Functional Scripting

汎用的関数型言語における系統的な外部リソース操作の原理と実装

大和谷 潔:学生番号 010119

2002 年 2 月 18 日

目的

- 関数型言語の信頼性
- スクリプティング言語の開放性

} を兼ね備えたプログラミング言語の開発

背景

- コンポーネント指向
- 現状のスクリプティング言語に欠ける信頼性
- 現状の関数型言語に欠ける開放性

コンポーネント指向

$\Rightarrow$ コンポーネントを組み合わせるアプリケーション構成手法が普及。

$\Rightarrow$ コンポーネントを結び付けるスクリプティング言語が登場。

$\Rightarrow$ 現状では、ad hocなスクリプティング言語が普及している。

$\Downarrow$

コンポーネントと言語の関連を科学的に検討する必要がある。

現状のスクリプティング言語

記述力 Syntax sugar、変数宣言不要

開放性 コンポーネントを取り込む機構

しかし、

$\Longrightarrow$ 安全性に欠ける

現状の関数型言語

記述力 型宣言が不要、高階関数、多相型関数

安全性 コンパイル時の型チェック

しかし、

$\Longrightarrow$ 開放性に欠ける

アプローチ

関数型言語と外部ライブラリとの連携を強化

1, 型システム
　オブジェクト指向など他の型システムの表現

2, 言語設計
　関数型スクリプティング言語 Amethyst

3, 処理系の実装
　外部ライブラリに対応して動的に調整可能な機構

1, 型システム

要求：オブジェクト指向クラスライブラリとの連携が重要。

- オブジェクト指向はコンポーネントの基礎
- 豊富なクラスライブラリ

問題:

1. オブジェクトの性質 (状態、実装の隠蔽) を表す型がない
2. オブジェクト指向の polymorphic 型システム

解決:

1. 外部オブジェクトを表す「オブジェクト型」を導入
2. オブジェクト指向でのサブタイプ関係を多相型レコード計算とオブジェクト型で表現

オブジェクト型

外部ライブラリが公開するオブジェクトを表現する型

$$(\alpha_1, \cdots, \alpha_k)\ \ T[\{lbl_1 : \tau_1, \cdots, lbl_n : \tau_n\}]$$

オブジェクト型の性質：

- 直接生成できない。

$$\times\ Person[\{Name = "YAMATO"\}]$$

- プリミティブ(外部関数)により生成

$$genPerson : \texttt{string} \rightarrow Person[\{Name : \texttt{string}\}]$$

- レコード型と同様にフィールド$lbl_1, \cdots, lbl_n$をもつ。

型システムへのオブジェクト型の組み込み

問題：レコード型と同様にオブジェクト型をパターンマッチしたい。

解法：フィールドの有無によって型を分類する。

多相型レコード計算 [Ohori]：
共通するフィールドをもつ型の集合 (**カインド**) を導入

$$kind \ ::= \ \{\!\{lbl_1 : \tau_1, \cdots, lbl_n : \tau_n\}\!\}$$

例：

$$\texttt{fun}\ getName\ x\ =\ \texttt{case}\ x\ \texttt{of}\ \{Name = n, \ldots\} \Rightarrow n$$

$$getName\ :\ \forall(\alpha :: \{\!\{Name : \beta\}\!\}, \beta).\alpha \to \beta$$

$Name$フィールドを持つすべての型を引数にとることができる。

同じフィールドを持つオブジェクト型とレコード型を同じカインドに。

$$\begin{matrix}\{Name : string\} \\ Person[\{Name : string\}]\end{matrix} \in \{\!\{Name : string\}\!\}$$

例：$getName$がレコードと$Person$の両方に適用できる。

$$getName \ : \ \forall(\alpha :: \{\!\{Name : \beta\}\!\}, \beta).\alpha \to \beta$$

$$getName\ \{Age = 30, Name = "YAMADA"\}$$
$$getName\ (genPerson("TANAKA")))$$

オブジェクト指向的polymorphism

オブジェクト指向型システムの特徴：

- クラスの継承関係から導出される型のサブタイプ関係
- サブタイプ関係をもとにした、制約されたpolymorphism

問題：オブジェクト指向的polymorphismを、関数型言語的型システムで表現

解法：クラスとカインドの類似に着目

型とカインドの関係 $\Longleftrightarrow$ 型とクラスの関係

カインド間の包含関係 $\Longleftrightarrow$ クラス間の継承関係

$\Downarrow$

レコードとオブジェクト型によってサブタイプ関係を表現

オブジェクト型

$$\alpha\ Obj[]$$

を用いる。
(値表現と無関係なαで、他の型システムの制約を表現。)

型が所属するすべてのクラスをレコードのフィールドで表現。

$$\begin{array}{rcl} & (\mathrm{OO}) & (\mathrm{ML}) \\ \texttt{class}\ A & \Longleftrightarrow & \{A : \texttt{unit}\}\ Obj \\ \texttt{class}\ B\ \texttt{inherits}\ A & \Longleftrightarrow & \{A : \texttt{unit}, B : \texttt{unit}\}\ Obj \end{array}$$

サブタイプ関係で制約されたOO-polymorphismを、カインドで制約されたML-polymorphismで表現。

$$(\mathrm{OO}) \quad (\texttt{class}\ A) \to \texttt{int}$$

$$(\mathrm{ML}) \quad \forall \alpha :: \{\!\{A : \texttt{unit}, \ldots\}\!\}.\alpha\ Obj \to \texttt{int}$$

2,Amethyst 言語

Standard ML + external 宣言 + カインド付多相型

ドメイン： ライブラリのカテゴリー

```
domain Java = imports "init" of "jni.so"
```

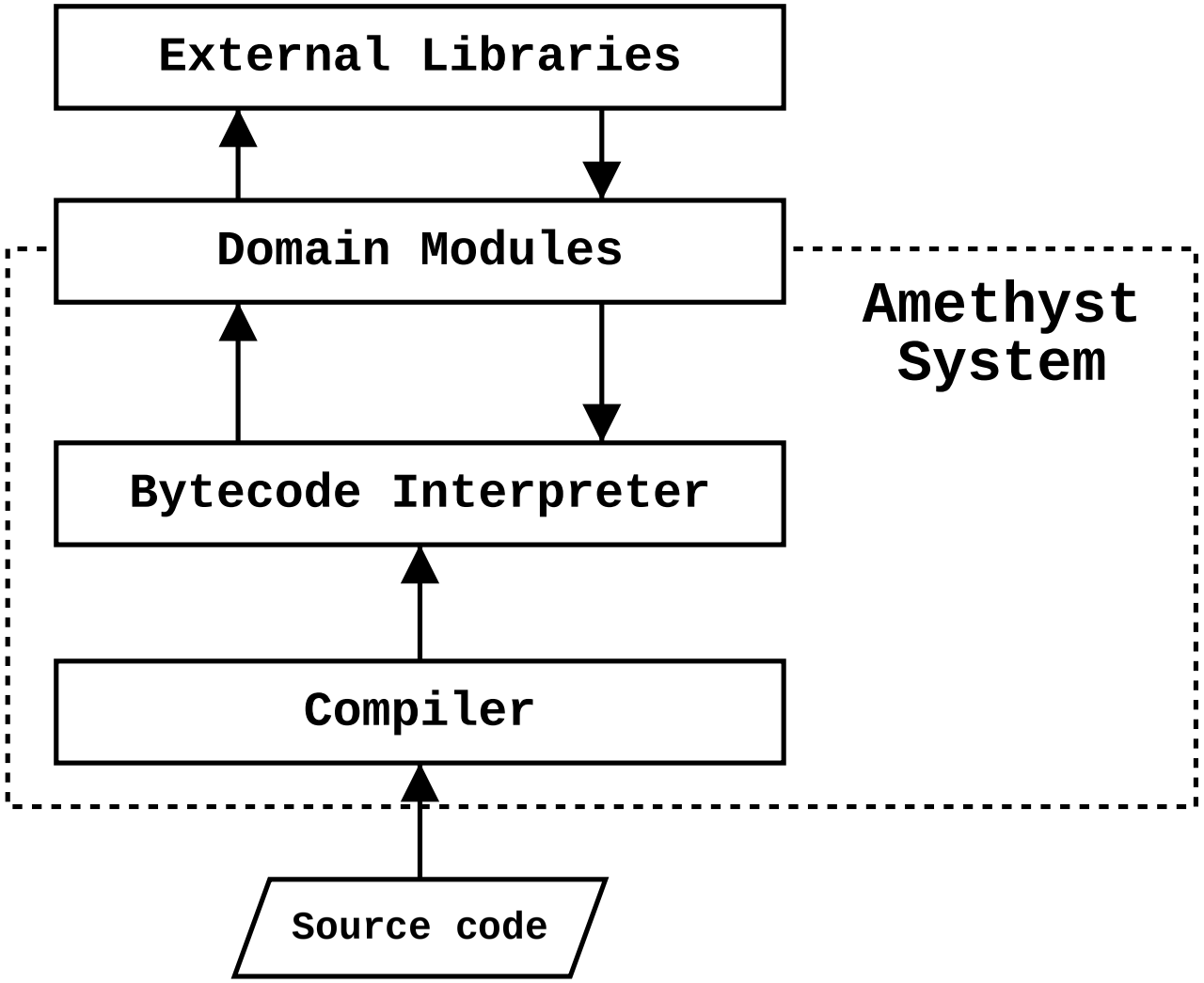
外部型：

```
external type Person = {Name:string,Age:int} imports "Person" of Java
```

外部変数：

```
external val genPerson : string -> Person imports "newPerson" of Java
```

3,実装


External Libraries
Domain Modules
Amethyst System
Bytecode Interpreter
Compiler
Source code

コンパイル

問題：

$$\texttt{fun}\ \ getName\ x = \#Name\ x$$

- xの型が呼び出しごとに異なる。
- xからフィールドを取出す方法がxの型に依存。
  (レコード)

$$getName \quad \{Name = "TANAKA", Salary = 100\}$$

  (Java オブジェクト)

$$getName \quad genPerson("YAMATO")$$

- フィールドを取出す個所で、フィールドの取出し方法を知りたい。
- レコード/オブジェクトを生成する個所では、取出し方法が分かる。

解決：(多相型レコード計算を応用)
レコード/オブジェクトを生成する個所からフィールドを取出す個所へ
フィールド取出しを実行する関数(セレクタ関数)を受け渡す。

例：

フィールドを取出す側：

$$\texttt{fun}\ getName\ x\ =\ \#Name\ x$$

$$\Rightarrow\ \texttt{fun}\ getName\ \ S\ \ x\ =\ S(x)$$

レコード/オブジェクトを生成する側：

$$getName\ (genPerson(\text{"}YAMATO\text{"}))$$

$$\Rightarrow\ getName\ \ Sel_{(Person,Name)}\ \ (genPerson(\text{"}YAMATO\text{"}))$$

ドメインモジュール

ドメインモジュールの役割： 外部ライブラリの物理的詳細を隠蔽

例：

- Java はオブジェクト指向。
- その外部インターフェイスである JNI は C 関数で構成。

⇓

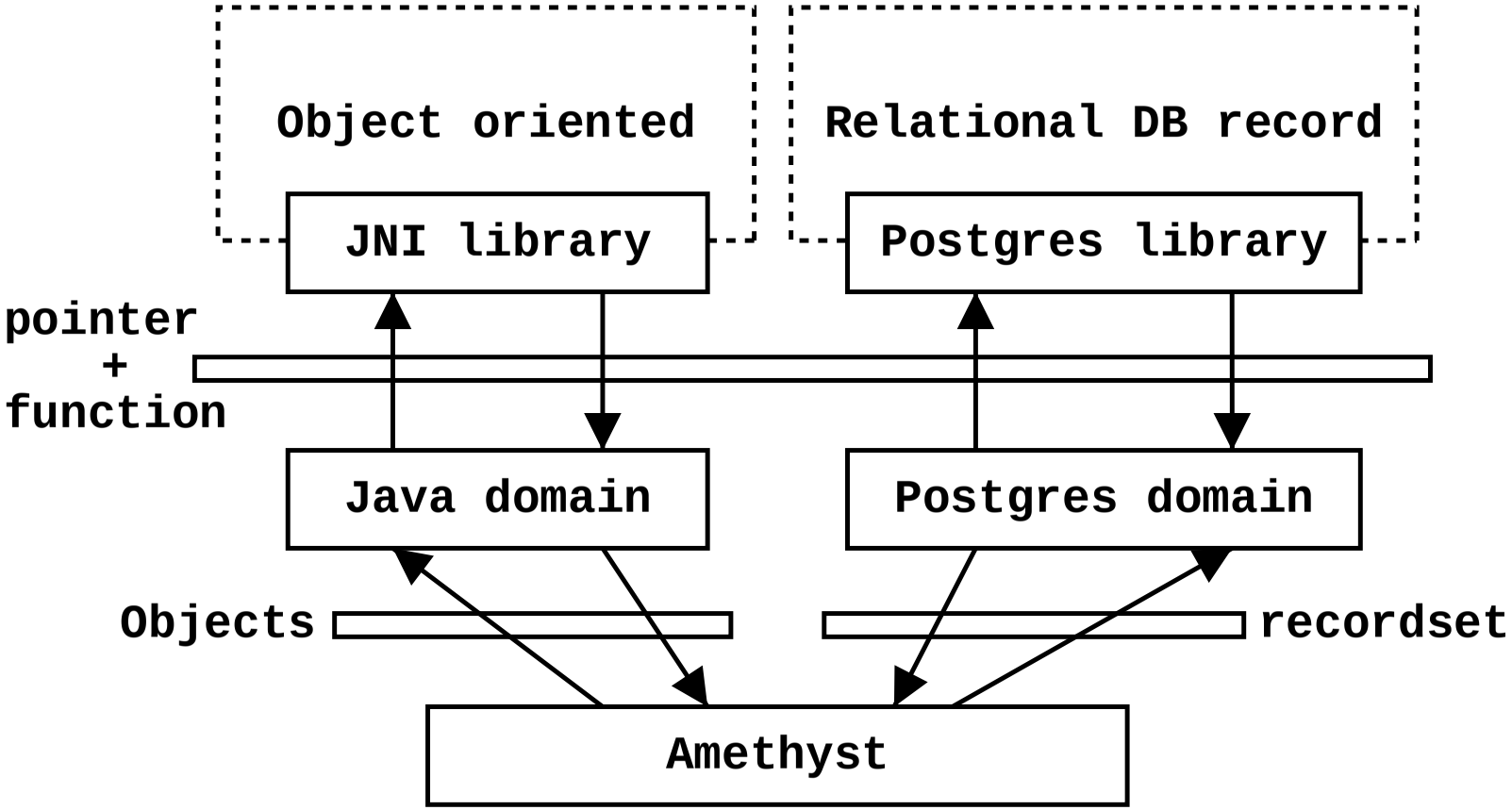
Java ドメインモジュールが

- 物理インターフェイス (C 関数) を隠蔽
- 本来の抽象モデル (オブジェクト指向) を再現

⇓

Amethyst 上で

「オブジェクト指向」的に Java オブジェクトを操作


Object oriented
Relational DB record
JNI library
Postgres library
pointer
+
function
Java domain
Postgres domain
Objects
recordset
Amethyst

複数のドメインモジュールが共存可能

```
domain Java = imports "init" of "jni.so"
external Person = {Name:string,Age:int} imports "Person" of Java
external val genPerson : string -> Person imports "newPerson" of Java

domain postgres = imports "init" of "postgres.so"
external emprec = {Name:string "NAME",Age:int "AGE" } imports "" of postgres
external val getEmp : string -> emprec =
                 imports "select NAME,AGE from EMP where @1" of postgres

fun getName r = #Name r

(getName {Name="YAMATO"},
 getName (genPerson("TANAKA")),
 getName (getEMP("SALARY > 1000")))
```

Amethyst システム

```
bash-2.04$ amethysti
>:load "pglib.ams"
domain postgres = imports "init" of pglib
       :
>:load "pgtest.ams"
external type  emprec = {Name:string "S:NAME",Rank:int "I:RANK"}
                                    imports "RECORD" of postgres
external fun  queryEmployee:connection -> string -> emprec =
         imports "query:select NAME, RANK from EMPLOYEE where @1" of postgres
       :
val getNames = fn : forall ('a,'b:{Name:'a,...}) => 'b dbrec -> 'a list
>val c = open "localhost" "testdb" "kiyoshiy" "kiyoshiy"
        handle e as (PGerror msg) => (print msg;raise e);
val c = ??? : connection
>val emp = queryEmployee c "RANK = 1";
val emp = ??? : emprec
>empnames = getNames emp;
val empnames = ["Carlos Ghosn"] : string list
```

今後の課題

型システム

XML など非レコード的なデータモデルへの拡張

コンポーネント記述

コールバックのために必要 (GUI のイベントハンドリングなど)