SANET - rilascio di un prodotto per il monitoraggio delle reti

Release 0.7.0

Luca Ferroni

27 November 2009

Indice

1 Introduzione **3**

2 In quale contesto nasce l'idea di SANET ? **5**
2.1 Chi . . . 5
2.2 Da cosa si parte . . . 5
2.3 L'esigenza . . . 6
2.4 Quando . . . 7

3 Le fasi di sviluppo **9**
3.1 Un primo tentativo fallito . . . 9
3.2 Fase 0: una migrazione graduale . . . 9
3.3 Fase 1: Django + RDBMS . . . 10
3.4 Fase 2: CLI e mappe . . . 11
3.5 Fase 3: Poller . . . 13
3.6 L'impulso del Master FOSSET0809 . . . 14

4 Uno sforzo importante: le mappe **17**
4.1 La scelta fra SVG o Flash . . . 17
4.2 La topologia della rete . . . 17

5 Rilasciare SANET alla comunità **25**
5.1 Le scelte principali . . . 25
5.2 Il primo rilascio alla ConfSL09 . . . 26
5.3 Andando verso il bazaar... . . . 28
5.4 Il secondo rilascio al termine del Master FOSSET0809 . . . 29

6 Verifica e prospettive **31**
6.1 Verifica . . . 31
6.2 Prospettive . . . 32

7 Appendice A - Osservazioni su NMS open source in lista cisco-nsp **33**

8 Appendice B - analisi della sicurezza di SANET **37**
8.1 Attacchi esterni . . . 37

8.2 Attacchi interni 38
8.3 Concludendo 39

9 Appendice C - SVG o FLASH ? 41

10 Appendice D: XML prodotto per una mappa con contenitori aperti 43

11 Indices and tables 47

Contents:

CAPITOLO 1

Introduzione

SANET è l'acronimo di Security Architecture NETwork ed è il software di monitoraggio per le reti sviluppato da Laboratori Guglielmo Marconi S.p.A. (LABS) a partire dal 2006.

Il 12 giugno 2009 SANET è stato rilasciato da LABS con la licenza di software libero AGPLv3 alla pagina http://sanet.sourceforge.net.

In questo documento si illustra il processo che ha portato alla realizzazione di SANET, le motivazioni del progetto, gli obiettivi che raggiunge, le fasi di sviluppo e le decisioni relative al rilascio come software libero.

Non si intende scrivere un manuale utente del software. Installazione ed utilizzo sono descritte nella documentazione online. Una panoramica del software con le sue peculiarità è riportata in appendice nell'articolo di presentazione scritto per la CONFSL09.

In particolare relativamente al rilascio della seconda versione viene mostrato il lavoro svolto nella **rappresentazione della mappe topologiche della rete** e il **processo di pacchettizzazione** avviato per facilitare l'installazione del software ad utenti meno esperti di sistemi.

Il tutto è concluso da una verifica del lavoro svolto e possibili future direzioni, sia dal punto di vista tecnico, sia di approccio alla comunità.

Viene riservata una corposa appendice con:

- una analisi sulla sicurezza del software

TODO * l'articolo di presentazione di SANET alla CONFSL09 * una mail girata nella mailing-list cisco-nsp che presenta alcuni aspetti tecnici avanzati soddisfatti da SANET * l'analisi comparativa effettuata nella scelta fra le tecnologie SVG e FLASH.

Buona lettura

In quale contesto nasce l'idea di SANET ?

In questo capitolo vedremo in quale contesto nasce SANET. Quando si è deciso di svilupparlo, chi lo ha deciso.

Cosa si è voluto realizzare e da cosa si è partiti, e perché è stato avviato un nuovo progetto evitando di adattare un programma open source esistente alle proprie esigenze.

2.1 Chi

LABS è un'azienda che si occupa di reti. Segue il processo di messa in funzione di una rete a partire dalla progettazione dell'infrastruttura fisica, per poi passare al livello rete, fino ad arrivare ai servizi.

Il cosiddetto *core business* dei LABS è l'assistenza sulle reti. Le reti hanno bisogno di monitoraggio e manutenzione continui e in reti complesse non si può fare a meno di avere tecnici specializzati per risolvere le più disparate problematiche che si presentano.

I problemi si possono verificare ad esempio a causa di:

- retaggi del passato
- deperimento dell'hardware
- cause estemporanee (ad esempio bachi in aggiornamenti software oppure gestione ntp leap second ?)
- worm/attacchi nella rete

2.2 Da cosa si parte

Il sistema di monitoraggio preesistente prendeva il nome di *pinger* ed era un monolitico script in Perl che disponeva di un linguaggio per la definizione dei controlli.

Il linguaggio era abbastanza basilare, ma consentiva di definire modelli abbastanza comodi per istruire il sistema su una serie di controlli da effettuare.

Il pregio di questo software consisteva proprio nella definizione duttile e dettagliata dei parametri da verificare per ogni controllo: a quali intervalli di tempo effettuarli, per quale soglia di tolleranza e a chi veicolare le segnalazioni. Tutto ciò in un modo relativamente semplice.

I difetti invece si possono riassumere in:

- incomprensibilità dei risultati dell'attività di monitoraggio per il cliente
- interfaccia grafica pressoché inesistente
- caricamento in memoria di tutti i controlli da effettuare e riavvio del software in caso di modifica della configurazione
- mancanza di *escalation* ossia di dipendenza dei controlli e quindi flood di allarmi per un unico problema
- limitazioni varie dovute alla ridotta espressività del linguaggio e del parsing dello stesso
- multiprocesso, ma non multithread con conseguente sovraccarico di memoria
- parsing dei controlli non resistente a *code injection*

È da notare che il tutto si è sempre basato su GNU/Linux e strumenti open source e anche la realizzazione stessa rimaneva open: i clienti più smaliziati potevano andare a curiosare nel codice e suggerire modifiche oltre ovviamente a definire i propri controlli.

Tuttavia non si può dire che questa situazione fosse ben definita, dato che, di fatto *pinger* era da sempre installato su appliance LABS e LABS stesso non ha mai esplicitato la licenza dello strumento in quanto era funzionale al servizio di assistenza delle reti che diveniva oggetto principale del contratto con il cliente.

Da questa considerazione si può desumere che LABS è sempre stato autore più o meno consapevole di software open, ma non aveva ancora tenuto in considerazione la possibilità di condividere apertamente la propria conoscenza, di creare una comunità intorno all'attività di monitoraggio delle reti che ha da sempre costituito un punto di eccellenza per l'azienda.

Ciò ovviamente era anche dovuto all'assenza di un settore di sviluppo software e quindi al modo *artigianale* in cui veniva sviluppato *pinger*. Per capire basta sapere che il versionamento del software avveniva tramite *diff* e *patch* ... che, a onor del vero, lo stesso Linus Torvalds ritiene un meccanismo ben più evoluto di CVS (si veda il video Linus Torvalds on Git), ma che comunque rasenta la definizione di versionamento.

In ultimo è necessario osservare che LABS stava avviando in quel periodo una piccola divisione di sviluppo software per portare avanti altri progetti.

2.3 L'esigenza

Le doti del sistema di monitoraggio in uso erano confermate dai casi di successo riscontrati presso i clienti LABS che tipicamente hanno sempre presentato infrastrutture di rete mediamente o molto complesse.

Da qui si è deciso di intraprendere una nuova strada, o meglio, di consolidare la strada già intrapresa: reingegnerizzare il sistema di monitoraggio valorizzandone i pregi, superandone i limiti e colmandone le lacune.

Si sa che nello sviluppo software "l'appetito vien mangiando" e, anche se non si aveva l'ambizione di realizzare il software di monitoraggio perfetto, si è capito che ristrutturando quello che già c'era a disposizione si sarebbe potuto:

- soddisfare nuove esigenze
- ottenere uno strumento più facile da mantenere ed estendere
- potenziare l'espressività e la tassonomia dei controlli da effettuare
- adattarsi o integrare nuove tecnologie

Non a caso la **soddisfazione** di nuove esigenze **del cliente*+ è stata elencata al primo posto: questa **è la motivazione che spinge le aziende a investire e innovare**.

Anche nel caso dei LABS, sebbene le motivazioni fossero molteplici, il momento dell'investimento, il cosiddetto *trigger* è stata la richiesta sempre più insistente di un'interfaccia grafica comprensibile: il cliente lamentava di non

essere consapevole in alcun modo della situazione della propria rete e che le informazioni estremamente dettagliate e tecniche fornite dalla scarna interfaccia di *pinger* non fossero per lui di alcuna utilità.

Quindi la decisione è stata di aprire gli orizzonti, coinvolgere il gruppo di sviluppatori, e progettare un software per il monitoraggio delle reti che superasse i limiti di *pinger* (con priorità per i limiti di rappresentazione grafica dei risultati), mantenesse il know-how pluriennale dell'azienda e anche il proprio *modus operandi*.

Trattandosi questa volta di un software vero e proprio più che di un insieme di script si inizia a valutare la possibilità di rilasciare il lavoro alla comunità. In azienda il terreno è fertile: si comprende l'importanza di condividere conoscenza, la ricchezza del servizio, della cultura e della prestazione professionale invece che il valore artificioso della licenza centellinata e pure si coglie marginalmente il vantaggio reciproco (per l'azienda a la comunità) che ne sarebbe potuto derivare.

La strada del software libero non è già tracciata, ma inizia a diffondersi nell'aria. Un merito che bisogna dare ai LABS è che continuando con contratti sul servizio di assistenza alle reti, non sarebbe sussistito alcun obbligo di rilascio del software: infatti tale software non sarebbe stato ceduto al cliente, ma sarebbe rimasto uno strumento di proprietà dell'azienda per offrire il servizio concordato. Inoltre la presentazione dei risultati sarebbe avvenuta tramite interfaccia web e quindi anche in questo caso si sarebbe rimasti nel pieno della legittimità a meno di integrare software basato su licenza Affero GPLv3 o con simili requisiti a tutela di libertà per l'utente.

2.4 Quando

Il lavoro di analisi, definizione obiettivi e progettazione della nuova architettura è iniziato nel 2005.

I più diffusi sistemi open source esistenti per il monitoraggio delle reti erano già conosciuti dagli esperti di rete LABS e non si è ritenuto opportuno fare un'approfondita valutazione di integrabilità con l'esistente.

In quel periodo ZenOSS , uno dei progetti open source più frequentati su SourceForge.net e più attivi al momento della scrittura di questo documento, non era ancora stato avviato.

Quindi, considerando anche l'ampia base di controlli sviluppati dall'azienda, i casi di successo riscontrati e le modalità operative assodate, ci si è diretti verso lo sviluppo di una nuova soluzione, accettando in questo modo di non approfittare di comunità già consolidate sui temi del network management.

Questa scelta ha ovviamente avuto un impatto decisivo sull'impostazione dello sviluppo della nuova piattaforma che voleva a questo punto evolvere dalla precedente senza rompere con il passato per continuare appunto, a consolidare la strada già intrapresa.

Si presentava lo scenario di rimpiazzare l'auto in corsa: non interrompere il servizio, ma sostituire gradualmente parti di software migliorandone gradualmente la qualità complessiva.

CAPITOLO 3

Le fasi di sviluppo

Evolvere un programma che porta con se i retaggi di un passato decennale oltre che l'approccio prettamente pratico del sistemista non è un percorso semplice, soprattutto se si intende reingegnerizzare il tutto in modo da minimizzare la rottura con il passato e farne un software manutenibile in cui sia chiara la separazione delle componenti.

Si decide di partire con lo sviluppo di *SANET*.

3.1 Un primo tentativo fallito

In principio si è approcciato il problema in modo abbastanza superficiale: presa coscienza delle parti da sostituire, si è scelto il linguaggio Python già in uso nell'azienda per altri progetti e si è demandata la prima stesura del nuovo codice ad un collaboratore temporaneo.

Questo approccio si è dimostrato fallimentare perché:

- aveva l'ambizione di sostituire in breve tempo tutta la vecchia piattaforma
- non si appoggiava ad alcun framework che potesse velocizzare lo sviluppo focalizzando l'attenzione sulle peculiarità di SANET
- basava la sua implementazione su uno storage in memoria e anche qui ne é derivata una perdita di tempo ed energie in problemi di concorrenza e persistenza delle informazioni
- si voleva implementare una politica di ereditarietà molto complessa fra istanze madri, categorie, e istanze collegate: in pratica si stava riscrivendo un linguaggio a oggetti

e inoltre non possiamo trascurare che dal punto implementativo queste sfide erano state sottovalutate dall'azienda che non ha dedicato risorse umane per seguire costantemente il collaboratore temporaneo che si è trovato probabilmente smarrito con in mano un linguaggio di programmazione che non conosceva e problematiche che non aveva mai affrontato dal punto di vista pratico.

3.2 Fase 0: una migrazione graduale

Una volta che la collaborazione temporanea è terminata, le persone coinvolte nel progetto si sono confrontate e, considerata la situazione, si è scelto di far tesoro dell'esperienza fatta, salvare il salvabile (che comunque non era trascurabile) e partire per una migrazione più graduale.

Si è pensato di appoggiarsi ad un RDBMS per la memorizzazione persistente dei dati, la gestione dell'atomicità e dell'isolamento delle interazioni, la coerenza dei dati (in quest'ordine proprietà DAIC di solito note come ACID !).

Inoltre avvalersi di un backend DB che sfruttasse il linguaggio SQL per l'interazione, avrebbe consentito in modo semplice di adattare le parti scritte in Perl man mano che la nuova implementazione in Python procedeva.

Vediamo quali fasi si sono susseguite nello sviluppo di SANET.

3.3 Fase 1: Django + RDBMS

Obiettivo visualizzazione: sfruttare le nuove tecnologie web per realizzare una interfaccia ricca (RIA) che potesse essere fruibile da vari browser e pertanto non richiedesse di essere installata su ogni postazione.

Da una disamina degli *application server* disponibili per Python si è scelto il nuovo framework Django che pure era solamente alla versione 0.96. Si è subito deciso di adottare la versione SVN che disponeva di funzionalità molto più avanzate della versione rilasciata e avrebbe dato anche la possibiltà di segnalare patch e essere sincronizzati con la comunità di sviluppatori.

Fra i pregi che hanno contribuito alla scelta:

- è basato su pattern modello-vista-template (MTV, una interpretazione del più comune MVC)
- è indipendente da un particolare backend DB: all'inizio si è usato MySQL ma poco dopo si è passati a PostgreSQL. Si voleva lasciare aperta già in principio questa possibilità date esperienze precedenti non troppo soddisfacenti con MySQL
- come dicono gli sviluppatori: Django è Python ! Non ci sono vincoli particolari da rispettare, o imponenti infrastrutture da costruire (come ad esempio accade in Zope). Inoltre il sistema è veramente molto duttile e si può decidere di utilizzare solamente una componente (ad esempio soltanto il modello come poi avviene con la CLI)

Lo *application server* è scelto e pertanto viene relegata ad esso una serie importante di problematiche di basso livello come:

- interazione con RDBMS
- interazione con socket
- parsing delle richieste HTTP
- serializzazione delle risposte HTTP
- astrazione del backend autenticazione: ogni infrastruttura di rete dispone di propri meccanismi

A questo punto si è definito il modello dei dati che verrà popolato dall'attività di monitoraggio di *pinger* (riadattato per l'esigenza): questi saranno i dati visualizzati nell'interfaccia web.

3.3.1 INIL, SITEATTR, FOREST

Il database è stato progettato in 3 parti logicamente suddivise:

- INIL (icon, node, interface, link): contiene le tabelle base che rappresentano la topologia della rete e le icone. In questa parte rientrano anche le configurazioni dei controlli, sebbene l'acronimo non le comprenda
- SITEATTR (siteattr): contiene una tabella con il registro globale di configurazione del sito (nome, ultima modifica, numero di thread, etc.)
- FOREST: (tree, container, node_tree_pos): contiene le tabelle per rappresentare alberi di contenitori, e per posizionare i nodi nei contenitori..

Nella prima fase l'implementazione di INIL si è concentrata sulle tabelle contenenti le informazioni di configurazione e di stato dei controlli. Per quello che riguarda i nodi e le interfacce di rete si è pensato di inserire solo le informazioni anagrafiche indispensabili alla rappresentazione, mentre i link sono stati totalmente relegati al momento dell'implementazione futura delle mappe topologiche.

Realizzare SITEATTR è stato ovviamente banale una volta deciso di memorizzare queste informazioni nel database è bastato dedicare una tabella con parametri e valori globali all'applicazione.

La parte FOREST è totalmente nuova rispetto a *pinger* e si è deciso di realizzarla subito. Essa è risultata di fondamentale importanza per catalogare le risorse della rete e quindi offrire all'utente viste specializzate dell'ambiente monitorato.

I contenitori sono strutturati ad albero in modo analogo alle directory strutturate nella gerarchia del file system. In questa metafora è come se i nodi di rete fossero i file da catalogare nelle directory. In SANET tuttavia l'espressività della categorizzazione è maggiore rispetto a quella del file system in quanto un nodo di rete può essere associato a diversi contenitori che non appartengono allo stesso albero. Ogni albero identifica una tipologia di categorizzazione e i contenitori una specifica categoria. Alberi comunemente usati sono relativi alla dislocazione geografica degli apparati, alla gerarchia di responsabilità, o ancora alla tipologia dei nodi di rete in esso contenuti. Classificare un nodo di rete in contenitori di alberi diversi in SANET significa applicare quei concetti di tag o categorizzazione duttile tipici delle folksonomie del Web 2.0.

Considerando che ogni pagina di SANET è la rappresentazione di ciò che è contenuto nella risorsa selezionata, usare in modo corretto i contenitori consente di offrire ad utenti specifici cruscotti, che di solito prendono il nome inglese di *dashboard*, tramite i quali osservare l'andamento delle risorse di rete di propria responsabilità o comunque le risorse di proprio interesse.

3.3.2 SANET visualizza lo stato della rete

Come risultato di questa fase viene soddisfatta la prima priorità: visualizzare lo stato della rete in modo comprensibile all'utente medio. Una svolta epocale per i LABS: da questo momento non sarebbero stati solo i tecnici ad avere visibilità totale e "macchinosa" degli eventi, ma tutti avrebbero avuto un'idea delle problematiche o quantomeno avrebbero saputo di qualcosa che non stava funzionando nel modo atteso.

...e anche di dove ciò stava accadendo: SANET infatti fin dalla prima versione implementa il meccanismo dei contenitori che era totalmente assente nel sistema precedente.

Pinger nel contempo era stato adattato all'interazione con lo RDBMS e continuava ad essere eseguito con le seguenti mansioni:

- interpretazione della configurazione per i controlli
- monitoraggio: verifica dello stato e aggiornamento misurazioni

Le caratteristiche dei controlli, il loro stato e il valore delle misurazioni venivano scritti nel database e letti dallo application server che ne poteva così produrre la rappresentazione

3.4 Fase 2: CLI e mappe

Il risultato raggiunto nella rappresentazione dell'attività di monitoraggio non era nemmeno paragonabile alla versione precedente, quindi le priorità stringenti erano soddisfatte.

A questo punto il lavoro è proseguito da un lato con il potenziamento dell'interfaccia web e quindi:

- le mappe topologiche delle reti (a cui è dedicata una sezione separata)
- interazione minimale: ricerca e gestione di appunti, note temporanee

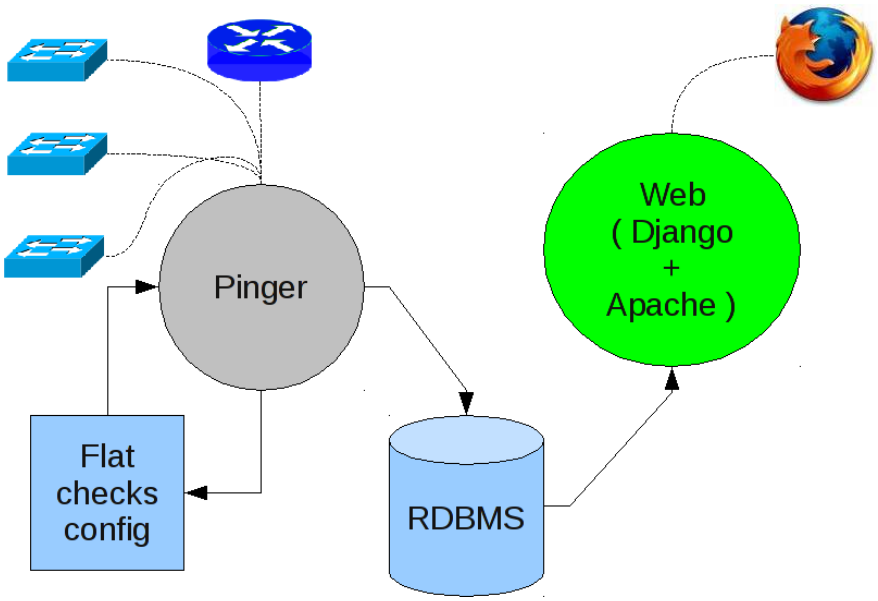

Figura 3.1: RDBMS + Web (legge conf e stato) + Pinger (legge conf, scrive conf e stato)

dall'altro con **l'implementazione della 'Command Line Interface' (CLI)**.

Con la CLI è stata colta l'occasione per potenziare l'espressività della tassonomia dei controlli definibili nel sistema e implementare un'interfaccia per l'operatore di rete esperto: non a caso l'interprete dei comandi implementato è simile a quello del sistema operativo Cisco IOS molto diffuso e apprezzato fra gli esperti di reti.

La CLI è realizzata interamente in Python e si appoggia allo stesso modello di dati costruito per la parte web. Ciò è stato un notevole pregio nell'aver scelto una soluzione come Django che implementa in modo chiaro la separazione delle componenti; oltre ovviamente al beneficio di utilizzare software libero che ci ha consentito di copiare le funzioni di inizializzazione di Django e riportarle nella procedura di inizializzazione della CLI.

L'interprete dei comandi è sviluppato in modo molto semplice e pratico. Anche qui si nota, come nel vecchio *pinger*, l'approccio sistemistico fatto di funzioni piuttosto che di classi ed ereditarietà, di variabili globali invece di attributi statici di classe, o ancor meglio di passaggi per riferimento.

Anche l'output della CLI viene prodotto su misura e in un primo momento non si pensa alla possibilità di astrarre il backend di output in modo da poter inizializzare lo stesso codice su backend testuale, ncurses, o grafico piuttosto che di socket di rete.

Per fortuna successivamente, appena possibile, non è stato troppo impegnativo l'intervento degli sviluppatori per aprire questo spiraglio nella rappresentazione dei dati, mentre purtroppo per le variabili globali o la strutturazione del codice ci si è dovuti accontentare dell'implementazione realizzata e che comunque, a onor del vero, funziona.

Con la CLI viene implementata nel database tutta la parte di configurazione di SANET (categorie, attributi, istanze) e quindi ristrutturato il vecchio sistema di *template* e definizione dei controlli.

La compatibilità è rotta, anche se la logica di fondo rimane simile.

I sistemisti al lavoro nelle installazioni in produzione di SANET si trovano disorientati e rallenta di molto il processo di aggiornamento delle installazioni da quella che era la versione 1.4 alla versione 2.0 (poi diventate 0.1.4 e 0.2.0 con il rilascio alla comunità open source).

In questa fase viene sottovalutato l'impatto di un tale aggiornamento e senza che nessuno se ne accorga, si interrompe il dialogo fra i sistemisti e gli sviluppatori, facendo sì che solo dopo alcuni mesi ci si accorga del mancato avanzamento delle installazioni in produzione.

In ogni caso, è stato raggiunto un altro importante obiettivo: il potenziamento della tassonomia dei controlli. Ora si possono definire molti più controlli con meno sforzo.

Pinger è stato adattato per leggere la nuova configurazione dal database e continua la sua attività come strumento di monitoraggio e quindi di aggiornamento dello stato.

La configurazione e la rappresentazione sono in mano a SANET. Notare che non viene provvisto, e ad oggi non è ancora implementato, un modo per configurare via web i parametri dei controlli: ciò è dovuto alla consapevolezza delle complesse realtà di rete gestite dall'azienda che non si possono normalizzare con l'esposizione di interfacce cosiddette *user-friendly*.

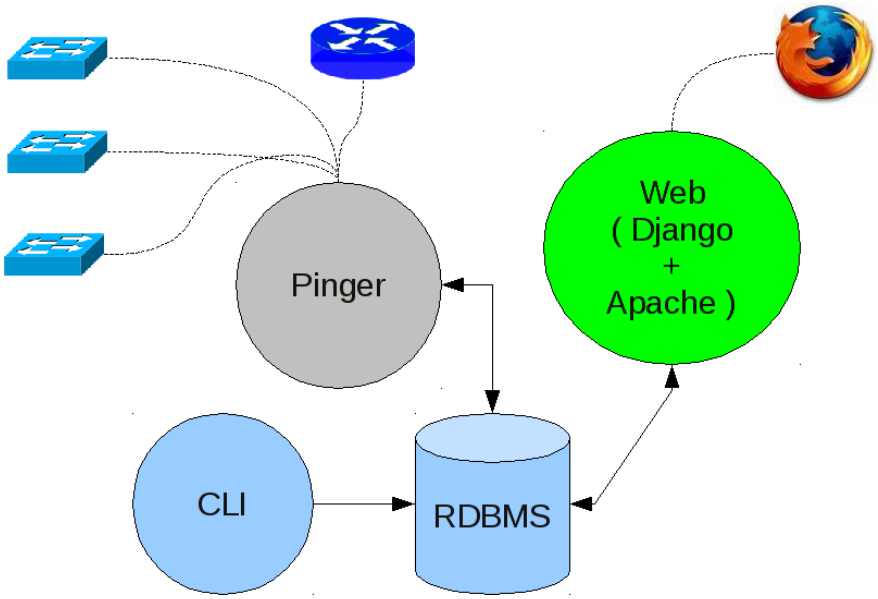

Figura 3.2: RDBMS + Web (legge conf e stato, scrive note e mappe) + CLI (scrive conf su RDBMS) + Pinger (legge conf e scrive stato su RDBMS)

3.5 Fase 3: Poller

Entra un nuovo sviluppatore nella squadra. **Obiettivo riscrivere 'pinger' in Python** e con questo:

- decurtare le ultime rimanenze di *pinger*
- aumentare l'espressività del linguaggio per la definizione delle espressioni con cui verificare lo stato ed effettuare le misurazioni
- avere un sistema più scalabile grazie al *multithread*

Se prima ci si era occupati dei meccanismi di ereditarietà fra i controlli e le categorie, ora ci si concentra sulla singola espressione da verificare. Si realizza un linguaggio con una propria grammatica, dotato di contesto e operatori con tipizzazione dinamica degli operandi. Questa nuova implementazione consente di esprimere ulteriori tipi di controlli; vegnono implementate:

- funzioni di adiacenza *bgp* ed *ospf*
- controlli su *ntp*
- interrogazione *WMI* tramite *wmic* per i *server Windows*
- esecuzione comandi esterni (e quindi integrazione *ZenPacks*, *JMX* o *plugins Nagios*)

- *wildcards* per *OID SNMP*
- operatori di *match* sottostringa

oltre al meccanismo di *escalation* che consente di ridurre al minimo il rumore per gli allarmi "a cascata".

Anche in questo caso si può sfruttare il modello di dati già usato dalla CLI e dallo application server e, vista l'elevata occupazione di memoria e la frequenza di operazioni di *update* nel database, si implementano meccanismi ad-hoc che sono più performanti di quelli offerti dal Django stesso.

Non mancano i *bug* nella libreria NetSNMP e nel suo *binding* Python: vengono segnalati, uno minore viene risolto, per il resto si trovano *workaround*.

Frattanto prosegue lo sviluppo delle mappe e il lavoro di promozione presso i clienti dell'azienda.

La realizzazione del *poller* è la svolta finale che consente al gruppo di procedere verso il rilascio. Ora il vecchio *pinger* è completamente sradicato e SANET lo sostituisce completamente superandone i limiti.

Un ulteriore miglioramento per la fruibilità dei dati è costituito dalla segnalazione degli allarmi tramite *feed RSS*, o dal recupero degli stessi tramite una semplice interfaccia *XML-RPC* che ricorda l'operazione *snmpwalk*.

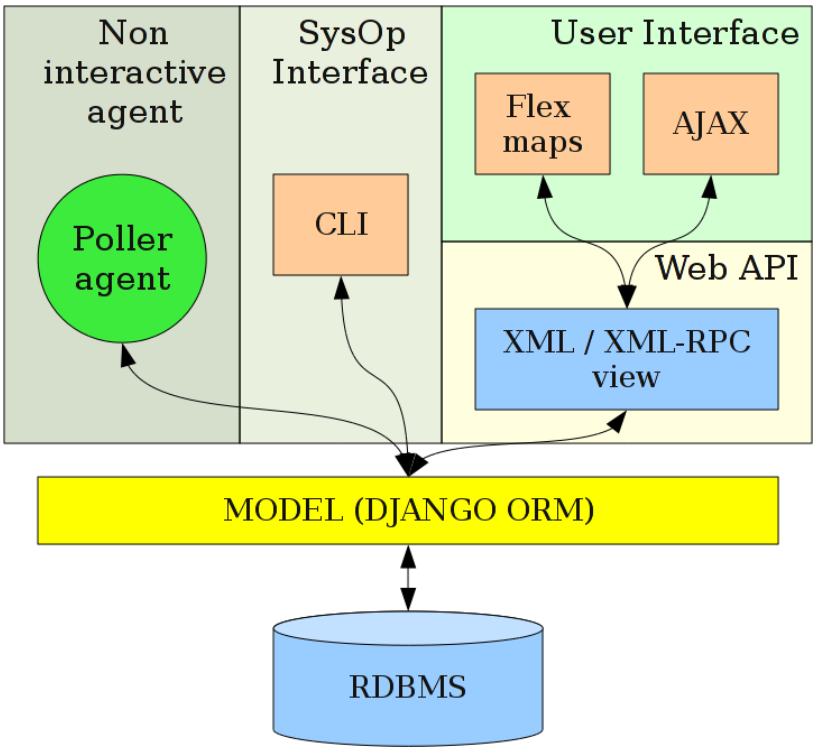

Figura 3.3: Architettura completa di SANET (RDBMS + Web + CLI + Poller)

3.6 L'impulso del Master FOSSET0809

Durante questo ultimo anno ho avuto modo di frequentare il Master FOSSET su proposta e con il sovvenzionamento dell'azienda.

Ritengo opportuno dedicare una breve sezione all'influenza che questa attività di formazione ha avuto sul processo di sviluppo di SANET.

Un apporto importante si è verificato nella filiera di sviluppo di tutto il software aziendale grazie al corso di *Strumenti di sviluppo collaborativo* e in particolare:

- uso degli *hooks* per i sistemi di versionamento
- strumenti di *literate programming*

In questo modo abbiamo potuto realizzare l'infrastruttura di sviluppo presentata in figura *Schema dell'infrastruttura di sviluppo*.

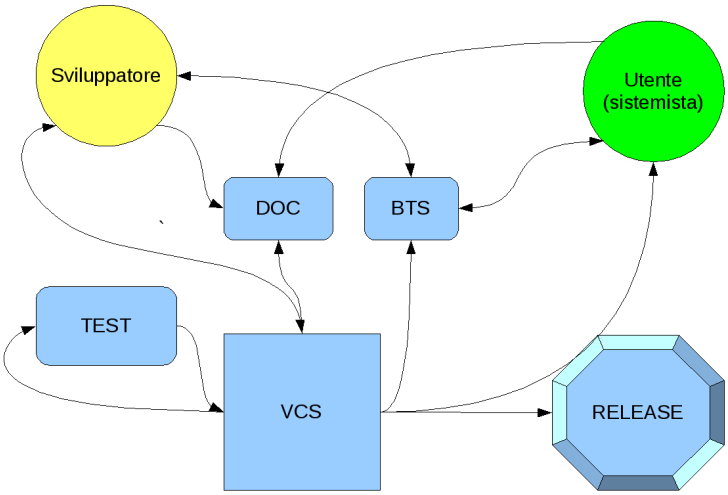

Figura 3.4: Schema dell'infrastruttura di sviluppo

Per quello che riguarda il corso di *Project management* invece, abbiamo cercato di mantenere, man mano che il team si allargava (con l'arrivo del nuovo sviluppatore) un approccio *Agile* alla risoluzione dei problemi, anche se non c'erano risorse per provare un vero e proprio *Scrum* come avrei voluto.

Altri tentativi di adozione sono stati:

- la tecnica del pomodoro
- *test driven development*

entrambi interessanti, ma purtroppo naufragati a causa del piccolo team autogestito e dal fatto che la crescita deve procedere per gradi.

Alcune sperimentazioni e introduzioni importanti sono derivate dai progetti:

- **Django history** realizzato per *Fondamenti di sistemi liberi* ci ha consentito di sperimentare un meccanismo versatile ed efficace per mantenere con Django uno storico temporale di alcune tabelle del database. L'implementazione realizzata supera alcuni dei limiti rilevati dall'autore Marty Alchin che purtroppo non ha più risposto ai miei messaggi. In ogni caso il modulo realizzato può essere tranquillamente integrato così come è in SANET per la gestione del log dei cambi di stato.
- **Syslog collector** per il corso di *Reti* è stato rilasciato come ulteriore prodotto LABS su http://syslog-agentx.sourceforge.net/ . Esso consente a SANET (e non solo) di implementare una serie innumerevole di nuovi controlli dato che esporta via SNMP informazioni su match di espressioni regolari in messaggi di *syslog*.

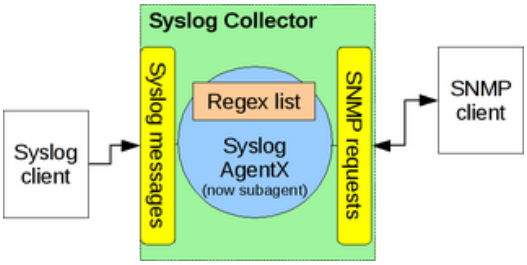

Figura 3.5: Syslog collector

- Il lavoro effettuato per il supporto ai *prepared statement* in Django con *backend* PostgreSQL per il corso di *Basi di dati e applicazioni web* è servito per capire che non avremmo avuto un aumento di prestazioni significativo con l'introduzione in SANET degli *statement* precompilati.

Uno sforzo importante: le mappe

Dedichiamo un capitolo all'implementazione ed evoluzione delle mappe di rete di SANET, dato che proprio questa evoluzione è parte consistente del project work rilasciato insieme con questo documento.

Dopo la rappresentazione dell'attività di monitoraggio in un'interfaccia web 2.0 si è deciso di implementare subito la visualizzazione delle mappe di rete per i contenitori.

4.1 La scelta fra SVG o Flash

La decisione di realizzare mappe interattive della rete ci ha subito posti davanti alla scelta fra le tecnologie grafiche per il web: SVG o Flash.

In passato (v. *Un primo tentativo fallito*) era stata implementata una versione basata su *Scalable Vector Graphic* (SVG) in modo da ottenere un risultato più aperto rispetto all'alternativa Flash di Macromedia ora Adobe.

Da quello che è stata la nostra esperienza, SVG risulta essere lento, e, paradossalmente, anche se è uno standard *più aperto* di Flash, Flash risulta essere più diffuso fra le varie piattaforme e, purtroppo, la compatibilità con Internet Explorer è un fattore critico in questi casi.

Inoltre, se da un lato lo standard SVG è comodo in quanto XML puro scrivibile comodamente anche tramite il sistema di template messo a disposizione da Django, dall'altro abbiamo molti esempi di applicazioni web che usano Flash, ma non abbiamo praticamente alcun caso reale in cui si usi SVG.

Per dettagli ulteriori sulla valutazione fatta si veda l'appendice *Appendice C - SVG o FLASH ?*.

Nel frattempo Adobe ha rilasciato il *framework Flex* con licenza Open Source, il che ci avrebbe consentito di implementare applicazioni in Flash senza acquistare licenze per costosi ambienti di sviluppo e anche di poter apportare modifiche a librerie secondo le esigenze del caso.

Considerata quindi anche la possibilità di mantere *vim*, il nostro IDE preferito, si passa a Flash o più precisamente a Flex.

4.2 La topologia della rete

Per rappresentare la topologia della rete si è deciso di partire dalla libreria *SpringGraph* di Mark Sheperd.

A questo punto è necessaria una premessa per capire la filosofia di SANET: con SANET si desidera vedere tutto e solo quello che si intende monitorare, quindi non solo si fa a meno dell'*autodiscovery* dei collegamenti di rete, ma anche

degli algoritmi di posizionamento automatico dei nodi sulla mappa. Entrambe le attività vengono affidate al sistemista di turno che ha il compito di configurare l'installazione specifica.

Per il posizionamento predefinito dei nodi di rete e dei contenitori si è usato, e si usa ancora oggi, *graphviz*. Le posizioni sono poi modificabili dai soliti sistemisti, o anche dall'utente tramite interfaccia grafica.

L'evoluzione interessante, che costituisce parte corposa in questo project work, si è verificata tra la prima e la seconda implementazione delle mappe di rete: vediamo ora quali erano caratteristiche e limiti della prima versione e come sono stati superati.

Inizialmente era stato creato uno strato superiore alla libreria che:

- non usasse gli algoritmi di posizionamento automatico
- posizionasse i vertici del grafo secondo le posizioni passate dal server
- rappresentasse risorse differenti: nodi e contenitori in questa fase sono entrambi vertici della mappa
- aggiornasse periodicamente lo stato degli elementi rappresentati: dei nodi, dei contenitori e dei collegamenti. Lo stato dei collegamenti poteva (e può ancor oggi) essere rapprentato anche da un gradiente (trattasi di una situazione temporanea o di un errore di configurazione), in cui le verifiche effettuate sulle estremità del link restituiscano uno stato differente.

Nella prima implementazione, quella del rilascio alla CONFSL09, ogni contenitore aveva associata un'unica mappa in cui venivano visualizzate le risorse direttamente incluse e i collegamenti fra esse.

Nelle figure *Mappa prima versione: collegamenti fra i contenitori (notare i blu)* e *Mappa prima versione: icone personalizzate* riportiamo alcuni esempi di mappe della prima implementazione.

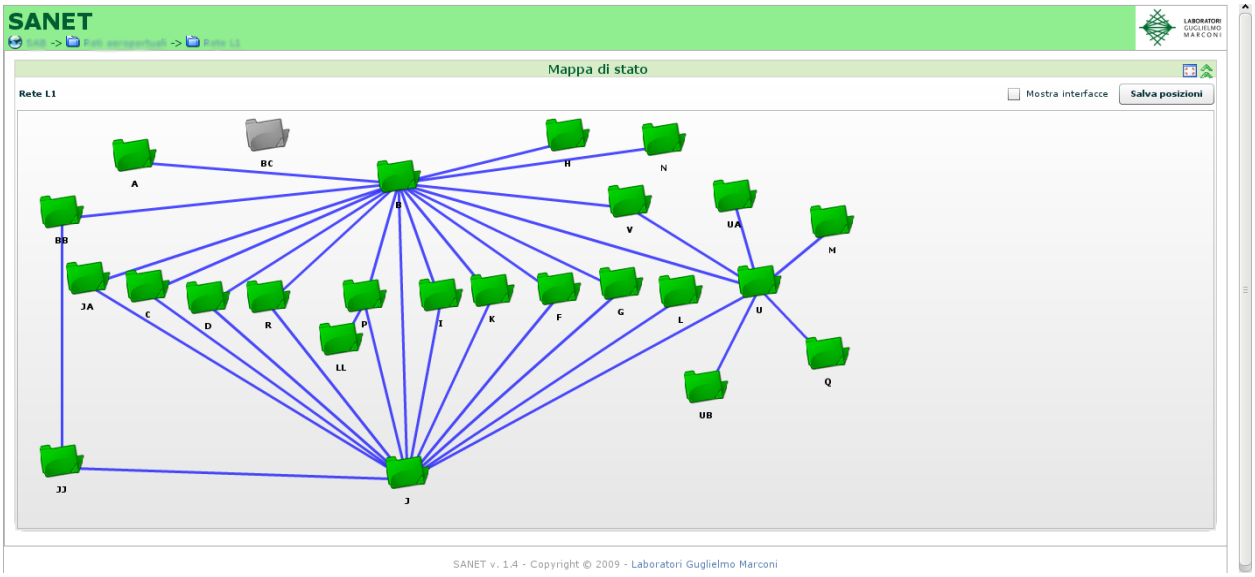

Figura 4.1: Mappa prima versione: collegamenti fra i contenitori (notare i blu)

Nella nuova versione abbiamo voluto introdurre le seguenti funzionalità:

- rappresentare contenitori aperti e risorse contenute (clustering)
- rappresentare mappe dei contenitori adiacenti al contenitore selezionato
- visualizzare le singole interfacce di rete e il loro stato

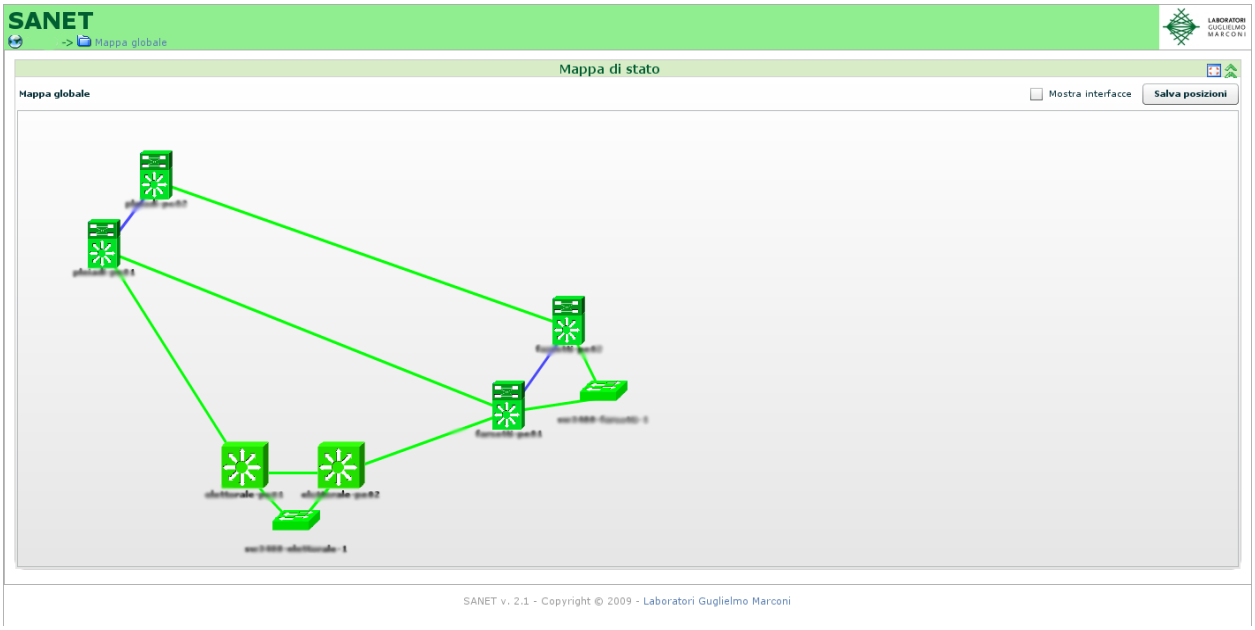

Figura 4.2: Mappa prima versione: icone personalizzate

- superare il limite del grafo semplice che non consentiva di visualizzare link multipli fra gli stessi due vertici (in caso di apparati con collegamenti ridondati o di contenitori con più apparati interconnessi veniva mostrato un arco blu)
- rappresentare differenti parametri: alternativamente alla rappresentazione dello stato, visualizzare il carico della rete o i valori relativi al protocollo Spanning Tree fra gli switch
- aprire e chiudere contenitori da interfaccia utente

4.2.1 La seconda implementazione

Per superare i limiti elencati si è inizialmente provata la libreria RaVis che è più avanzata rispetto a *SpingGraph*, ma purtroppo le esigenze di personalizzazione, con particolare riguardo al *clustering*, hanno fatto sì che tornassimo ad utilizzare la soluzione precedente già conosciuta.

Quindi, seppur basandoci sulla stessa libreria, siamo ripartiti con una riscrittura totale dell'implementazione precedente, integrando ovviamente di volta in volta alcune parti che ritenevamo congrue alle nuove esigenze.

Lato server

Innanzi tutto è stata ristrutturata la parte server.

È stata creata una nuova applicazione Django denominata *map* che si appoggia ad un modello di dati ibrido: da una parte esso include le classi *MapEnv* e *MapOpenContainer* che si appoggiano a tabelle nel database, dall'altra le classi *MapRoot*, *OpenContainer*, *OpenNode*, *ClosedContainer*, *ClosedNode*, *ClosedIface*, *MapEdge* che non hanno un corrispettivo nel database, ma fungono da classi *proxy* per gli oggetti di SANET e i corrispondenti *dot* di *graphviz*.

MapEnv contiene la lista delle mappe disponibili nel sistema con gli attributi relativi.

Essa ha 2 campi fondamentali:

- *bound_container*: una chiave esterna che identifica a quale contenitore è associata la mappa, ossia relativamente a quale contenitore deve essere visualizzata. Possono esistere più mappe associate allo stesso contenitore.

Per soddisfare l'esigenza di rappresentazione di mappe di adiacenza si è pensato di introdurre la possibilità di associare mappe arbitrarie ad ogni contenitore

- *open_containers*: che è un campo molti-a-molti attraverso il quale ogni mappa identifica i contenitori aperti da rappresentare. Le risorse visualizzate nella mappa saranno tutte quelle direttamente incluse nei contenitori aperti oggetto di questa relazione. In questo caso la tabella referenziata è proprio quella della classe *MapOpenContainer* che pertanto non ha bisogno di descrizione: il suo scopo è infatti di rappresentare la relazione molti-a-molti che esiste fra *MapEnv* e *Container*.

Di seguito il semplice schema E-R dell'applicazione *map*.

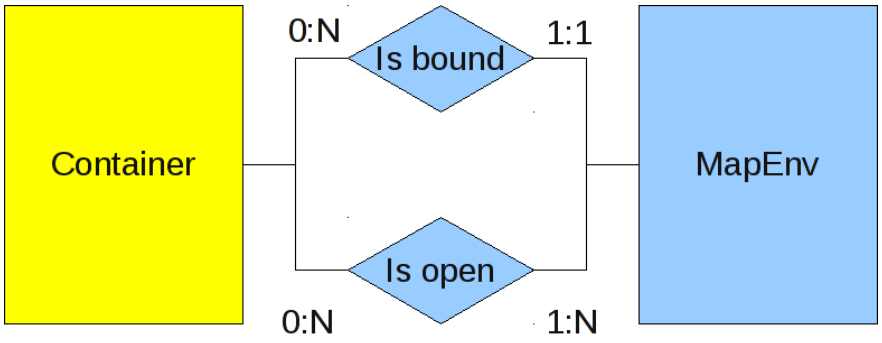

Figura 4.3: Schema E-R per le mappe

Come dicevamo *MapRoot*, *OpenContainer*, *OpenNode*, *ClosedContainer*, *ClosedNode*, *ClosedIface*, *MapEdge* non si appoggiano al database, ma fungono da classi *proxy* verso due tipi di oggetti: una risorsa di SANET (contenitore, nodo, interfaccia) e il corrispondente oggetto *pydot* che è il binding python per il *dot* di *graphviz*.

MapRoot rappresenta la radice della mappa e fa da *proxy* rispetto agli oggetti di classe *MapEnv* e *pydot.Dot*. Ottiene le informazioni dall'istanza *MapEnv* e procede ad istanziare gli altri oggetti di classi derivate da *BaseDot* (che sarebbero poi gli *Open** e *Closed**). Infine processa tutti i link delle interfacce istanziate nella mappa e istanzia gli oggetti di classe *Edge*. *MapRoot* fa anche qualcosa di più: crea un container radice virtuale per ogni mappa in modo da avere coerenza anche in caso di contenitori aperti appartenenti a due gerarchie di contenitori differenti. In questo modo la rappresentazione di mappe arbitrarie è ricondotta al medesimo problema di rappresentazione della mappa di un singolo contenitore con possibilità di contenitori aperti innestati.

Le classi *Open** e *Closed** derivano da una classe comune *BaseDot* che ha il compito di isolare proprio la logica di *proxy* verso le risorse di SANET e di *pydot*.

Istanziando un oggetto di classe *Open* si avrà a che fare con una risorsa di SANET e un'istanza di *pydot.Cluster*. Mentre istanziando un oggetto di classe *Closed* si avrà a che fare con una risorsa di SANET e un'istanza di *pydot.Node*. Ciò significa che è stata completamente scorporata la natura intrinseca della risorsa di SANET dalla sua rappresentazione: se si vuole aprire un contenitore si passerà da un oggetto *ClosedContainer* a un oggetto *OpenContainer* che avrà fra i suoi attributi la medesima risorsa di class *Container*.

Allo stesso modo per i nodi: se si desidera esplodere un nodo, ossia visualizzare tutte le interfacce all'interno in modo dettagliato per poter, in futuro, spostare il cavo UTP o la fibra da un'interfaccia a un'altra, è possibile farlo passando semplicemente da un *ClosedNode* a un *OpenNode*.

MapEdge è una classe che include un oggetto *pydot.Edge* più un'interfaccia sorgente e una destinazione.

Le mappe attuali utilizzano:

- *OpenContainer* per rappresentare i contenainer aperti
- *ClosedContainer* per rappresentare i container chiusi
- *ClosedNode* per rappresentare i nodi

- *MapEdge* per gli archi

La gerarchia di *MapRoot* viene serializzata in un documento XML. La prima versione prevedeva solamente elementi di tipo:

- **<node>**: vertice del grafo, poteva essere un nodo o un contenitore
- **<iface>**: elemento contenuto in **<node>** con la funzione di elencare solamente i nomi delle interfacce incluse nel nodo
- **<edge>**: conteneva attributi per identificare gli endpoint di tipo <node> e gli endpoint di tipo <iface>

Nella nuova versione è stato introdotto il tipo **<subgraph>** che altro non è che la serializzazione di un'istanza di *OpenContainer*. In questa versione è rilevante l'innestamento degli elementi XML che stabiliscono in quale contenitore aperto si trovano gli oggetti rappresentati. Ciò è fondamentale sotto 2 aspetti:

- il posizionamento di ogni elemento che deve essere calcolato relativamente al proprio contenitore aperto
- lo spostamento di ogni elemento che deve considerare i limiti del proprio contenitore aperto. Attualmente l'implementazione espande il contenitore aperto dell'elemento se esso viene spostato oltre il limite del lato destro e del lato inferiore dello stesso.

In *Appendice D: XML prodotto per una mappa con contenitori aperti* riportiamo un esempio di XML prodotto dalla nuova versione di SANET per rappresentare una mappa con contenitori aperti:

Una nota importante sulle interfacce di rete: è l'istanza *ClosedNode* che serializza le interfacce contenute nel nodo. Come vedremo le interfacce sono fondamentali per superare in modo apparente il limite del grafo semplice: purtroppo spesso non sono visualizzate e rendono più corposo lo XML prodotto.

Lato Flex

Quando si è andati ad intervenire sulla parte Flex, abbiamo cercato di evitare qualunque modifica alla libreria SpringGraph originale, per poter installare gli aggiornamenti (di minor version) e i bugfix in modo trasparente.

Ciò non è stato possibile a causa di alcune modifiche necessarie ad implementare i meccanismi in oggetti derivati, quindi sono state effettuate alcune variazioni alle *signature* delle funzioni e degli attributi che in alcuni casi sono passati ad esempio da *private* a *protected*. Queste personalizzazioni sono state segnalate all'autore che in un primo momento aveva risposto dicendo che non era contrario alle modifiche, ma lui aveva risolto in precedenza in altri modi e avrebbe comunque verificato la nostra patch. Da quel momento in poi non si è più ricevuta alcuna risposta.

Molti sforzi sono stati dedicati all'implementazione delle mappe lato Flex. Flex è un framework molto potente, ma molto complesso.

Di seguito spieghiamo la logica che risiede dietro i file più significativi:

- *SANETMap.mxml*: questo il file applicazione. Il main. Contiene l'interfaccia grafica applicativa con i controller grafici e le funzioni (astratte) che servono a recuperare i valori desiderati (**valori di stato, spanning tree, carico di rete**). In questa nuova versione sono stati introdotti 3 slider per: **zoom, curvatura degli archi e trasparenza dello sfondo dei contenitori aperti**. È presente un **check selezionabile per la visualizzazione delle interfacce di rete e dei loro nomi**. Infine un bottone per il **salvataggio permanente delle posizioni dei nodi**.
- *MapInfoProxy.as*: proxy del tipo di richiesta specifica. Tutti gli *handler* di invio e ricezione si registrano in questa classe. Il controller grafico che seleziona il tipo di valore da visualizzare attiva il canale specifico di richiesta/ricezione.
- *NetGraph.as*: struttura del grafo di cui poi VisualNetGraph.as conterrà la rappresentazione. Include il un metodo statico per costruire il grafo dai dati XML ricevuti.
- *VisualNetGraph.as*: rappresentazione grafica della mappa. Qui vengono innestati i contenitori aperti (a partire dal contenitore radice virtuale) e le risorse in essi incluse. Inoltre vengono ospitati gli archi che devo-

no poter connettere risorse presenti anche in container aperti differenti e disattivato l'algoritmo automatico di posizionamento.

- *SANETEdge.as*: rappresentazione degli archi. Qui si gestisce la direzione e il gradiente. 2 trucchi sono stati usati per superare le limitazioni del grafo semplice: il primo riguarda la direzione e il secondo i link multipli. La direzione viene passata nell'elemento XML <edge> e viene ricalcolata proprio in questo file. Invece l'intuizione che sta dietro al superamento dei link multipli fra due vertici, consiste nella consapevolezza che le interfacce di rete odierne (*wired*) se connesse, interconnettono sempre e solo 2 endopoint. In questo caso va benissimo il grafo semplice. Quindi nelle nuove mappe l'arco non è più costruito fra 2 nodi o 2 container chiusi, ma fra le interfacce che essi stessi contengono siano esse visibili o invisibili.
- *NestedItem.as*: implementa la logica dell'innestamento degli elementi XML spiegata sopra.
- *SANETViewFactory.as*: in fase di rendering, istanzia l'oggetto *Visual** specifico relativamente ai dati XML di un elemento di *NetGraph.as* dato in input
- *VisualBaseNetGraphElement.mxml*: classe base per gli elementi della mappa. Definisce il canvas e gli handler per gli eventi nell'interazione con l'utente (mouseover, mouseout, doubleClick, ...)
- *VisualCloseContainer.as*: rappresentazione di contenitore chiuso. È usato anche per la rappresentazione dei nodi chiusi. Dal punto di vista dell'interfaccia sono icone che hanno un'etichetta, un url, stessi handler per la gestione di eventi e conoscenza delle interfacce di rete incluse (*VisualIface.as*)
- *VisualOpenContainer.as*: rappresentazione di container aperto. Un riquadro con posizionamento diverso dell'etichetta, bordi colorati con il proprio stato (derivante dalle risorse incluse nello stesso) e conoscenza degli oggetti in esso contenute (*VisualOpenContainer.as*, *VisualCloseContainer.as*)
- *VisualIface.as*: rappresentazione grafica delle interfacce di rete. Esse sono cerchi posizionati al centro del proprio *VisualCloseContainer.as*. Di esse non è possibile salvare le posizioni. È molto interessante poter visualizzare lo stato di una singola interfaccia di rete.

Purtroppo al momento della scrittura di questo documento manca l'implementazione lato server del recupero dei valori quali carico di rete e stato dello spanning tree e quindi, sebbene l'implementazione lato client sia praticamente pronta, tali parametri non possono essere rappresentati nella versione 0.4 di SANET.

SANETMap.swf è il file compilato dell'applicazione Flash, occupa circa 400KB ed è l'unico che viene scaricato dal browser (e messo in cache) per la visualizzazione delle mappe in SANET.

Un'ultima nota su come ancora una volta il software libero ci è stato di aiuto e in particolare il modulo di terze parti Flex Thunderbolt che consente di effettuare il log dei messaggi nell'estensione FireBug di Firefox (estensione fondamentale per ogni sviluppatore web).

Interazione fra Javascript e Flex

A corredare il tutto non poteva mancare un modulo per interagire con le mappe direttamente da Javascript e viceversa. Questo è il modulo Flex Ajax Bridge provvisto da Adobe stessa (file *bridge/FABridge.as*) ed è stato fondamentale per dare coerenza nel tooltip informativo delle risorse e nel menu contestuale.

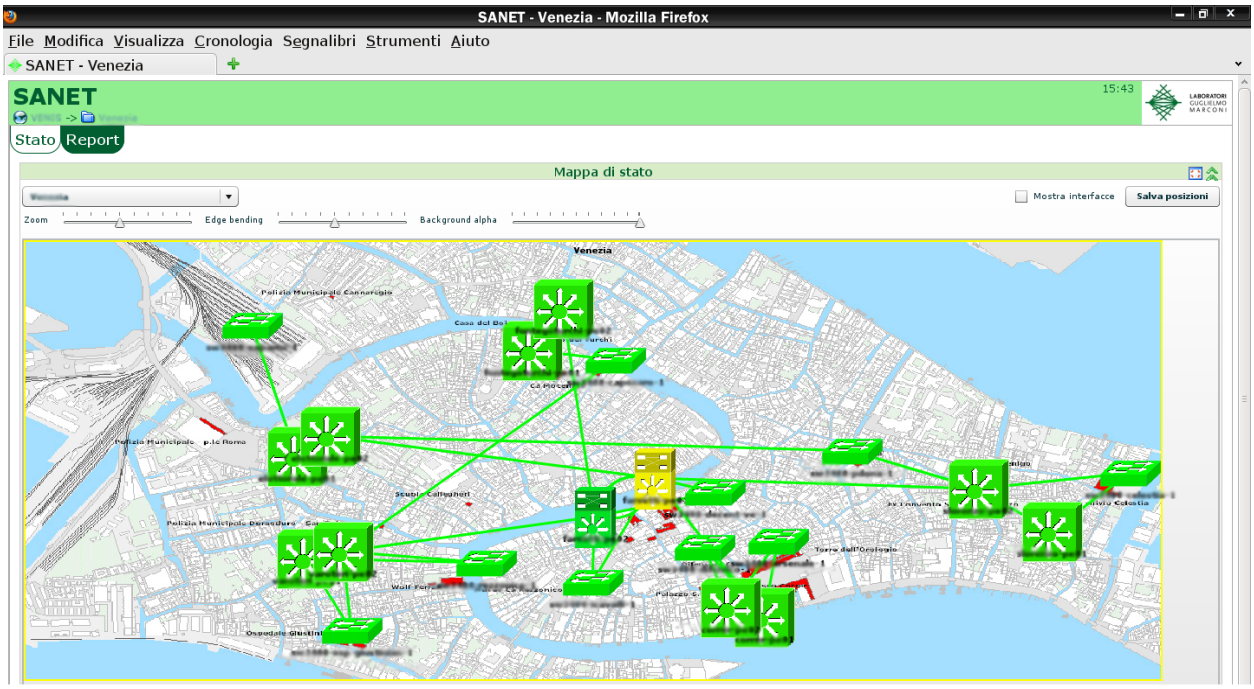

Figura 4.4: Mappa con sfondo georeferenziato (precaricato)

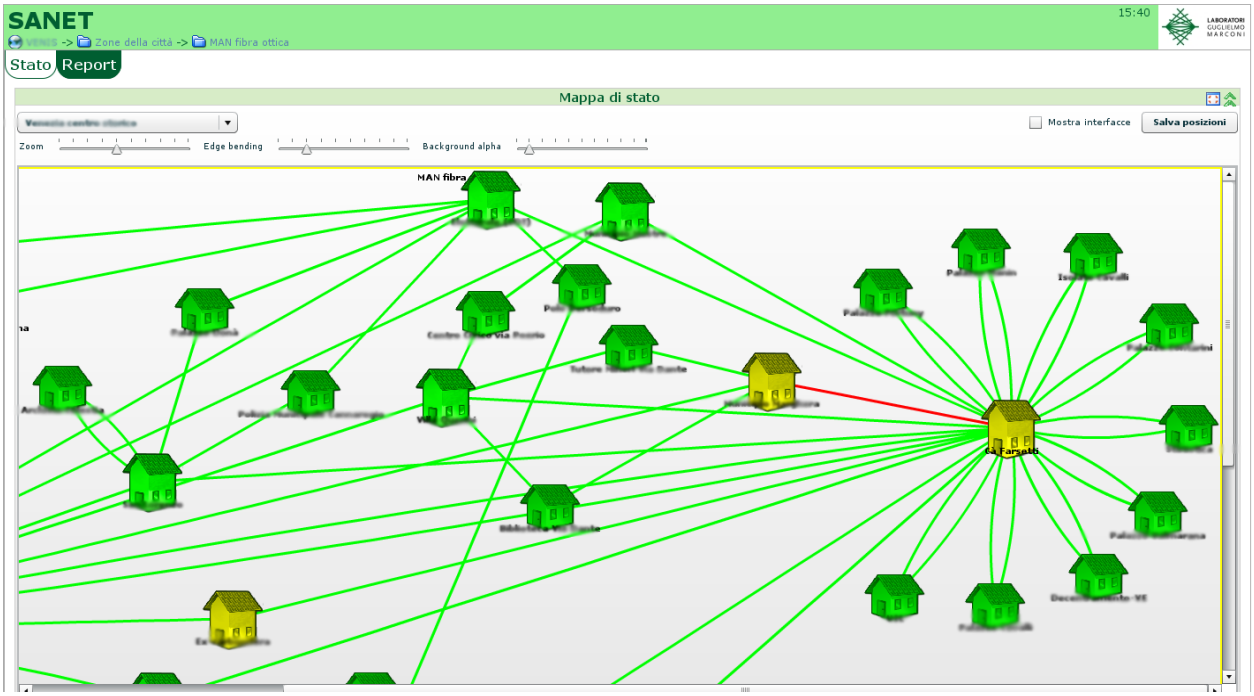

Figura 4.5: Dettaglio di una MAN: notare gli archi doppi che partono dal grande edificio giallo a destra

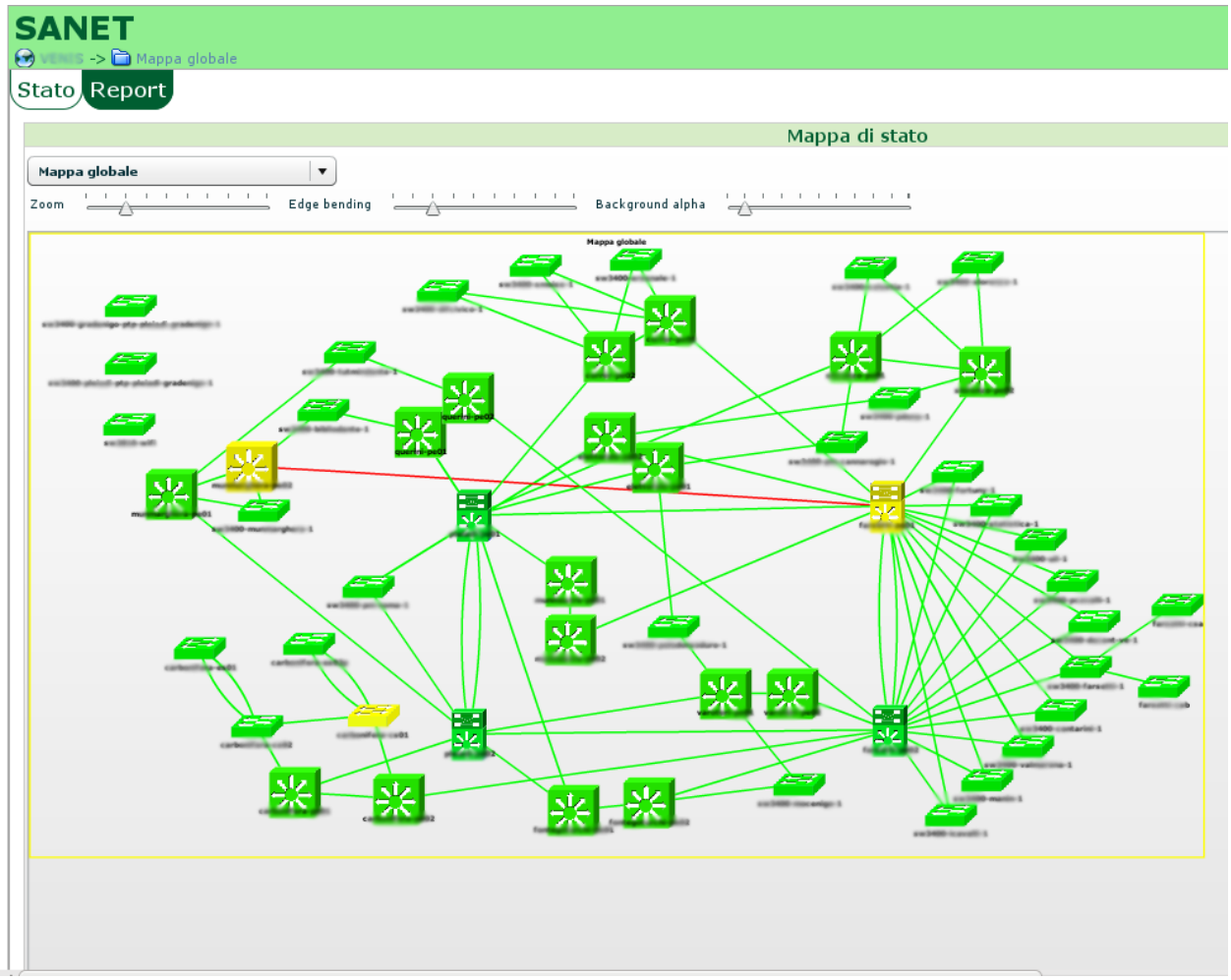

Figura 4.6: Mappa di apparati: anche qui si notano archi doppi fra apparati ridondati

CAPITOLO 5

Rilasciare SANET alla comunità

Il processo di rilascio è una fase cruciale nella vita di un software libero. È il momento in cui il software esce dalla sua nicchia ed entra a far parte della comunità.

Tecnicamente non si può dire che è il momento in cui il software libero nasce perché un software è libero a prescindere dalla sua distribuzione, ma si può tranquillamente dire che il rilascio è il momento in cui il software entra a far parte della comunità del software libero.

Per uno sviluppatore vuol dire mettersi in gioco, vuol dire trovare il tempo per spiegare ed ascoltare. Per un'azienda vuol dire pubblicità, e vuol dire anche costi.

Un'azienda che non ha bisogno di pubblicità non è interessata a rilasciare il software. Anche se un po' di pubblicità non guasta mai, bisogna considerare i costi ad essa collegata.

Rilasciare un software libero per un'azienda, significa porsi degli interrogativi e fare delle scelte che altrimenti non avrebbe dovuto fare, significa un impegno a documentare adeguatamente il software, significa creare dei canali di comunicazione con la comunità, significa porre attenzione alle richieste della comunità.

Da tutto ciò quindi si deduce che non c'è solo un discorso di pubblicità, ma anche di qualità del software e di apertura mentale di chi crede che il confronto sia in ogni caso costruttivo.

È per questo che secondo me bisogna credere nelle aziende che rilasciano i propri prodotti e pongono attenzione alla comunità, mentre bisogna andare cauti con chi sostiene di offrire soluzioni open source, ma che in realtà non giocano la propria parte secondo queste regole.

5.1 Le scelte principali

Riassumiamo le scelte fondamentali che LABS ha preso per essere pronta al rilascio di SANET:

- **Rilasciare tutto, o tenere un piccolo sottoinsieme di funzionalità "separate"**: non è raro trovare prodotti liberi di aziende che mantengono separate una serie di funzionalità per la versione *enterprise* con licenza meno aperta. Questo approccio può derivare da una serie di considerazioni quali vantaggio competitivo, collaborazioni esterne, o il meno pulito di posizionare i cosiddetti *specchi per allodole*. LABS ha deciso di rilasciare tutto il software realizzato e anche una corposa libreria di controlli. Questa è senza dubbio una scelta di grande apertura che sottolinea la confidenza nel proprio operato e la consapevolezza che la ricchezza di un software simile consiste soprattutto nel servizio di personalizzazione e assistenza h24 che si può offrire a strutture complesse che necessitano di esperti sistemisti di rete... e la professionalità non si fa con qualche riga di codice in più

- **Quale licenza applicare**: dato che SANET è un'applicazione web, abbiamo optato per la AGPLv3. L'intenzione è di evitare il *problema Google*: un'azienda potente che inglobi SANET fornendo un servizio di rete basato su di esso, ma senza essere obbligata a restituire il codice alla comunità e tantomeno a noi
- **Dove sviluppare il codice** o meglio dove gestire lo sviluppo del codice e la comunità degli utenti: la risposta è stata Sourceforge.net. Il portale di sviluppo software open source per definizione. Sourceforge.net offre il sistema di versionamento Subversion (e ora anche GIT) e la recente possibilità di installare applicazioni esterne quali TRAC, MediaWiki o Wordpress, ci hanno convinto che sarebbe stata una scelta appropriata. Infatti, in particolare grazie a TRAC, abbiamo potuto portare avanti il progetto senza cambiare il nostro modo di lavorare
- **Quando uscire con la prima release**: una decisione non banale. Abbiamo deciso di uscire non appena ci fossimo liberati totalmente del vecchio *pinger*. In aiuto ci è venuta anche l'occasione della ConfSL09: un motivo in più per accellerare i lavori e rilasciare entro il 13 giugno 2009.

5.2 Il primo rilascio alla ConfSL09

Sabato 13 giugno 2009 è il primo rilascio ufficiale di SANET.

Momento storico per i LABS e in particolare per me che per qualche anno avevo sviluppato software libero con il sogno, o l'ambizione di raggiungere questo obiettivo.

Cosa viene fatto per il rilascio di questa prima versione ?

Il minimo indispensabile:

- viene inserito un file LICENSE con la licenza AGPLv3
- viene creato un tarball dell'ultima versione stabile
- viene rilasciato il pdf dell'articolo per la CONFSL09 come guida base per l'utente TODO ripportato in appendice ?
- viene creata la home page del progetto: http://sanet.sourceforge.net
- viene attivato e impostato l'ambiente TRAC con le prossime milestone
- viene creato e registrato il canale irc #sanet nella rete freenode

La strada è tracciata. SANET è su Sourceforge.net.

Ora inizia un percorso: dalla cattedrale si va verso il bazaar. Riportiamo a questo proposito la figura *From the Cathedral to the Bazaar* riferita allo studio *From the Cathedral to the Bazaar: An Empirical Study of the Lifecycle of Volunteer Community Projects* di *Andrea Capiluppi e Martin Michlmayr*.

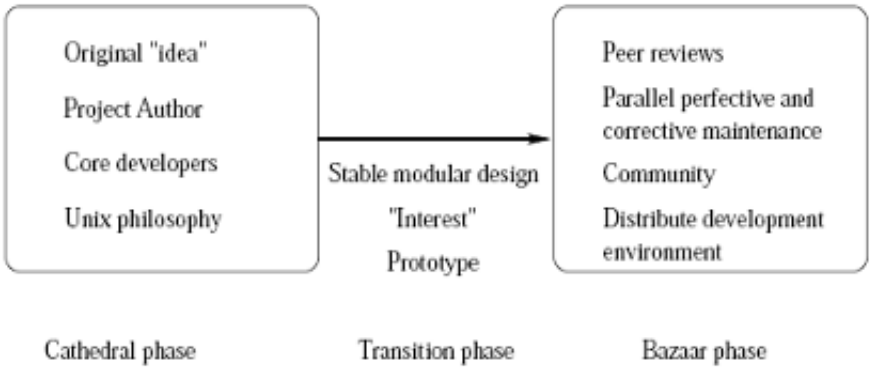

Figura 5.1: From the Cathedral to the Bazaar

Nello studio i due autori spiegano come ogni progetto disseminato nel bazaar parta da una prima fase caratterizzata dall'approccio a cattedrale.

Cosa c'è nella cattedrale ?

- Rappresentazione della rete
  - Nodi
  - Interfacce
  - Controlli
    - Target
    - Misure
  - Sito
  - Contenitori
- Poller (agente di controllo)
  - Legge la configurazione
  - Esegue i controlli
  - Agisce al verificarsi di determinate condizioni
- Logica dei controlli
  - Target UP
  - Target DOWN
  - Target FAILING
  - Target UNCHECKABLE
  - Target INACTIVE (trasparente)
  - 2 limiti:
    - Valore
    - Tolleranza temporale
- Libreria dei controlli
  - Nodo
    - Raggiungibilità (MTU configurabile)
    - Occupazione CPU, FS, RAM, VMEM
    - Reboot
    - Presenza di un processo
    - Raggiungiblità TCP
    - Sincronizzazione con server NTP
    - Adiacenza BGP, OSPF
    - Match di un URL con una espressione regolare
    - WMI
  - Interfaccia (supporta variazione di ifIndex)

        - Stato
        - Numero di errori
        - Pacchetti non unicast ricevuti
        - Full duplex
        - Traffico (supporta contatori a 32 e 64 bit)
        - STP
        - Variazione di stato
        - Variazione costo root bridge
        - Variazione porta root bridge
- CLI per la configurazione
    - Creazione e gestione di categorie di nodi, interfacce, controlli
    - Creazione e gestione di nodi, interfacce e controlli
    - Creazione e gestione di alberi e contenitori
    - Quando controllare
    - Quando e a chi mandare la segnalazione
    - Sospendere il controllo di un nodo
    - Snmpwalk integrato
- Interfaccia web
    - Visualizzazione dello stato e delle misure
    - Feed RSS
    - Mappe

5.3 Andando verso il bazaar...

Alla ConfSL09 il rilascio è stato annunciato come *Open Source Prerelease* a causa della mancanza di un'adeguata documentazione e dell'esternazione del repository Subversion per lo sviluppo.

Ci siamo subito concentrati nel colmare queste lacune e quindi:

- la documentazione è stata completata e tradotta in inglese
- abbiamo trasferito su Sourceforge tutto il repository Subversion con la storia dello sviluppo, rimediando ad alcuni *errori di giovinezza*: abbiamo eliminato alcune password che erano state inserite in passato e la licenza è stata applicata in modo retroattivo
- abbiamo riportato nel TRAC di Sourceforge i bug applicativi

Fatto il nuovo *tarball* con i primi *bugfix*, ci siamo anche confrontati internamente sullo stato del software: quello che noi consideravamo versione 2.x non poteva essere considerato alla stessa stregua dalla comunità del software libero.

Perciò abbiamo deciso di effettuare il *downgrade* di versione dalla 2.3 alla 0.2.3: SANET è funzionante, ma è ancora in evoluzione e soprattutto non ha ancora la *confezione* necessaria per essere almeno 1.0.

5.4 Il secondo rilascio al termine del Master FOSSET0809

Siamo a inizio novembre e SANET è andato molto avanti rispetto al rilascio di giugno. C'è stato tutto il lavoro sulle mappe (fino ad agosto), ma non solo. Il *poller* integra molti più controlli, ed è stato realizzato un modulo per la reportistica.

In questo periodo la crescita della comunità non è stata fra le priorità LABS che ha preferito spingere sulle nuove funzionalità.

A cinque mesi dal rilascio si contano:

- 120 download dell'applicazione e 56 dell'articolo realizzato per la ConfSL09
- un canale IRC frequentato solo da sistemisti LABS
- un repository Subversion che è molto più avanti del tarball

Colgo l'occasione col dire che io, lo sviluppatore principale del progetto, ho interrotto il rapporto di lavoro dipendente con i LABS il 30 settembre. Questo aspetto è molto importante e darà adito ad alcune riflessioni che però lascio alla sezione *Verifica e prospettive*.

Di cosa ha bisogno SANET ora ?

Ho pensato di curare il rilascio di questa nuova versione, la 0.3.9.

I cambiamenti sono stati molti e ci avviciniamo alla 0.4. È giunto il momento di realizzare la procedura di installazione che si occupi di verificare se tutte le dipendenze del sistema sono soddisfatte.

È abbastanza frequente che si verifichino errori a causa di vecchie librerie, o mancanza di alcuni prerequisiti.

L'evoluzione naturale del rilascio del software sarebbe di ampliare il bacino di utenza e ampliare i canali di comunicazione con la comunità. Per fare questo è innanzi tutto importante pacchettizzare il software per una distribuzione. Un'altra idea sarebbe di aprire un blog specializzato.

Al momento, considerata l'evoluzione dei rapporti, non si è pensato di proseguire riguardo a questi ultimi due passi. Si intende discuterne con LABS che detiene il diritto di paternità del software e quindi l'interesse nella diffusione dell'implementazione.

In questa fase abbiamo quindi congelato lo sviluppo e creato il file standard setup.py per la distribuzione di applicativi python. Lo script verifica le dipendenze e installa il software. Inoltre è stato adottato pip per la generazione dell'elenco di librerie python richieste con le rispettive versioni. Per quello che riguarda la verifica delle dipendenze fra applicativi i manutentori di moduli python suggeriscono di occuparsene usando i gestori pacchetti delle distribuzioni specifiche.

Abbiamo cercato di andare oltre, per verificare le dipendenze rispetto ad applicativi esterni. Sicuramente lo strumento per eccellenza a questo fine sono gli *autotools*, ma l'idea è giunta tardi e, alla data di stesura di questo documento, non c'è stato modo di provarli. È stato invece realizzato lo script ad-hoc *install_requirements.sh* che verifica la presenza di corrette librerie NET-SNMP e PostgreSQL che sono elementi cruciali del sistema. La sua esecuzione è stata integrata nel *setup.py* per mantenere comunque la procedura standard di installazione pacchetti python. Ripeto che il prossimo passo sarà ripassare gli *autotools* e provare con quelli.

Il senso di creare la 0.3.9 (alpha 1) è quello di avere margine per alcune modifiche grafiche nell'integrazione nell'interfaccia del modulo dei report che necessita di alcune migliorie prima della 0.4.

Verifica e prospettive

Indubbiamente SANET esiste, funziona e tecnicamente si può dire che è uno degli strumenti più avanzati nel panorama dei Network Management System Open Source.

Sicuramente può e deve ancora migliorare sotto l'aspetto della comunità, delle performance, della distribuzione degli agenti, della personalizzazione degli eventi.

Potrebbe anche interagire con sistemi di *ticketing* per la gestione del *workflow* di risoluzione degli allarmi.

In questo capitolo si intende proporre una verifica e analizzare alcuni possibili prospettive future relativamente al processo di sviluppo e rilascio e agli attori attualmente interessati al buon funzionamento e al miglioramento di SANET.

6.1 Verifica

Guardarsi alle spalle dopo qualche anno di lavoro non è semplice. Neanche le dinamiche che hanno portato all'evoluzione di SANET sono semplici, come non sono semplici le dinamiche della comunità Open Source.

Vivendo dall'interno questo processo ci si rende conto che alla fine le uniche cose che contano veramente sono le priorità aziendali. Chi è nel mondo dell'Open Source da anni non può non sapere che è questa la regola che vale per i prodotti di successo. È il trionfo di un sano liberismo in cui le priorità aziendali sono anche quelle che portano a un miglioramento sociale e comunitario.

Purtroppo le priorità aziendali, accompagnate da un contesto sociale in cui la mancanza di tempo e la fretta predominano, a volte portano a compiere scelte affrettate e si perdono inevitabilmente occasioni di crescita derivanti da un confronto più approfondito con lo stato dell'arte e la comunità del software libero.

Nel complesso io credo che la forte esperienza dei LABS nella gestione delle reti e il precedente sistema di monitoraggio *pinger*, abbiano giocato un ruolo fondamentale nella decisione di non puntare ad integrarsi a soluzioni esistenti. Questa scelta potrebbe essere criticata anche considerando che LABS storicamente non è un'azienda di sviluppo software e quindi sarebbe stato più produttivo avvalersi della forza della comunità invece che creare e mantenere un nuovo programma. Allo stesso modo potrei dire che con una unità di sviluppatori più corposa si sarebbe potuto indagare meglio sulle peculiarità degli altri sistemi di monitoraggio perché si sa ... la barriera di entrata è più alta se si deve entrare in un progetto avviato. Soprattutto se ci si deve entrare con un know-how importante da integrare.

Considerati questi aspetti, sebbene io stesso fossi promotore di un approccio più aperto ed integrato con le soluzioni esistenti, comprendo le scelte che l'azienda ha fatto.

Ora che il prodotto è maturo per un uso interno, credo che sia il momento di aprirsi in modo più deciso alla comunità. In questo modo ci si potrebbe avvalere anche di collaboratori esterni interessati allo sviluppo di SANET che potrebbero apportare contributi in documentazione, codice, pubblicità o nuove idee senza gravare sui costi aziendali.

Riguardo alla tecnologia usata ritengo che siano state fatte scelte appropriate:

- Python è un linguaggio di programmazione versatile e veloce che consente programmazione funzionale, ad oggetti e a metaclassi
- Django è un framework web che non snatura python e ne sfrutta ottimamente le potenzialità
- PostgreSQL è potente e comunque la scelta di appoggiarsi ad un RDBMS è di indubbia utilità
- Flash, per quanto riproducibile ottimamente solo con il plugin proprietario di Adobe è stupefacente
- RRD è il punto di riferimento per le serie temporali e il consolidamento dei dati
- RSS + XML-RPC e folksonomie sono tecnicismi del Web 2.0 che adattano SANET anche alla consultazione tramite strumenti esterni

6.2 Prospettive

Un altro punto su cui bisogna porre molta attenzione sono secondo me le prospettive che si aprono a partire da uno strumento di software libero creato a cattedrale da un'azienda e poi rilasciato nella sua interezza.

Cosa succederà ?

Il rilascio sicuramente è ancora una scommessa, per l'azienda e per me che sono stato uno dei principali autori di SANET. Date le logiche aziendali che hanno condotto fin qui il progetto, non mi sembra che sia fra le priorità dell'azienda far crescere la comunità di sviluppatori intorno a SANET.

Ma il punto è un altro: SANET nasce come prodotto all'interno di LABS. Effettuando il rilascio alla comunità LABS, pur non rinunciando al diritto (inalienabile) di paternità dell'opera, di fatto offre un ruolo centrale al prodotto su cui convogliare l'interesse in modo diretto: per accedere al programma, alla conoscenza e alle funzionalità da esso implementate non si deve più necessariamente passare attraverso LABS.

È in questo, come si è ricordato più volte, che si denota una grande apertura di mentalità dell'azienda.

Ora, l'interesse è catalizzato dal prodotto e di fatto, ogni attore del mercato può *dare una spinta* verso il miglioramento.

Ad esempio io in questo momento, nella scrittura di questo progetto finale di Master, sto scrivendo una documentazione sull'evoluzione di SANET, e ho realizzato la procedura di installazione del software pur non essendo alle dipendenze dell'azienda.

Poniamo caso che l'azienda beFair da me costituita nutra interesse per l'attività di monitoraggio delle reti, o individui l'esigenza di un cliente che possa essere soddisfatta con degli aggiornamenti su SANET, ebbene, sarebbe interesse dell'azienda beFair e dell'azienda LABS discutere del miglioramento ed integrarlo nella versione ufficiale di SANET.

Questo è il punto fondamentale e la grande apertura del rilascio alla comunità: siamo tutti curiosi di vedere come si evolverà questa situazione.

Appendice A - Osservazioni su NMS open source in lista cisco-nsp

La mailing-list **cisco-nsp** è una lista di discussione per persone che usano apparati Cisco in Network Service Provider.

Riportiamo qui di seguito un messaggio che include, nella parte quotata, alcune osservazioni sui sistemi di monitoraggio Open Source e la risposta dell'Ing. Michele Bergonzoni, sistemista senior in reti di telecomunicazioni (nda: direi più un *network god*), nonché autore unico di *pinger* e coautore di *SANET*.

Il messaggio assume un significato di particolare importanza in quanto riesce a cogliere gli aspetti tecnici più nascosti per cui si è deciso di sviluppare un sistema alternativo per il monitoraggio delle reti.

Inizia qui il messaggio reperibile all'indirizzo [c-nsp] Free NMS Tools:

```
Michele Bergonzoni bergonz at labs.it
Fri Jul 17 09:26:28 EDT 2009

    * Previous message: [c-nsp] Free NMS Tools
    * Next message: [c-nsp] Free NMS Tools
    * Messages sorted by: [ date ] [ thread ] [ subject ] [ author ]

Saku Ytti <saku at ytti.fi> said:

> To me all OSS NMS solutions out seem like they are made by
> coder-in-server-admin not coder-in-network-admin, and as such seem to
>  have much more integration with servers than with network

This is one of the reasons why over the years we developed sanet, the
other being that many NMSs are very chatty and tend to keep you up all
night when relayed on pagers and SMS.

Sanet is OSS but in prerelease, meaning that we use it and it works, but
its documentation is not quite complete and it is not easy to install.
If you are willing to setup many python packages by hand and to explore
funcionalities without a concise HOWTO, or if you are just interested in
the OIDs, you can find it at sanet.sf.net, the SVN version being much
better (expecially for maps and reports) than the downloadable version.

We use it mainly in multivendor corporate networks, but we have one case
of cisco MPLS carrier network.
```

```
> Why don't they ship with MIBs or just specific OIDs for few top
> vendors important traps etc?

sanet has a a library of checks for common cisco, HP, fortigate and
other vendor's OIDs. Sorry we don't collect traps nor syslog in the
sanet DB, we usually transform traps to syslog (net-snmp snmptrapd) and
collect syslog (we are accustomed to grepping the results).

> Adding appropriate reaction classification.

Sanet does not react. You can trivially achieve that by binding scripts
to emails, etc., but we are quite scared of this kind of triggering and
we don't do it (yet).

> People want NMS to automatically monitor BGP

In the library there is the check for the BGP neighborship state:

"1.3.6.1.2.1.15.3.1.2.$peer_ip:$community@$node == 6"

it is not "automatic" because in sanet you have to decide all the
monitoring that you want it to do.

> OSPF

We have the OSPF neighborship state check:

"1.3.6.1.2.1.14.10.1.6.byRegexpUnique(1.3.6.1.2.1.4.20.1.2,^$ifindex$).0:$linked_community@$linked_no
== 8"

but it works only for point-to-point links. I'm sure we can make it better.

> IS-IS

Sorry no IS-IS here, but of course you can define your own if you know
the OIDs. Please contribute it back if you do.

> LDP

We have an LDP neighborship check:

"1.3.6.1.2.1.10.166.4.1.3.2.1.2.byBinaryIP(1.3.6.1.2.1.10.166.4.1.3.2.1.5:$community@$node,$peer_ip)
== 2"

> status of some other CPU/memory than just control-plane

Well, for IOS we usually check processor memory and IO memory. OIDs and
suggestions are very, very welcome.

> Other thing that annoys me is how SNMP pollers are implemented,
> they're blocking

You are definitely right. Our poller is multithreaded but each thread is
blocking, with adjustable timeouts.

> While having SNMP poller poll 140k OID per second on 386 class PC is
> rather trivial, using two process strategy, where single process
```

```
> spews packets outs, and another listens what comes back, completely
> asynchronous

It was not so trivial for us, so we made it synchronous. The tricky part
is to collect all the SNMP vars used to form an expression in the same
moment (of course with some approximation), remembering what you asked
for at each poll cycle. It is trivial if you just check variables
against ranges, but we build complex expressions with current and past
variables.

Anyway, patches are welcome...

> I've also only seen alarms based on absolute values of different
> counters

sanet can combine current and past (last poll cycle) vars, like this
expression for a threshold on broadcast packets received:

"((1.3.6.1.2.1.2.2.1.12.$ifindex:$community@$node -
1.3.6.1.2.1.2.2.1.12.$ifindex#$node) / $delta) < $threshold"

($delta is the time in second since last poll)

> This type of 'trending' module should be relatively easy, and could
> be reused by any counter values.

This is a good idea, I will try to think about how this can fit into our
existing software or if a new check type is needed for that.

> I demoed zenoss with 27 routers and it froze trying to poll their
> interface (granted there are very many interfaces)

We measure installations from the number of targets (yes/no checks) and
measures (graphs). One of our big ones is:

root at XXXXXX:~# sanet-cli
Benvenuti in SANET 2 su XXXXX

sanet# sh ver
...
Configuration defines 831 interfaces, 523 nodes, 409 links, 9868
targets, 2089 measures.
Targets summary: 9 down, 1 failing, 38 uncheckable, 0 out of time, 9820 up
Measures summary: 2042 updated in last 2 mins, 2089 in last 5 mins, 2089
in last 30 mins

(this is running on a XEN VM, I/O being the bottleneck)

I'm sure people on this list will appreciate the configuration via CLI
(web is used for displaying the status), which is shamelessly copied
from IOS. This was "sh ver", and in order to configure monitoring you
start with "conf t". You will probably appreciate physical maps (a /30
is a line, not a line with a cloud in between), NTP checks, IPv4/IPv6
pings with adjustable payload length, iface designation by name, MAC,
IP, CDP neighbor, route, IOS description, etc (no ifindex blues).

Hope this helps,
                                    Bergonz
```

```
--
Ing. Michele Bergonzoni - Laboratori Guglielmo Marconi S.p.a.
Phone:+39-051-4392826 Fax:+39-051-6153683 e-mail: bergonz at labs.it
alt.advanced.networks.design.configure.operate
```

Appendice B - analisi della sicurezza di SANET

In questa appendice si vuole effettuare una breve disamina delle problematiche di sicurezza di SANET.

SANET è un software Web il che vuol dire tutto e non vuol dire niente. Diciamo che dialoga all'esterno tramite il protocollo HTTP.

Oltre ad ascoltare richieste HTTP, SANET effettua disparate richieste sulla rete da cui si aspetta determinate risposte. Tali richieste vengono fatte molto spesso tramite protocollo SNMP, ma è altrettanto possibile che siano implementati comandi ad-hoc per interagire con qualunque altro esotico servizio.

Infine SANET dispone della CLI di configurazione che è accessibile in locale e per il quale si applica una politica di *trust* degli utenti del sistema.

Quindi gli attacchi esterni possono provenire da:

- richieste malformate veicolate sulle porte HTTP
- informazioni ricevute in risposta alle richieste effettuate per la rete monitorata

Mentre gli attacchi interni da:

- *code injection* in comandi impartiti nella CLI locale
- *sql injection* nel database locale

oltre ai più classici *Denial Of Service* per l'esterno e *bug* di libreria di cui segnaleremo solo un caso critico.

8.1 Attacchi esterni

Iniziamo col dire che SANET soffre ancora di problemi di performance. È un sistema *CPU-bound* (oltre che *Network-bound* per quello che riguarda il monitoraggio delle condizioni di rete), per cui non è difficile realizzare un attacco Denial Of Service e fare in modo che altri utenti non possano accedere al sistema di monitoraggio e visualizzare lo stato della rete.

Tolto il problema del Denial Of Service bisogna considerare il tipo di operazioni che vengono esposte via web:

- richiesta generica di un url
- salvataggio note e posizioni degli elementi nella mappa: queste operazioni scrivono dati nello RDBMS

La richiesta generica di un url potrebbe dare problemi in caso di pacchetto HTTP malformato, oppure di corpo della richiesta forgiato ad-hoc per operazioni di code injection.

SANET è basato su Django che nelle installazioni in produzione è gestito tramite l'handler *mod_python* di Apache. Quindi per quello che riguarda i livelli ISO/OSI fino al livello HTTP , la sicurezza è garantita dall'implementazione dello stack TCP/IP del sistema operativo di installazione (GNU/Linux per noi), mentre per il livello HTTP abbiamo Apache Web Server.

Il contenuto del pacchetto HTTP invece viene trasmesso a livello applicativo anche in questo caso molto robusto:

- del web server non occorre dire altro
- Django dispone di un ottimo sistema di quoting e sanitizzazione dell'input grazie alla libreria cgi di Python
- inoltre i parametri valorizzati compilando i *form* nella canonica interfaccia web, vengono serializzati tramite la libreria *Javascript Prototype* che ne garantisce il corretto quoting nell'url della richiesta in caso di una *GET*, o nell'url e nel *payload* in caso di una *POST* (operazioni HTTP più comuni). Certo, questa è una considerazione esclusivamente marginale se si pensa che in un attacco informatico di solito l'input non viene fornito tramite l'interfaccia web applicativa.

Rimane la possibilità di iniezione di codice attraverso risposte malformate alle richieste di monitoraggio di *poller*. In questo caso le richieste SNMP sono state incapsulate e attualmente si appoggiano ai binding python di NetSNMP. La nota positiva è che essendo incapsulati possono essere facilmente rimpiazzati da altri moduli che implementano richieste SNMP, la nota negativa è che ogni tanto l'implementazione attuale raramente va in *Segmentation fault*. Questo problema si è verificato per siti molto grandi, dove incide molto la latenza della rete nell'effettuazione dei controlli.

Dopo una indagine approfondita del problema non se ne è riuscita ad individuarne la causa, e il problema (che io sappia) non è stato sottoposto alla comunità di sviluppo per mancanza di informazioni sulla riproducibilità e la particolarità delle reti in cui questo problema si verifica. È stato comunque realizzato un *kludge* con un processo padre che ha il solo compito di supervisionare l'esistenza del figlio che effettua i controlli di rete e riavviarlo in caso non esista più.

Gli altri controlli di rete sono implementati da funzioni apposite e il parsing dei risultati effettuato di volta in volta a seconda del tipo di controllo configurato. In genere il valore di ritorno viene considerato come stringa o come uno specifico tipo di dato dipendentemente dall'operatore definito nel controllo e relativamente a questo correttamente interpretato. In ogni caso mai valutato.

Il sistema non riceve trap SNMP dagli apparati.

Non si può chiudere la sezione sugli attacchi esterni senza porre in evidenza il rischio relativo alla compromissione del DNS cui *poller* dirige numerose richieste. Se infatti venisse compromesso il DNS interrogato, si potrebbero indirizzare a piacimento tutte queste richieste ad esempio ad un singolo host e generare un DoS. Questo considerato anche che in reti mediamente complesse il processo dispone di circa 40 thread che fanno richieste in parallelo.

8.2 Attacchi interni

Come si accennava prima, è stata adottata una politica di sicurezza *trust* per gli utenti del sistema locale, quindi è ovvio che se qualcuno riuscisse ad entrare nel sistema con un account autorizzato, molto probabilmente riuscirà a far fare a SANET quello che vorrà.

Ma l'altra domanda che ci si pone è se SANET possa o meno presentare un pericolo reale per il sistema, o meglio, se sia possibile ottenere un accesso privilegiato sfruttando un problema di sicurezza di SANET.

Una prima considerazione da fare è che il *poller* viene eseguito come utente *root* del sistema, perché deve poter forgiare pacchetti RAW per fare ICMP ECHO REQUEST (ping). Questo già potrebbe essere facilmente migliorabile nel caso si avesse a disposizione un kernel Linux che supporti le *capabilities* e in particolare si potrebbe creare un account con la capability *CAP_NET_RAW*.

Il fatto che tutti i controlli di rete vengano eseguiti da un processo con utente *root*, rende il sistema nudo di fronte a un exploit del *poller*.

Inoltre è necessario porre particolare attenzione alla configurazione della access list del database: se l'utente non privilegiato può scrivere la configurazione di SANET nel database (tramite la CLI che non ha alcun controllo all'accesso), a questo punto l'attacker può configurare SANET , e quindi il *poller* per i propri fini.

Le espressioni di controllo dello stato dispongono di un proprio linguaggio con un parser sviluppato ad-hoc e quindi si può affermare che sono protette da iniezione di codice, però ci sono particolari espressioni (i.e: *wmic* ed *exec*), che eseguono programmi esterni. È necessario verificare che le directory e i file che sono già presenti e pronti per essere eseguiti, non siano modificabili dall'utente non privilegiato, che altrimenti troverebbe una facile scorciatoia per eseguire codice arbitrario nel sistema.

Infine è doveroso considerare che la macchina su cui viene eseguito il processo *poller* deve, per definizione, raggiungere numerosi apparati di rete e server, più di quanto tipicamente possa fare un server qualsiasi nella rete del cliente: ciò rende più appetibile questa macchina come bersaglio per chi voglia usarla come testa di ponte per altri attacchi o attività di ingegneria sociale.

8.3 Concludendo

SANET è un sistema protetto dagli attacchi esterni, ma molto vulnerabile dagli attacchi interni. Per questo si consiglia di ridurre i privilegi dell'utente di esecuzione del *poller* sfruttando la capability *CAP_NET_RAW*, oppure installando il sistema di monitoraggio in una macchina virtuale eseguita con i diritti di utente non privilegiato e con un proprio stack TCP/IP con cui produrre i pacchetti RAW di cui necessita per il corretto funzionamento.

Appendice C - SVG o FLASH ?

A favore di SVG:

- Standard W3C, basato su XML invece che su un formato binario. È progettato esplicitamente per lavorare con altri standard W3C come ad esempio CSS, DOM e SMIL(Synchronized Multimedia Integration Language)
- Le potenzialità sono equivalenti al Flash
- Meglio supportato sui cellulari (anche se nella versione tiny: http://www.w3.org/TR/SVGMobile12/)
- È vero che si trovano meno realizzazioni in SVG, ma ci sono più risorse cui attingere e di conseguenza un più rapido sviluppo implementazioni di visualizzatori SVG http://wiki.svg.org/Viewer_Implementations
- Scrivibile come un file di testo e quindi direttamente con i template di Django (questo è notevole)

A favore di FLASH invece:

- Plugin Flash gira su tutte le piattaforme più diffuse (ora anche 64bit), ma la chiave di volta (purtroppo) è la compatibilità con Internet Explorer
- Supporto per Adobe SVG viewer è stato abbandonato dal 1 gennaio 2009 http://www.adobe.com/svg/eol.html
- Adobe ha rilasciato Flex: una piattaforma open source per lo sviluppo di applicazioni Flash MVC
- Sembra essere più leggero e quindi produrre animazioni più fluide

Appendice D: XML prodotto per una mappa con contenitori aperti

Di seguito un exempio di XML per una mappa con alcuni sottocontenitori aperti. Per i singoli elementi sono stati lasciati solo gli attributi xml più significativi.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sanet_map [
        <!ELEMENT graph (subgraph, node, edge)>
        <!ELEMENT subgraph (subgraph, node, edge)>
        <!ELEMENT node EMPTY>
        <!ELEMENT edge EMPTY>
]>
<graph id="map_root" name="Sottocontenitori aperti" bb="">
        <subgraph id="container-2" name="Sedi e rack" icon="/img/container128x128.png" bg="">
                <subgraph id="container-3" name="01 container" icon="/img/container128x128.png" bg="
                        <node id="node-1" name="node-1" icon="/img/node128x128.png" y="55.0">
                                <iface id="iface-4" href="/sanet/state/iface/4/"/>
                                <iface id="iface-1" href="/sanet/state/iface/1/"/>
                                <iface id="iface-2" href="/sanet/state/iface/2/"/>
                                <iface id="iface-3" href="/sanet/state/iface/3/"/>
                        </node>
                        <node id="node-2" name="node-2" icon="/img/node128x128.png" y="53.0">
                                <iface id="iface-6" href="/sanet/state/iface/6/"/>
                                <iface id="iface-7" href="/sanet/state/iface/7/"/>
                                <iface id="iface-5" href="/sanet/state/iface/5/"/>
                                <iface id="iface-8" href="/sanet/state/iface/8/"/>
                        </node>
                        <node id="node-13" name="node-13" icon="/img/node128x128.png" y="176.0">
                                <iface id="iface-9" href="/sanet/state/iface/9/"/>
                                <iface id="iface-10" href="/sanet/state/iface/10/"/>
                        </node>
                </subgraph>
                <subgraph id="container-4" name="02 container" icon="/img/container128x128.png" bg="h
                        <node id="node-39" name="node-39" icon="/img/node128x128.png" y="18.0">
                                <iface id="iface-30" href="/sanet/state/iface/30/"/>
                                <iface id="iface-28" href="/sanet/state/iface/28/"/>
                                <iface id="iface-29" href="/sanet/state/iface/29/"/>
                        </node>
```

```
                <node id="node-40" name="node-40" icon="/img/node128x128.png" y="18.0">
                        <iface id="iface-33" href="/sanet/state/iface/33/"/>
                        <iface id="iface-31" href="/sanet/state/iface/31/"/>
                        <iface id="iface-32" href="/sanet/state/iface/32/"/>
                </node>
                <node id="node-41" name="node-41" icon="/img/node128x128.png" y="91.0">
                        <iface id="iface-35" href="/sanet/state/iface/35/"/>
                        <iface id="iface-38" href="/sanet/state/iface/38/"/>
                        <iface id="iface-34" href="/sanet/state/iface/34/"/>
                        <iface id="iface-40" href="/sanet/state/iface/40/"/>
                        <iface id="iface-44" href="/sanet/state/iface/44/"/>
                        <iface id="iface-41" href="/sanet/state/iface/41/"/>
                        <iface id="iface-36" href="/sanet/state/iface/36/"/>
                        <iface id="iface-37" href="/sanet/state/iface/37/"/>
                        <iface id="iface-39" href="/sanet/state/iface/39/"/>
                        <iface id="iface-42" href="/sanet/state/iface/42/"/>
                        <iface id="iface-43" href="/sanet/state/iface/43/"/>
                </node>
                <node id="node-43" name="node-43" icon="/img/bigswitch128x128.png" y="194.0">
                        <iface id="iface-87" href="/sanet/state/iface/87/"/>
                        <iface id="iface-86" href="/sanet/state/iface/86/"/>
                        <iface id="iface-84" href="/sanet/state/iface/84/"/>
                        <iface id="iface-76" href="/sanet/state/iface/76/"/>
                        <iface id="iface-275" href="/sanet/state/iface/275/"/>
                        <iface id="iface-85" href="/sanet/state/iface/85/"/>
                        <iface id="iface-78" href="/sanet/state/iface/78/"/>
                        <iface id="iface-73" href="/sanet/state/iface/73/"/>
                        <iface id="iface-74" href="/sanet/state/iface/74/"/>
                        <iface id="iface-75" href="/sanet/state/iface/75/"/>
                        <iface id="iface-68" href="/sanet/state/iface/68/"/>
                        <iface id="iface-69" href="/sanet/state/iface/69/"/>
                        <iface id="iface-276" href="/sanet/state/iface/276/"/>
                        <iface id="iface-247" href="/sanet/state/iface/247/"/>
                        <iface id="iface-72" href="/sanet/state/iface/72/"/>
                        <iface id="iface-79" href="/sanet/state/iface/79/"/>
                        <iface id="iface-80" href="/sanet/state/iface/80/"/>
                        <iface id="iface-81" href="/sanet/state/iface/81/"/>
                        <iface id="iface-82" href="/sanet/state/iface/82/"/>
                        <iface id="iface-83" href="/sanet/state/iface/83/"/>
                </node>
        </subgraph>
        <subgraph id="container-6" name="04 container" icon="/img/container128x128.png" bg="
                <node id="node-68" name="node-68" icon="/img/internet128x128.png" y="0.0">
                        <iface id="iface-251" href="/sanet/state/iface/251/"/>
                        <iface id="iface-252" href="/sanet/state/iface/252/"/>
                        <iface id="iface-253" href="/sanet/state/iface/253/"/>
                        <iface id="iface-254" href="/sanet/state/iface/254/"/>
                        <iface id="iface-255" href="/sanet/state/iface/255/"/>
                        <iface id="iface-256" href="/sanet/state/iface/256/"/>
                        <iface id="iface-257" href="/sanet/state/iface/257/"/>
                        <iface id="iface-258" href="/sanet/state/iface/258/"/>
                        <iface id="iface-268" href="/sanet/state/iface/268/"/>
                </node>
        </subgraph>
        <node id="container-5" name="container-5" icon="/img/container128x128.png" y="14.0">
                <iface id="iface-239" href="/sanet/state/iface/239/"/>
                <iface id="iface-241" href="/sanet/state/iface/241/"/>
                <iface id="iface-240" href="/sanet/state/iface/240/"/>
```

```
                </node>
                <node id="container-7" name="container-7" icon="/img/container128x128.png" y="63.0"/>
                <node id="container-8" name="container-8" icon="/img/container128x128.png" y="4.0"/>
                <node id="container-19" name="container-19" icon="/img/container128x128.png" y="0.0">
                        <iface id="iface-270" href="/sanet/state/iface/270/"/>
                        <iface id="iface-269" href="/sanet/state/iface/269/"/>
                        <iface id="iface-267" href="/sanet/state/iface/267/"/>
                        <iface id="iface-266" href="/sanet/state/iface/266/"/>
                        <iface id="iface-265" href="/sanet/state/iface/265/"/>
                        <iface id="iface-261" href="/sanet/state/iface/261/"/>
                        <iface id="iface-263" href="/sanet/state/iface/263/"/>
                        <iface id="iface-264" href="/sanet/state/iface/264/"/>
                </node>
        </subgraph>
        <edge fromID="node-1" toID="node-2" fromIfaceID="iface-2" toIfaceID="iface-5" pos=""/>
        <edge fromID="node-1" toID="node-2" fromIfaceID="iface-3" toIfaceID="iface-8" pos=""/>
        <edge fromID="node-2" toID="node-13" fromIfaceID="iface-6" toIfaceID="iface-9" pos=""/>
        <edge fromID="node-13" toID="node-41" fromIfaceID="iface-10" toIfaceID="iface-43" pos=""/>
        <edge fromID="node-39" toID="node-40" fromIfaceID="iface-30" toIfaceID="iface-33" pos=""/>
        <edge fromID="node-43" toID="container-5" fromIfaceID="iface-68" toIfaceID="iface-239" pos="
        <edge fromID="node-43" toID="container-5" fromIfaceID="iface-69" toIfaceID="iface-240" pos="
        <edge fromID="node-43" toID="container-5" fromIfaceID="iface-75" toIfaceID="iface-241" pos="
        <edge fromID="container-19" toID="container-19" fromIfaceID="iface-263" toIfaceID="iface-270
        <edge fromID="container-19" toID="container-19" fromIfaceID="iface-266" toIfaceID="iface-269
        <edge fromID="container-19" toID="node-68" fromIfaceID="iface-267" toIfaceID="iface-268" pos=
</graph>
```

Indices and tables

- *Indice*
- *Indice dei Moduli*
- *Cerca*