Lightweight LanguageとしてのSAD

Workshop SAD2006(2006/09/05-07)

森田 昭夫
KEKB Commissioning Group

Contents

- Lightweight Languageってなに?
- SADってなに?
- SADで何が出来るの?
- SADの開発体制
- SADのこれからの課題

Lightweight Languageってなに?

- Lightweight Language = 軽量言語
  - Web関連技術として最近流行の言語
    - 代表例は Perl, Ruby, Python, PHP...
    - もちろん、/bin/shなんかもその仲間
  - 軽量とは言語としての軽さ(処理の軽さでは無い)
  - 特徴
    - インタプリタ型で手軽に動かせる
    - 型を持たない or 動的な型システム
    - 強力なライブラリ郡
  - 要するに...
    - 手軽かつ簡潔に書けて、手軽に試せる

KEKBでいつも使ってるSADも LLの仲間では?

SADってなに？

歴史を紐解けば...

- Strategic Accelerator Design
  - 1986年ごろから KEKで開発されてる加速器設計用コード
    - 名前は、Methodical Accelerator Designへの対抗w
  - Project Driven Development
    - 必要とする人が必要とする機能を実装する
  - Kitchen Sink Approach
    - 必要な道具は全て取り込む
  - 1994年ごろに Mathematicaに似た Script言語インタプリタが実装された
    - 加速器モデルを内蔵した Lightweight Languageの出現

SADで何が出来るの?

- LL的側面
  - SADScriptインタプリタ
    - リスト操作&Mathematicaもどきの文法
    - 遅延評価とパターンマッチングベースの関数評価
      - 関数渡し、多相性を持った関数定義
    - Object Oriented Programmingのサポート
- 加速器設計コード的側面
  - ビームライン定義(MAINレベル)
    - DRIFT, BEND, QUAD, SEXT, OCT, DECA, DODECA, MULT, CAVI, TCAVI, MAP, MARK...
  - ユーザー定義のエレメント
    - SADScriptによる写像や転送行列定義による
  - InteractiveなGeometry/Optics Matching(FFSレベル)
  - Tracking/Synchrotron Radiation/Emittance計算
  - Dynamic Aperture Survey

SADScript入門(1/*)

- Mathematica likeな関数型言語
  - 文法は、ほぼMathematica互換
    - 参考文献は、SAD/Tkinter ManualやMathematicaの入門書
  - Symbolic Manipulationはサポートしない
  - 組み込み型
    - String　長さを持ったUnibyte String(null文字表現できる)
    - Real　倍精度浮動小数点
    - Symbol　シンボルそのもの
    - リスト　型を持ったリスト
      - この型をリストの第0要素、又はHeadと呼ぶ(Head[]で参照できる)
      - 型自身は、シンボルとして表現される(普通のリストの型はList)
      - リストの中身には、SADScriptで表現可能な型が全て入る
      - 他の組み込み型も、長さ0のリストと見なせる
    - Integerや Boolean型を持たない
      - これらは、Real型で代用する
    - 配列やハッシュは、リストや関数で表現する

SADScript入門(2/*)

- 式
  - 評価の対象物、実行の単位
  - プログラムは式もしくは連結された式である
  - 式の評価は、遅延評価をされた対象を除き、簡約可能なものは簡約し、簡約化出来ない部分はそれ自身を値とする
    - ようするに、適用可能な簡約規則が無くなるまで式を簡約する
- 変数
  - なんらかの値に束縛されたシンボル
- 演算子
  - 関数型の構文を簡略化する構文糖
  - 無くても、問題ないがいろいろ便利です
    - 糖分の使い過ぎると自分でも読めなくなりますw
  - 例えば...
    - Times[a, Plus[b, c]] → a * (b * c)
    - Map[func, {a, b, c}] → func/@{a, b, c}

SADScript入門(3/*)

- 関数
  - 式に遅延評価で束縛されたシンボル
    - 束縛する際にパラメータ(引数)を含ませることが可能
    - 評価時にパラメータのパターンマッチングが行われる
      - 多相型の関数定義が可能
      - パラメータを含んだハッシュと解釈することも出来る
  - 関数という概念自体(純粋関数)もシンボルに束縛できる
  - 例えば...
    - f[x_Real] := x^2 + 2 * x + 1　(f[x]を x^2+2x+1へ束縛)
      - f[1] → 1^2 + 2 * 1 + 1 → 4
    - f = (#^2 + 2 * # + 1)&　　(fに 純粋関数を束縛)
      - f[1] → (#^2 + 2 * # + 1)&[1] → 1^2 + 2 * 1 + 1 → 4
    - f[x_Real] := x * f[x - 1]; f[0] := 1; f[x_] := x
      - f[3] → 3 * f[2] → 3 * (2 * f[1]) → 3 * (2 * (1 * f[0])) → 3 * (2 * (1 * 1)) → 6
      - f["test"] → "test"

SADScript入門(4/*)

- シンボルのスコープ
  - グローバルスコープ
    - 大域的な名前空間
    - Block[]構文で、一次的に別の値を割り当てることが可能
    - Block[]構文が作用しないグローバルシンボルも有るので注意($FORMなどのインタプリタ提供の特殊なシンボル)
  - モジュールスコープ
    - Module[]構文で導入される、ローカルな名前空間
    - スコープの終わりで、シンボルが破壊される(Destructorが呼ばれる)
  - クラススコープ
    - Class[]で定義されるクラスローカルな名前空間
  - インスタンススコープ
    - Class[]で定義されるインスタンスローカルな名前空間
    - クラスのインスタンスごとに独立した名前空間
  - クラス、インスタンススコープからの重複するグローバルシンボルへの参照は、Literal[]による修飾で解決可能

SADScript入門(5/*)

- 代表的なString操作関数
  - 長さを測る　StringLength[]
  - 文字列の結合/切り出し　StringJoin[],str[begin,end]
  - 文字の挿入/削除　StringInsert[],StringDrop[]
  - 部分文字列の位置検索　StringPosition[]
  - 文字列パターンの照合　StringMatchQ[]
  - コードへの変換　ToCharacterCode[]
  - コードからの変換　FromCaracterCode[]
  - 式からの変換　ToString[]
  - 式への変換　ToExpression[]

パターンマッチした部分を抽出/置換する関数は
提供されていない

SADScript入門(6/*)

- 代表的な組み込みリスト操作関数
  - 長さを測る　Length[]
  - リストの結合/切り出し　Join[],Take[],Drop[]
  - リストの積/和/排他積　Intersection[], Union[], Complement[]
  - 要素へのアクセス　Part[](演算子版 foo[[index]])
  - 要素の取り出し　Extract[]
  - 要素の追加　Append[], Prepend[]
  - 要素の挿入/削除/入れ換え　Insert[],Delete[],ReplacePart[]
  - 要素の整列　Sort[]
  - 多重リストの平坦化　Flatten[]
    - Flatten[{a,b,c},{1,2,3}] → {a,b,c,1,2,3}
  - 多重リストの並べ替え　Thread[]
    - Thread[{a,b,c},{1,2,3}] → {{a,1},{b,2},{c,3}}

SADScript入門(7/*)

- 代表的な構造的演算子
  - リスト全体に作用する演算子
  - 関数を作用させるもの(高階関数)
    - Apply[f, list]　　listの全要素を引数列としたfの評価
    - Map[f, list]　　listの各要素にfを作用させた結果のリスト
    - MapThread[f, list]　≡ Map[Apply[f, #]&, Thread[list]]
    - Select[list, f]　　fを作用させるとTrueを返す要素のリスト
    - SelectCases[list, {f1, f2...}]　≡ Map[Select[list, #]&, {f1, f2...}]
  - パターンを作用させるもの
    - Count[list, pattern]　　patternに照合する要素数
    - Position[list, pattern]　　patternに照合する要素の位置リスト
    - Cases[list, pattern]　　patternに照合する要素リスト
    - DeleteCases[list, pattern]　patternに照合しない要素リスト
    - SwitchCases[list, {pattern1, pattern2...}]
      ≡ Map[Cases[list, #]&, {pattern1, pattern2...}]

SADScript入門(8/*)

- 制御構造
  - If[cond, A, B, C]
    - condがTrue(0以外のReal)、False(Real[0])、それ以外に応じてA,B,Cのどれかを評価した値を返す
      - If[False, A, B, C] := B
      - If[_Real, A, B, C] := A
      - If[_, A, B, C] := C
    - If-Then-Elseのおばけを作りたくない場合の構文
      - Which[cond1, body1, cond2, body2...]
      - Switch[cond, case1, body1, case2, body2...]
  - ループ構文や脱出構文
    - Do[body, {loop symbol, begin, end, step}]
    - Whilte[cond, body]
    - For[initialize, cond, update, body]
    - Scan[func, list]　　listの各要素にfuncを適用(Doループより速い)
    - Break[]　　ループを抜ける
    - Return[expr]　　関数評価を中断し、exprの評価値を返す

SADScript入門(9/*)

- 数学関数
  - Min[], Max[]
  - Conjugate[], Re[], Im[], Abs[], Sign[], Round[], Floor[], Ceiling[]
- 初等関数
  - 平方根/三角関数/指数関数/対数関数
- 特殊関数
  - ベッセル関数/ガンマ関数
- Fourier変換
  - Fourier[], InverseFourier[]
- ベクトル/行列演算
  - Dot[], Transpose[], Det[]
  - LinearSolve[], SingularValues[], Eigensystem[]
- 求根/最小化
  - FindRoot[], DownhillSimplex[]
- 疑似乱数
  - Random[], GaussRandom[]

SADScript入門(10/*)

- グラフィック
  - Graphics[], Show[], Plot[], FitPlot[], ListPlot[], HistoPlot[], ColumnPlot[], ListContourPlot[], ListDensityPlot[]
- 入出力
  - OpenRead[], OpenWrite[], OpenAppend[]
  - TCPOpen[], TCPAccept, SelectUnit[]
  - Close[]
  - Read[], Skip[], Write[], WriteString[]
- 時刻取得/操作
  - Date[], FromDate[], ToDate[], DateString[]
- プロセス制御
  - GetPID[], GetUID[], GetGID[], TimeUsed[]
  - SetDirectory[], Directory[], HomeDirectory[]
  - Fork[], Wait[], Pause[]
  - System[], Environment[]
  - OpenShared[], Shared[]

SADScript入門(11/*)

- Class[]入門(#/*)
  - Object Oriented Programmingに必要なクラスを定義する
    - class = Class[{super classes}, {class scope symbols}, {instance scope symbols}, class/instance function definitions]
    - instance = class[constructor arguments] (construction)
    - instance =. (destruction)
    - クラスの内部実装を隠蔽する機能は備わっていない
  - クラス/インスタンスシンボルは@で参照
    - instance@variable, instance@method[arguments]
  - Constructor[],Destructor[],クラス/インスタンス関数の定義はClass[]の第4引数に書く
    - Constructor[]は多相的に定義可能
    - 導出クラスにConstructor/Destructorが存在する場合、基底クラスのConstructor/Destructorは呼び出されない(特別扱いしない)
      - 明示的に`スコープ演算子を用いて呼び出さなければならない
      - SuperClass`Constructor[arguments]

SADScript入門(12/*)

- GUI Toolkit(Tkinter)
  - Tk ToolkitをSADScript上へマップしたもの
  - 基本的には、Tcl/Tkと1:1
    - がんばれば、Tkで実現できるどんなGUIでもデザイン出来る
  - 参考文献: SAD/Tkinter Manual

Example: Hello World

```
FFS;
w = Window[];
t = TextLabel[w, Text->"Hello World"];
TkWait[];
```

SADScript入門(13/*)

- GUI Toolkit(KBFrame)
  - KEKB用に開発されたTkinterラッパー(KEKB Frame)
  - KBFで始まるシンボルのリストでGUIをデザインを表現する
    - 段組みやフレームの入れ子など
  - 標準化されたLook&FeelとWindow間のインターフェイスを提供する
    - 便利なGUI部品: Number/String EntryやCursor付き Entryなど
  - 参考文献: http://www-kekb.kek.jp/Documentation/KBFrame/

Example: Hello World

```
FFS;
w = KBMainFrame["HELLOWORLD", f, Title->"Hello World"];
w[AboutMessage] = "Sample Script \"Hello World\"";
cf = KBFComponentFrame[f, Add->{
  KBFText[Text->"Hello World!"],
  Null[]}];
TkWait[];
```

SADScript入門(14/*)

- EPICS Channel Access
  - 制御に必要なEPICSレコードへのアクセス手段の提供
    - EPICS CA Client Libraryが必要
  - 関数ベースのインターフェース
    - Ca(Open|Read|Write|Close)[]関数
    - 同期型のインターフェイス
  - クラスベースのインターフェース
    - CaMonitorクラス
    - 非同期型のインターフェイス
    - ユーザー定義のコールバック関数に状態変化を通知

SADScript入門(15/*)

Ca(Open|Read|Write|Close)[]関数

- レコードを読み出す
  - {val, status, severity, timestamp} = CaRead[record_String]
- レコードに書き込み、CaRead[]で読み返した値を返す
  - {val, status, severity, timestamp} = CaWrite[record_String, newval_]
- unix likeなOpen/Closeスタイルもサポートする
  - Ca(Read|Write)[]は、record_Stringがid_Realに置き換わる
  - レコードを開き、記述子(chidオブジェクト)を得る
    - id = CaOpen[record_String]
  - レコード記述子を閉じる
    - CaClose[id_Real](実際には何もしない)
- リストを使いまとめてアクセスすることも出来る
  - {val1, val2,...} = First/@CaRead[{record1, record2,...}]
  - CaWrite[{record1, record2,...}, {newval1, newval2,...}]

SADScript入門(16/*)

CaMonitorクラス

- インスタンス生成
  - instance = CaMonitor[record_String, ValueCommand:>callback1, ConStatCommand:>callback2]
- レコード読みだし
  - val = instance@Value[]
    - timestamp = instance@TimeStamp[], severity = instance@Severity[], constat = instance@ConStat[]
- レコードに書き込み(非同期)
  - instance@Put[newval_]
- コールバックの呼び出し
  - Value/Severityの変化　　callback1が評価される
  - ConStatの変化　　callback2が評価される
- インスタンスの破壊
  - instance =.

SADの実装

- 実装は Fortran + C
  - 基幹機能は Fortranで実装(インタプリタも!)
  - 外部ライブラリとの糊はC
  - Fortranの標準化されていない拡張を使ってる
    - Fortran77/90/95等の言語仕様に抵触するコードも...
- 環境依存な設計
  - 構造体など無く、データの長さはハードコード
    - sizeof(real) == 2 * sizeof(integer) == 8 * sizeof(character)
    - pointerは integerへ収納できる
    - って、今どきのマシン[LP64]じゃ、破れてるじゃん orz
  - Time Zoneは JST決め打ち
    - 中国のユーザーから、1時間時計がずれてると文句が...
- 築20年のコード
  - 最初の実行環境は、HITAC
  - HP735/755、DEC8000を経て Tru64/Linux/MacOS Xへ...

SADの開発体制(1/3)

- Project Driven Developmentな開発体制
  - 欲しいものは、欲しい人が実装する
  - ユーザー＝開発者という環境で開発が進んで来た
  - コードの開発・保守を専門に行う開発部隊は存在しない
    - 純粋な外部ユーザーのためのサポートを行う仕組みは未実装w
- Ad HocでKitchen Sink Approachな開発方針
  - 必要になった機能は、取り合えず組み込む
    - 追加機能の汎用性や他の機能との直交性は犠牲になっている
    - 予約語が何時の間にか増える(ユーザーコードの互換性を破壊
  - 必要でなくなった機能は、保守しないけど消さない
    - ユーザーが使おうとしたときに動くかどうか怪しい
  - ようするに...
    - その場で必要な機能を実装、後は野となれ山となれw

SADの開発体制(2/3)

- 系統だったテストを行わない文化
  - 単体試験、退行試験ってなに?な文化
    - 大きな機能拡張時には、致命的なバグが混入すること多数
- Documentが残らない文化
  - その場限りの実装をして、文書化や保守が行われない
    - commitlogは version番号だったりする
    - モジュールAPIやデータ構造の解説は何処にもない
      - 保守されてないコードも有るので、データ構造への異なる参照を行うコードがありどれが正しいか分からないことも orz
- 開発への新規参入時の敷居が高い
  - Programmer's Guide的な文書は存在しません
  - 正面から解析するには巨大に成り過ぎた
    - しかも、間違ったコードやMagic Numberだらけのコードから実装者の意図を見抜く洞察力が必要
  - 現実的には、弟子入りして匠の技を盗むしか...

SADの開発体制(3/3)

- 開発環境
  - 主に MacOS X 10.4/PowerPC G5上
    - KEKBの運転環境をターゲットに開発
    - Redhat Linux 7.3と Tru64 UNIX V5.1Bもサポート
    - 標準コンパイラは、GCC 3.4.5(gcc/g77)
  - 履歴管理は CVSを使用
    - 公式Repositoryは、acsad[34]上に存在
    - アクセス権は、sadistグループ所属のSAD Clusterユーザー
      - 登録ベースで、17人
      - 活発にcommitしてるのは数人
    - MAIN trunkの他に幾つかのpersonal branchやtest branchがある
      - ckallen2　　RMS Envelope Simulation by C.K.Allen
      - Sun-Solaris　　Solaris移植 by N.Yamamoto
      - noboru-32bits　　2GB境界問題の修正 by N.Yamamoto
      - amorita　　移植用 by A.Morita(現在はSubversionへ移行)

SADのこれからの課題(1/3)

- 世の中の流れへ対する対応
  - ILP32 → LP64
    - データ構造に関する仮定の破れ
  - GCC 3.x → GCC 4.x
    - GCC4への移行にともなって Fortranコンパイラを標準サポートしない環境が増えている(インストールが面倒になる
  - 取り残されると、動かせる環境が無くなる
    - 最後の手段は、仮想マシン上で...
- 今後の機能拡張・保守に関する課題
  - 内部モジュールのデータ構造・APIの文書化
  - 特定条件用のコードの削減・共通化
    - コンパイラの最適化を期待し、保守性を優先すべきでは?

SADのこれからの課題(2/3)

- これからのSADに欲しいLL的機能(結構個人的理由)
  - 関数やビームラインエレメントの動的な定義
    - 標準の名前空間を汚染せずに特定用途の機能を追加できる
    - ユーザー定義の機能拡張や新機能の実装テストに有用
    - 関数定義は試験的実装有り(DynamicLink/Call/Unlink)
  - POSIX ThreadとかMPIとかの並列化サポート
    - fork(2),wait(2)は有るけど、wait4(2)が無いので不便
    - Shared[]関数での固定長なIPCは不便
    - POSIX Threadは、インタプリタの内部状態を COMMONから追い出さないと難しそう
  - Just In Timeコンパイラ
    - 関数定義から C/Fortranコードを生成できるならDynamic Loaderと組合せでJITもどきが作れる
  - GPL or OSS互換なライセンス
    - 現状のライセンスではGPLなものをリンク出来ません
      - 最近流行のGUI Toolkit(Qt/GTK+)はGPL

SADのこれからの課題(3/3)

- ユーザーコミュニティの活性化
  - 掲示板とかWikiが有りますけどみんな使ってます?
    - 直接会うと不満やバグの指摘をするけど、e-mailや掲示板に報告してくれない人々が多い
  - ユーザーからのバグ報告やパッチ寄贈とか見掛けない...
    - 全員βテスターなはずなんですがw
    - 機能追加の(具体的な)提案も見掛けない気が
  - ユーザーからの要望が上がってこない・反応がない
    - 講習会をやって欲しいとか言う話も聞かない
    - 掲示板の質問に答えてもその後の反応がない
      - 事例集の蓄積にならない

現状のまとめ

- SADは築20年の古いコード
  - あちこち保守が必要な部分が有るが、手が回っていない
  - 代替品となるコードは無いので、これからも使われる?
  - 文書化が行き届いてないので、ユーザーによる新機能の実装には大きな壁がある
- ユーザーコミュニティの活性が低い
  - 掲示板の活性は昔より低くなってる
    - Release Announceと一部の開発者間の経過報告しか orz
  - Wikiは出来たばかりで、まだ何とも言えない
  - 手元で見付かるバグの数からすると、不具合報告が上がってこないのは、使っているユーザーが少ないからでは...