

Lisp 応用事例

- 自動車衝突実験データベースシステムの開発 -

東京都新宿区新宿 2-4-3 フォーシーズンビル
(株) 数理システム システム技術部
黒田寿男

概要

Common Lisp を軸に、自動車衝突実験データベースシステムを開発した。

本システムは、自動車の衝突実験データを採取する際の実験担当者の利便を図る一方、自動車設計者にはそれを自身の端末から自由に検索・参照・解析できるようにしたものである。

開発に先立ち利用者要求をヒアリングし分析した結果、本システムをクライアント・サーバー型のアーキテクチャーとしてとらえ、以下の要素技術の組み合わせで実現した。

1. サーバーの主開発言語を Common Lisp とし、DBMS に Common Lisp 上の OODBMS を採用した。
2. クライアント・サーバー間の通信に CORBA を適用した。
3. クライアントプログラムを Java Applet として開発した。

本稿ではシステム構成において上記要素技術を選択した経緯を踏まえ、うち、サーバー開発の中心的役割をはたした 1. と 2. について、実用的システム開発者の立場から評価する。

1 システム概要

1.1 衝突データ処理

衝突データ解析処理に用いられる主な装置は、ダミーに内蔵されたトランスデューサー、車載計測装置、リモートコントローラー (パーソナルコンピューター)、データ処理用ホストコンピューターおよびデータ参照用の端末 (パーソナルコンピューター) から成る。車載計測装置はトランスデューサーからの電気信号を増幅するシグナルコンディショナー、プレサンプリングフィルター、AD 変換器およびデータメモリーが一体となって構成されている [1]。

本システムは、上記リモートコントローラーからホストコンピューターへのデータ登録の簡便を図り、またそれらをネットワークを介した端末から検索・参照・解析が行なえるようにしたものである。

1.2 システム要求概要

本システム開発にあたって強く要求されたものに次がある。

- データベースが頑強であること。衝突データはハンドメイドで作った試作自動車を衝突させ破壊することで得られる。これには数億円の金がかかれており、一度採取したデータを紛失することは許されない。
- 計測器データ以外に、衝突の様子を収めた静止画および動画をデータベースにとりこみ参照したい。設計初期段階では技術者による頻繁な参照があり、後にはレポート作成等に必要となる。

- 衝突安全性について法規で定められた基準が数百あり、それぞれに対し傷害値計算やグラフ描画の方法が決められている。これらを定義記述するための言語枠組みを作り、利用者に解放する。
- データの登録・参照には、ネットワーク (LAN) を利用したインターフェースを与える。新規にインストールの不要な Web ブラウザーなどを利用できるとなるとよい。

与えられた開発期間は要求分析の期間を含めて約7ヶ月であった。

2 開発方針

システム開発にあたってまず次の三点を大枠として決めた。

- 本システムをクライアント・サーバー型のものとして実現すること。
- 開発と運用の負荷軽減のため、サーバーには製品化された DBMS を利用すること。
- クライアントプログラムは、インストールの手間とプラットフォーム依存度合を減らすため、Java Applet として実現する。

2.1 開発言語の撰択

サーバー開発言語には当初 Java を考えた。これは製品版 RDBMS が持っている開発者用言語インターフェースの一つに Java があったためである¹。

一方、法規基準記述の枠組みを作るという要求のあったことから、シンタクス設計とパーサー開発の軽減のために Lisp の利用を考えた。法規基準記述のシンタクスを S 式にすることで、S 式の意味付けに専念できると判断したためである。また、サーバーにはデータベース機能に加えて、デジタルフィルターや傷害値計算などの数値演算機能が要求されており、実行効率に関してネイティブコンパイラーを持った処理系のほうが有利である、という見通しがあった。

以上のことから製品版の Common Lisp を中心に Java を DBMS インターフェース言語として組み込むという方針を、ひとまずとった。

2.2 データベース

製品版 DBMS として最初に候補に上げたのは RDBMS(Oracle) である。

しかし、ベンチマークの結果、Java インターフェースによる BLOB(Binary Large Object) のアクセスが異常に遅く、使いものにならないことがわかった。また利用者要求を分析していく過程において、衝突データを RDBMS の枠に当てはめるのが非常に難しいことがわかってきた。

上の理由から RDBMS の利用を見直し、DBMS に Allegro Common Lisp 上の OODBMS(オブジェクト指向型 DBMS)、AllegroStore を採用することを決めた。

2.3 CORBA

クライアント・サーバー間の通信には CORBA 製品を適用した。

開発期間が限られているため、開発言語が各々異なるクライアントとサーバーの両者間に独自のプロトコルを設計/実装している時間がないと判断したことが最大の理由である。

サーバー側は Allegro Common Lisp 上の ORBLink[3, 4] を、クライアント側には JavaORB² を使った。

¹ 他に C インターフェースが提供されていたが仕様が直感的でなく理解しづかったこと、サーバーを C で書いているだけの人的・時間的余裕がないと判断したため、却下した。

² <http://dog.exoffice.com>

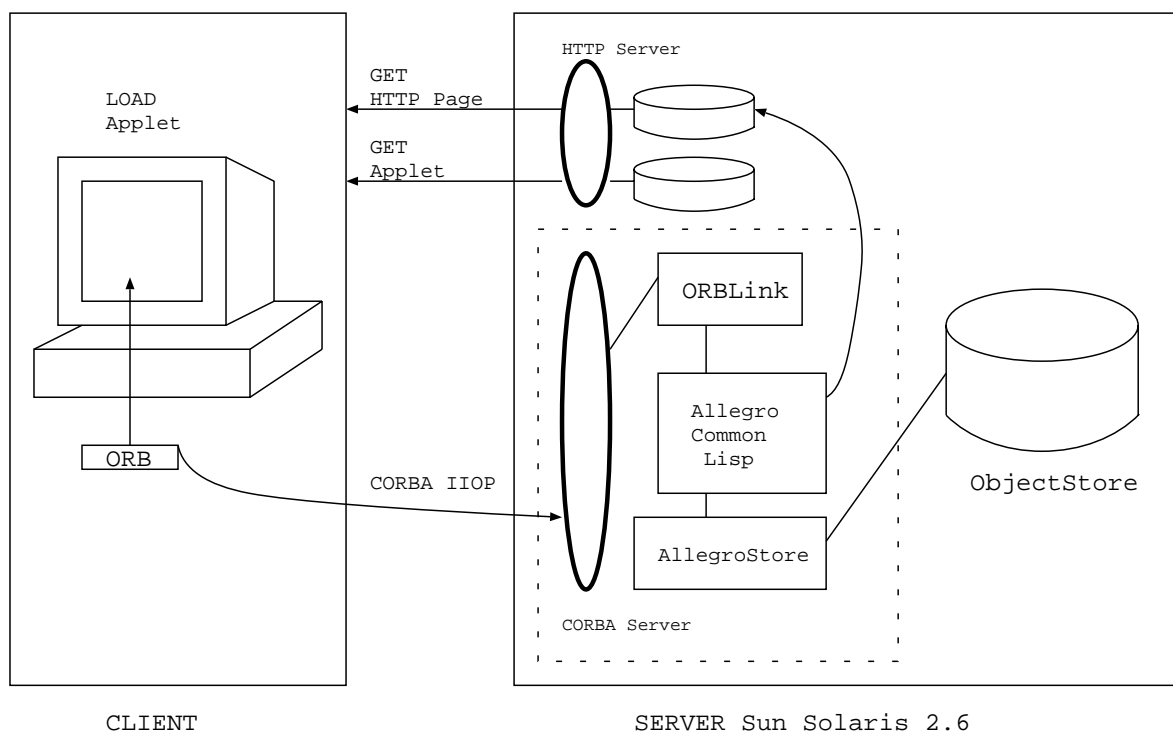


図 1: クライアント・サーバー構成図

3 システム構成

3.1 ハードウェア構成

ハードウェアは、ホスト機となる Sun450(Solaris 2.6)、1 テラバイトの RAID5 ハードディスクシステム StorEdge A3000(35 ギガ*8*5 本)、テープ記憶装置に DTF2(Digital Tape Format) ドライブ、テープ操作ロボット DMS(Digital Mass Storage system)36 スロットが与えられた。

3.2 クライアント・サーバー構成

クライアント・サーバー構成を図 1 に示す。図の点線で囲まれた部分が本システムにおけるサーバー部の核である。この部分はプログラム変更なしにマルチプロセス化が可能で、かつ、ネットワーク上の任意のノードにそのプラットフォームを問わず置くことができる。

4 データベース

4.1 スキーマ定義と CORBA

AllegroStore の永続オブジェクトの記述書式に従ってスキーマを定義した。

CORBA オブジェクトインプリメンテーションを、データベーススキーマと 1:1 対応を持つ一時 (非永続) オブジェクトとしてサーバープロセスに置いた。

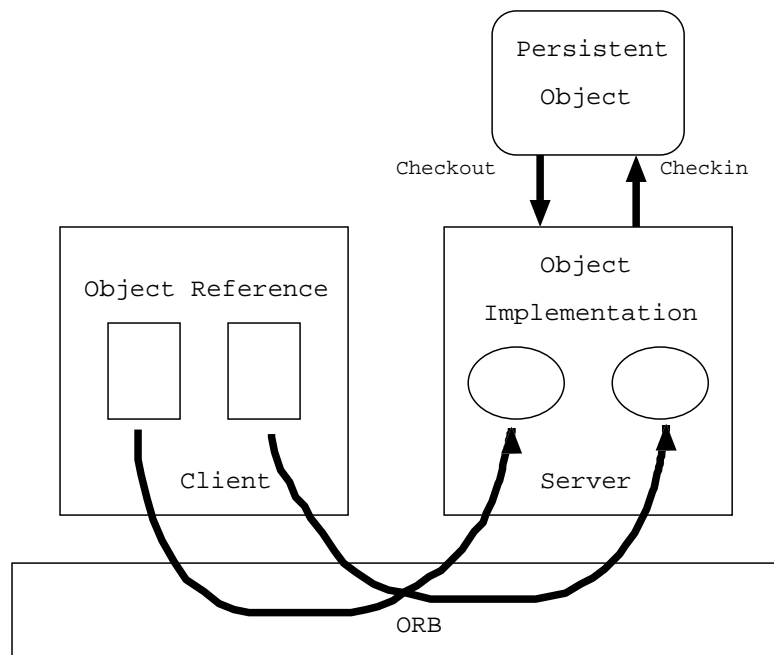


図 2: チェックイン・チェックアウト機構

4.2 チェックイン・チェックアウト

一時オブジェクトは、クライアントとデータベース間の緩衝として位置付けられる。

クライアントのチェックアウト要求により、データベースの内容が一時オブジェクトへ読み出される。チェックアウトの間、クライアントは一時オブジェクトの内容を自由に書き換えることができる。必要な書き換えが終われば、クライアントはチェックイン要求を出し、一時オブジェクトの内容がデータベースへ書き出される。

永続オブジェクト、一時オブジェクトと、クライアントとの関係を図2に示す。

永続オブジェクトと一時オブジェクトの定義、および、チェックイン・チェックアウトメソッドは Common Lisp のマクロを用いて簡便に定義できるようにした。

マクロを使ったスキーマ定義の例

```
(defschema "TEST" (test-po root-po)
  (test-snap root)
  (test-impl test-servant schema-impl)
  ((test_date (type fixnum) (:documentation "予定日")))
  (test_mode (type symbol)
    (choices car2car car2mdb) (default car2car))
  (test_count (type symbol))
  (test_target (type "TESTTARGETSEQ") (listp t)
    (inst test-target) (ref ref))
  )
  (dbname *root-database*))
```

スキーマ定義のマクロ出力例

```

(PROGN

  (DEFCLASS TEST-PO (ROOT-PO)
    ((TEST_DATE :ALLOCATION :PERSISTENT :ACCESSOR TEST_DATE
      :INITARG :TEST_DATE)
     ....)
    (:METAClass PERSISTENT-STANDARD-CLASS))

  (DEFCLASS TEST-SNAP (ROOT)
    ((TEST_DATE :ACCESSOR TEST_DATE :INITARG :TEST_DATE)
     ....))

  (DEFMETHOD CHECK-IN ((OBJ-PO TEST-PO) (OBJ TEST-SNAP) &KEY EXP-USER
    LAZYP)

    (DECLARE (IGNORE EXP-USER LAZYP))
    (IF (NEXT-METHOD-P) (CALL-NEXT-METHOD))
    (IF (SLOT-BOUND P OBJ 'TEST_DATE)
      (SETF (TEST_DATE OBJ-PO) (TEST_DATE OBJ))
      (SLOT-MAKUNBOUND OBJ-PO 'TEST_DATE))
    ....
    OBJ)

  (DEFMETHOD CHECK-OUT ((OBJ-PO TEST-PO) &KEY OBJ EXP-USER LOCKP LAZYP)
    (LET ((OBJ
      (OR OBJ (MAKE-INSTANCE 'TEST-SNAP
        :INSTANCE-ID (INSTANCE-ID OBJ-PO)
        :DB-NAME (DB-NAME (DATABASE-OF OBJ-PO))))))
      (AND (SLOT-BOUND P OBJ-PO 'TEST_DATE)
        (SETF (TEST_DATE OBJ) (TEST_DATE OBJ-PO)))
      ....
      (IF (NEXT-METHOD-P)
        (CALL-NEXT-METHOD OBJ-PO :OBJ OBJ :EXP-USER EXP-USER
          :LOCKP LOCKP :LAZYP LAZYP))
      OBJ))

  (DEFCLASS TEST-IMPL (TEST-SERVANT SCHEMA-IMPL) NIL)

  (OMG.ORG/CORBA:DEFINE-METHOD TEST_DATE ((THIS TEST-IMPL))
    ....)
  (DEFMETHOD (SETF OMG.ORG/OPERATION::TEST_DATE) (NEWVAL (THIS TEST-IMPL))
    ....)

  .....
)

```

4.3 IDL

サーバーがもつオブジェクトインプリメンテーション (一時オブジェクト) とそのメソッド構成を、そのまま IDL に書き写したものをクライアントインターフェースとした。

IDL の例

```

interface SchemaBase : ObservableSchema {
    void checkout()
        raises (CheckoutFailureException, ServerFailureException);
    void checkin()
        raises (CheckinFailureException, ServerFailureException);
    void cancel();
    .....
};

interface Test : SchemaBase {
    attribute string test_date;
    attribute string test_mode;
    attribute string test_count;
    attribute TestTargetSeq test_target;
    TestTarget test_target_ref(in short n);
        // 試験ターゲット N 番目参照
    TestTarget push_test_target();
        // 試験ターゲットのプッシュ
    .....
}

```

4.4 静止/動画像

AllegroStore が持つ BLOB 機能を使って静止および動画像をデータベースへ入れ、他のデータとともに一元管理/運用させている。

静止画は 40K ~ 200Kbyte 程度の JPEG あるいは BMP フォーマットのもの。一回の試験につき 0 ~ 100 程度の個数ある。動画はオリジナルが 100 ~ 300Mbyte 程度の AVI フォーマットで、これを Cinepa Codec に圧縮し、約 1/10 のサイズにし、データベースへしまつて、常時参照可能としている。

5 ファイルのアップロード・ダウンロード

Java Applet プログラミングにおいてはセキュリティ上の問題でローカルファイルのアクセスを許していない。しかし、サーバー上のデータをローカルにコピーしてきて、独自に加工を加えたり、またローカルに作った文書ファイルをサーバーに登録しておいたりする機能はごく自然な要求として考えられる。

これに答えるため、HTTP を利用したファイル受け渡しを実現した [7, 8]。図 3 に動画のアップロード画面を示す。

5.1 CL-HTTP

最初 HTTP サーバーには apache³ を使って開発を進めた。

しかし、ファイルのアップロード・ダウンロードのための CGI プログラムと、サーバープログラムとの通信が複雑になるのを避けるため、途中で CL-HTTP サーバー⁴ の利用に切り替えた。

apache を利用するのに比べ、CGI 機能をサーバープロセス中に取り込めるため、エラー処理などを少ない労力で書くことができた。また、継がった相手によって動的に HTML の内容を変える、などの木目の細かい対応をしている。

³ <http://www.apache.org/>

⁴ <http://www.ai.mit.edu/projects/iip/doc/cl-http/home-page.html>

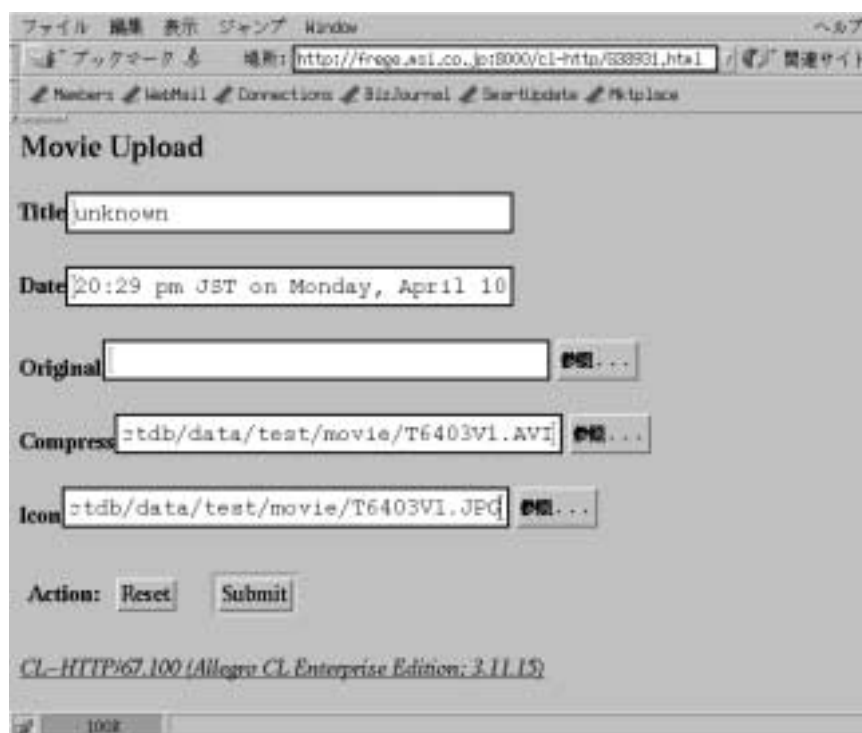


図 3: 動画アップロード画面

6 法規基準記述

法規基準記述の枠組みとなる傷害値計算と波形グラフ定義には、CLOS(Common Lisp Object System)の書式をそのまま応用した。これにより可読性を失なうことなく、シンタクスとセマンティクスの開発を軽減することができた。

6.1 傷害値計算

傷害値計算には、頭部傷害指数である HIC(Head Injury Criteria) や、胸部障害指数である TTI(Thoracic Trauma Index)、VC(Viacous Criterion) などがある。これらは Common Lisp 関数として定義される。

6.2 波形グラフ定義

波形グラフ定義は以下の要素からなる。

- GD (Graph Definition) グラフ要素を配置する。
- LD (Layout Definition) GD で定義するグラフを一個以上配置する。
- UP (User interface Parameter) ユーザーパラメーターを設定する。

それぞれを CLOS のクラスとして定義した。

レイアウト定義例

```
(defclass driv-head-layout (xyz-layout)
  ((Value1 :accessor Value1
           :initform (make-instance 'label-factory
                                   :pos '(60.0 215.0)
                                   :text ""
                                   :height 5.0
                                   :width 3.0
                                   :charSpace 0.02
                                   :color 'white))
   .....
   (LineV1 :accessor LineV1
           :initform (make-instance 'line-factory
                                   :pos '(nil nil nil 230.0)
                                   :clipIndicator t
                                   :clipRect '(0.0 380.0 0.0 250.0)
                                   :color 'white))
   .....
   (Result11 :accessor Result11
             :initform (make-instance 'label-factory
                                     :pos '(60.0 220.0)
                                     :text ""
                                     :height 5.0
                                     :width 3.0
                                     :charSpace 0.02))
   .....
  ))
```

6.3 グラフ描画と印刷

6.3.1 波形グラフ描画

波形グラフは先の法規基準記述にのっとり、サーバー側で線やラベルなどの描画基本要素を生成し、CORBA を介して Java クライアントに送りだす。

図 4 に波形グラフ描画の例を示す。

6.3.2 波形グラフ印刷

レポート等に利用される波形グラフ印刷には縦横比に厳密性が要求される。波形グラフ印刷はサーバーが一括して受け持ち、PDF を生成することで、どのクライアントからも一定品質の印刷物が得られるようにはかった。

PDF の生成には ClibPDF⁵ を利用し、Allegro Common Lisp の FFI(Foreign Function Interface) から制御した。

7 データベース管理

バックアップを含めたデータベース管理には、AllegroStore のコマンド群を利用し、一元化している。

バックアップは年 3 回のフルバックアップと、日/週/月毎のオンラインバックアップを行なうようにした。

⁵ <http://www.fastio.com/>

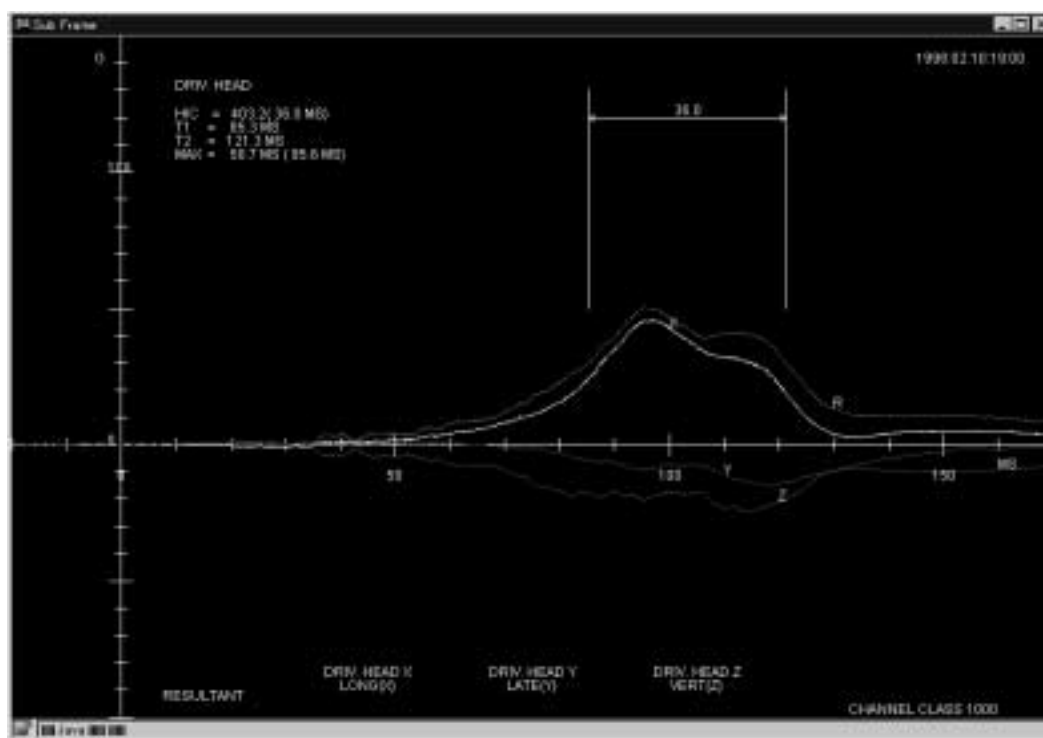


図 4: 波形グラフ描画の例

万一の障害発生時には、最初に最重要データのみをリストアし、あとはオンラインリストア機能を用いることで、ユーザーの体感的な障害回復時間を短縮している。

8 評価

Allegro Common Lisp、AllegroStore、ORBLink という要素技術の組み合わせでサーバーシステムを開発し、以下のような効果を得ることができた。

8.1 開発効率

- Common Lisp マクロ機能によるメタプログラミングを活用し、コーディング量を減らすことができた。
- OODBMS を選択したためにデータの出し入れ、特に画像データのそれについて、開発工数面とパフォーマンス面の両方で効率をあげることができた。
- マルチスレッド・マルチプロセスへの対応がプログラマー負荷なしに実現できた。

8.2 運用/保守性

- データを DBMS に一元管理することで、運用を簡便にすることができた。
- オペレーティングシステムの版改正などともなうプログラム修正を最小限に留めることができています。

8.3 可用性

- オンラインバックアップ/リストア機能によりシステムの可用性を高めることができた。

8.4 移植性・拡張性

- クライアント・サーバーを通して、オブジェクト指向の枠組みで開発できたことで、全体に見通しの良いシステムとなっている。
- 移植性を高め、ヘテロジニアスなシステム拡張を可能にしている。

8.5 信頼性・頑強性

- 開発効率があがったぶん、負荷試験や異常系の試験に多くの時間を割くことができ、結果としてシステムの信頼性を高めることができた。

9 まとめと問題点

以上のように、Common Lisp を軸とした開発はおおむね成功を収めることができた。以下のような問題点があったことを報告しておく。

9.1 AllegroStore の頑強性

AllegroStore をマルチプロセス化して利用した場合に、負荷をかけると AllegroStore が “Segmentation Violation” を起こす、という問題があった。

また、ある程度大きな BLOB オブジェクトを登録していくと、“ERR_BEYOND_SEGMENT_LENGTH” というエラーを起こすため、一つのデータベースに登録可能な BLOB が制限される、という不具合があった。

9.2 データベース管理コマンド群の Lisp バインディング

AllegroStore がエンジンとして使っている ObjectStore の管理コマンドまわりが Lisp バインディングされておらず、定期バックアップの自動化の実現に手間がかかった。

9.3 ORBLink の使い方

CORBA パッケージを使うと、二度目以降のファイルロードが異常に遅くなる、という問題があった。

Lisp の記述と IDL の記述が二度手間であった。Lisp 記述から IDL を自動生成してくれるような機能が望まれた。

9.4 開発機の限定

商用の言語処理系一般に言えることだが、機械毎にライセンス購入しないと、開発時のプラットフォームを分散させることができない。そのため、複数人数で開発する際、一つのマシンに負荷が偏る、といった問題があった。

9.5 その他

今回、不特定多数のクライアント機からのアクセスを想定して、クライアントプログラムを Java Applet で書いたが、仮にクライアント機へフリーあるいは非常に廉価で Common Lisp 処理系をインストールすることができれば、CLIM(Common Lisp Interface Manager) などの GUI ツールを用いての実現が考えられた。その場合 AllegroStore がもつクライアント・サーバー機能を用いれば、CORBA は必要なく、今よりシンプルにシステムが構成できたと考えられる。

参考文献

- [1] 自動車技術シリーズ7 自動車の計測解析技術 (社) 自動車技術会 編集, 1998
- [2] Allegro Store manual version 1.3 June, 1999, Franz Inc.
- [3] ORBLink 1.0: USER GUIDE, Franz Inc.
- [4] IDL/LISP MAPPING VERSION 1.0, Franz Inc.
- [5] Allegro CL HTML documentation, Franz Inc.
- [6] ロバート・オーファリ、ダン・ハーキー著, Java & CORBA C/S プログラミング, 日経BP社
- [7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, "Hyper Transfer Protocol – HTTP/1.1", RFC2616, June 1999.
- [8] E. Nebel, L. Masinter, "Form-based File Upload in HTML", RFC1867, November 1995.
- [9] MIRA Data Viewer V3.0, Feb 1997, Motor Industry Research Association and the University of Central England in Birmingham.