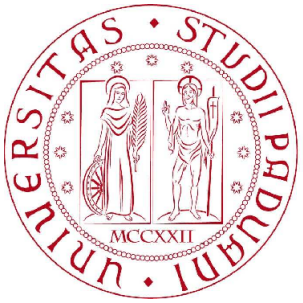

Università degli studi di Padova

Dipartimento di Matematica Pura ed Applicata

Report Tecnologie Open Source

Studente: Dissegna Stefano

Anno Accademico 2009-2010

Indice

1 Introduzione

Parrot è una macchina virtuale progettata per compilare ed eseguire efficientemente bytecode per linguaggi di programmazione dinamici, quali Perl, Python, PHP e Ruby.

1.1 Storia

Parrot nasce all'interno del progetto per implementare Perl 6, la nuova versione del linguaggio Perl. Il progetto poi diventa più ambizioso e si propone come target per tutti i linguaggi di programmazione dinamici. Il compilatore da Perl 6 a Parrot diventa un progetto a sè stante, denominato Rakudo. Il forte legame con Perl 6 comunque resta. Infatti, fino al Febbraio del 2009, con la release "Vienna" di Rakudo, i due progetti convivevano nello stesso repository. Inoltre molti degli sviliuppatori collaborano ad entrambi i progetti.

1.2 Origine del nome

Il nome deriva da un pesce d'aprile [1] organizzato congiuntamente dalle comunità Perl e Python, in cui annunciavano un nuovo linguaggio, chiamato per l'appunto Parrot, che sarebbe stato il frutto dell'unione dei due linguaggi. Visto l'obiettivo dichiarato del progetto, che si prefigge di creare una macchina virtuale su cui implementare, tra gli altri, i linguaggi Perl e Python, Parrot è diventato il nome del progetto.

2 Descrizione del progetto

Parrot supporta nativamente le feature più comuni dei linguaggi dinamici moderni: creazione, modifica, istanziazione di classi o funzioni a runtime, tipi dinamici, valutazione di codice durante l'esecuzione, etc. Il bytecode interpretato (o, nei piani futuri del progetto, compilato) dalla macchina virtuale può essere generato a partire da file testuali contenenti istruzioni assembly. La macchina è basata su registri, diversamente da un approcio più diffuso, adottato ad esempio dalla JVM, basato su stack. Il codice assembly può accedere ad un numero infinito (o, meglio, non limitato a priori) di registri, che vengono poi mappati dal compilatore su registri reali oppure in memoria. Il bytecode che gira sulla VM può essere generato dai compilatori HLL (High Level Languages, ovvero linguaggi ad alto livello che hanno Parrot come target) in tre modi differenti:

- Generazione di assembly a basso livello, chiamato PASM (Parrot Assembly). Questa modalità di generazione di codice veniva ammessa agli inizi del progetto, ma al momento è altamente sconsigliata, in quanto necessita di trattare esplicitamente aspetti importanti come le calling convention, che sono soggetti a modifiche da parte del core team senza seguire il normale ciclo di deprecazione (descritto più avanti).

- Generazione di assembly ad alto livello, chiamato PIR (Parrot Intermediate Representation). Si tratta fondamentalmente di uno strato di zucchero sintattico sopra al codice PASM e si occupa di nascondere alcuni dettagli di basso livello. Il codice generato può essere salvato in un file che potrà poi essere pre-compilato in un file di bytecode oppure direttamente caricato ed eseguito dalla VM. Il codice può anche essere generato dentro ad una stringa e poi compilato (o interpretato) direttamente in memoria.

- Creazione di un albero sintattico (PAST, Parrot Abstract Syntax Tree) che rappresenta il codice. A partire da quest'albero verrà generato automaticamente il codice PASM da inviare alla VM. Questa è la modalità consigliata, ed è quella seguita dal progetto Rakudo.

Oltre alla macchina virtuale, Parrot include diversi tool per aiutare gli HLL developer nello sviluppo di un compilatore per il proprio linguaggio. L'insieme di questi tool si chiama PCT (Parrot Compilers Toolkit). I principali tool messi a disposizions sono:

- Uno script Perl per la generazione in automatico dello scheletro di un compilatore. Questo scheletro contiene i file necessari per il build automatico, l'esecuzione di test, script di installazione/disinstallazione, una grammatica d'esempio per il linguaggio ed altro.

- Un motore per la generazione in automatico di parser, chiamato PGE (Parrot Grammar Engine). Simile a Yacc, permette di descrivere in modo dichiarativo la grammatica di un linguaggio. PGE è fortemente integrato con il resto del sistema, e può essere caricato, ed utilizzato, dinamicamente nella VM. Questo rende possibile, tra le altre cose, modificare la grammatica di un linguaggio durante l'esecuzione di un programma.

- Un motore per attraversare gli alberi sintattici generati dal PGE e per generare alberi PAST. Questo motore si chiama TGE (Tree Grammar

Engine). Il TGE, come anche il PGE, è utilizzabile in congiunzione con un qualsiasi linguaggio in grado di eseguire su Parrot, oppure direttamente con il linguaggio PIR.

- Un semplice linguaggio di programmazione, chiamato NQP (Not Quite Perl). NQP è un sottoinsieme estremamente ristretto di Perl 6. Supporta i costrutti di base (cicli, if, switch), definizione di funzioni, un sottoinsieme del sistema ad oggetti, i tipi di base (numeri, stringhe, array, tabelle di hash) e non dispone di una libreria di base oltre a quanto è già fornito da Parrot stesso. Permette di annegare codice assembly (PIR) all'interno dei sorgenti per le volte in cui è necessario accedere a funzionalità a basso livello. NQP è utile soprattutto per la fase di bootstrapping di un linguaggio, e viene utilizzato da Rakudo in congiunzione con PGE e TGE per l'implementazione di Perl 6.

2.1 Problematiche

Al momento, Parrot è una piattaforma ancora immatura. Durante lo sviluppo di un linguaggio basato su Parrot [2] ho riscontrato diversi problemi:

- Lentezza. Al momento la VM è estremamente lenta, almeno di uno o due ordini di grandezza (a seconda del tipo di programma) rispetto ad implementazioni di linguaggi già relativamente lente come CPython. Le prestazioni possono variare fortemente tra una release e l'altra. Ad esempio, a distanza di una sola release (1 mese), il tempo d'esecuzione del benchmark che utilizzavo per valutare le prestazioni è raddoppiato. Dopo alcuni mesi è tornato alle prestazioni originarie.

- Ogni sottosistema è costantemente soggetto a cambiamenti radicali. Ad esempio, in questi mesi è appena iniziato il design di un nuovo sistema di threading, e sono in cantiere la riscrittura del sottosistema di I/O e del GC (garbage collector). Se da un punto di vista questo è un bene per l'innovazione della VM, dopo circa 9 anni di vita di un progetto ci si aspetta una maggiore stabilità di componenti così fondamentali. Questa costante volontà di rivedere l'architettura di base è una precisa scelta che ha caratterizzato Parrot fin dagli inizi e che, secondo il mio parere, ne ha causato il forte ritardo nella realizzazione di una versione abbastanza stabile da essere utilizzabile in un ambiente di produzione.

- Molti sottosistemi essenziali sono afflitti da talmente tanti bug da essere inusabili. L'esempio più eclatante è il supporto per il multithreading. È praticamente impossibile utilizzarlo senza incappare in un bug. Un

altro caso è la compilazione JIT (Just In Time): il bytecode era più veloce quando interpretato che quando compilato tramite il sistema JIT. Questo, unito al fatto che il codice era impossibile da manutenere in quanto gli sviluppatori che che vi avevano lavorato all'inizio al momento non partecipano più al progetto, ha portato alla decisione di rimuovere questo sottosistema e di rimpiazzarlo in futuro con uno scritto ex-novo e basato su LLVM [3].

- Sebbene la documentazione sia abbondante, spesso è difficile orientarvisi, ed è molto facile trovare on-line documentazione obsoleta che fa riferimento a versioni out-dated. L'unica documentazione sicuramente aggiornata è quella disponibile all'interno dello stesso repository dei sorgenti. Inoltre alcuni documenti fanno riferimento a feature non ancora implementate.

3 Mission

"Parrot is a virtual machine aimed at running all dynamic languages"

L'obiettivo è di creare una macchina virtuale in grado di supportare tutti i linguaggi di programmazione dinamici, in particolare il linguaggio Perl 6.

Lo stato dell'arte nell'implementazione di linguaggi di programmazione è estremamente avanzato, e ad una nuova implementazione, specialmente per un progetto che non dispone delle risorse tipiche delle grandi aziende, possono essere necessari anche 10 anni per implementare funzionalità considerate di base per un linguaggio moderno, come un garbage collector, un sistema ad oggetti, le eccezioni, supporto unicode, primitive per la concorrenza. Semplicemente riuscire a mettersi in pari rispetto ad altre implementazioni può essere considerata una grande conquista. Inoltre per poter essere utile un nuovo linguaggio ha bisogno di un ecosistema di librerie che permetta agli sviluppatori di risolvere problemi reali senza dover reinventare continuamente la ruota.

Con queste premesse, startare un nuovo linguaggio è tutt'altro che banale. Parrot ha come obiettivo di cambiare radicalmente la situazione. Fornendo un'implementazione delle funzionalità di base, permette ai nuovi linguaggi di occuparsi solo delle proprie caratteristiche distintive. Non dovendosi più preoccupare delle feature di base già ampiamente consolidate, il designer del linguaggio può permettersi di sperimentare nuovi costrutti e nuove idee. Perdere anni sperimentando un nuovo linguaggio è un rischio difficile da prendere, mentre perdere poche settimane è tollerabile. Inoltre, già dal

giorno 0 il linguaggio avrebbe a disposizione un'ampia collezione di librerie utilizzabili.

Il tempo necessario a rilasciare la prima versione di un nuovo linguaggio può passare da 10 anni a poche settimane. L'implementazione più veloce è stata durante una presentazione (un lightning talk di 5 minuti) in cui il presentatore ha implementato le basi del linguaggio LOLCODE (un linguaggio chiaramente scherzoso) partedo da zero fino al completamento.

4 Licenza

Parrot utilizza come licenza la Perl Artistic License 2.0. Si tratta di una licenza approvata dall'OSI. L'utilizzatore di Parrot dispone quindi dei seguenti diritti e doveri:

- Possibilità di utilizzare e modificare il software senza restrizioni fintantochè lo stesso non viene redistribuito.
- Possibilità di ridistribuire (anche a pagamento) copie del software originale, a patto che venga mantenuta la stessa licenza.
- Possibilità di ridistribuire una versione modificata del software a patto che le modifiche siano documentate e che almeno una delle seguenti condizioni sia verificata:
  - Rendere la modifica disponibile al possessore del copyright sotto la licenza originaria.
  - Il nome deve essere differente dall'originale a la sua installazione non deve interferire con l'installazione dell'originale.
  - Rendere il sorgente disponibile sotto la licenza originaria oppure una licenza che permetta di copiare, modificare e ridistribuire la versione modificata applicando gli stessi termini di licenza applicata a chi riceve la licenza di una copia e che qualsiasi lavoro derivato da essa sia liberamente disponibile.
- Se si ridistribuisce una versione compilata dell'originale, è necessario indicare le istruzioni per ottenere il sorgente originale.
- È possibile linkare o comunque aggregare il software originale con altri software (anche commerciali).

Si tratta quindi di una licenza liberale, adatta ad una macchina virtuale che, per sua natura, presenta come caso d'uso il linking con un prodotto che la usa o comunque la ridistribuzione della macchina virtuale congiuntamente ad altri prodotti. L'utilizzatore della macchina virtuale è lo sviluppatore di implementazioni di linguaggi di programmazione, mentre l'utente finale è lo sviluppatore che usa queste implementazioni. Una licenza restrittiva come la GPL di fatto non permetterebbe lo sviluppo di implementazioni proprietarie di linguaggi di programmazione che utilizzino la macchina virtuale Parrot, o di linguaggi open source integrati con prodotti proprietari, ad esempio come linguaggio di scripting.

Un altro motivo per l'utilizzo di questa licenza è che è la stessa adottata dal progetto Perl.

5 Mercato

Parrot si rivolge a tutti gli implementatori di linguaggi di programmazione dinamici. Quando Parrot era nato, nel 2001, non esistevano macchine virtuali con il preciso intento di supportare linguaggi dinamici. Le due VM più diffuse erano la JVM e .NET (e lo sono ancora), ma entrambe avevano lo scopo di supportare dei linguaggi di programmazione (Java e C#) molto più statici rispetto a Perl, Python o Ruby.

Tra gli utenti si possono individuare due sottocategorie:

Gli sviluppatori di nuovi linguaggi. Implementare un linguaggio di programmazione dinamico partendo da zero è un compito complesso: richiede lo sviluppo di un garbage collector efficiente, una parser, un compilatore (oppure un interprete), un sistema runtime, librerie di base ed altro. Inoltre, una volta ottenuta una buona implementazione ogni nuovo linguaggio deve affrontare il problema della carenza di librerie. Uno dei motivi del successo dei linguaggi che si basano sulla JVM (come Clojure, Scala o Groovy) è la capacità di accedere in modo semplice e diretto all'enorme bacino di librerie disponibili per Java. Altro problema per un nuovo linguaggio è la mancanza di una community iniziale che utilizzi il linguaggio riportandone i problemi allo sviluppatore e che contribuisca allo sviluppo.

Parrot, se raggiungesse il suo obiettivo, permetterebbe di risolvere, o quantomeno di alleviare, questi problemi:

Implementazione: lo sviluppatore del linguaggio non dovrebbe occuparsi del garbage collector o di buona parte dei sistema runtime e delle librerie di base, in quanto queste sono fornite direttamente da Parrot. Inoltre avrebbe a disposizione dei tool che semplificano la scrittura del parser e del compilatore.

Librerie: i linguaggi che girano su Parrot hanno la possibilità di interoperare tra di loro in modo trasparente. Uno degli esempi più utilizzati dalla community è che sarà possibile scrivere una classe in Python, realizzare una sottoclasse in Ruby e instanziare la sottoclasse in Perl. Se quest'obiettivo fosse raggiunto ogni linguaggio di programazione avrebbe accesso fin da subito a tutte le librerie esistente per questi tre linguaggi di programmazione senza dover scrivere binding ad hoc.

Community: ogni nuovo linguaggio entrerebbe a far parte della community Parrot.

Linguaggi di programmazione già esistenti. Linguaggi già esistenti potrebbero ottenere anch'essi i vantaggi sopra descritti da un'implementazione per Parrot. Lo sviluppo dei componenti comuni a molti linguaggi sarebbe compito del progetto Parrot, e si libererebbero così risorse da dedicare al miglioramento di altre parti più caratterizzanti del linguaggio.

Esiste una lista regolarmente aggiornata sulle implementazioni di linguaggi al momento disponibili per Parrot [4]. Molti di essi sono o abbandonati o in stato "dormiente". Ci sono vari motivi per questo, non ultimo il fatto che molti sono solamente degli esperimenti, ma molte ragioni si possono trovare nel fatto che Parrot, al momento, è ancora lontano dal realizzare ciò che ha promesso: una macchina virtuale solida, veloce, adatta a tutti i linguaggi dinamici.

Il "cliente" principale di Parrot al momento è senz'altro Rakudo [5], l'implementazione principale di Perl 6. Supportare Rakudo d'altronde è lo scopo iniziale per cui è stato fondato. Rispetto ad altri HLL Rakudo ha il vantaggio di essere seguito molto strettamente dalla comunità Parrot, tanto che vi sono sviluppatori in comune, e spesso le priorità di sviluppo vengono decise in base a ciò che serve a Rakudo.

6 Alternative

Esistono diverse alternative a Parrot. Nessuna di esse si propone direttamente come target per sviluppatori di linguaggi, ma esistono per supportare un linguaggio specifico, e non offrono, oltre alla VM, tool integrati per la costruzione di compilatori come Parrot. In generale però si tratta di piattaforme molto più mature, e questo le rende target molto attraenti.

L'alternativa a Parrot più utilizzata è probabilmente la JVM. Non offre un supporto diretto a linguaggi dinamici e non offre feature che non servano al linguaggio Java. Dispone però di un sistema ad oggetti sufficientemente flessibile, moltissime librerie ed è potenzialmente molto efficiente. L'efficienza

viene persa man mano che la semantica del linguaggio si discosta da quella di Java. I linguaggi più maturi che girano sulla JVM sono (oltre a Java):

- Scala: si tratta di un linguaggio statico, quindi non della tipologia a cui si rivolge Parrot.
- Groovy: un linguaggio invece fortemente dinamico, che dimostra come una macchina virtuale pensata per un linguaggio statico possa essere un buon ambiente anche per un linguaggio dinamico, sebbene le prestazioni in questo caso soffrano parecchio.
- Clojure: un dialetto del Lisp pensato per la concorrenza. Per attenuare il problema delle prestazioni dovute ai tipi dinamici, permette al programmatore di annotare le variabili con un tipo statico e generare quindi codice più efficiente.
- Molti linguaggi già esistenti dispongono anche di un'implementazione per la JVM: Ruby (JRuby), Python (Jython), Common Lisp (Armed Bear Common Lisp), Scheme (Kawa), PHP (Quercus) ed altri.

Visto il successo negli ultimi anni di questi linguaggi la Sun (ora Oracle) ha aggiunto un po' di supporto per i linguaggi dinamici nelle ultime versioni della JVM, come ad esempio il bytecode invokedynamic, oltre a creare un progetto sperimentale, denominato Da Vinci, che si propone di modificare la JVM per offrire un migliore supporto ai linguaggi dinamici. Inoltre la Sun organizza una conferenza incentrata sui linguaggi che hanno come target la JVM, il JVM Language Summit.

Un'altra alternativa valida è PyPy [6]. Non si tratta esattamente di una macchina virtuale, ma di un progetto di ricerca sofisticato, sponsorizzato in passato dall'unione europea, che permettere di scrivere un interprete nel linguaggio RPython, un sottoinsieme analizzabile staticamente del linguaggio Python, e di convertire questo interprete in un eseguibile che lo trasformerà in automatico in un compilatore JIT sfruttando tecniche di partial evaluation e tracing. Il linguaggio che il progetto PyPy vuole implementare con questa tecnica è Python, di cui esiste una versione completamente compatibile, fatta (parziale) eccezione per le estensioni C. Gli autori di PyPy comunque vogliono rivolgere il progetto anche ad altri linguaggi di programmazione. Esiste un'implementazione abbastanza completa del Prolog che utilizza PyPy.

7 Modello di buisness

Il progetto Parrot esiste all'interno di una fondazione non a scopo di lucro. Gli sviluppatori lavorano su base puramente volontaria, fatta eccezione per

alcuni sotto-progetti specifici in cui sono usate delle donazioni alla fondazione da parte di aziende od altre organizzazioni per permettere ad alcuni sviluppatori di lavorare a tempo pieno per un determinato periodo di tempo.

NLNet, una fondazione olandese, nel 2005 ha finanziato con 70 000 dollari lo sviluppo con l'obiettivo di raggiungere la release 1.0 di Parrot. In particolare lo sviluppo si è incentrato sui seguenti componenti:

- AST interface
- Bytecode format
- Character sets
- Compiler tools
- Concurrency model (Threads)
- Events
- Exceptions
- Garbage collection
- I/O
- Multimethod dispatch
- Object support
- PMCs (core data types)
- Security model

Una descrizione più dettagliata è disponibile nel sito della Parrot Foundation [7]. Contributi allo sviluppo sono arrivati anche dalla Google Summer Of Code. Google ogni estate finanzia degli studenti per lavorare durante l'intera stagione ad un obiettivo di interesse generale per un progetto open source. Ogni studente riceve 4500 dollari, e gli viene affiancato un mentore (che a sua volta riceve 500 dollari) che ha il compito di seguirlo e di indirizzarlo nel suo progetto. Per l'estate 2010 Google ha approvato i seguenti progetti, relativamente a Parrot:

- Parrot on RTEMS.
- Hybrid threads on Parrot.

- A PAST optimization framework for Parrot.
- NFG and single-representation strings for the Parrot Virtual Machine.
- Implementing an Instrumentation Tool for Parrot VM.

È possibile seguire lo sviluppo di questi progetti durante l'estate oltre che tramite i soliti canali del progetto, anche attraverso il blog ufficiale [8] che funge da aggregatore per i blog degli studenti.

7.1 La fondazione

La Parrot Foundation è una fondazione no-profit costituita per i seguenti scopi:

- Proteggere la proprietà intellettuale (IP) della macchina virtuale Parrot e dei tool, librerie e implementazioni di linguaggi annessi. L'IP è reso disponibile sotto licenze open source.
- Coltivare una comunità open source e incoraggiare lo sviluppo di un ecosistema di tool, librerie, estensioni, applicazioni e linguaggi costruiti intorno a Parrot.
- Supportare lo sviluppo di Parrot e le attività ad esso collegate (eventi, presentazioni, etc.) tramite delle sovvenzioni.
- Fornire al progetto un'infrastruttura tecnica, legale ed organizzativa.

La fondazione è membership-based. Per diventare membri votanti sono necessarie due raccomandazioni da parte di membri attuali della fondazione, e viene concessa soltanto ad individui che hanno dato un proprio contribuito al progetto almeno due volte tramite documentazione, codice o l'implementazione di un linguaggio, dimostrando quindi il proprio impegno e il proprio interesse al successo di Parrot. Infine è necessario inviare un contribution agreement firmato (una copia è disponibile a http://www.parrot.org/files/parrot_cla.pdf).

Gli sponsor diventano membri della fondazione tramite una donazione annuale, ed in base all'ammontare della donazione acquisiscono privilegi diversi. Gli sponsor restano comunque membri non votanti.

I principali sponsor della fondazione sono:

- *ActiveState*: ActiveState sviluppa e commercializza distribuzioni e ambienti di sviluppo per i principali linguaggi di programmazione dinamici: Perl, Python, Tcl.

- *BBC*: Internamente la BBC ha investito molto negli anni passati nel linguaggio di programmazione Perl, che utilizza per sviluppare buona parte del software di cui ha bisogno. Parrot funge da base per l'implementazione principale di Perl 6, quindi è negli interessi della BBC il successo di questo progetto.

- *NLNet Foundation*:

  "NLnet Foundation financially supports organizations and people that contribute to an open information society."

- *Mozilla Foundation*: la Mozilla Foundation ha già finanziato in passato lo sviluppo di Perl 5, e questo commitment continua anche con i progetti Parrot e Rakudo.

Il boarding viene eletto annualmente. Attualmente è composto da:

- Allison Randal
- Chairman Jerry Gay
- President Will Coleda
- Vice President Jeff Horwitz
- Treasurer Shane Warden
- Secretary Patrick Michaud

8 Gestione dell'informazione

Il progetto si avvale di diversi canali di comunicazione tra gli sviluppatori del progetto e gli utenti. I canali interni non sono strettamete divisi dai canali verso gli utenti. Infatti, essendo gli utenti del progetto sviluppatori essi stessi, e quindi con una certa conoscenza tenica, c'è una forte cross-pollination tra i progetti realizzati dagli utenti ed il progetto Parrot.

Sito Web Il canale principale verso l'esterno è il sito web: http://www.parrot.org/ dove vengono annunciate le release, sono visualizzate le news ed il calendario dei prossimi eventi relativi al progetto. Sono inoltre presenti link alla documentazione, ai repository dei sorgenti e informazioni su come accedere agli altri canali di comunicazione.

IRC Uno strumento importante, e molto usato, è IRC. Sia gli sviluppatori, sia gli utenti sono spesso collegati. Oltre alle normali discussioni non programmate, periodicamente gli sviluppatori di Parrot si incontrano in chat ad un orario prefissato per discutere di scelte architetturali, o della direzione del progetto, in particolare in vicinanza delle release. La chat è necessaria in quanto gli sviluppatori si trovano in zone geografiche molto distanti, e quindi di solito non è possibile tenere una discussione faccia a faccia. Nella home page del sito è presente un calendario con le date e gli orari degli incontri programmati in chat.

Mailing lists Esistono due mailing list: una per gli sviluppatori di Parrot ed una per gli utenti. La prima, pubblica ed aperta a tutti, è riservata a discussioni sullo sviluppo del sistema, mentre la seconda è a disposizione degli sviluppatori di implementazioni di linguaggi di programmazione basati su Parrot. Questa seconda mailing list è stata aperta recentemente (l'anno scorso) per incentivare la comunicazione tra gli utilizzatori di Parrot, così da potersi scambiare idee, problemi e soluzioni. Molti degli sviluppatori di Parrot seguono comunque anche questa mailing list per rispondere alle domande degli utenti. Viene inoltre usata per annunci pubblici come il rilascio di una nuova versione.

Wiki Il progetto dispone di un wiki [9]. Utilizza trac come sistema di wiki [10]. In esso vengono messe a disposizione degli utenti e degli sviluppatori diverse informazioni:

- Documenti per gli implementatori di linguaggi basati su Parrot.
- Le parti del progetto su cui si sta concentrando lo sviluppo durante la settimana corrente.
- Diverse istruzioni su come contribuire al progetto.
- Una lista di task e progetti da realizzare.
- Descrizioni di possibili scelte architetturali ancora in fase di studio.

Bug Tracker Parrot utilizza trac anche per gestire i ticket. Ogni ticket viene associato ad un componente del sistema, ad esempio il core, il garbage collector, i diversi compilatori, etc. I ticket sono divisi in base alla versione afflitta dal problema oppure, se il bug riguarda la versione correntemente in fase di sviluppo, al posto della versione viene indicato il trunk SVN. Alcuni

ticket fanno riferimento ad una milestone, ovvero alla versione di Parrot entro la quale è necessario che siano risolti. I ticket possono essere di diverse tipologie: cage, bug, deprecation (ovvero indica la necessità di deprecare una particolare feature oppure di rimuoverne una già deprecata in precedenza), todo, patch, feature (ovvero il ticket fa riferimento ad una nuova funzionalità da implementare). Un particolare tipo di ticket è RFC (Request For Comment). I ticket di questo tipo vengono aperti dagli sviluppatori per chiedere opinioni su problematiche relative allo sviluppo del sistema. Anche i ticket, come il resto del progetto, sono completamente pubblici. Questo offre un vantaggio agli utenti di Parrot, che possono vedere lo stato di avanzamento della risoluzione di problemi che interferiscono con l'implementazione del proprio linguaggio.

Blog Molti degli sviluppatori di Parrot hanno un blog in cui postano, in modo regolare, notizie che riguardano il progetto, informazioni sullo sviluppo del componente di cui si stanno occupando, idee per il futuro e informazioni su come contribuire e task adatti ai novizi. I blog dei vari contributors sono aggregati all'indirizzo http://planet.parrotcode.org

Twitter È possibile seguirte Parrot su twitter, in cui vengono annunciate notizie o eventi che riguardano il progetto.

9 Interazione con la comunità

Durante lo sviluppo il team di Parrot mantiene una particolare attenzione ai propri utenti, sopprattutto verso il progetto Rakudo, e le priorità e le feature da implementare o deprecare vengono decise in base ad essi, oltre che alla normale tabella di marcia del progetto.

Sia nel sito web, sia nel wiki è disponibile una lista dei progetti che utilizzano Parrot come macchina virtuale [4]. Gli sviluppatori di Parrot spesso interagiscono con gli autori di questi linguaggi di programmazione, anche tramite la sottomissione di patch, per aiutarli a seguire gli standard del progetto per quanto riguarda la codifica, i test ed il sistema di build.

Per ogni linguaggio viene tenuta traccia dello stato del progetto (attivo, inattivo, dormiente), della versione di Parrot con cui è compatibile, il linguaggio in cui è implementato. In particolare, tengono traccia del rapporto test di unità passati / test scritti. Questo rapporto viene usato come metrica sulla maturità del progetto. Infatti Parrot pone molto l'accento sul Test Driven Development, e quindi da molta importanza ai test anche per i progetti che lo utilizzano.

10 Processo di rilascio

Lo sviluppo si basa su release costanti. Ogni terzo martedì del mese viene rilasciata una nuova versione. Esistono due tipi di release: supported e developer.

Le release supported avvengono ogni 3 mesi. La prima release è a Gennaio, ed è numerata X.0.0, e le successive release supported dell'anno sono le numero X.3, X.6, e X.9. Release di questo tipo ricevono aggiornamenti per bug e fix di sicurezza critici, hanno delle deprecazioni documentate tra una release e l'altra, sono intese per il packaging nelle distribuzioni e si rivolgono agli utenti finali (agli utilizzatori dei linguaggi di programmazione basati su Parrot).

Le release developer si rivolgono invece agli sviluppatori di linguaggi. Esse seguono lo sviluppo nel trunk SVN e non sono oggetto di aggiornamenti per il fix di bug successivi al rilascio. I fix appariranno nella release del mese seguente.

Questa divisione è iniziata dopo la release 1.0 del progetto. Prima le release seguivano tutte il modello developer. Il rilascio delle versioni supported è stata pensata per facilitare l'inclusione di Parrot all'interno delle distribuzioni GNU/Linux e *BSD.

La macchina virtuale è in continuo sviluppo. Spesso le modifiche sono sostanziali e hanno un impatto significativo sugli sviluppatori HLL. Ad esempio, negli ultimi mesi stanno pensando di cambiare il linguaggio utilizzato per definire i tipi di base dei linguaggi. Al momento è un'estensione del linguaggio C, mentre in futuro sarà (probabilmente) un linguaggio creato ad-hoc (chiamato Lorito). Queste modifiche continue, sebbene essenziali per il progresso del progetto, creano molti problemi agli sviluppatori HLL che si trovano a dover continuamente modificare il proprio codice per seguire i recenti sviluppi della macchina virtuale. Per alleviare il problema, a partire dalla release 1.0 seguono un processo di deprecazione ben definito:

- Le deprecazioni vengono annunciate con il rilascio delle release supported come parte dell'annuncio della release ed all'interno del file DEPRECATED.POD contenuto nel repository principale. Ad ogni deprecazione viene associato un numero di release in cui verrà rimossa la feature deprecata.

- Una volta che una feature è deprecata, potrebbe essere rimossa in una qualsiasi release developer successiva. Gli sviluppatori di Parrot cercano comunque, ma non garantiscono, di ritardare la rimozione fino al rilascio della successiva release supported.

Quindi, se dopo una relase supported una feature non è stata deprecata, gli sviluppatori HLL possono confidare nel fatto che sarà ancora presente almeno fino alla prossima release supported.

Fino ad un paio di anni fa il progetto non aveva release costanti, ma ogni release veniva decisa caso per caso. Secondo gli sviluppatori, passare al nuovo metodo ha permesso di rivitalizzare il progetto, dando degli obiettivi ben precisi e incentivando a mantenere il codice nel repository funzionante rispetto alla compilazione e ad i test. I risultati positivi ottenuti hanno convinto anche il progetto Perl a seguire questa nuova metodologia a partire dalla release 5.12.

11 Strumenti di sviluppo

Versionamento Il sistema di versionamento utilizzato è SVN. Dato che il progetto era iniziato nei primi anni 2000 all'epoca SVN era il VCS open source standard de facto, utilizzato (insieme a CVS) da quasi tutti i progetti open source. Negli ultimi anni sono nati nuovi sistemi basati su un workflow distribuito. In particolate il sistema git [11] sta avendo un grande successo, e anche Parrot sta considerando di passare a questo sistema di versionamento. Esiste già un mirror del repository principale che utilizza git [12]. Il sito web su cui si trova questo mirror è un sito di social code hosting ovvero, oltre alle funzionalità di hosting di repository di base, permette di seguire gli aggiornamenti di un progetto tramite feed RSS, seguirne gli sviluppatori e, soprattutto, sfrutta la natura decentralizzata di git per permettere a chiunque disponga di un account di fare il fork di un qualsiasi progetto in modo estremamente semplice. Questo favorisce la sperimentazione da parte degli utenti, che possono fare un fork, apportare le proprie modifiche e, se lo ritengono opportuno, creare delle patch da sottoporre al core team di Parrot per una revisione ed un'eventuale inclusione nel repository principale sempre tramite l'interfaccia del sito. Al momento esistono 6 fork solo su github. Molti di questi sono degli studenti che partecipano alla Google Summer of Code, e che quindi portano avanti il proprio progetto in un fork.

Al momento, la discussione se passare o no a git è ancora in corso, anche se in molti sarebbero favorevoli al passaggio.

L'utilizzo di SVN si basa sul seguente workflow:

- Lo sviluppo avviene quasi completamente direttamente nel trunk, fatta eccezione per alcune feature sperimentali (come il garbage collector incrementale) che vengono mantenute in branch separati.

- Il secondo marted di ogni mese, ovvero una settimana prima della release mensile, il trunk viene copiato in un tag che diventa la release.
- Lo sviluppo riprende dal punto 1.

Il passaggio a git potrebbe consentire dei workflow diversi. Uno dei possibili workflow ipotizzati è il seguente:

- Ognuno ha un branch locale su cui lavorare. Con ognuno si intende chiunque, non solo i core developers.
- Esiste un master integration branch a cui hanno accesso solo gli sviluppatori del core team. Questo branch serve ad integrare i vari branch locali. Sul master quindi non si fanno modifiche dirette, ma solo dei pull dai branch locali (ovvero vengono scaricate e viene fatto il merge tra le modifiche locali ed il master).
- Dall'integration branch, al momento della release, vengono selezionate le feature stabili dentro al release master branch (ovvero un branch per le release). Da questo branch si ricava poi un tag che diventa la release del mese.
- La release del mese diventa il nuovo master integration branch. Le modifiche rimaste nel vecchio integration branch e non presenti nel release branch potrebbero essere aggiunte al nuovo se considerate degne o se sono maturate.
- Viene rimosso il vecchio integration branch e il ciclo riparte del punto 1.

I vantaggi di questo workflow, secondo il core developer che lo ha proposto, sono:

- Si può puntare ad una maggiore qualità delle release, in quanto le modifiche non passano tutte direttamente dall'integration al release branch, ma vengono selezionate.
- Facilita il coinvolgimento di altre persone nello sviluppo, in quanto tutti, non solo i committer designati, avrebbero un branch locale su cui lavorare liberamente.
- Il release manager, ovvero la persona in carica della release mensile, avrebbe più controllo sulla release stessa e potrebbe darle una direzione in un modo che non è possibile con il workflow attuale basato su SVN.

Integrazione continua Parrot utilizza BuildBot [13] come sistema di integrazione continua. Si basa su un server centrale, detto master, che controlla il repository SVN principale. Quando avvengono delle modifiche, avvisa delle macchine slave che eseguono il build e lanciano tutti i test.

Test Il sistema di build contiene dei target per l'esecuzione di test. Il modo più semplice è utilizzare il comando make test. I test verranno eseguiti ed al termine verrà visuallizato un report. Un esempio di tale report (con dei test falliti) è il seguente:

```
Failed Test                    Stat Wstat Total Fail  List of Failed
-------------------------------------------------------------------------------
t/pmc/packfile.t                  1   256    34   48  11-34
t/pmc/packfileannotations.t       1   256    17   32  2-17
t/pmc/packfileconstanttable.t     1   256    16   32  1-16
t/pmc/packfiledirectory.t         1   256    20   32  5-20
t/pmc/packfilefixupentry.t        1   256     3    6  1-3
t/pmc/packfilefixuptable.t        1   256     3    6  1-3
t/pmc/packfilerawsegment.t        1   256     7   14  1-7
9 tests and 654 subtests skipped.
Failed 7/347 test scripts. 85/12431 subtests failed.
Files=347, Tests=12431, 131 wallclock secs (58.80 cusr + 29.08 csys = 87.88 CPU)
Failed 7/347 test programs. 85/12431 subtests failed.
```

Un altro target del sistema di test è make smoke. Questo target si occupa di eseguire i test come make test, e poi invia i risultati ad un server smolder [14]. I risultati sono visualizzabile nella pagina di status [15]. Questa pagina contiene un elenco degli ultimi test inviati. Per ognuno sono indicate informazioni sul sistema su cui è stato eseguito (architettura del processore, sistema operativo, etc), la revisione SVN di Parrot che è stata testata, il numero di test totali eseguiti, la percentuale ed il numero di test eseguiti con successo, falliti oppure saltati. È possibile inoltre visualizzare in dettaglio, per ogni report inviato, quali test hanno avuto successo e quali invece sono falliti.

I test si trovano all'interno della directory t e sono scritti in Perl oppure in PIR e seguono il protocollo TAP (Test Anything Protocol). Per aiutare nella scrittura dei test sono disponibili due moduli: Parrot::Test per i test scritti in Perl e Test;More per quelli scritti in PIR.

Report di bug La distribuzione di Parrot contiene uno strumento apposito, chiamato parrotbug, utilizzabile da linea di comando per inviare alla

mailing list un bug report. Lo strumento è pensato per essere utilizzato interattivamente. Una volta avviato chiede delle informazioni sul bug (un riassunto, la categoria del bug, etc.) ed invia una mail contenete le informazioni rilevanti.

Build Il sistema di build è un sistema personalizzato basato su Makefile. Al momento questo sistema si basa pesantemente, sia per la configurazione, sia per l'esecuzione, sulla presenza di un interprete Perl. Questa dipendenza verrà in futuro rimossa. La release in cui questo dovrebbe avvenire, secondo i piani, è la 3.0, denominata per l'appunto"independence".

Sistema di configurazione La configurazione si divide in steps. Ogni step contiene diversi prompts, probes (per controllare cosa è presente nel sistema) e generations. La descrizione di questi step si trova sotto la directory config. Ogni step si compone di un solo file .pm (un modulo Perl) e di eventuali file .c, .in, etc. di supporto. Nella directory config/init sono presenti gli step di inizializzazione. Questi step vengono eseguiti per primi ed hanno il compito di preparare le strutture dati del sistema di configurazione. I prompts servono a chiedere input all'utente. La guida per gli sviluppatori di Parrot consiglia di utilizzarli solo quando strettamente necessario, in quanto rendono il passo di configurazione interattivo ledendo così all'automatismo del sistema. Gli step generations servono a creare gli artifatti che serviranno dopo la configurazione, e sono l'output del sistema di configurazione. Questi artifatti in genere includono Makefile, file header C, etc. I template dei file da generare terminano con l'estensione .in.

Documentazione La documentazione viene scritta utilizzando il formato POD (Plain Old Documentation), un formato standard per i progetti legati al linguaggio Perl. POD è un linguaggio di markup molto semplice, ed esistono strumenti per leggere direttamente un file da terminale (perldoc) oppure per generare file in altri formati: plain text, HTML, pagine man ed altri.

12 Divisione dei ruoli

All'interno del progetto Parrot gli sviluppatori possono assumere i seguenti ruoli:

- Architect: si occupa del design globale del sistema. Ha sempre l'ultima parola per quanto riguarda scelte che coinvolgono aspetti importanti del sistema. Ha la responsabilità di assicurarsi che i documenti che

descrivono il design siano aggiornati. Al momento, il ruolo di Architect viene svolto da Allison Randal.

- Pumpking: è il lead developer, e definisce gli standard di codifica da seguire. Aiuta gli altri sviluppatori a coordinarsi.
- Release Manager: controlla il processo di release. Controlla quando è possibile aggiungere nuove feature e quando il codice deve essere congelato per il testing ed il debugging.
- Committer: ha accesso in scrittura al repository SVN. Si diventa committer dopo aver contribuito con molte patch ed essere stati attivi durante le discussioni del progetto.
- Metacommitter: come il committer, ma ha la possibilità di far diventare committer altri sviluppatori. L'Architect ed il Pumpking sono anch'essi metacommitter.

Questi ruoli si possono suddividere ulteriormente:

- Core developer: lavora su sottosistemi interni della macchina virtuale. I core developer necessitano di una buona conoscenza del linguaggio C.
- Compiler developer: lavorano sui frontend dei diversi compilatori che fanno parte di Parrot. Si occupano di compilatori di supporto alla macchina virtuale, non dei compilatori costruiti su di essa.
- High-Level Language developer: uno sviluppatore che si occupa dell'implementazione di un linguaggio che ha come target Parrot. In genere un HLL developer non fa parte del team di Parrot, anche se in passato solitamente ne faceva parte, in quanto i primi linguaggi venivano scritti dagli sviluppatori stessi di Parrot. Fino a non molto tempo fa, il repository principale di Parrot manteneva anche questi linguaggi sotto la directory languages.
- Build manager: si occupa di manutenere i tool di build utilizzati dagli altri sviluppatori.
- Testers: si occupano di scrivere i test.
- Platform porters: uno degli obiettivi di Parrot è di funzionare su un numero elevato di piattaforme. I Platform Porters controllano che Parrot sia compilabile, eseguibile e che superi i test nelle diverse piattaforme supportate. Si occupano anche di creare e distribuitre pacchetti precompilati di Parrot per le diverse piattaforme.

Riferimenti bibliografici

[1] http://www.perl.com/pub/a/2001/04/01/parrot.htm

[2] http://gituhub.com/stefano/primitivearc/

[3] http://llvm.org/

[4] http://trac.parrot.org/parrot/wiki/Languages

[5] http://rakudo.org/

[6] http://codespeak.net/pypy/

[7] http://parrot.org/foundation/grants/nlnet-grant

[8] http://www.parrot.org/blog

[9] http://trac.parrot.org/parrot

[10] http://trac.edgewall.org/

[11] http://git-scm.com/

[12] http://github.com/leto/parrot

[13] http://buildbot.net/

[14] http://sourceforge.net/projects/smolder/

[15] http://smolder.plusthree.com/app/public_projects/details/8