data Storage Method*) – podatki se lahko shranjujejo na različne načine, večinoma se za shrambo uporabljajo različne podatkovne baze (MySQL, Oracle, PostgreSQL, IBM DB2, SQLite itd);

- **podpora protokolu IPv6 –** zahteva določa, ali aplikacija lahko nadzira računalnike in naprave, ki uporabljajo protokol IPv6, pridobiva podatke, naslovljene preko protokola IPv6, in teče na strežniku, ki podpira omenjeni protokol.

3. PROGRAMSKA REŠITEV ZABBIX

3.1. Splošno o programu

Zabbix je odprtokodna programska rešitev za sistemsko nadzorovanje in obveščanje. Ustvaril jo je Alexei Vladishev in leta 2001 izdal prvo verzijo, ki je bila, tako kot vse dosedanje verzije, izdana pod licenco za prosto programiranje GNU GPL (angl. *General Public Licence*). Zaledni del programa je spisan v programskem jeziku C, medtem ko je spletni čelni del sistema spisan v jeziku PHP. Eno leto po izdaji prve uradne stabilne verzije programa z oznako 1.0 leta 2005 je bila v Rigi ustanovljena organizacija Zabbix SIA.. Podobno kot ostala podjetja, ki izdajajo odprtokodne produkte, se organizacija poleg aktivnega razvijanja programa osredotoča predvsem na zagotavljanje poslovnih storitev. Te vključujejo planiranje sistema, ki mora zagotoviti vse potrebe naročnika, integracijo končnega sistema s poljubnimi zahtevami ter nudenje ekspertne tehnične podpore s svetovanji in različnimi usposabljanji.

3.2. Zgradba programa in lastnosti

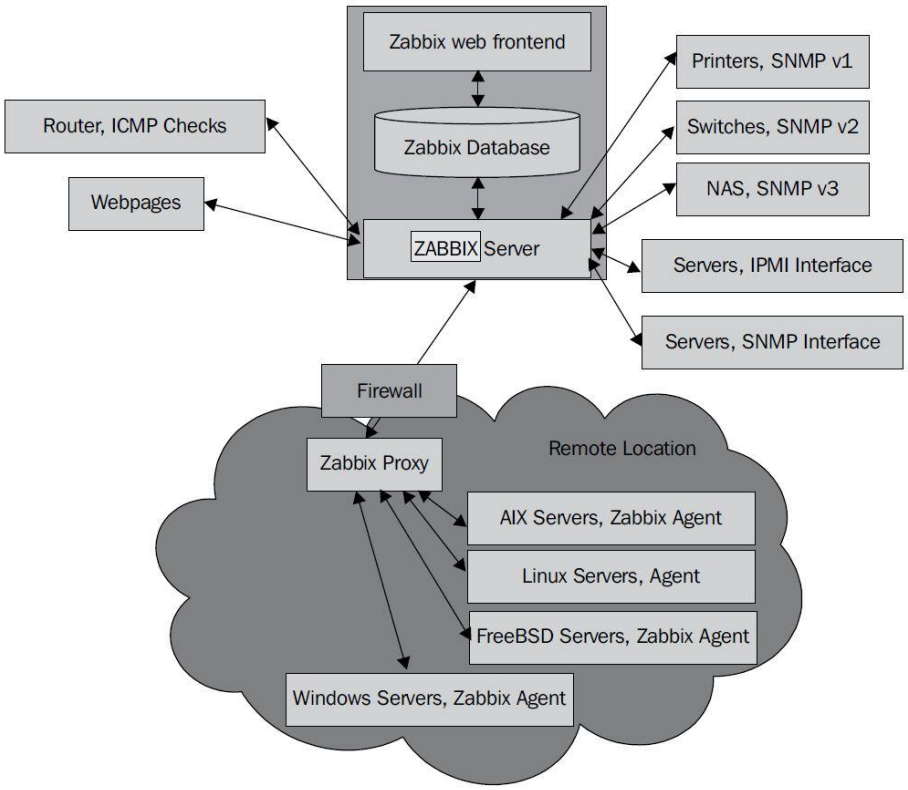
Program ponuja več načinov za nadzorovanje vseh različnih vidikov poslovne infrastrukture in bodočih dodatkov. Na kratko ga lahko opišemo kot delno distribuiran (angl. *semi-distributed*) nadzorni sitem s centraliziranim sistemom vodenja[3].

Lastnosti programske rešitve Zabbix:

- centraliziran in enostaven spletni vmesnik,
- strežnik teče na večini platform tipa Unix, kot so Linux, AIX, FreeBSD, OpenBSD, NetBSD, SCO Open Server in Solaris,
- agenti so na voljo za večino platform Unix in verzij Microsoft Windows,
- zmožnost direktnega nadzorovanja naprav preko SNMP protokola (verzije 1, 2 in 3) ter IPMI protokola;
- vgrajene grafične in druge vizualizacijske zmožnosti,
- fleksibilna konfiguracija z vključenimi predlogami (angl. *templates*),

- podpora različnim tipom podatkovnih baz (Oracle, MySQL, PostgreSQL, IBM DB2,SQLite),
- pošiljanje obvestil od izrednih dogodkih.

Če si ogledamo shemo preprostega omrežja, ki ga v celoti nadzorujemo z Zabbixovo programsko rešitvijo, lahko opazimo, da je strežnik Zabbix umeščen v sredino (Slika 3) skupaj s podatkovno bazo (angl. *database*) ter čelnim delom sistema (angl. *frontend*) s spletnim vmesnikom, ki omogoča dostop do zalednega dela sistema in nastavitev. Centralni del je povezan in omogoča hkratno komunikacijo z več napravami, s katerimi lahko komunicira s pomočjo programskih agentov Zabbix, preko posredovalnih strežnikov Zabbix (angl. *Zabbix proxy*), neposredno preko različnih protokolov za nadzorovanje (SNMP v1/v2/v3, IPMI) in osnovnih zahtev ICMP. Orodje hkrati omogoča nadzor spletnih strani.



Slika 3: Shema omrežja z osrednjim strežnikom Zabbix in lokalnim omrežjem na oddaljeni lokaciji.

3.2.1. **Programski agenti Zabbix**

Kot je že bilo omenjeno, programski agenti so osnova mrežnega nadzorovanja in na voljo za veliko različnih platform. Agenti lahko delujejo pasivno, ti na zahtevo strežnika zberejo podatke in mu pošljejo odgovor, ali aktivno, ko zajemajo podatke ves čas ter jih v določenih intervalih ali na zahtevo sami pošljejo strežniku. Vzpostavitev neposredne povezave med agentom in strežnikom je sicer najenostavnejši in najučinkovitejši način za zanesljivo nadzorovanje, a v nekaterih primerih ni mogoč – nekatere enote v večjem lokalnem omrežju so lahko popolnoma izolirane od ostalih in od zunanjega omrežja, tudi požarni zidovi lahko povzročajo težave.

3.2.2. **Namestniški strežnik Zabbix**

Za primere omrežij, varovanih s požarnimi zidovi, uporabljamo posebne namestniške strežnike Zabbix (angl. *Zabbix proxy*), ki namesto centralnega strežnika sami poskrbijo za pridobivanje podatkov z vseh naprav, ki so lahko del določenega zaprtega lokalnega omrežja. Z njihovo pomočjo tako odpade potreba po individualni povezavi od centralnega strežnika do posamezne naprave in obratno. Namestniški strežnik namreč sam pridobi vse podatke ter jih zbrane pošlje centralnemu strežniku. Enostavna je tudi konfiguracija požarnega zidu, ki varuje omrežje pred zunanjimi vdori, potrebno je definirati le ustrezno pravilo, ki bo omogočalo nemoteno komunikacijo med centralnim ter namestniškim strežnikom. Podobno kot agenti so tudi namestniški strežniki na voljo za različne tipe platform.

3.2.3. **Protokol SNMP**

Sledijo še rešitve, ki za nadzorovanje ne potrebujejo namenskih agentov. V ta namen programska rešitev Zabbix podpira tudi nekatere protokole za nadzor, med katere sodi SNMP (angl. *Simple Network Management Protocol*), ki je eden od najbolj priljubljenih in uveljavljenih protokolov[5]. Protokol je na voljo v več verzijah (SNMPv1, SNMPv2 ter najnovejša SNMPv3), ki so od prve predstavitve protokola dve desetletji nazaj izhajale postopoma. Kljub temu, da samo poimenovanje protokola namiguje na funkcionalnosti upravljanja sistema, je njegova poglavitna naloga prav nadzorovanje. Protokol je najbolj razširjen med napravami z vgrajenim sistemom (angl. *embedded device*), kjer poganjanje polno funkcionalnega operacijskega sistema in instalacija programskih agentov ni možna. Najpogostejši predstavniki naprav, ki že v osnovi podpirajo SNMP in ga tudi najpogosteje izkoriščajo, so tiskalniki ter omrežna stikala. Ostale naprave, ki prav tako podpirajo omenjeni protokol so različni usmerjevalniki, naprave za brezprekinitveno napajanje (UPS, angl. Uninterruptible Power Supply), omrežno priklopljeni pomnilniki (NAS, angl. Network

Attached Storage), senzorji za zaznavanje temperature ter vlažnosti v strežniških omarah in druge.

3.2.4. **Protokol IPMI**

Medtem ko je SMNP še vedno zelo priljubljen protokol in v večini, ko govorimo o napravah z možnostjo omrežnega povezovanja ter nadzorovanja, je IPMI (angl. *Intelligent Platform Management Interface*) relativno novejši protokol, ki prav tako omogoča nadzorovanje naprav. IPMI je navadno implementiran kot ločen gostiteljski nadzorovalni modul z neodvisnim operacijskim sistemom, čigar glavna lastnost je, da lahko ponuja podatke o nadzoru tudi takrat, ko naprava ne deluje (vseeno mora biti priklopljena na električno napajanje). V zadnjem času priljubljenost protokola zaradi cenejših IPMI modulov narašča, v celoti ga podpira tudi Zabbix.

3.2.5. **Zahteve ICMP**

Po pregledu protokolov za mrežni nadzor si oglejmo še zahteve protokola ICMP, ki je eden od osnovnih protokolov internetnega sklada protokolov (angl. *Internet Protocol Suite*). Te zahteve uporabljamo takrat, ko se želimo zgolj prepričati, ali je neka naprava, ki sicer ne omogoča instalacije agentov ali protokolov za nadzorovanje, dosegljiva ali ne. Takšne naprave so usmerjevalniki, preklopniki in ostale omrežne naprave. Zahteva ICMP za preverjanje dosegljivosti se imenuje *ping*. Poleg informacije o dostopnosti lahko z ukazom pridobimo tudi podatek o latenci med poslano in prejeto zahtevo.

3.3. Prenos programske rešitve

Na uradni spletni strani za prenos programske rešitve Zabbix[4] imamo na voljo več različnih namestitvenih paketov z različnimi različicami programske rešitve. Osnovo predstavlja paket z izvornimi datotekami (angl. *Zabbix sources*), ki vsebuje vse ključne dele sistema – osnovni Zabbixov strežnik, namestniške strežnike, agente za različne platforme ter spletni čelni del sistema s uporabniškim vmesnikom. Namestitev iz izvornih datotek večinoma vedno predlaga najnovejšo verzijo programa, ki ponuja nove možnosti in izboljšave ter več nadzora nad namestitvijo, za kar pa praviloma zahteva velik delež pozornosti pri nameščanju in konfiguraciji. Tako je treba ustrezno konfigurirati Zabbixov strežnik in vse ostale pripadajoče komponente, da lahko vsi delujejo na isti napravi in nemoteno dostopajo do centralne podatkovne baze.

Zaradi tega imamo poleg paketov z izvornimi datotekami na voljo že ustvarjene namenske naprave (angl. *Appliance*), ki so predhodno že nameščene, potrebno jih je le uvoziti v sistem ter konfigurirati. Izdajanje novih verzij pri namenskih napravah ni tako pogosto kot pri tistih v izvornih datotekah, zato so namenjene predvsem evalvaciji sistema in testiranju posameznih funkcij programske rešitve. Paketi namenskih naprav so na voljo v različnih formatih:

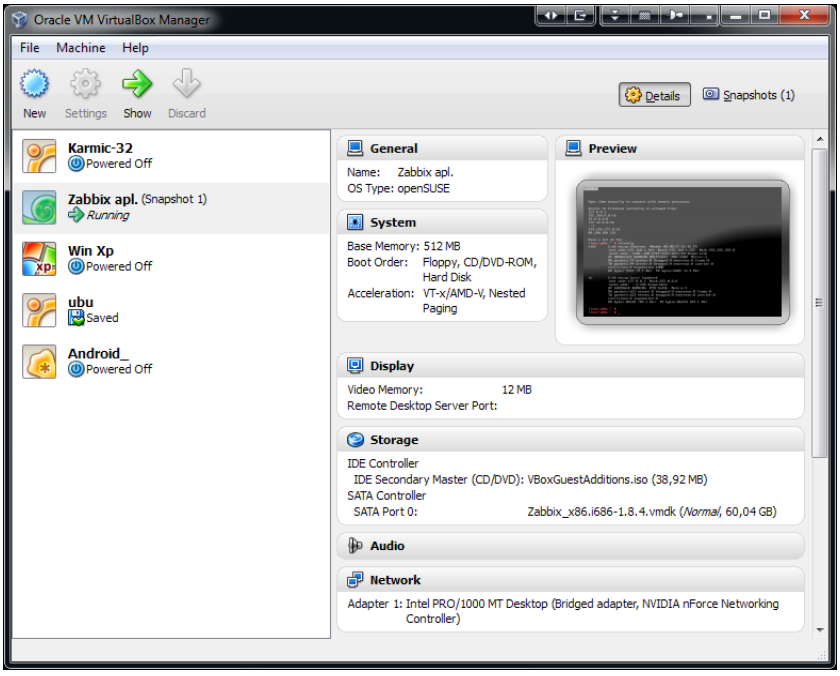
- VDMK (angl. *Virtual Machine Disk)* – format za ustvarjanje in izvoz navideznih diskov, ki jih uporabljajo navidezne naprave v programu WMware. Uvoz navideznega diska formata *vdmk* je možen tudi v programu VirtualBox;
- OVF (angl. *Open virtualisation format*) – odprti standard za ustvarjanje in distribucijo navideznih naprav ter programske opreme, ki teče na teh napravah;
- Live CD/DVD (format ISO) – avtonomen zagonski operacijski sistem, nahaja se v nekompresirani ISO datoteki, ki jo zapišemo na fizični medij, iz katerega se sistem ob zagonu samodejno zažene;
- Hard disk/Flash image (format RAW) – Samostojna datoteka, ki predstavlja celotno vsebino in strukturo nekega medija v neobdelani in nekompresirani obliki. Datoteka se zapiše na trdi disk ali bliskovni pomnilnik in predstavlja isto strukturo map ter particij kot disk, iz katerega je bila datoteka ustvarjena;
- Xen guest – Virtualna naprava, namenjena posebnim procesorskim arhitekturam kot so IA-32, Itanium, ARM itd.

3.4. Namestitev in konfiguracija strežnika

Osrednji strežnik je najpomembnejša komponenta, saj vzpostavlja povezavo z odjemalci ali namestniškimi strežniki, pošilja zahteve po podatkih, te podatke še dodatno procesira ter jih predstavi v človeku razumljivi obliki. Običajno je, da proces agenta, ki teče na strani klienta, praviloma pošilja podatke le enemu strežniku.

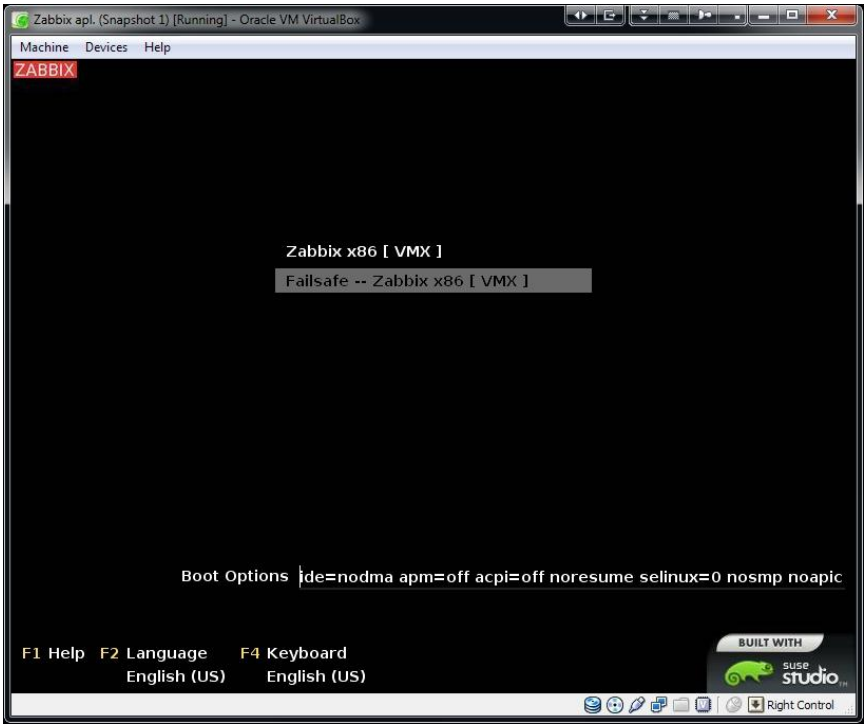
Pri naši namestitvi smo se zaradi majhnosti omrežja in števila nadzorovanih naprav odločili uporabiti že ustvarjeno namensko napravo v formatu *vdmk,* ki je osnovana na distribuciji Open Suse operacijskega sistema Linuxa z zalednim delom MySQL. Virtualni strežnik je tako popolnoma zadoščal potrebam izdelave aplikacije in testiranju posameznih komponent programske rešitve. Odločili smo se namestiti takrat najnovejšo verzijo z oznako *Stable*, s katero je bila označena takratna verzija 1.8.4.

Za uvoz virtualne naprave v operacijskem sistemu Windows smo uporabili program VirtualBox[6]. Za uspešen uvoz navideznega diska je potrebno ustvariti novo virtualno napravo, ki ji določimo ime in tip operacijskega sistema. Izbrali smo Linux distribucijo Open Suse. Nato sledi še določanje pripadajočega delovnega pomnilnika ter izbira navideznega diskovnega pogona, kjer izberemo lokacijo, kamor smo shranili našo datoteko *vdmk*. Navidezni pogon še potrdimo kot zagonski disk (angl. *Boot Hard Disk*), s čimer je uvoz pri koncu (Slika 4).



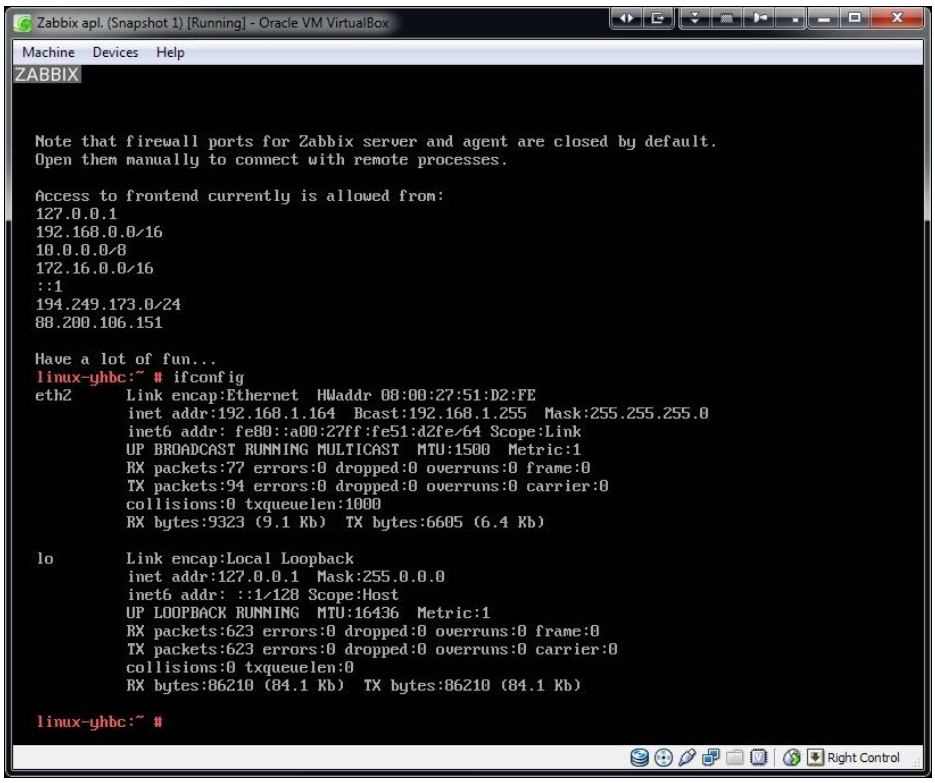
Slika 4: Program VirtualBox z uvoženim navideznim strežnikom Zabbix.

Po uspešnem uvozu lahko poizkusimo zagnati novo ustvarjeni navidezni strežnik. Tako se nam odpre novo okno, ki prikazuje zaslon navidezne naprave, na katerem se izvede zagonski postopek. Kmalu se prikaže zagonsko okno programa, ki omogoča, da program zaženemo s posebnimi ukazi, ki onemogočijo ali določijo nekatere posebne načine delovanja strojne opreme, ki morda niso podprti in lahko povzročijo neuspešen zagon programa (Slika 5). Za začetek izberemo prvo, standardno možnost in počakamo, da se strežnik dokončno naloži.



Slika 5: Izbira zagonskih možnosti ob zagonu strežnika.

Po uspešni zagonski proceduri nam strežnik prikaže nekaj osnovnih informacij in zažene vmesnik za prijavo. Za polni dostop do strežnika je potrebno vnesti kombinacijo uporabniškega imena in gesla, ki nas v primeru ustreznih podatkov pripelje do ukazne lupine, preko katere lahko že opravljamo vse dodatne nastavitve (Slika 6).



Slika 6: Ukazna lupina strežnika Zabbix.

V tem trenutku je strežnik postavljen in pripravljen za nadzorovanje. Z ukazom `ifconfig` lahko preverimo njegovo omrežno stanje, v odgovoru nam sistem vrne podatke o uporabljenih omrežnih vmesnikih. Poiskati je potrebno ustrezni vmesnik in podatek o dodeljenem IP-naslovu. S pomočjo tega naslova lahko nato preko spletnega brskalnika dostopamo do čelnega dela sistema. Če navidezni strežnik preko omrežnega vmesnika ne pridobi IP-naslova, lahko poskusimo z ukazoma *ifdown <ime omrežnega vmesnika>* ter nato *ifup <ime omrežnega vmesnika>*, ki od usmerjevalnika zahtevata ponovno dodelitev IP-naslova. Dodatno je priporočljivo z ukazom *date* na strežniku nastaviti tudi ustrezni datum in uro ter izbrati ustrezno časovno območje, saj lahko v nasprotnem primeru pride do nekonsistentnosti pri dnevniških zapisih in generiranju grafov.

3.5. Namestitev in konfiguracija odjemalcev

Proces, ki teče v ozadju operacijskega sistema in streže s podatki, imenujemo odjemalec Zabbix ali agent. Zabbix ponuja agente za večino operacijskih sistemov in distribucij. Sami smo za testiranje uporabili operacijska sistema Windows XP/7 ter Linux distribucijo Ubuntu. Dodatno je bil preizkušen tudi odjemalec, ki deluje na brezžičnem usmerjevalniku Linksys WRT54GL. Slednji je med uporabniki zelo priljubljenem, saj je, za razliko od drugih usmerjevalnikov, osnovan na prirejeni različici operacijskega sistema Linux. Usmerjevalnik omogoča tudi enostavno posodabljanje vgrajene programske opreme (angl. *firmware*) preko spletnega vmesnik. Zaradi teh lastnosti je množica računalniških navdušencev ustvarila veliko prostodostopnih modificiranih verzij sistema, ki omogočajo instalacijo posebnih dodatkov. Ti izboljšajo nekatere obstoječe funkcionalnosti usmerjevalnika in omogočajo namestitev raznoraznih dodatkov. Eden od teh je tudi agent Zabbix.

Ob namestitvi strežnika je sicer priporočljivo na ciljne nadzorovane naprave namestiti enako ali starejšo verzijo agenta Zabbix. Velja namreč, da starejše verzije agentov praviloma dobro delujejo z novejšimi verzijami strežnika Zabbix, v obratni smeri pa lahko pride do konfliktov. Nekatere starejše različice strežnikov namreč ne podpirajo določenih funkcij, ki jih ponujajo agenti novejših verzij, kar lahko privede do motenega ali onemogočenega nadzora posamezne naprave.

3.5.1. **Linux**

Za namestitev in testiranje agenta v operacijskem sistemu Linux smo se odločili za standardno namizno distribucijo Ubuntu različice Natty Narwall - 11.04. Za izdelavo okolja in konfiguracijo smo uporabili vgrajeni vmesnik z ukazno vrstico (CLI, angl. c*ommand line interface*) Terminal.

Prvi korak obsega namestitve zajema ustvarjanje novega uporabnika, poimenujemo ga »zabbix«:

```
sudo adduser zabbix
```

Po potrditvi nas sistem vpraša še za geslo, ki ga dvakrat vnesemo in potrdimo.

Nato novo ustvarjenega uporabnika dodamo v sudoer:

```
addgroup zabbix admin
```

Prijavimo se kot novi ustvarjeni uporabnik:

```
su – zabbix
```

Ustvarimo domačo mapo »zabbix« v imeniku `/home` ter v imeniku `/var/log/`, ki je namenjen shranjevanju dnevnikov posamičnih procesov ali sistemskih dnevnikov.

```
sudo mkdir /etc/zabbix
sudo mkdir /var/log/zabbix
```

Ustvarjenima mapama, vključno z vsemi podmapami, še spremenimo lastnika, in sicer uporabniku »zabbix« skupine »zabbix«:

```
sudo chown -R zabbix.zabbix /etc/zabbix/
sudo chown -R zabbix.zabbix /var/log/zabbix
```

Sedaj je na vrsti prenos izvornih datotek Zabbix, ki vključujejo agente in strežniške komponente v obliki stisnjenih arhivov »Tarball« (s končnico *.tar.gz*). Za prenos uporabimo ukaz `wget`, ki zažene orodje za prenašanje datotek. Če imamo datoteke že prenesene, ta korak izpustimo in datoteke iz arhiva le razširimo s pomočjo ukaza `tar`. Zraven ukazov uporabimo tudi navedene parametre:

```
wget
http://downloads.sourceforge.net/project/zabbix/ZABBIX%20Latest%20Stable/1.
8/zabbix-1.8.tar.gz?use_mirror=freefr
wget
http://www.zabbix.com/downloads/1.8/zabbix_agents_1.8.linux2_6.i386.tar.gz

tar zxvpf zabbix-1.8.tar.gz
tar zxvf zabbix_agents_1.8.linux2_6.i386.tar.gz
```

Zatem kopiramo vse komponente in njihove namestitvene datoteke, ki smo jih v prejšnjem koraku odpakirali/razširili, v imenika `/etc/zabbix/` ter `/usr/bin/`. V slednjem se nahajajo vse izvršljive datoteke, ki jih sistem uporablja pri zagonu ali za izredna popravila sistema.

```
cp misc/conf/zabbix_agent* /etc/zabbix/

sudo cp bin/zabbix_get /usr/bin/
sudo cp bin/zabbix_sender /usr/bin/
sudo cp sbin/zabbix_agent /usr/sbin/
sudo cp sbin/zabbix_agentd /usr/sbin/
```

Na vrsti je urejanje datoteke `zabbix_agentd.conf`, ki je osnovna konfiguracijska datoteka za proces agenta. Datoteko odpremo z enim od urejevalnikom besedil (*nano*, *vim*, *gedit* in ostali):

```
nano /etc/zabbix/zabbix_agentd.conf
```

Pri osnovni konfiguraciji in delovanju je potrebno urediti vrstico »Server« in vnesti ustrezni naslov IP našega strežnika Zabbix, oziroma jih ločiti z vejico, če želimo, da agent pošilja informacije več strežnikom:

```
### Option: Server
#     List of comma delimited IP addresses (or hostnames) of Zabbix
servers.
#     No spaces allowed. First entry is used for receiving list of and
sending active checks.
#     If IPv6 support is enabled then '127.0.0.1', '::127.0.0.1',
'::ffff:127.0.0.1' are treated equally.
#
# Mandatory: yes
# Default:
# Server=

Server=192.168.3.27
```

Uredimo še polje »Hostname«, kjer določimo ime našega gostitelja. Če se v nastavitvah strežnika, kljub istemu naslovu IP, ime gostitelja ne bo ujemalo, strežnik ne bo mogel nadzorovati agenta.

```
### Option: Hostname
#     Unique, case sensitive hostname.
#     Required for active checks and must match hostname as configured on
the server.
#     System hostname is used if undefined.
#
# Default:
# Hostname=system.hostname

Hostname=TestniLinux
```

Uredimo še datoteko »Services« v imeniku `/etc/services`:

```
sudo nano /etc/services
```

In v datoteko dodamo številki vrat (angl. *port*) obeh procesov `zabbix_agent` in `zabbix_trap`.

```
zabbix_agent 10050/tcp # Zabbix ports
zabbix_trap 10051/tcp
```

Sedaj je potrebno urediti še zagon agenta Zabbix, ki se naj zažene skupaj z operacijskim sistemom. *Init*, okrajšava za *initialization*, je proces v operacijskih sistemih Linux, ki se zažene kot prvi, skupaj s operacijskih sistemom, ob ustavitvi tega procesa pa se ustavi (angl. *halts*) tudi operacijski sistem. Vse zagonske skripte proces *Init* bere in izvaja iz mape `/etc/init.d`. V to mapo je potrebno s pomočjo ukaza `cp` kopirati ustrezne skripte, ki ustrezajo uporabljeni distribuciji sistema.

```
sudo cp misc/init.d/ubuntu/zabbix-agent.conf /etc/init.d
```

Za testni zagon agenta ni potreben takojšen ponovni zagon sistema, agenta lahko poskusimo zagnati z ukazom:

```
zabbix_agentd start
```

Preden dodamo novega gostitelja, preko spletnega vmesnika na strežniku preverimo, ali proces ustvarjenega agenta že teče na sistemu:

```
ps -aux | grep zabbix
```

Kot ustrezen odgovor moramo dobiti eno ali več vrstic, ki označujejo agenta z vsemi njegovimi pripadajočimi kopijami procesa `zabbix_agentd`. Kopije so tipa *pre-forked*, kar pomeni, da se ob zagonu procesa agenta samodejno ustvari več kopij agenta, ki so zmožni popolnoma neodvisno prejemati in odgovarjati zahteve. V primeru, da določen agent v določenem trenutku ne more odgovoriti na zahtevo, se le-ta pošlje naslednjemu in tako naprej. Število *pre-forked* kopij se nastavi s pomočjo vrstice `StartAgents` v datoteki `zabbix_agentd.conf`:

```
zabbix 742 0.0 0.0 6184 1016 ? SN 12:28 0:00 /usr/sbin/zabbix_agentd
zabbix 743 0.0 0.0 6184 1016 ? SN 12:28 0:00 /usr/sbin/zabbix_agentd
zabbix 744 0.0 0.0 6184 1016 ? SN 12:28 0:00 /usr/sbin/zabbix_agentd
zabbix 745 0.0 0.0 6184 1016 ? SN 12:28 0:00 /usr/sbin/zabbix_agentd
zabbix 746 0.0 0.0 6184 1016 ? SN 12:28 0:00 /usr/sbin/zabbix_agentd
```

3.5.2. **Microsoft Windows**

Instalacija v operacijskem sistemu Windows je do neizkušenega uporabnika, ki je vajen tega operacijskega sistema, dokaj prijazna in hitra. Okensko verzijo agenta lahko namestimo z dveh lokacij – agent se nahaja v izvornih datotekah v imenikih `.\bin\win32` in `.\bin\win64`, v katerih sta na voljo različici za 32- in 64-bitna Okna. Slednja sta dostopna tudi kot samostojni enoti brez izvornih datotek strežnika Zabbix in ostalih komponent, ki jih najdemo na uradni spletni strani za prenos[4]. Izbira je prosta, zaradi usklajenosti z verzijo strežnika in agenta Zabbix pa je agenta priporočljivo namestiti iz mape z izvornimi datotekami.

Ko se odločimo za različico, ki ustrezna arhitekturi našega operacijskega sistema, jo prekopiramo v poljubno mapo na disku. Priporočljivo je ustvariti kar standardni imenik `C:\Zabbix` in vanjo prekopirati vsebino mape. Poleg izvedljivih binarnih datotek potrebujemo tudi standardno konfiguracijsko datoteko, ki jo najdemo na lokaciji `.\misc\conf\zabbix_agentd.win.conf` in jo prekopiramo v mapo, kjer se nahaja agent. Za urejanje datoteke jo odpremo z enim od urejevalnikom besedil, zadostuje že vgrajeni program Beležnica (angl. *Notepad*).

Sedaj je potrebno spremeniti pot do dnevniške datoteke, ki se praviloma nahaja v isti mapi, kjer je agent. Tako uredimo vrstico z oznako `LogFile` :

```
LogFile=c:\zabbix\zabbix_agentd.log
```

Podobno kot v opisu nastavitev v poglavju 3.5.1 je prav tako treba urediti vrstici z imenom »`Server=`« ter »`Hostname=`« ter jim dodati pripadajoče vrednosti.

Sedaj lahko poskusimo pognati agenta z zagonskim ukazom v ukazni vrstici, kjer mu z opcijskim elementom `-c` podamo absolutno pot do konfiguracijske datoteke:

```
C:\zabbix>zabbix_agentd.exe -c c:/zabbix/zabbix_agentd.win.conf
```

Odgovor:

```
zabbix_agentd.exe [1464]:
!!!ATTENTION!!! ZABBIX Agent started as a console application.
!!!ATTENTION!!!
```

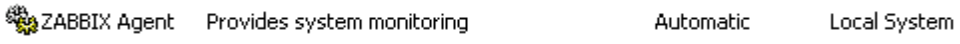
Proces agenta se sicer uspešno zažene, a nas hkrati opomni, da se je zagnal kot aplikacija v upravljalniku (angl. *console application*). Agent je namreč namenjen temu, da se namesti kot storitev, ki teče v ozadju operacijskega sistema. Za ta postopek je potrebno ukazu dodati zastavico `-i` (angl. *install*):

```
C:\zabbix>zabbix_agentd.exe -c c:/zabbix/zabbix_agentd.win.conf -i
```

Odgovor:

```
zabbix_agentd.exe [4376]: Service "ZABBIX Agent" installed successfully.
zabbix_agentd.exe [4376]: Event source "ZABBIX Agent" installed
successfully.
```

Dodatno lahko namestitev agenta Zabbix v obliki storitev preverimo tudi v Nadzorni plošči (Slika 7), razdelek Storitve (angl. *Control Panel, Services section*):



Slika 7: Novo dodana storitev agenta Zabbix v nadzorni plošči.

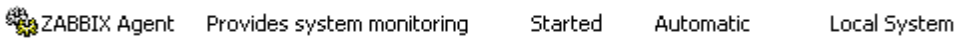
Storitev agenta se sedaj zažene samodejno s sistemom, a se trenutno še ne izvaja. Poženemo jo lahko s klikom na storitev in izbiro možnosti »Start« ali pa v ukazni vrstici izvedemo ukaz:

```
C:\zabbix>zabbix_agentd.exe -c c:/zabbix/zabbix_agentd.win.conf -s
```

V odgovoru lahko vidimo, da je storitev uspešno zagnana:

```
zabbix_agentd.exe [5608]: Service "ZABBIX Agent" started successfully.
```

Isti rezultat opazimo s tudi v Nadzorni plošči (Slika 8), storitev ima oznako delovanja :



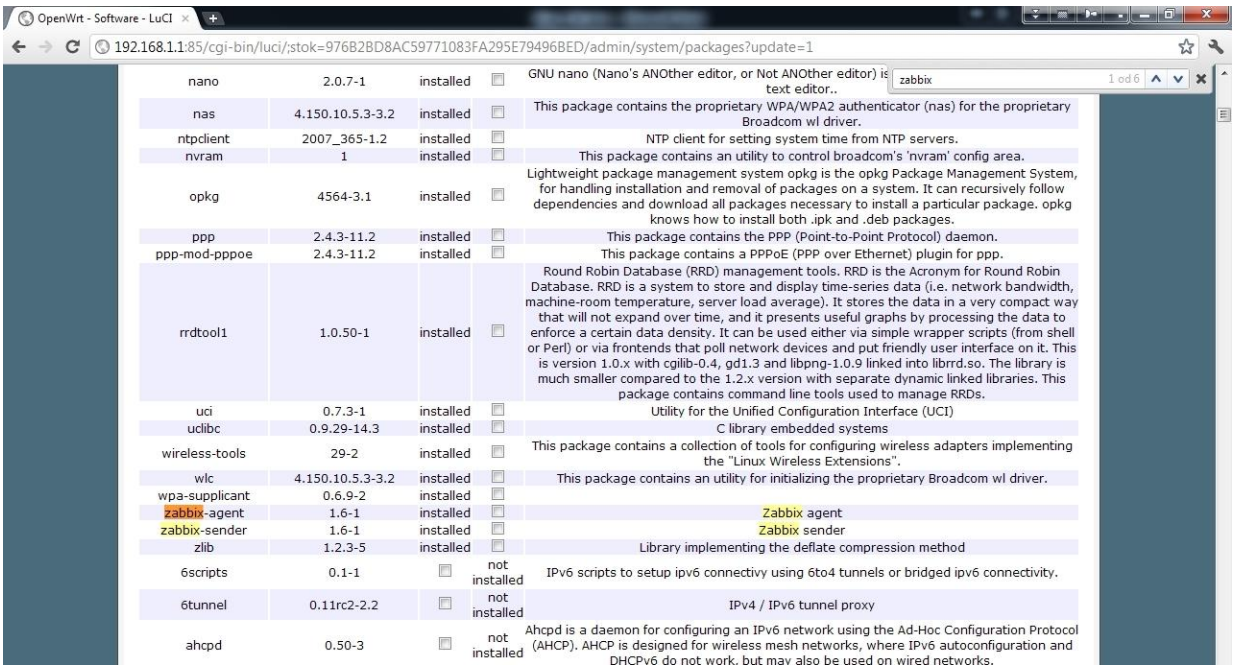
Slika 8: Zagnana storitev agenta Zabbix v nadzorni plošči.

Sedaj lahko novo ustvarjenega agenta dodamo v sistem in začnemo z nadzorom.

3.5.3. **Usmerjevalnik Linksys WRT54GL (programska oprema OpenWrt)**

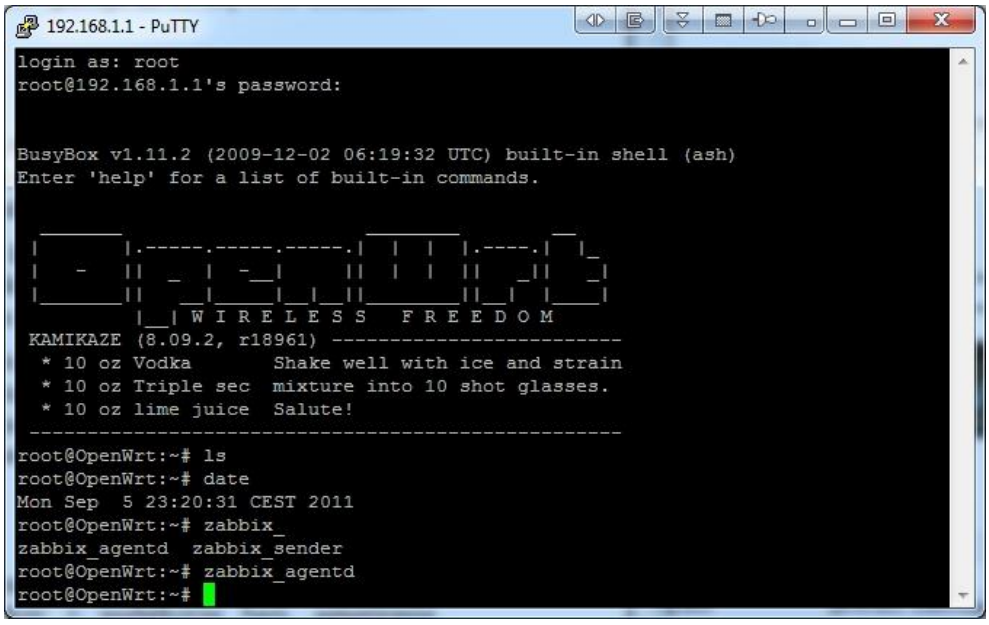
Za prikaz delovanja agenta na usmerjevalniku Linksys WRT54GL je bila uporabljena distribucija OpenWrt verzije Kamikaze 8.09.2, prilagojena za omrežni čip podjetja Broadcom z jedrom 2.4 (uradna oznaka *brcm-2.4*). Za začetek je treba iz uradne spletne strani distribucije OpenWrt[7] prenesti ustrezen paket, ga razširiti v izbrano mapo ter ga preko obstoječega spletnega vmesnika usmerjevalnika namestiti. Po ponovnem zagonu usmerjevalnika se je najbolje napotiti do vgrajenega grafičnega spletnega vmesnika *LuCI* (*Lua Configuration Interface*), ki omogoča najenostavnejšo namestitev dodatnih paketov. Za ta proces je potrebno usmerjevalniku že zagotoviti dostop do interneta.

Po vnosu IP-naslova usmerjevalnika v spletnemu brskalniku se nam prikaže prijavna stran vmesnika, kjer se prijavimo z ustreznim uporabniškim imenom in geslom našega usmerjevalnika. Po uspešnem postopku se nam odpre osrednja stran vmesnika, kjer v zgornjem meniju izberemo možnost »Administration«, ki nam omogoči naprednejše nastavitve usmerjevalnika. Tako lahko preko možnosti menija »Overview« izberemo podmožnost »LuCI Components«, ki nam omogoča pregled že nameščenih programskih paketov. Za namestitev paketa Zabbix je potrebno izbrati možnost »Update package lists« ter izmed množice paketov poiskati in izbrati vnos »zabbix-agent« (Slika 9). Za zaključek namestitve označimo potrditveno polje v vrstici ter potrdimo z gumbom »Perform Actions« na koncu dokumenta.



Slika 9: Pogled spletnega vmesnika usmerjevalnika in podstrani za nameščanje dodatnih komponent.

Ko je paket nameščen, je za delovanje potrebno še nekaj korakov konfiguracije. Za ta proces se je treba povezati do prednameščenega strežnika SSH (angl. *Secure Shell*), ki nam omogoča povezavo z oddaljenim računalnikom ali napravo ter dostop do njegove celotne hierarhije datotečnega sistema. Za vzpostavitev povezave SSH imamo na voljo veliko različnih programov. Sami smo se odločili za program PuTTY[11] in se z njim povezali do strežnika (Slika 10). Za povezavo je treba vnesti IP-naslov strežnika SSH, ki je identičen naslovu usmerjevalnika, ter ustrezno uporabniško ime in geslo.



Slika 10: Program PuTTY z aktivno povezavo do usmerjevalnika preko strežnika SSH.

Ob namestitvi paketa preko spletnega vmesnika se že avtomatično izvede skript *Makefile*, ki samodejno ustvari uporabnika in novo skupino *zabbix*, prekopira datoteke v ustrezne mape in ustvari privzete konfiguracijske datoteke. Za delovanje je potrebno slednje le še urediti, zato odpremo datoteko `zabbix_agentd.conf` in uporabimo enega od nameščenih urejevalnikov besedil:

```
nano /etc/zabbix/zabbix_agentd.conf
```

Uredimo vrstico »Server« in vnesemo ustrezni IP-naslov našega strežnika Zabbix oziroma jih ločimo z vejico, če želimo, da agent pošilja informacije več strežnikom:

```
### Option: Server
#     List of comma delimited IP addresses (or hostnames) of Zabbix
servers.
#     No spaces allowed. First entry is used for receiving list of and
sending active checks.
#     If IPv6 support is enabled then '127.0.0.1', '::127.0.0.1',
'::ffff:127.0.0.1' are treated equally.
#
# Mandatory: yes
# Default:
# Server=

Server=192.168.3.27
```

Uredimo še polje »Hostname«, kjer določimo ime našega usmerjevalnika. Če se v nastavitvah strežnika, kljub istemu naslovu IP, ime gostitelja ne bo ujemalo, strežnik ne bo mogel nadzorovati agenta.

```
### Option: Hostname
#     Unique, case sensitive hostname.
#     Required for active checks and must match hostname as configured on
the server.
#     System hostname is used if undefined.
#
# Default:
# Hostname=system.hostname

Hostname=Linksys_router
```

Sedaj moramo poskrbeti še za nastavitev pravic, saj privzete nastavitve ne omogočajo dostopa do konfiguracijske datoteke.

```
chown -R zabbix.zabbix /etc/zabbix/
chown -R zabbix.zabbix /var/log/zabbix
```

Agent je tako pripravljen za uporabo z usmerjevalnikom. Če bi želeli, da se agent zažene skupaj z usmerjevalnikom, moramo podobno kot pri navodilih za Linux v mapo *init.d* kopirati ustrezni skript:

```
cp misc/init.d/ubuntu/zabbix-agent.conf /etc/init.d
```

Za testni zagon agenta ni potreben takojšen ponovni zagon sistema, agenta lahko poskusimo zagnati s ukazom:

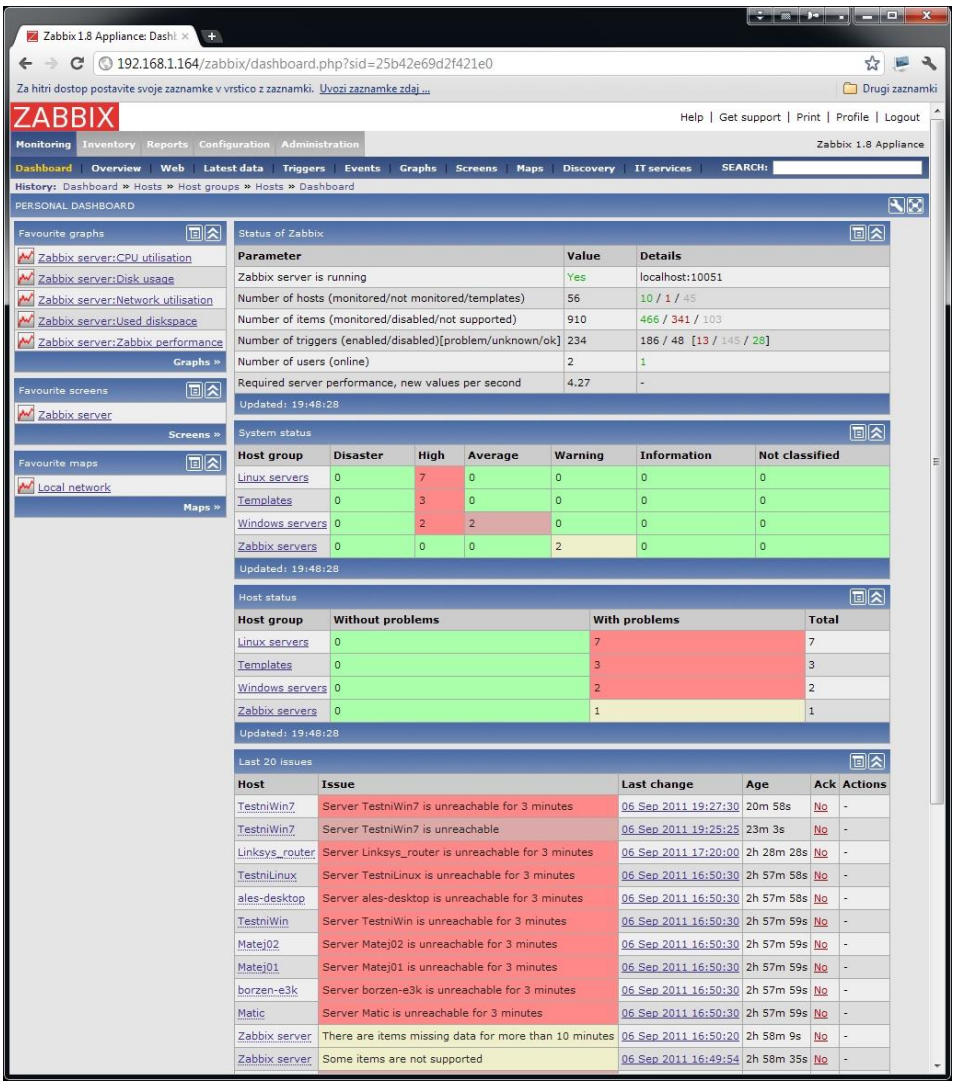
```
zabbix_agentd start
```

Kot ustrezen odgovor moramo dobiti eno ali več vrstic, ki označujejo agenta z vsemi pripadajočimi kopijami procesa `zabbix_agentd`:

```
 1038 zabbix    2184 S N  zabbix_agentd
 1039 zabbix    2184 S N  zabbix_agentd
 1040 zabbix    2188 S N  zabbix_agentd
 1041 zabbix    2196 S N  zabbix_agentd
 1042 zabbix    2184 S N  zabbix_agentd
 1043 zabbix    2216 S N  zabbix_agentd
```

3.6. Izgled spletnega programskega vmesnika programa Zabbix

Za dostop do programskega spletnega vmesnika je potrebno vnesti ustrezen IP-naslov strežnika. Če strežnik poganjamo na našem računalniku, je to IP-naslov privzetega omrežnega vmesnika. Po vnosu uporabniškega imena in gesla se nam v primeru vnosa pravilnih podatkov prikaže pozdravno sporočilo. Nato se odpre spletni vmesnik s privzetim pogledom, imenovanim »Personal dashboard« (Slika 11),ki vsebuje vse najosnovnejše informacije o delovanju in konfiguraciji strežnika ter podatke o delovanju in možnih izrednih dogodkih na nadzorovanih odjemalcih.



Slika 11: Privzeti pogled spletnega vmesnika Zabbix.

Za osnovno upravljanje s programom in izbiro osnovnih možnosti spletnega vmesnika imamo na vrhu glavni meni (Slika 12), ki prikazuje osnovne možnosti programa skupaj s pripadajočimi podmožnostmi, ki se prikažejo ob pomiku miškinega kazalca na določeno kategorijo.

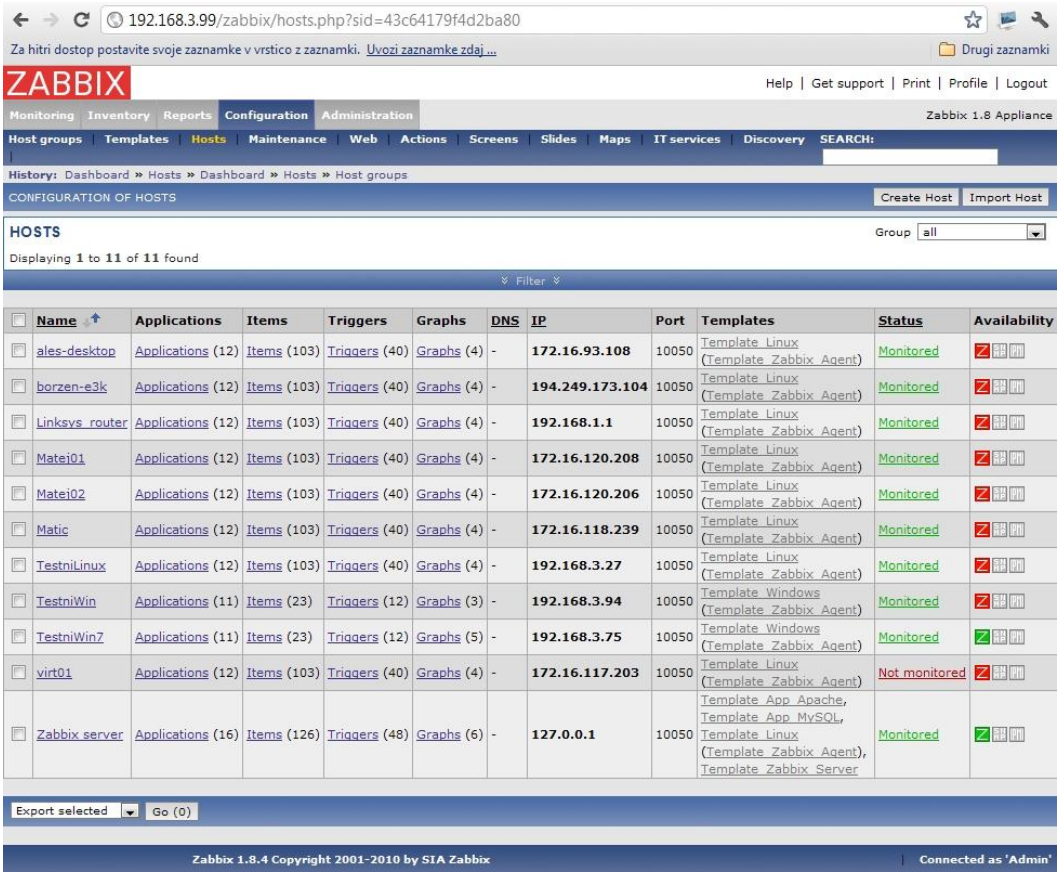
Slika 12: Osnovne možnosti glavnega menija razdeljene po kategorijah.

Kategorije osnovnega menija obsegajo:

- **Nadzor** (angl. *Monitoring*) – izbrana kategorija vsebuje večino možnosti, ki se ukvarjajo s nadzorom. V tem delu najdemo vse pridobljene podatke o odjemalcih, možne napake, grafe ter pregled delovanja sistema;
- **Popis** (angl. *Inventory*) – v tem razdelku najdemo popis podatkov o različnih nadzorovanih odjemalcih. Podatki se vnesejo ročno preko kategorije Configuration| Hosts| Profile;
- **Poročila** (angl. *Reports*) – v kategoriji lahko pregledujemo različna poročila o preteklih dogodkih, razpoložljivosti ter pregled sumiranih podatkov in njihovih predstavitev v vizualni obliki;
- **Nastavitve** (angl. *Configuration*) – urejanje vseh parametrov in nastavitev, ki so potrebni za učinkovito delovanje sistema spletnega vmesnika, nadzorovanja, pošiljanja opozoril itd. Razdelek omogoča tudi dodajanje novih predlog, zemljevidov za prikaz lokacij strežnikov ter dodajanje odjemalcev in skupin za grupiranje odjemalcev glede na poljubne parametre;
- **Administracija** (angl *Administration*) – razdelek je posvečen administraciji samega programa, ki obsega dodajanje uporabnikov in urejanje njihovih pravic, izbor avtentikacij, ki jo uporablja program, pregled in urejanje dnevnikov dostopa do spletnega vmesnika in procesiranih podatkov, urejanje in dodajanje poljubnih skript ter spreminjanje lokalnih nastavitev in nastavitev instalacije.

Postopek dodajanja nove nadzorovane naprave

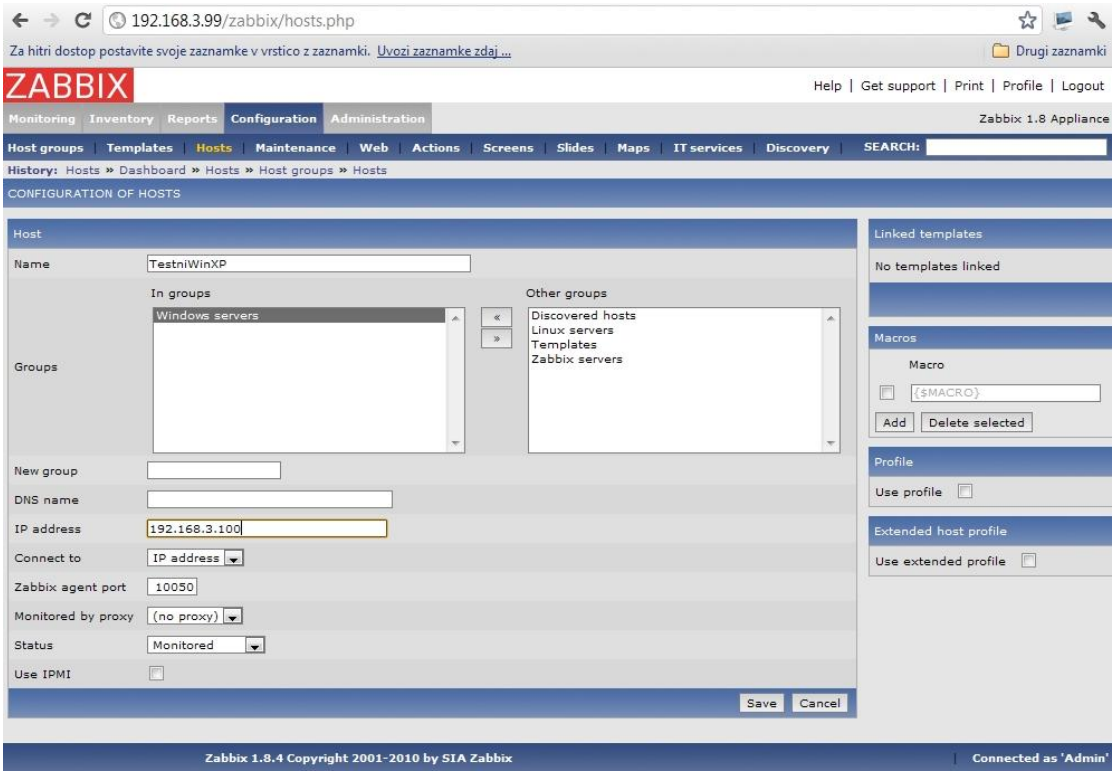
Dodajanje novih naprav za nadzorovanje je osnoven korak, ki smo ga v diplomski nalogi večkrat omenjali. Za dodajanje moramo v glavnem meniju spletnega vmesnika izbrati možnost *Configuration*. Odpre se nam pogled prve izbrane možnosti podmenija, privzeta je *Host Groups*, ki nam predstavlja vse obstoječe skupine s primerki odjemalcev v teh skupinah. Za dodajanje nove naprave oziroma odjemalca je potrebno v meniju izbrati možnost *Hosts*, ki nam prikaže seznam vseh vnesenih odjemalcev v sistemu (Slika 13). Poleg imena vnesene naprave imamo na voljo še nekaj podrobnosti o vsaki napravi (število elementov *Items*, število dodeljenih sprožilcev *Triggers*, definirani grafi za napravo), IP-naslov in številka vrat, status nadzora odjemalca (nadzorovan/ni nadzorovan) ter razpoložljivost odjemalca (je/ni dosegljiv).



Slika 13: Seznam vseh vnesenih odjemalcev v pogledu spletnega vmesnika.

Za dodajanje novega odjemalca v desnem zgornjem kotu izberemo možnosti *Create Host*, s katero ustvarimo novega odjemalca. Poleg nje imamo na voljo tudi opcijo *Import Host*, pri kateri novega odjemalca uvozimo iz predhodno izvožene datoteke z vsemi informacijami o odjemalcu. Predpostavimo, da take datoteke nimamo na voljo, zato izberemo prvo opcijo.

Odpre se nam pogled *Configuration of Hosts* z vnosnimi polji za vnos podatkov in izbiro skupine, v katero lahko umestimo novega odjemalca (Slika 14). Pri vnosu podatkov sta najpomembnejša le ime odjemalca, IP-naslov ter številka vrat za vzpostavitev povezave. Vsi ostali podatki so opcijski in jih v prvem delu, ko se želimo le prepričati ali strežnik uspešno vzpostavi povezavo ter prepozna odjemalca, ni potrebno vnesti.



Slika 14: Dodajanje novega odjemalca v pogledu spletnega vmesnika.

Po potrditvi novo vnesenega odjemalca se ta nahaja v seznamu vseh vnesenih odjemalcev (Slika 13). Potrebno je še nekoliko počakati, da se ikona v stolpcu obarva zeleno, kar pomeni, da je strežnik uspešno vzpostavil povezavo s odjemalcem, ki je tako pripravljen na nadzorovanje. V primeru, da ikona ostane rdeča, se nanjo pomaknemo z miškinim kazalcem, kar nam prikaže dodatne informacije o izvoru in tipu napake, s pomočjo katerih lahko hitreje odpravimo vzroke neuspešne povezave.

4. OPERACIJSKI SISTEM ANDROID

4.1. Zgodovina

Zgodovina operacijskega sistema Android sega v leto 2003, ko se je skupina razvijalcev odločila izdelati operacijski sistem za bodoče pametne telefone (angl. *smartphone*). Ta bi ponujal stabilno ogrodje za pametnejše osebne naprave, ki bi bile bolj seznanjene z lokacijo svojega uporabnika ter njegovimi željami.

Podjetje Android Inc. je avgusta 2005, prevzelo podjetje Google Inc. in ga skupaj z razvijalci ter ostalimi zaposlenimi priključilo kot hčerinsko družbo. S tem so prevzeli tudi vse vložene patente podjetja.

Novembra 2007 se je ustanovila t.i. *Open Handset Alliance*, aliansa podjetij, ki je, poleg idejnega vodje Googla, obsegala določene proizvajalce mobilnih telefonov, izdelovalcev komponent za prenosne naprave in ponudnikov mobilne telefonije (Intel, LG, HTC, Motorola, Nvidia, Qualcomm, Samsung Electronic, T-Mobile, Texas Instruments in ostali). Glavni cilj združenja je bil nadaljnji skupni razvoj odprtih standardov za mobilne in prenosne naprave, ki bi uporabniku ponudili novo uporabniško izkušnjo. Za ta namen je bila že isti dan pripravljena in v zgodnji vpogled izdana prva verzija razvojnega okolja Android.

Uradna predizdajna verzija (RC, angl. *release candidate*) z oznako 1.0 je bila sicer objavljena šele slabo leto kasneje in sicer oktobra 2008, nekaj dni po izidu prvega telefona z operacijskim sistemom Android, T-Mobile G1. Dobro sprejeta platforma je istega leta botrovala k pristopu 14 novih podjetij, ki so se vključile v alianso (ARM Holdings, Asustek Computer Inc., Garmin Ltd, Sony Ericsson, Toshiba Corp. in ostala).

Od oktobra 2008 je Android na voljo brezplačno in licenciran kot odprtokodna programska oprema. Hkrati je Google pod licenco Apache izdal tudi celotno izvorno kodo, ki vsebuje sklad mreže ter telefonije. Do danes je bila zadnja izdana verzija 3.2 »Honeycomb«, ki je v osnovi namenjena tabličnim računalnikom in ostalim napravam z zasloni večjih dimenzij. To morda kaže na namen Googla, da platformo Android nekoč približa tudi osebnim računalnikom.

4.2. Lastnosti operacijskega sistema

Android je operacijski sistem, ki je zasnovan na modificirani verziji Linuxa. Že takoj ob izidu je Google želel, da bi operacijski sistem Android ostal odprt in brezplačen. Večina izvorne kode je namreč izdana pod odprtokodno licenco Apache, kar pomeni, da lahko vsak, ki želi uporabljati Android, enostavno prenese celotno izvorno kodo in uporablja sistem. S tem je operacijski sistem postal zanimiv tudi za različne proizvajalce mobilnih telefonov in drugih podobnih naprav, ki operacijskemu sistem priredijo uporabniški vmesnik, dodajo določene kontrole ter lastne aplikacije, s čimer ustvarijo prepoznaven izdelek, ki jih loči od ostalih na trgu.

Od dneva izida operacijskega sistema je bil eden glavnih ciljev omogočati aplikacijam, da delujejo interaktivno in omogočajo medsebojno uporabo lastnih komponent. Medsebojna raba se ne nanaša samo na storitve, ampak tudi na deljenje podatkov ter gradnikov uporabniškega vmesnika.

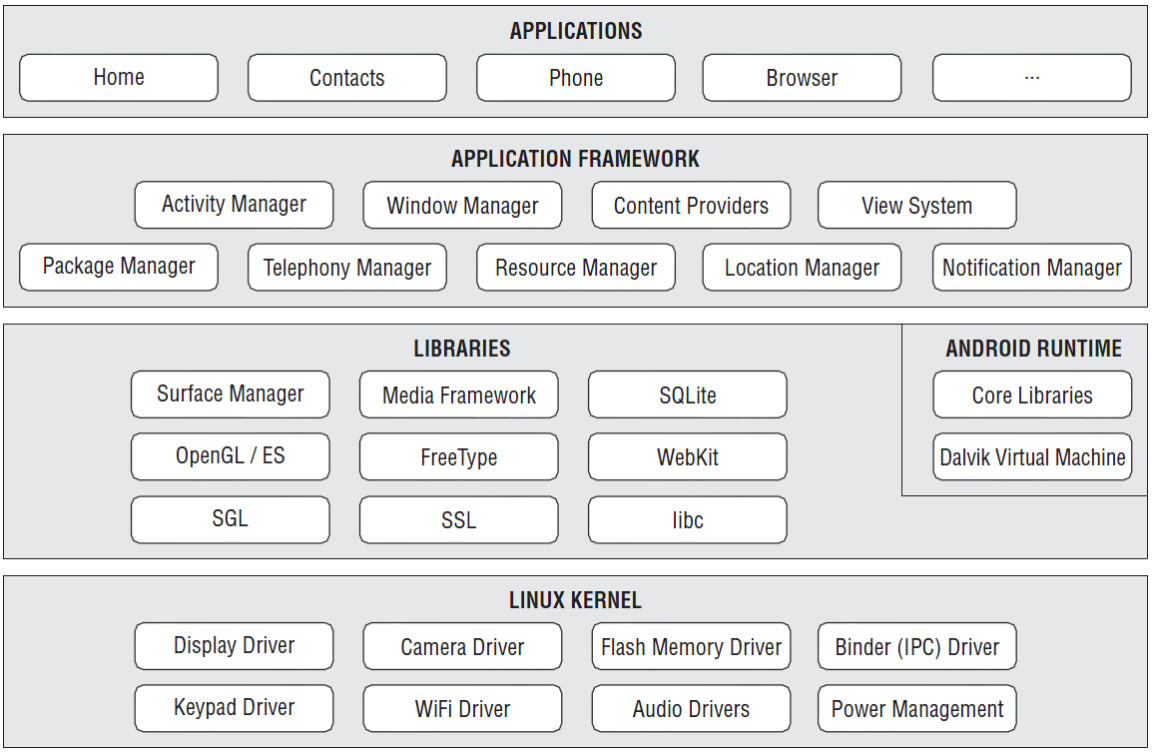
Poleg odprtokodne zasnove in proste dostopnosti Android prav tako nima točno specificiranih strojnih in programskih zahtev. Vseeno pa Android odlikujejo naslednje osnovne zmožnosti:

- **povezljivost** – sistem podpira mobilne tehnologije kot so GSM/EDGE, IDEN, CDMA, EV-DO, UMTS ter tehnologije Bluetooth, WiFi, LTE, WiMAX itd;
- **baza podatkov** – uporaba SQLite, enostavne, relacijsko zasnovane podatkovne baze za shranjevanje podatkov;
- **sporočanje** – pošiljanje in prejemanje kratkih ter multimedijskih sporočil (SMS, MMS);
- **vgrajen brskalnik** – osnovan na tehnologiji WebKit za upodabljanje spletnih strani (angl. *rendering*) ter podpora jeziku Javascript s pomočjo pogona Chrome V8;
- **multimedija** – predvajanje video datotek v formatu H.263, H.264 (preko vsebnika 3GP ali MP4), MPEG-4SP, AMR, AMR-WB (preko vsebnika 3GP), zvočnih datotek formata AAC, HE-AAC (preko vsebnika MP4 ali 3GP), MP4, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, in BMP;
- **podpora večdotičnosti** – možna pri napravah z večdotičnim zaslonom, omogoča lažjo interakcijo s aplikacijami, ki podpirajo večdotičnost;
- **podpora večopravilnosti** – na sistemu se lahko izvaja več aplikacij hkrati;
- **deljenje mobilne podatkovne povezave** – podpora delovanju v načinu brezžične dostopne točke za deljenje mobilne internetne povezave;
- **sistem določanja lokacije** – s pomočjo modula GPS (angl *Global Positioning System*) lahko uporabnik enostavno pridobi natančne podatke o trenutnem položaju lokaciji.

4.3. Arhitektura

Platforma Android zagovarja idejo o splošnonamenskem in vseprisotnem računalništvu za prenosne naprave. To je celovita platforma, ki uporablja sklad protokolov na osnovi operacijskega sistem Linux za nadzor naprav, pomnilnika in procesov. Knjižnice[8] v Androidu obsegajo telefonijo, video, grafiko, izdelavo celovitega uporabniškega vmesnika (UI, angl. *user interface*) ter številne ostale poglede.

Za razumevanje arhitekture operacijskega sistema Android si je najbolje ogledati diagram posameznih slojev operacijskega sistema Android (Slika 15):



Slika 15: Pregled arhitekture operacijskega sistema Android po slojih.

- **jedro Linux** (angl. *Linux kernel*) – jedro platforme Android je osnovano na jedru Linux, ki je odgovorno za upravljanje z gonilniki naprav, dostopom do sredstev, upravljanju z napajanjem in ostalih nalog, ki jih običajno opravlja OS;
- **knjižnice** (angl. *Libraries*) – na naslednjem nivoju se nahajajo standardne knjižnice (večina jih je spisana v jezikih C/C++) kot so OpenGL (gradnja naprednih grafičnih aplikacij), WebKit (podpora za brskalnike), FreeType (pretvarjanje običajnih pisav v rasterizirane/vektorske pisave), Secure Socket Layer (sloj varnih vtičnic, protokol za varno povezavo med strežnikom in odjemalcem), SQLite (relacijska podatkovna baza) in Media (knjižnice za predvajanje ter zajemanje zvočnih ali video datotek);

- **izvajalnik kode** (angl. R*untime*) **–** nahaja se na istem sloju z ostalimi knjižnicami in ponuja množico osrednjih knjižnic (angl. *core libraries*), ki omogočijo razvijanje aplikacij v programskem jeziku Java. Za ta namen izvajalnik vključuje tudi virtualni stroj Dalvik (angl. *Dalvik virtual machine*), ki omogoča aplikaciji, da se izvaja v svojem lastnem procesu kot primerek (angl. *instance*) virtualnega stroja Dalvik;

- **aplikativno ogrodje** (angl. *Application Framework*) – ponuja različna programska ogrodja operacijskega sistema Android, ki jih razvijalec lahko uporabi v aplikaciji in jih spreminja po svojih željah;

- **aplikacije** (angl. *Applications*) – na najvišjem nivoju se nahajajo že prednameščene poglavitne aplikacije (brskalnik, pregledovalnik slik, delo s kontakti ….) ter aplikacije, ki se namestijo na željo uporabnika.

Androidovo programsko razvojno ogrodje (Android SDK, angl. *Android Software Development Kit*) podpira večino značilnosti tehnologije Java. Če jo primerjavo s standardno platformo Java, znano pod imenom Java Platform Standard Edition (Java SE), lahko vidimo, da Android SDK ne vsebuje programske knjižnice za izdelavo okenskih uporabniških programov AWT (angl. *abstract windowing toolkit*) ter njene naslednice Swing. Namesto teh knjižnic Android SDK vsebuje lastno razširljivo ogrodje za izdelavo uporabniškega vmesnika.

Ker se aplikacije programirajo v programskem jeziku Java, je pričakovano, da se za interpretiranje ukazov v obliki bitne kode uporablja posebno verzijo Javanskega navideznega stroja JVM (angl. *Java Virtual Machine*). Takšna navidezna naprava tipično ponuja tudi prepotrebno optimizacijo, ki ji omogoča, da doseže enake primerljive zmogljivosti s jeziki, ki uporabljajo prevajalnik namesto intepreterja, katerih primerka sta C ali C++.

Pri načrtovanju operacijskega sistema Android je bilo potrebno veliko energije vložiti predvsem v načrtovanje, kako še dodatno izboljšati obstoječi javanski intepreter. Ta mora delovati tudi na prenosni napravi, ki zelo zaostaja za računskimi ter pomnilniškimi sposobnostmi klasičnih računalnikov. Ob izidu prve naprave Androida je ta v tem pogledu za običajnim namiznim računalnikom zaostajala od osem do deset let. Zaradi teh razlogov je Google preučil obstoječo implementacijo JVM in jo izboljšal na vseh kritičnih območjih.

Rezultat je nov optimizirani javanski navidezni stroj, ki omogoča učinkovito in tekoče izvrševanje prevedenih javanskih razredov v skladu z zmožnostmi in omejitvami prenosne naprave (pomnilnik, procesor, napajanje). Imenuje se Dalvik VM (virtualni stroj Dalvik).

V prvem koraku virtualni stroj Dalvik najprej združi vse zgenerirane datoteke, ki predstavljajo posamezne javanske razrede, v eno ali več izvršljivih datotek Dalvik (angl. *Dalvik Executable*, s končnico .dex). Pri tem ponovno uporabi katerekoli ponavljajoče se podatke iz več razrednih datotek in pri tem učinkovito zmanjšuje prostorske zahteve (nekompresirano) za polovico v primerjavi z običajnimi javanskimi arhivskimi datotekami (JAR, angl. *Java ARchive files*). Za osnovno primerjavo, datoteka .dex privzetega spletnega brskalnika v

Androidu je velika okoli 200 kilobajtov, velikost ekvivalentne .jar verzije pa je okoli 500 kilobajtov.

V naslednjem koraku je Google občutno izboljšal tudi sistem čiščenja pomnilnika (angl. *garbage collection*) v navideznem stroju Dalvik, a se je sprva odločil opustiti tehnologijo prevajalnika JIT (angl. *just-in-time*). JIT je hibridni pristop, ki za izvajanje kode uporablja poseben interpreter, ki ne interpretira celotne datoteke z bitno kodo naenkrat, ampak datoteko intepretira po delih in interaktivno prevede v strojni jezik. Razlog v učinkovitosti tega delovanja je v tem, da Java vseskozi nadzira izvajanje kode na različnih delih in s pomočjo predpomnjenja ponovno izvaja že predhodno prevedeno kodo.

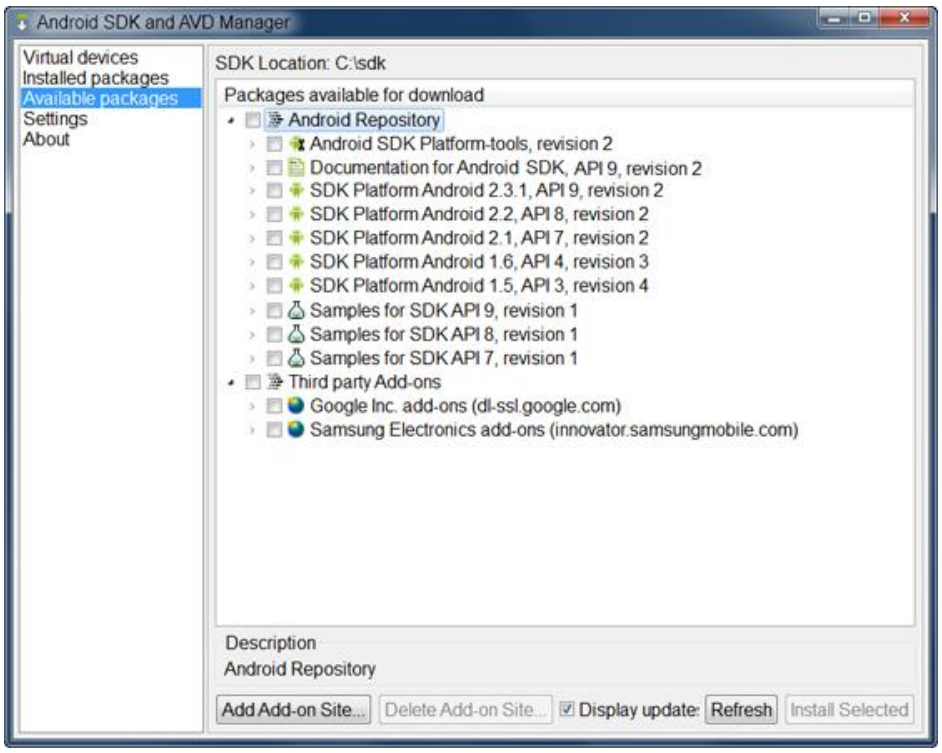
Zadnji korak optimizacije navideznega stroja Dalvik je drugačno generiranje kode v zbirnem jeziku (angl. *assembly-code*), kjer se za shranjevanje podatkov uporabljajo registri namesto klasične skladovne arhitekture. Razlog za uporabo registrov se upravičuje z manjšim naborom ukazov za shranjevanje podatkov v registrih in manipulacijo z njimi kot pri skladu, a so zaradi naslavljanja izvornega ter ponornega registra ti ukazi ponavadi daljši. Ta tehnologija je primerna predvsem za intepreterje virtualnih strojev, pri katerih se teži k čim redkejšemu pošiljanju in sprejemanju ukazov.

Optimizacija delovanja se nadaljuje tudi na ostalih aspektih operacijskega sistema Android. Za definiranje grafičnih vmesnikov in njihovo izgradnjo se uporabljajo datoteke XML (angl. *Extensible Markup Language*), ki se pred uporabo pretvorijo v binarne datoteke, še preden se upodobijo na napravah.

4.4. Lastnosti razvojnega ogrodja in namestitev

Za potrebe razvijanja aplikacij Android potrebujemo prosto dostopno razvojno ogrodje Android, imenovano Android SDK (*Software Development Kit)*, ki vsebuje vsa potrebna orodja za izdelavo in testiranje Android aplikacij. A preden lahko začnemo z razvojem Android aplikacij, si moramo prenesti Javino razvojno ogrodje, imenovano Java SE Development Kit (JDK). Za uspešno delovanje Android SDK zahteva JDK verzije 5 ali novejšo različico, ki jo prenesemo z uradne spletne strani[9].

Po uspešni namestitvi Javinega razvojnega okolja si prenesemo še Androidovo razvojno okolje. Ta je na voljo[10] za večino najpriljubljenejših operacijskih sistemov, med katere sodijo Windows (Windows XP, Windows Vista in Windows 7), Mac OS X (samo za Intel) ter Linux (samo Intel). V našem primeru smo uporabili instalacijski paket za operacijski sistem Windows 7. Ob končanju namestitve se samodejno izvede program *SDK and AVD Manager* (Slika 16). Ta nam omogoča pregled že instaliranih različic platforme Android ter nas obvešča o njihovih posodobitvah ter novih različicah platforme Androida, ki jih morebiti še nimamo nameščenih. Pri izdaji posameznih različic operacijskega sistema jim Google dodeli ustrezno oznako ter pripadajoč identifikator razvijalskega dela (*API level*). Prva različica je imela oznako 1.0 z identifikatorjem API 1, naslednja 1.1 z identifikatorjem API 2 Itd. Poleg tega so vsem različicam zaradi oglaševanja in boljše prepoznavnosti, začenši s 1.5, dodelili posebna imena, ki predstavljajo imena slaščic, urejenih po abecedi (Cupcake, Donut, Eclair, Froyo, Gingerbread…).



Slika 16: Osnovno okno orodja *SDK and AVD manager*.

Poleg instalacije posameznih platforme so nam v programu na voljo tudi različni paketi s primerki aplikacij, ki predstavijo novosti in izboljšave določene različice operacijskega sistema, in ustrezna dokumentacija. V kolikor želimo, lahko namestimo tudi posebne knjižnice, imenovane *Third party add-ons*, ki sicer niso sestavni dela Android SDK-ja, a ponujajo posebne dodatke in uporabo storitev posameznih podjetij, ki jih lahko s pridom uporabimo v naši aplikaciji.

Po namestitvi izbranih različic sistema Android nam program omogoča tudi ustvarjanje virtualnih naprav Android (AVD, angl. *Android Virtual Device*).

5. RAZVOJ APLIKACIJE

5.1. Metodologija dela

Pri procesu zbiranja zahtev za aplikacijo smo morali najprej preučiti arhitekturo sistema za nadzorovanje ter njegove osnovne specifikacije. Pri tem so bile v veliko pomoč že ustvarjene virtualne naprave in programski agenti za nadzorovanje, ki so prihranili veliko časa pri vzpostavitvi sistema za nadzor. Pomembno vlogo pri analiziranju funkcionalnosti aplikacije je igral predvsem Zabbixov programski vmesnik, saj so bile funkcionalnosti aplikacije pogojno odvisne od obsega preddefiniranih funkcij tega vmesnika. Vmesnik se je sicer izkazal za zelo stabilnega in dokaj enostavnega, a potrebuje še nekaj dodelav pri uporabi dodatnih filtrov za določene funkcije ter pri njihovih pomanjkljivih opisih v dokumentaciji.

Ker izkušenj z razvojem na platformi Android še nismo imeli, smo uporabili tehniko razvoja programske opreme s pomočjo prototipov. Pri tej tehniki so znane osnovne zahteve, na pa tudi vsi detajli (vhodni in izhodni podatki, podrobnosti glede procesiranja, uporaba algoritmov itd). Pri tem smo se najprej osredotočili na vzpostavitev učinkovite komunikacije med napravo ter sistemom za nadzorovanje in ob vsakem pomembnem mejniku prototipu dodali nove funkcionalnosti. Tako se je prototip, skupaj s spremembami v uporabniškem vmesniku, postopoma razvijal v končni produkt.

5.2. Orodja in tehnologije

5.2.1. **Razvojno okolje**

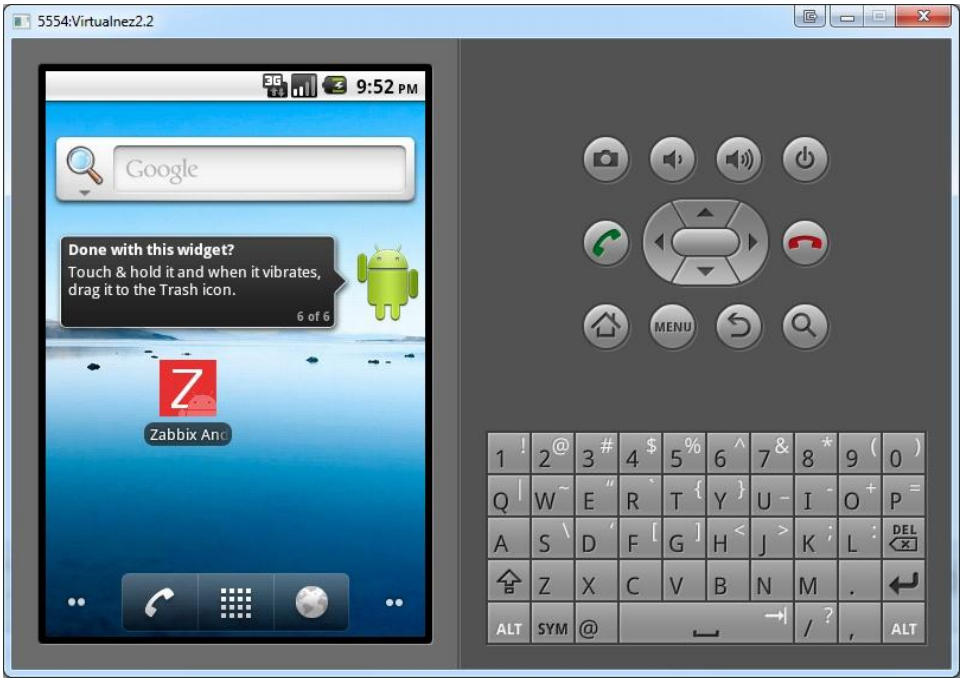
Podobno kot pri ostalih programskih jezikih tudi pri programiranju za operacijski sistem Android zadostuje že tekstovni urejevalnik in nameščeno razvojno ogrodje Android, kjer aplikacijo prevedemo in poženemo kar iz ukazne lupine. A podobno kot pri ostalih kompleksnejših novodobnih programskih jezikih si lahko delo precej olajšamo z uporabo okolja za razvoj programov (IDE, angl. *Integrated Desktop Enviorment*). Prednost takih orodij se pokaže pri razvoju programov zahtevnih programov in aplikacij, ki uporabljajo kompleksne knjižnice, za katere je potrebno podrobno delovanje in hierarhija razredov. Prednosti se kažejo pri razširjanju (angl. *Override*) obstoječih knjižnic, branju zaglavnih datotek in pripravi pomoči za argumente funkcij, samodejnega dokončanja kode (angl. *auto-complete*), integracija s servisi za nadzor verzij (SVN, Git …) in ostale[12].

Pri pripravi delovnega okolja smo se tako oprli na programsko orodje Eclipse IDE, in sicer verzijo s kodnim imenom Helios, ki je eno od uradno podprtih razvojnih okolij. Program poleg ostalih funkcij že v osnovi omogoča enostavno dodajanje novih programskih paketov in vtičnikov. Na spletni strani Android SDK najdemo tudi uradni vtičnik ADT (*Android Development Tools*) za Eclipse[13], ki olajša razvijanje Android aplikacij, saj ponuja vsa orodja za enostavno programiranje in testiranje Android aplikacij ter njihovih grafičnih konstruktov.

Poleg ostalih funkcij omogoča tudi testiranje s pomočjo razhroščevanja, kot ga poznamo pri običajnih Javanskih aplikacijah, in uporabo emulatorja, ki ga lahko enostavno ustvarimo in poženemo preko našega razvojnega okolja.

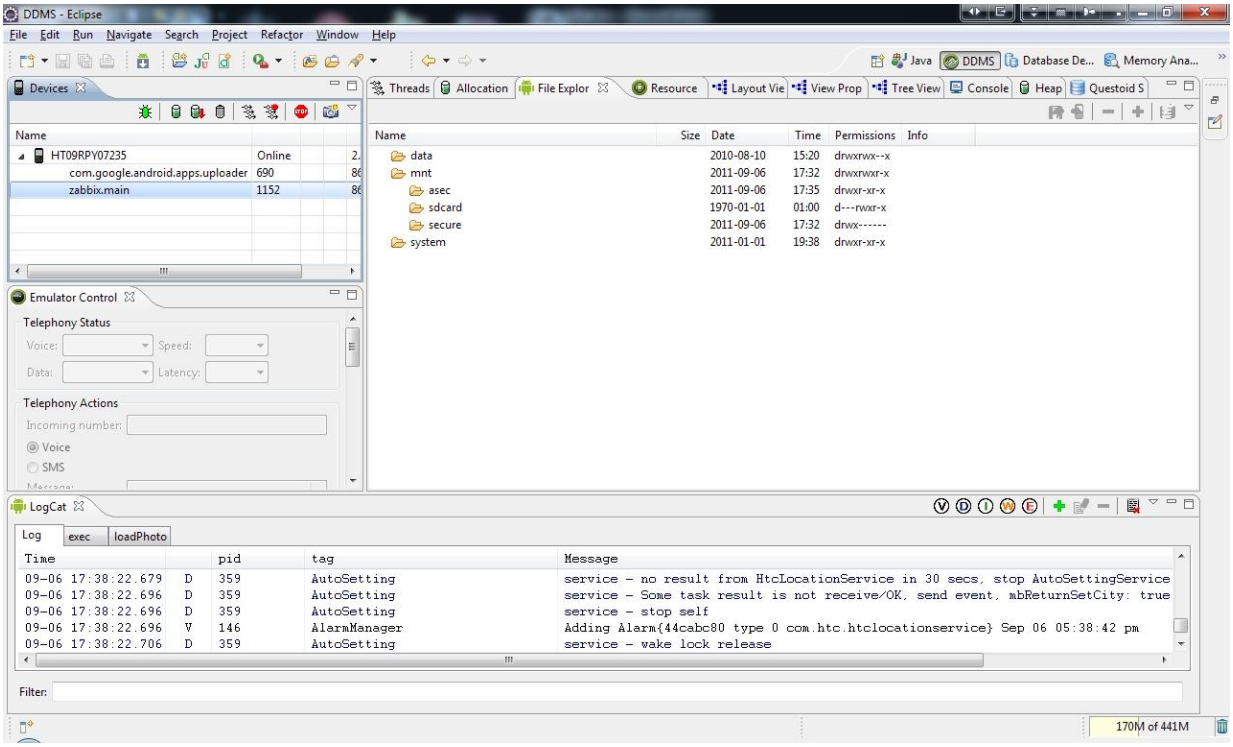
5.2.2. **Emulator, razhroščevalnik**

Emulator oziroma virtualna naprava Android (AVD, angl. *Android Virtual Device)* je standardni del razvojnega ogrodja, ki omogoča razvijanje in testiranje naše aplikacije, četudi ne posedujemo fizične naprave. Emulator je po obnašanju, zmožnostih ter programski in strojni zasnovi ekvivalenten fizični napravi z operacijskim sistem Android in nam omogoča instanten vpogled v obnašanje, izgled ter interaktivnost naše aplikacije. Poleg tega lahko posameznemu primerku virtualne naprave Android določimo veliko poljubnih lastnosti: podprto različico razvojnega okolja, velikost ter resolucijo zaslona, velikost pomnilnika ter velikost pomnilnika za shranjevanje podatkov itd. Primer virtualne naprave je viden na spodnji sliki (Slika 17).



Slika 17: Primer virtualne naprave z operacijskim sistemom Android 2.2 .

Ogrodje Android SDK nam poleg možnosti poganjanja aplikacije ponuja tudi zelo uporaben razhroščevalnik imenovan *Dalvik Debug Monitoring Server* (v nadaljevanju DDMS), ki ponuja številna orodja za vpogled v ozadje delovanja naše aplikacije. Za preklop v razhroščevalnik v okolju Eclipse je potrebno klikniti na pogled DDMS, ki se privzeto nahaja v meniju z ostalimi pogledi zgoraj desno (Slika 18). DDMS tako omogoča analiziranje aktivnih procesov, vpogled v stanje na skladu in kopicah, pregled aktivnih niti in njihovo pavziranje, raziskovanje po datotečni strukturi sistema, pregled trenutno priključenih fizičnih ali navideznih naprav Android ter simulacijo nekaterih dogodkov (dohodni klic ali tekstovno sporočilo), ki drugače na navidezni napravi niso mogoči. Poleg tega je preko razhroščevalnika poenostavljeno zajemanje trenutne slike na zaslonu naprave, preučevanje izpisov dnevnikov na napravi v realnem času (z orodjem *LogCat*), kreiranje map v datotečni strukturi naprave in dodajanje datotek na napravo ter iz nje.



Slika 18: Pogled orodja DDMS v razvojnem okolju Eclipse.

5.2.3. **Nadzor verzij preko strežnika SVN**

Za potrebe izdelave rezervne kopije projekta in beleženje poteka izdelave aplikacije smo projekt redno shranjevali na strežnik Apache Subversion (v nadaljevanju SVN). Ta princip se sicer pogosteje uporablja za projekte, pri katerih sodeluje večja skupina ljudi, saj omogoča lažje urejanje projekta in njegovih datotek ter preprečuje nekonsistentnost podatkov. Tudi v tem primeru smo uporabili prosto dostopni vtičnik Subclipse[14], ki omogoča shranjevanje in osveževanje stanja projekta z dostopom do strežnika SVN neposredno v okolju Eclipse.

Dostop do projekta je bil tako možen kjerkoli, zelo enostavno se je dalo tudi razveljaviti nezaželene spremembe v projektu in s pomočjo dnevnikov voditi evidenco o napredku izdelave aplikacije.

5.3. Povezava do programskega vmesnika

Nekatere spletne aplikacije in storitve ponujajo enostaven način dostopanja, konfiguriranja ter manipuliranja z dano aplikacijo oziroma storitvijo preko čelnega dela sistema (angl. *frontend*), neposredno v brskalniku. Problem pa nastane, ko bi želeli podatke deliti tudi z neko napravo, s katero bodisi ni možno bodisi ni smotrno dostopati do čelnega dela sistema, saj je ta procesorsko ali podatkovno preveč zahteven.

Definiranje enostavnega tekstovnega programskega vmesnika poenostavi takšne zahteve. Programski vmesnik (angl. API, *Application Programming Interface*) definira natančno določeno zbirko pravil in specifikacij, ki jih morajo upoštevati programi, da lahko komunicirajo s strežnikom API in tako dobijo ustrezno povratno informacijo. Natančno je potrebno definirati tudi tip formata zahtev in odgovorov, najpogosteje se sicer uporablja format XML (angl. *Extensible Markup Language*) ali JSON[15] (angl. *Java Script Object Notation*).

V primeru Zabbixovega programskega vmesnika gre za sporočila formata JSON-RPC (*JSON Remote Procedure Call*). Deluje tako, da se sproži zahteva, klic za oddaljeni postopek do strežnika, ki je implementiran na osnovi tega protokola. Odjemalec je v tem primeru tipično programska oprema ali aplikacija, ki kliče določeno metodo, ki je implementirana na oddaljenem sistemu. V enem klicu je lahko podanih več vhodnih parametrov v obliki polja ali objekta, prav tako lahko metoda v odgovoru vrne več izhodnih podatkov. Za klic oddaljene metode in pošiljanje zahteve v formatu JSON se uporabljajo metode protokola HTTP ali prenos sporočil preko vtičnika TCI/IP.

Primer zahteve JSON-RPC 2.0:

```
{
    "jsonrpc": "2.0",
    "method": "user.authenticate",
    "params": {
        "user": "Admin",
        "password": "zabbix"
    },
    "auth": null,
    "id": 0
}
```

V primeru zahteve JSON-RPC lahko hitro izluščimo ključne elemente. Najprej je treba z nizom `"jsonrpc"` definirati, da gre v tem primeru za sporočilo protokola JSON-RPC 2.0.

Naslednja vrstica `"method"` ciljnemu strežniku že pove, katero funkcijo mora poklicati s podatki, ki jih pridobi v vrstici `"params"`. V našem primeru so podatki podani v obliki objekta tipa *named parameters*, ki je sestavljen iz ključa za označbo parametra funkcije in pripadajoče vrednosti. Podatek bi lahko bil tudi v obliki *positional parameters*, kjer uporaba ključa kot oznaka imena parametra odpade, namesto nje je potrebno le vnesti vrednosti v obliki polja (`[admin,zabbix]`) v določenem zaporedju, ki označuje parametre po vrsti. Sledi še vrstica `"auth"`, ki vsebuje identifikator posamezne seje. Ker se seja ustvari šele ob uspešni prijavi v sistem, je v klicu prijave prazna (angl. *null*). Za identifikacijo posamezne zahteve skrbi vrstica `"id"`, ki jasno označuje, kateri zahtevi pripada prejeti odgovor na zahtevo.

Odgovor na zahtevo:

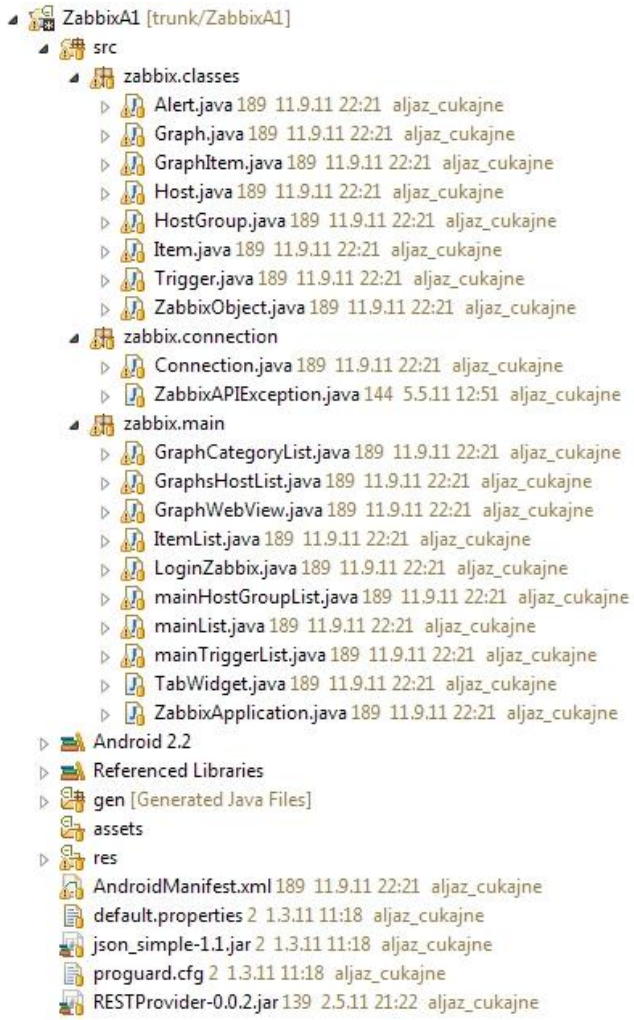
```
{
    "jsonrpc": "2.0",
    "result": "13f28ca608a4b12c83a32d749229da71",
    "id": 0
}
```

V odgovoru nam tudi sama ciljna naprava ali strežnik z nizom `"jsonrpc"` pove, v katerem formatu pošilja odgovor. Rezultat poizvedbe se nahaja v vrstici `"result"`, ki je lahko, odvisno od zahteve, v katerikoli znanem formatu. V našem primeru dobimo v odgovoru le identifikator seje `auth` v obliki niza, ki ga bomo vključili v vse ostale zahteve. Da odgovor pripada ustrezni zahtevi, lahko preverimo tako, da primerjamo prejeti identifikator `"id"` s poslanim v zahtevi.

5.4. Programski razredi in logika

Rezultat našega dela, aplikacija imenovana Zabbix Android, je namenjena napravam, ki imajo nameščen operacijski sistem Android verzije 2.2 ali novejši. Za uporabo programa in vseh njenih funkcij zadostuje že običajna mobilna ali druga naprava s standardnimi strojnimi specifikacijami in strojno opremo. Dostop do podatkov je možen preko brezžičnega Wi-Fi vmesnika, ki mora imeti dostop do omrežja z internetno povezavo, ali z uporabo mobilnega omrežja. Za shranjevanje slik posameznih grafov potrebujemo tudi vstavljeno spominsko kartico, sliko grafa pa lahko sicer shranimo tudi v notranji pomnilnik telefona. Aplikacija deluje tako v pokončnem kot tudi ležečem načinu delovanja, med katerima naprava praviloma samodejno preklopi, ko jo obrnemo.

Pri programiranju aplikacije smo se celoten projekt odločili razdeliti na posamezne dele. Tako smo medsebojno povezane razrede združili v pakete glede na funkcijo, ki jo opravljajo, oziroma entitete, ki jih predstavljajo (Slika 19).



Slika 19: Struktura datotek projekta.

- **Aktivnost Login**

Ko odpremo aplikacijo, je pogled aktivnosti Login prvi, ki ga naprava prikaže na zaslonu (seveda le v primeru, da aplikacije že nismo prej zagnali, saj je ta lahko shranila svoje stanje). To je vstopna aktivnost, ki od uporabnika zahteva vnos ustreznih podatkov, in mu v primeru uspešne prijave omogoči prijavo v sistem in s tem dostop do vseh funkcij naše mobilne aplikacije (Slika 20).

Pri prijavi je uporabnik dolžan vnesti spletni oziroma IP naslov strežnika, na katerem teče Zabbix, ter zraven dodati celotno pot do ustrezne datoteke programskega vmesnika aplikacije (ta je privzeta v formatu PHP in se nahaja na lokaciji `./zabbix/api_jsonrpc.php`). Uporabnik nato vnese še uporabniško ime in geslo, ki sta identična tistima, s katerima dostopa do spletnega vmesnika sistema Zabbix (opozorilo: dostop do programskega vmesnika je pri nekaterih računih izklopljen, omogoči ga lahko skrbnik v nastavitvah spletnega strežnika Zabbix).

Slika 20: Aktivnost Login.

Slika 21: Aktivnost Login z aktiviranim menijem.

Če uporabnik ob prijavi označi možnost »Remember Me« (možnost je privzeto obkljukana), se podatki za prijavo shranijo na SD kartico in jih ob naslednji prijavi ni potrebno ponovno vnašati. Aktivnost vsebuje še izvlečni meni, ki se ga aktivira s pripadajočo tipko »Menu« (Slika 21). Meni ponuja dve možnosti, »Forget login data«, s katero se vsi shranjeni podatki o

naslovu strežnika, uporabniškemu imenu ter geslu zbrišejo, ter »Exit« s katero aplikacijo zapremo.

Ker bi uporabniško ime in geslo lahko potrebovala tudi katerikoli druga aktivnost v aplikaciji, je v aplikaciji priporočljivo ustvariti razred tipa *Application*, ki hrani referenco do razreda skozi celotni življenjski cikel aplikacije, vse dokler aplikacije na zapremo. Ko ustvarimo razred, ki razširja razred *Application*, je potrebno definirati ustrezne spremenljivke ter pripadajoče procedure za nastavljanje ter vračanje podatkov iz razreda. Ko se hočemo v določeni aktivnosti sklicevati na ustvarjeni razred, ustvarimo novo spremenljivko, ki ji vrnemo kontekst (angl. *Context*) naše aplikacije z ukazom `getApplicationContext()`. Nato lahko preko konteksta aplikacije dostopamo do vseh njenih metod ter z njimi spreminjamo ali beremo podatke.

Ob pritisku na tipko »Login« se izvede posebna koda, ki zažene novo nit (angl. *Thread*[16]), ki v določenih intervalih s klici metode `checkNetworkConnection()` preverja, ali imamo vzpostavljeno povezavo s omrežjem. Vse dokler ne vzpostavimo povezave, nas program s pomočjo indikatorja napredka, ki izhaja iz razreda *ProgressDialog*, obvešča o neaktivni povezavi. V kolikor uporabnik ne more vzpostaviti povezave, lahko prijavo prekine z ustrezno tipko za zapiranje aktivnosti »Back«. Ko je povezava z omrežjem vzpostavljena, se preko logike v niti pokliče ustrezno metodo za preverjanje prijave, nit pa se uspešno zaključi.

V primeru, da je v prijavni obrazec uporabnik vnesel napačne podatke, se sproži izjema (angl. *Exception*), ki prikaže ustrezno opozorilo in opis napake, ki jo aplikacija dobi od strežnika v primeru neuspele prijave. Vsa opozorila so že predhodno definirana v programskem vmesniku, zato jih je v aplikaciji potrebno le razčleniti iz odgovora in prikazati na zaslonu.

Ob ustreznih podatkih za prijavo in uspešni prijavi v sistem nas sistem obvesti s kratkim pojavnim sporočilom, ki ga implementiramo s pomočjo razreda *Toast*.

- **Aktivnost Main**

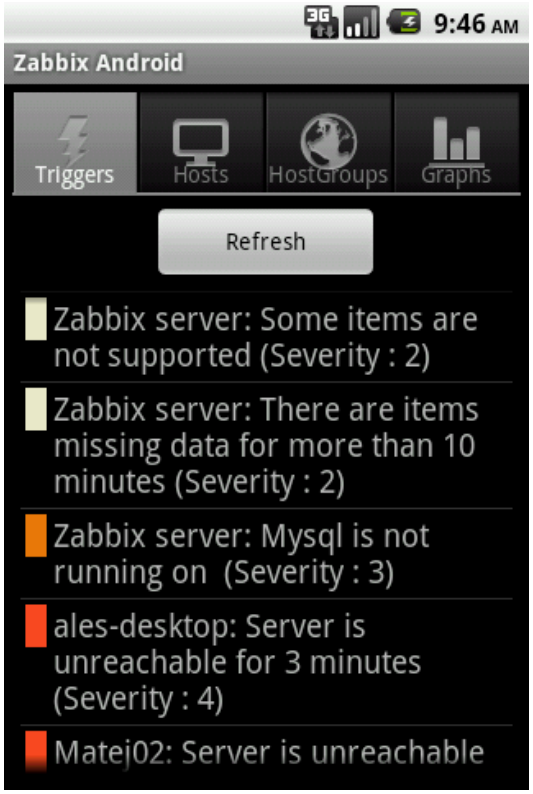
Uspešni prijavi v sistem sledi osrednja aktivnost `MainTabActivity`. Njena posebnost je, da razširja razred `TabActivity`, ki je prilagojen za uporaba gradnika `TabWidget`. Ta ima že definiran poseben pogled, čigar razporeditev elementov spominja na nekakšen brskalnik z zavihki. Njegova odlika je, da ima vse aktivnosti aplikacije prikazane z indikatorji v obliki ikone in lastnim imenom. Vse aktivnosti aplikacije so hitro dostopne z glavne strani, med njimi lahko enostavno preklapljamo s pritiskom na indikator določenega zavihka (Slika 22).

Pri implementaciji gradnika z zavihki ni potrebno definirati nobenega dodatnega poslušalca (angl. *listener*), ki bi nadzoroval, ali je določen zavihek izbran ali ne, saj za to poskrbi gradnik sam. Potrebno je le pridobiti gostitelja zavihkov z ukazom `getTabHost()` ter mu določiti lasten podrazred `TabSpec`, ki predstavlja posamezen zavihek. Temu nato določimo ime in

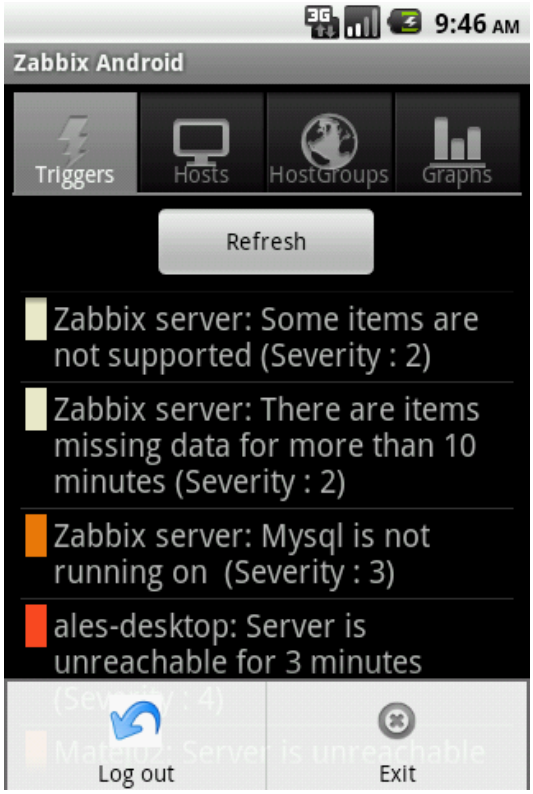
ikono indikatorja ter definiramo, katero namero (angl *intent*) bo izbira določenega indikatorja sprožila.

Inicializacija razreda `TabHost` in dodajanje novega zavihka:

```
// Inicializacija lastnosti zavihka TabSpec za vsak zavihek in dodajanje
le-tega razredu TabHost

tabHost = getTabHost();  // Instanca aktivnosti gostitelja TabHost
TabHost.TabSpec spec;   // Razred TabSpec za opis lastnosti razreda TabHost
Intent intent;          // Določitev namere (Intent) za vsak zavihek


intent = new
Intent().setClass(this,mainTriggerList.class).putExtras(podatki);
//določitev klica razreda za namero, določitev dodatnih podatkov
spec = tabHost.newTabSpec("triggers").setIndicator("Triggers",
       res.getDrawable(R.drawable.ic_menu_flash)).setContent(intent);
// določitev lastnosti zavihka, imena, indikatorja namere
tabHost.addTab(spec);
// dodajanje zavihka gostitelju »tabHost«


// Dodamo še ostale zavihke

intent = new Intent().setClass(this, mainList.class).putExtras(podatki);
spec = tabHost.newTabSpec("hosts").setIndicator("Hosts",
       res.getDrawable(R.drawable.ic_menu_monitor)).setContent(intent);
tabHost.addTab(spec)
```



Slika 22: Aktivnost Main s pogledom podaktivnosti »Triggers«.



Slika 23: Aktivnost Main z aktiviranim menijem.

Prav tako kot aktivnost Login tudi aktivnost Main vsebuje izvlečni meni (Slika 23), ki ponuja možnost »Logout« za takojšno odjavo iz sistema in vračanje na aktivnost Login ter možnost »Exit«za izhod iz aplikacije.

V tem trenutku lahko izbiramo med eno od štirih podaktivnosti (»Triggers«, »Hosts«, »HostsGroups«, in »Graphs«), ki so v nadaljevanju podrobno opisane.

➢ Podaktivnost Triggers

Je privzeto izbrana podaktivnost, ki se prikaže ob uspešni prijavi v sistem. Aktivnost prikazuje seznam vseh sprožilcev (angl. *triggers*), ki se aktivirajo ob pojavitvi nekega izrednega dogodka, ali prečkanje določene pragovne meje nekega senzorja in sprožijo napako (Slika 22). Sprožilce lahko enostavno določamo in spreminjamo v nadzorni plošči spletnega vmesnika Zabbix, v razdelku Configuration| Host | Triggers. Poleg imena odjemalca, na katerem se je zgodila napaka ter kratkega opisa napake, se poleg nje pojavi tudi oznaka, identificirana s pripadajočo barvo in kodo za resnost napake (»Severity«) v opisu posamezne napake (Slika 24). Zabbix pozna pet osnovnih stopenj za označevanje resnosti napake ter eno dodatno za napake, ki še niso kvalificirane.

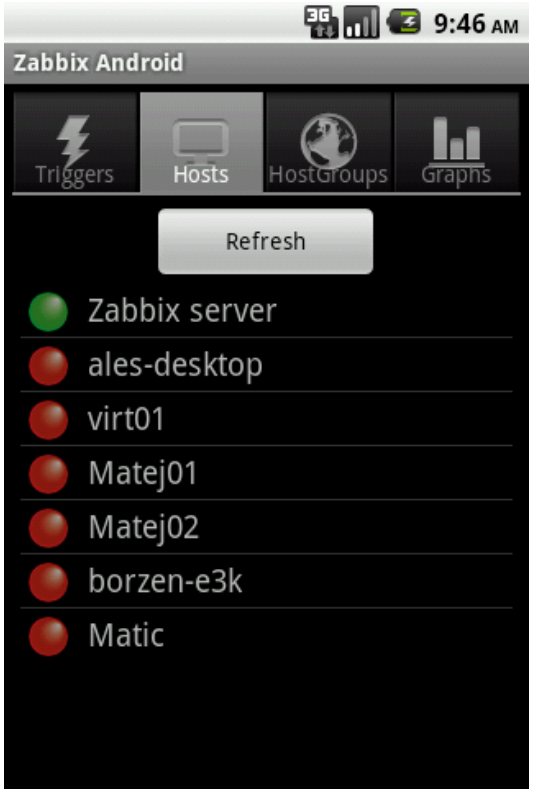
Severity	Status	Name
Disaster	Enabled	SSL certificate on {HOSTNAME} expired
High	Enabled	SSL certificate on {HOSTNAME} expires in less than 7 days ({ITEM.VALUE} days remaining) **Depends on :** Template_zext_ssl_cert : SSL certificate on Template_zext_ssl_cert expired
Average	Enabled	SSL certificate on {HOSTNAME} expires in less than 15 days ({ITEM.VALUE} days remaining) **Depends on :** Template_zext_ssl_cert : SSL certificate on Template_zext_ssl_cert expires in less than 7 days (days remaining)
Warning	Enabled	SSL certificate on {HOSTNAME} expires in less than 30 days ({ITEM.VALUE} days remaining) **Depends on :** Template_zext_ssl_cert : SSL certificate on Template_zext_ssl_cert expires in less than 15 days (days remaining)
Information	Enabled	SSL certificate on {HOSTNAME} expires in less than 60 days ({ITEM.VALUE} days remaining) **Depends on :** Template_zext_ssl_cert : SSL certificate on Template_zext_ssl_cert expires in less than 30 days (days remaining)
Not classified	Enabled	SSL certificate on {HOSTNAME} expires in less than 90 days ({ITEM.VALUE} days remaining) **Depends on :** Template_zext_ssl_cert : SSL certificate on Template_zext_ssl_cert expires in less than 60 days (days remaining)

Slika 24: Tabela stopenj resnosti napake v sistemu Zabbix.

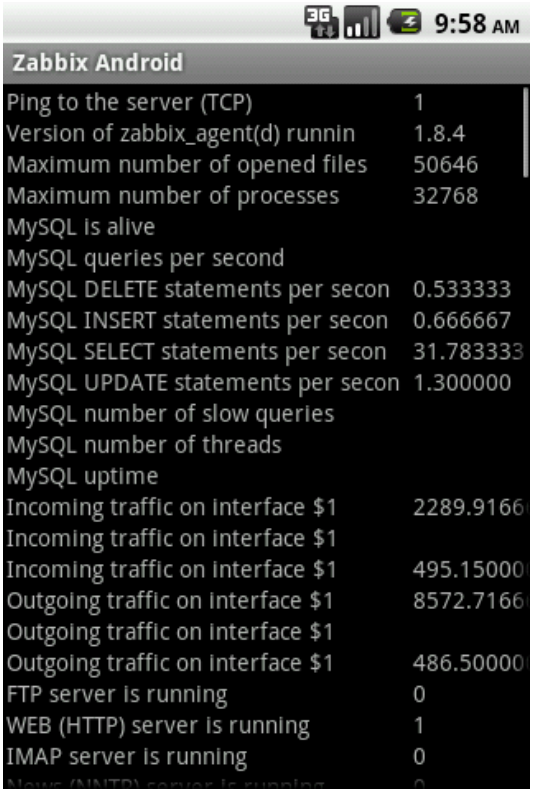
Prirejeno po viru: [18].

- **Podaktivnost Hosts**

Izbrana podaktivnost prikazuje seznam vseh strežnikov, računalnikov, mrežnih naprav ali drugih odjemalcev, ki jih imamo dodane za nadzorovanje (Slika 25). Poleg imena posameznega odjemalca se nahaja tudi ikona, ki identificira, ali je določena naprava pod nadzorom (ikona je obarvana zeleno) ali ne (ikona je obarvana rdeče). Ob izboru posameznega odjemalca s seznama se nam odpre tabela z različnimi informacijami o delovanju in lastnostih izbranega odjemalca (Slika 26).



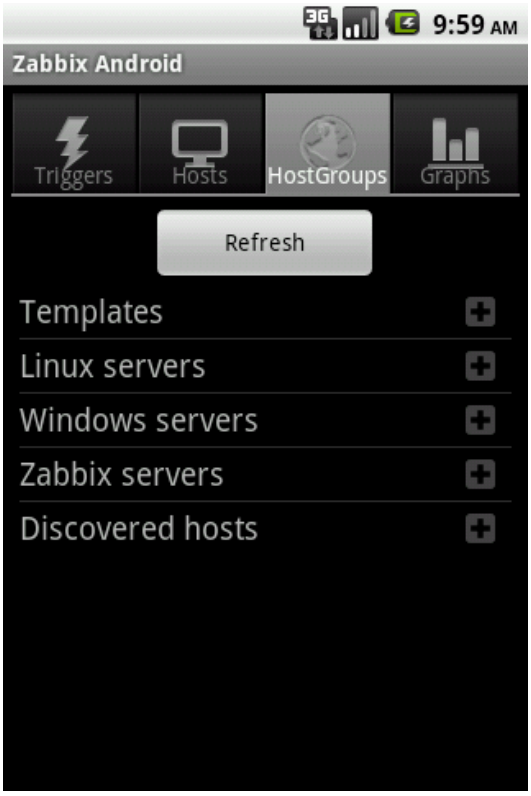
Slika 25: Podaktivnost »Hosts«.



Slika 26: Prikaz tabele podatkov o izbranem odjemalcu.

- **Podaktivnost HostGroups**

Naslednja podaktivnost prikazuje vse skupine odjemalcev, ki so dodani v sistem Zabbix kot nadzorovane (Slika 27). Ob izboru določene skupine se prikažejo vsi odjemalci, ki spadajo v to skupino. Izpis odjemalcev je identičen tistemu pri aktivnosti »Hosts«, ob kliku na posameznega odjemalca se nam pokaže seznam vse njegovih lastnosti. Možno je, da določena skupina nima nobenega odjemalca, ki bi spadal v določeno skupino. V tem primeru se odpre sporočilo o praznem seznamu.

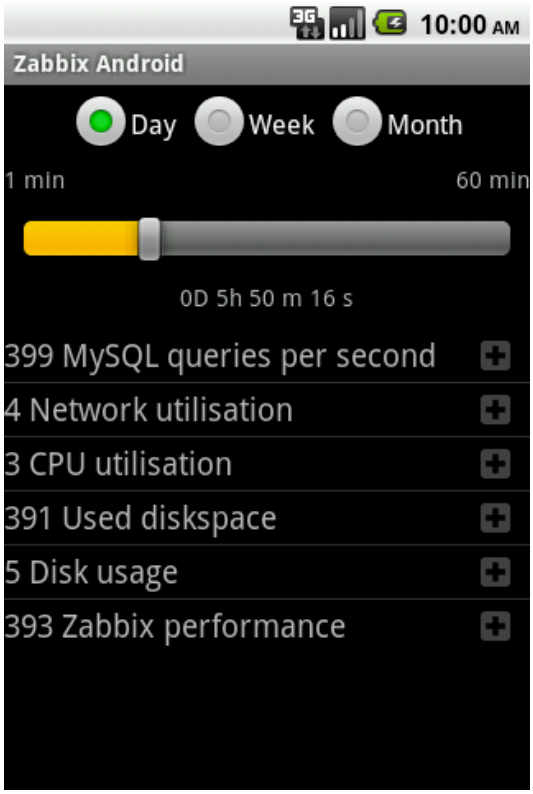


Slika 27: Podaktivnost »HostGroups«.

- **Podatkivnost »Graphs«**

Podaktivnost nam prikaže delano preoblikovani seznam vseh nadzorovanih odjemalcev (Slika 28). Z izbiro določenega odjemalca se nam odpre nova podaktivnost s pregledom vseh tipov grafov (Slika 29), ki jih lahko prikažemo za določenega odjemalca. Vsak tip grafa je prikazan z enolično identifikacijsko številko ter svojim imenom, ki povesta več o pomenu podatkov na obeh oseh.

Slika 28: Podaktivnost »Graphs«.

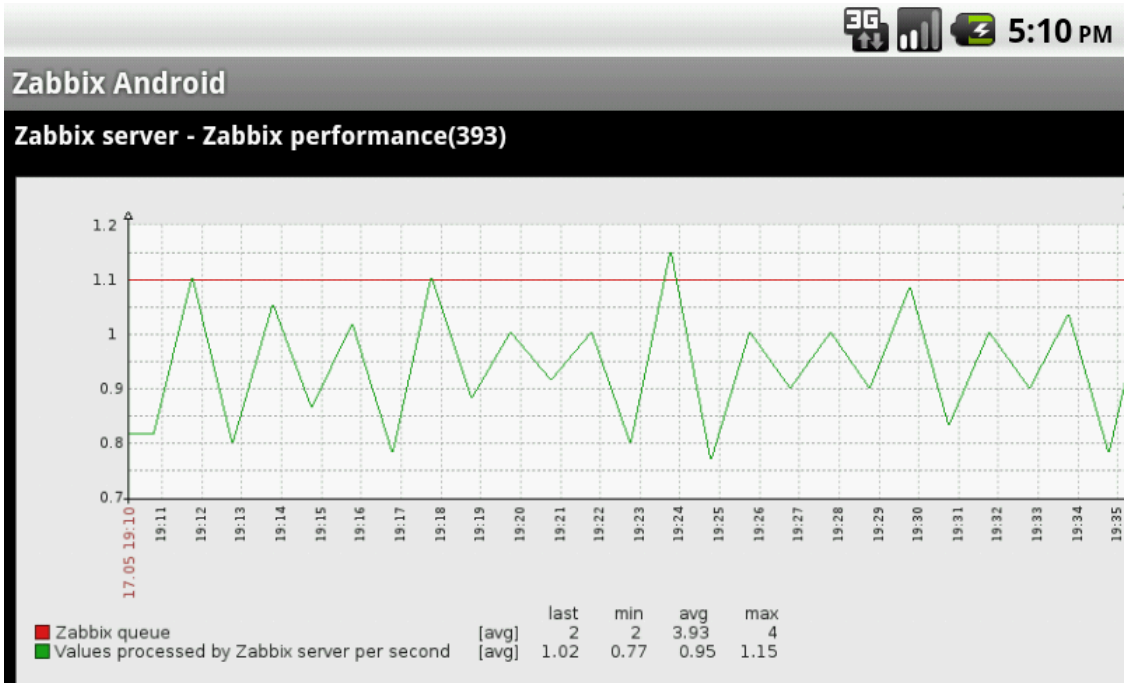


Slika 29: Podaktivnost »Graphs« z možnostjo izbire poljubnega grafa.

Zaradi večje fleksibilnosti vmesnika imamo na voljo tudi tri različne časovne intervale, ki jih izbiramo s pomočjo izključujočih se potrditvenih stikal. Izbira določenega stikala ustrezno poveča širino časovnega intervala, s pomočjo drsnika pa lahko enostavno izberemo časovno obdobje, v katerem želimo prikazati določen graf. Meje trenutnega intervala so označene nad drsnikom, prav tako lahko enostavno odčitamo izbrano časovno obdobje iz tekstovne oznake pod drsnikom.

Z izbiro določene kategorije prikažemo ustrezen graf v izbranem časovnem obdobju (Slika 30). Za boljšo preglednost grafa je napravo priporočljivo orientirati v ležeči položaj. Sliko grafa je možno tudi umeriti (v nadaljevanju skalirati, angl. *scale*) s pomočjo integriranih gumbov za povečevanje in zmanjševanje »+« in »-«, ki se nahajata v spodnjem desnem delu

zaslona. Na večini naprav lahko uporabimo tudi integrirano gesto *pinch-to zoom*, pri kateri pritisnemo s palcem in kazalcem na zaslon ter ju za povečevanje slike pomaknemo narazen, za pomanjševanje pa skupaj.

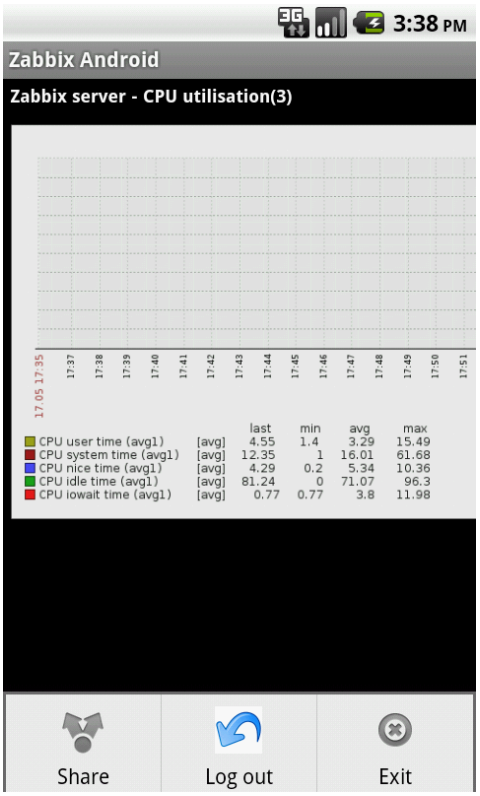


Slika 30: Prikaz izbranega grafa v ležečem načinu.

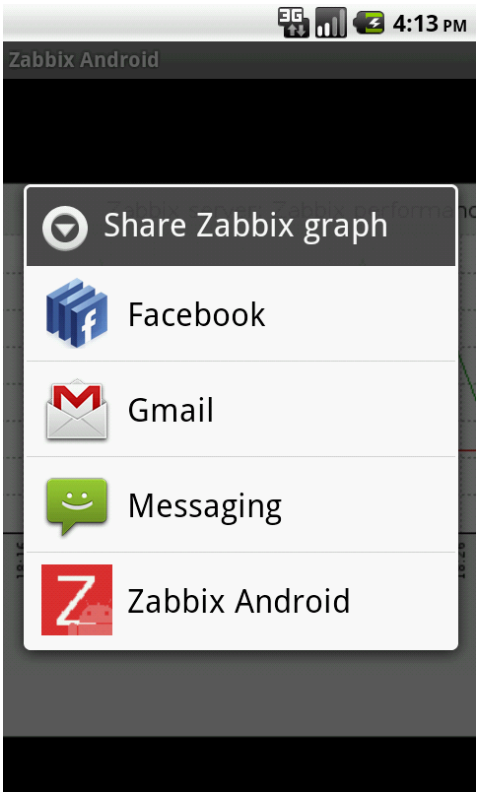
Tudi pri tej podaktivnosti imamo prilagojen meni, ki nam poleg ostalih standardnih funkcij omogoča še dodatno možnost »Share« (Slika 31). Ta kliče posebno namero (angl. *intent*) z že definirano akcijo `Intent.ACTION_SEND`. Ob klicu te namere nam naprava vrne seznam vseh aktivnosti, ki so registrirane za odziv na klic specifične akcije (Slika 32). V našem primeru nam vrne seznam aktivnosti, ki nudijo možnost pošiljanja oziroma objavo slike grafa preko različnih medijev (MMS odjemalec, aplikacija za elektronsko pošto itd.).

Metoda za deljenje slike preko namere:

```
public void Share(){

      Intent share = new Intent(Intent.ACTION_SEND);
      //namera za klic storitve deljenja/pošiljanja

      share.setType("image/jpeg");
      share.putExtra(Intent.EXTRA_STREAM,Uri.parse(SD_PREFIX+SD_FILENAME));
      //določitev tipa podatkov(slika formata jpeg) in določitev slike

      startActivity(Intent.createChooser(share, "Share Zabbix graph"));
      //klic za izbiro aktivnosti deljenje slike grafa
    }
```



Slika 31: Aktivnost »Graph« z menijem.



Slika 32: Klic storitve za deljenje »Share Zabbix graph«.

5.5. Optimizacija aplikacije

Veliko pozornosti je bilo namenjeno tudi optimizaciji podatkovnih in grafičnih struktur. Nekatere naprave so namreč strojno omejene ali v nekem času le preveč obremenjene z ostalimi aplikacijami. V takih primerih pridejo optimizacije še bolj do izraza. Ker včasih v aplikaciji pride do izrednih dogodkov, ki sprožijo napake in v večini primerov povzročijo sesutje aplikacije, smo se trudili, da do takih dogodkov ne bi prišlo. Če se napaka vseeno zgodi, aplikacija uporabnika samo opozori o vrsti napake ter poskuša delovati nemoteno naprej.

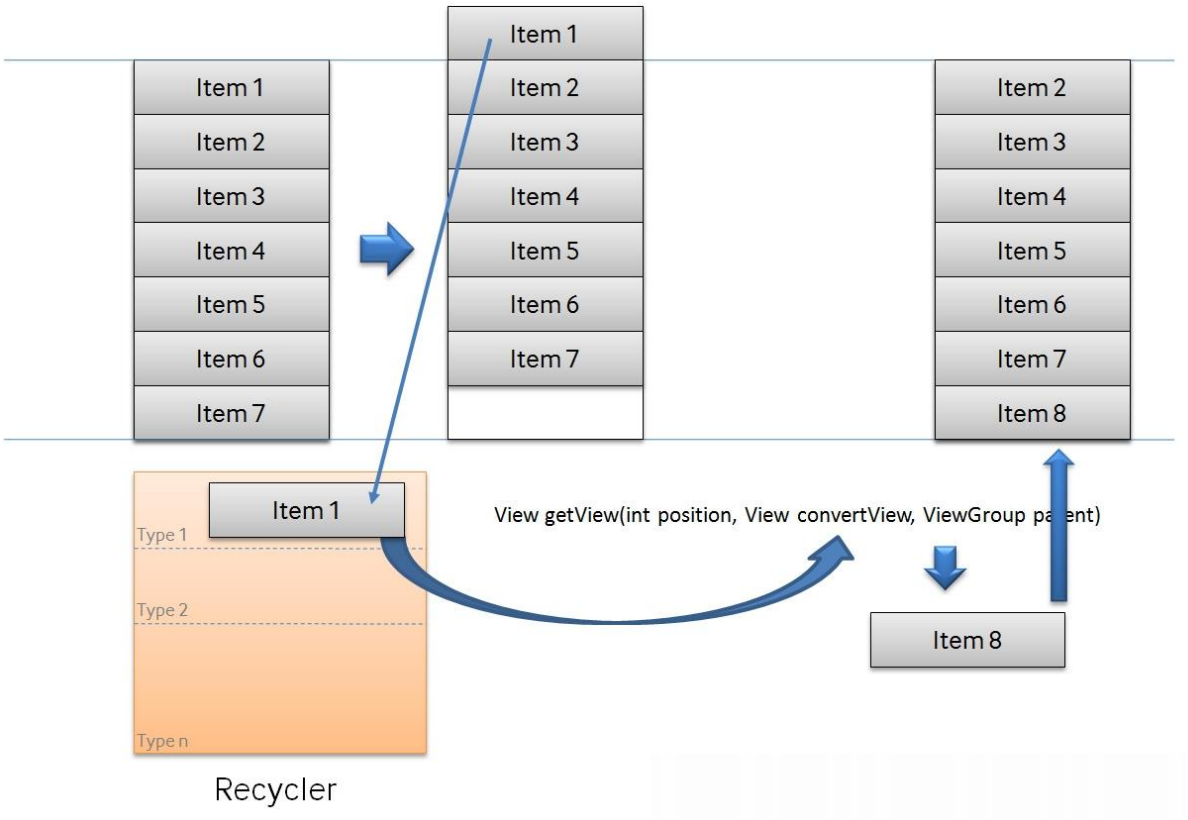
Tekom projekta so bile uporabljene naslednje optimizacije:

- Recikliranje izdelanega pogleda ter uporaba razreda *ViewHolder*

V aplikaciji se veliko uporablja razred *ListView* za prikaz podatkov v obliki seznama elementov, razporejenih po vrsticah. Elemente seznama s pomočjo adapterja razberemo iz izbrane tabele, podatkovne baze ali drugih virov in dinamično dodamo vse zapise iz vira v seznam. V primeru, da želimo spremeniti izgled seznama, je potrebno v datoteki XML opisati izgled in postavitev elementov v posamezni vrstici. V datoteko poljubno dodajamo ustrezne elemente, urejamo njihovo postavitev in jim dodelimo identifikacijski element (imenovan »android:id«). Sedaj je potrebno adapterju še določiti, da bere podatke v ustrezne elemente novoustvarjene vrstice. To pomeni, da je preko standardne metode adapterja *getView* za vsako vrstico treba ustvariti nov pogled (angl. *View*) v obliki nove vrstice, vse elemente vrstice referencirati in jim določiti ustrezne podatke ter preko razreda *LayoutInfalter* pogled vrstice dodeliti našemu seznamu.

Takšen pristop ni napačen in deluje. Problem lahko nastane pri pogostem osveževanju podatkov in pomikanju po seznamu. Adapter namreč ne ustvari vseh pogledov vrstic naenkrat, temveč le tiste, ki jih lahko prikaže na zaslonu, kar skrajša čas, ki preteče od zahteve po seznamu do prikaza le-tega na zaslonu. To seveda pomeni, da bo adapter pri pomikanju ustvarjal nove in nove poglede, tudi tiste, ki jih je pred časom že izdelal. Tu si pomagamo z razredom *ViewHolder*[20] in metodama razreda *View*, *setTag()* in *getTag()*.

Optimizacija deluje tako, da vsak izdelan pogled z referencami do elementov pogleda shrani v primerek razreda *ViewHolder* in pogledu dodeli oznako (angl. *tag*). Ko se premikamo po seznamu, sedaj ne ustvarjamo že obstoječih pogledov z isto vsebino, ampak obstoječ pogled s klicem metode *getTag()* samo zamenjamo (angl. *recycle*) z že ustvarjenim pogledom (Slika 33).



Slika 33: Recikliranje vrstice z vsebino in njena ponovna umestitev v seznam.

Prirejeno po viru: [21].

- Možnost dodajanja slike gradniku *TextView* ter gnezdenje tipov razporeditve

Pri izdelovanju uporabniškega vmesnika v Androidu moramo biti pozorni pri uporabi ustreznega tipa postavitve gradnikov (angl. *layout*) v našem programu in morebitnemu gnezdenju tipov postavitev. Samo gnezdenje tipov postavitev sicer ne predstavlja posebnega problema, tudi kompleksne postavitve s različnim nivojem gnezdenja in številom gradnikov se glede ustreznost postavitve normalno izrišejo. Vseeno pa je potrebno vedeti, da se z vsakim novim dodatnim (nepotrebnim) tipom postavitve povečuje čas izrisa posamezne uporabniškega vmesnika. Za analizo postavitve uporabniškega vmesnika lahko uporabimo orodje *Hierarchy Viewer*, ki je na voljo kot paketna datoteka, ki se nahaja na lokaciji *./tools/ hierarchyviewer.bat* v instalacijskem direktorju našega razvojnega ogrodja ter v razvijalskem okolju Eclipse pod pogledom DDMS (Slika 18).

S pomočjo orodja lahko v nekaterih primerih našo kodo spremenimo na tak način, da dosežemo rezultat gnezdenega tipa razporeditve le z uporabo posebnega tipa razporeditve ali preprostega gradnika, ki za enak rezultat ne potrebuje gnezdenja razporeditev. Temu enostavno pripnemo vse gnezdene gradnike, ki se v uporabniškem vmesniku vseeno nahajajo na enakih pozicijah in s tem ohranijo postavitev. Primer tega je gradnik TextView, ki poleg osnovne funkcije, prikazovanje tekstovne označbe, omogoča tudi pripenjanje slike na štirih

sosednjih pozicijah (levo,zgoraj, desno,spodaj). V primeru, da želimo ustvariti vrstico s tekstom in sliko, tako ni potrebno definirati posebnega tipa razporeditve, ki bi vseboval gradnik tekstovne označbe in gradnik slike, ampak ga enostavno nadomestimo z enim gradnikom tekstovne označbe, ki nase veže dodatni gradnik slike. S tem smo pri izrisu naše postavitve prihranimo en gradnik – tip postavitve.

5.6. Testiranje aplikacije

Za pravilno delovanje aplikacije na različnih napravah z različnimi strojnimi specifikacijami in velikostmi zaslona je potrebno slediti osnovnim načelom pri programiranju v operacijskem sistemu Android[22]. Priporoča se uporaba linearne (angl. *linear layout*) ali relativne razporeditve (angl. *relative layout*) namesto absolutne razporeditve (angl. absolute layout), kjer namesto fiksne lokacije elementa določimo, koliko odstotkov zaslona zasede posamezen del in za koliko je odmaknjen od drugih elementov. Za ta postopek se priporoča uporaba navidezne enote *density independent pixel*, ki za osnovo vzame velikost zaslonske pike (v nadaljevanju piksel) na standardnem zaslonu z gostoto 160 *dpi* (število pikslov na enoto inč, angl. *dots per inch*), nato pa preveri še našo resolucijo zaslona ter velikost piksla ustrezno priredi. Rezultat je odmik, ki je proporcionalen velikosti in resoluciji zaslona. Podobno velja tudi za izrisovanje slik. Da bi se izognili postopku skaliranja vsakega posameznega elementa ob izrisu, ki je relativno draga operacija, imamo za različne velikosti zaslonov v imeniku `/drawable/` našega projekta že predpripravljane mape z označbami gostot zaslona. V te je potrebno umestiti iste slike, a v različnih resolucijah (v razdelku *Supporting Multiple Screens* uradne spletne strani Android SDK[19] so na voljo uradni dogovori o tem, kako je potrebno skalirati slike in v katero mapo jo umestimo).

Pri testiranju je najlažje uporabiti fizično napravo z Androidom. Ker pa je v zadnjem času na trgu veliko naprav z različnimi velikostmi zaslona in verzijami operacijskega sistema Android, se za takšno preizkušanje priporoča uporaba emulatorja. Takšne virtualne naprave, ki tečejo na računalniku in se uporabljajo za razvijanje in testiranje aplikacij, imenujemo tudi AVD-ji (angl. *Android Virtual Device*). S pravilnimi konfiguracijami je možno aplikacijo testirati pod pogoji, ki jih je težko poustvariti na fizični napravi. Tudi delovanje in interakcija s sistemom sta v principu podobna delovanju fizične naprave, le da uporabljamo miškin kazalec namesto fizičnega dotika zaslona, tipkovnico za morebiten vnos besedila ter razne dodatne funkcije, kot so spreminjanje orientacije zaslona ali pritisk tipke za klic ali izklop telefona.

Program je bil testiran na različnih virtualnih napravah:

- Android 2.2, ločljivost 240X320, ldpi,
- Android 2.2, ločljivost 320X480, mdpi,
- Android 2.2, ločljivost 480X800, hdpi.

Za testiranje smo uporabili tudi naslednje fizične naprave:

- HTC Wildfire (Android 2.2.1, zaslon 3.2", ločljivost 240X320 pik),
- HTC Desire HD (Android 2.3, zaslon 4.3", ločljivost 480X800 pik),
- HTC Flyer (Android 2.3.1, zaslon 7", ločljivost 1024X600 pik),
- Asus EEE Transformer (Android 3.2, zaslon 10", ločljivost 12800X800 pik).

Pri razvijanju in testiranju aplikacije smo opazili, da je že osnovno delovanje emulatorja kljub sodobni strojni opremi (štirijedrni procesor Intel Core i7, 6 Gb rama, Intel SSD disk) relativno počasno, saj ta med delovanjem zasede velik delež sistemskih virov. Tudi s hitrejšo strojno opremo ne bi veliko pridobili, emulator namreč teče na operacijskem sistemu računalnika, a podobno kot fizična naprava uporablja ARM arhitekturo s pripadajočimi ukaznimi kodami. Zaradi tega jih mora pretvoriti v enakovredne x86 ukaze, s katerimi lahko upravlja tudi naš procesor. Fizične naprave imajo pogosto tudi ločene grafične pospeševalnike ki skrbijo za izris grafike in grafičnih elementov. Emulator te lastnosti ne omogoča ne glede na to, katero grafično kartico imamo v računalniku. Poleg tega fizična naprava vse vhodno-izhodne podatke piše v lastni flash ROM pomnilnik, ki je veliko hitrejši od zapisa v navidezno diskovno datoteko emulatorja, katere hitrost je dodatno odvisna od tehnologije delovanja diska.

Aplikacijo smo imeli priložnost testirati tudi na dveh tabličnih računalnikih z velikostjo zaslona 7" in 10". Pri obeh je aplikacija delovalo normalno. Zaradi uporabe ustreznih gradnikov za postavitev in uspešne prilagoditve uporabniškega vmesnika za različne velikosti zaslona in ločljivosti se je le-ta izrisal zvezno in nepopačeno ter s tem ohranil vse funkcionalnosti.

5.7. Težave pri razvoju

Pri razvijanju aplikacije se je pojavilo tudi nekaj težav bodisi na strani programa Zabbix in njegovega programskega vmesnika bodisi pri uporabi in razširjanju standardnih knjižnic in razredov v Androidu. Vse težave so bile tekom razvijanja aplikacije omiljene ali v celoti odpravljene.

- Pridobivanje slike grafa preko programskega vmesnika

Prva težava se je pojavila ob pridobivanju grafov za določen strežnik. Zabbixov programski vmesnik sicer omogoča pridobivanje podatkov o vseh grafih, ki jih lahko prikažemo za določen strežnik ali napravo, vendar se slike grafa preko klica posamezne metode programskega vmesnika ne da enostavno pridobiti. Problem je bil rešen z zahtevo do ustrezne strani PHP (angl. *Hypertext Preprocessor*), ki se drugače kliče v spletnem vmesniku in ob uporabi ustreznih podatkov v odgovor vrne sliko v formatu PNG (angl. *Portable Network Graphics*). Poleg tega moramo biti za takšno zahtevo tudi uspešno prijavljeni v sistem, kar strežnik ugotovi tako, da preveri prijavni piškotek (angl. *login cookie*), ki je zahtevi pripet.

Da bi lahko realizirali takšno zahtevo v mobilni aplikaciji, smo uporabili ustrezno knjižnico, ki omogoča uporabo programske arhitekture REST[17] (angl. *Representational state transfer*). Tako smo ukaz realizirali z enim od standardnih klicev te HTTP POST, ki ob zahtevi do ustrezne prijavne strani PHP in ustreznih uporabniških podatkih vrne prijavni piškot, ki ga shranimo v shrambo za piškote, kot lahko imenujemo razred *Cookie Storage*. Nato za vsako zahtevo po določenem grafu poleg zahteve same iz shrambe piškotov vzamemo naš prijavni piškot ter ga pripnemo zahtevi. Rezultat zahteve je ustrezna slika grafa posamezne naprave.

- Uporaba gradnika *TabActivity* za izdelavo zavihkov

TabActivity je standardni gradnik za implementacijo posebnega uporabniškega vmesnika, ki ima praviloma od tri do štiri glavne aktivnosti, ki so predstavljene kot zavihki z ikonami. Seveda je možno dodati še več ikon, a pri napravah z manjšimi zasloni lahko hitro pride do prenasičenosti, razen če uporabimo drsnik. Njegova posebnost je, da se aktivnosti preprosto preklapljajo s pritiskom na ustrezno ikono in imenom, ki predstavlja določeno funkcijo v programu. Uporabnik tako v vsakem trenutku vidi vse možne funkcije programa in lahko s pritiskom na ikono preklaplja med njimi, trenutno izbrana aktivnost pa je jasno vidna.

Težava nastane, ko hočemo iz trenutno vidne aktivnosti poklicati novo aktivnost. Pri klicu same aktivnosti seveda ne pride do težav, se pa ob tem naloži nov pogled aktivnosti, ki prekrije obstoječi pogled aktivnosti z zavihki, tako da interakcija z zavihki do zaprtja trenutne aktivnosti ni več možna. Problem bi lahko rešili tako, da bi se v vsakemu zavihku izvajala le

ena aktivnost, ob dogodkih v aktivnosti pa bi se menjali le pogledi. Po kratkem premisleku vmesnika nismo spreminjali, saj se posamezna aktivnost ne veji v toliko aktivnosti, da bi bilo to potrebno.

- Klic *getItems()* za dostop do vseh postavk določene naprave

Eden od možnih klicev zahteve predvideva tudi vračanje posameznih informacij o odjemalcu, kjer je vsaka informacija identificirana s svojim objektom, poimenovanim *Item*. Ta objekt vsebuje veliko informacij, kot so identifikacijska številka, lastno ime, seznam preteklih vrednosti in ostale. Problem nastane takrat, ko bi želeli vrniti le ime in trenutno vrednost določenega *Itema*, saj ne moremo izbrati nobenega filtra v zahtevi, ki bi vrnil le te dve informaciji. Posledica tega je poleg velike količine prenesenih podatkov z vsemi objekti in njihovimi informacijami tudi precejšen porabljen čas za razčlenitev vseh podatkov v teh objektih. Težava je opazna zlasti pri napravah s slabšo strojno opremo, ki bi v času razčlenitve podatkov kazala znake neodzivnosti, kar je za uporabnika lahko zelo moteče.

Edina rešitev bi bila možnost filtriranja določenih informacij za posamezni objekt *Item*, ki bo mogoče na voljo v naslednjih različicah programa Zabbix. Težavo bi bilo mogoče omiliti tudi z nastavljanjem posebnega filtra, ki določenih objektov ne bi razčlenjeval, a s tem bi funkcijo, ki naj bi prikazala vse informacije o odjemalcu, preveč oklestili.

Težavo z navidezno neodzivnostjo smo sicer omilili s pomočjo razreda *ProgressDialog*, ki ob klicu metode, dostopanju in prenašanju podatkov ter njihovem razčlenjevanju prikaže standardni indikator napredka, ki ga lahko uporabnik tudi kadarkoli prekine med delom ter se vrne na prejšnjo aktivnost. Ob uspešni operaciji se le-ta zaključi, podatki pa se prikažejo na zaslonu.

6. VIRI IN LITERATURA

[1] N. Agoulmine, *Autonomic Network Management Principles*, Burlington: Academic Press; First edition, 2010, pogl. 1.

[2] A. Clemm, *Network Management Fundamentals*, Indianapolis: Cisco Press, 2007.

[3] R. Olups, *Zabbix 1.8 Network Monitoring*, Birmingham: Packt Publishing Ltd., 2010.

[4] (2011) Zabbix Download. Dostopno na: http://www.zabbix.com/download.php.

[5] D. Mauro, K. Schmidt, *Essential SNMP*, Cambridge: O'Reilly Media; Second Edition, 2005.

[6] (2011) VirtualBox, a powerful x86 and AMD64/Intel64 virtualization product. Dostopno na: http://www.virtualbox.org/.

[7] (2011) OpenWrt: a Linux distribution for embedded devices. Dostopno na: https://openwrt.org/.

[8] Wei-Meng Lee, *Android™ Application Development*, Indianapolis, 2011, str. 57-217.

[9] (2011) Java SE Downloads. Dostopno na: http://www.oracle.com/technetwork/java/javase/downloads/index.html.

[10] (2011) Android SDK Download. Dostopno na: http://developer.android.com/sdk/index.html.

[11] (2011) Putty: Telnet and SSH client. Dostopno na: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html.

[12] (2011) Razvoj programov v okolju Eclipse. Dostopna na: http://hpc.fs.uni-lj.si/eclipse.

[13] (2011) ADT Plugin for Eclipse. Dostopno na: http://developer.android.com/sdk/eclipse-adt.html.

[14] (2011) Subclipse for Eclipse. Dostopna na: http://subclipse.tigris.org/.

[15] (2011) JSON (JavaScript Object Notation). Dostopno na: http://json.org/.

[16] (2010) Java Docs; Thread. Dostopno na: http://download.oracle.com/javase/1.4.2/docs/api/java/lang/Thread.html.

[17] L.Richardson, S. Ruby, D. Heinemeier Hansson, *RESTful Web Services*, Cambridge: O'Reilly Media; First Edition, 2007, pogl. 2, 3, 4.

[18] (2011) Zabbix Triggers. Dostopna na: http://aperto.fr/cms/images/zabbix/zabbix-ssl-cert-triggers.png.

[19] (2011) Android Dev Guide: Supporting Multiple Screens. Dostopno na: http://developer.android.com/guide/practices/screens_support.html.

[20] S. Komatineni, D. MacLean, S. Hashimi, *Pro Android 3*, New York: Apress, 2011, str. 188-193.

[21] (2011) HowTo: ListView, Adapter, getView and different list. Dostopno na: http://android.amberfog.com/?p=296.

[22] (2011) Android Developers: Hello, Views. Dostopna na: http://developer.android.com/resources/tutorials/views/index.html.

KAZALO SLIK