全国共同利用実験用 DB システムにおける Ajax を用いた UI の実装

○山田研二 A)

A)大阪大学 レーザーエネルギー学研究センター

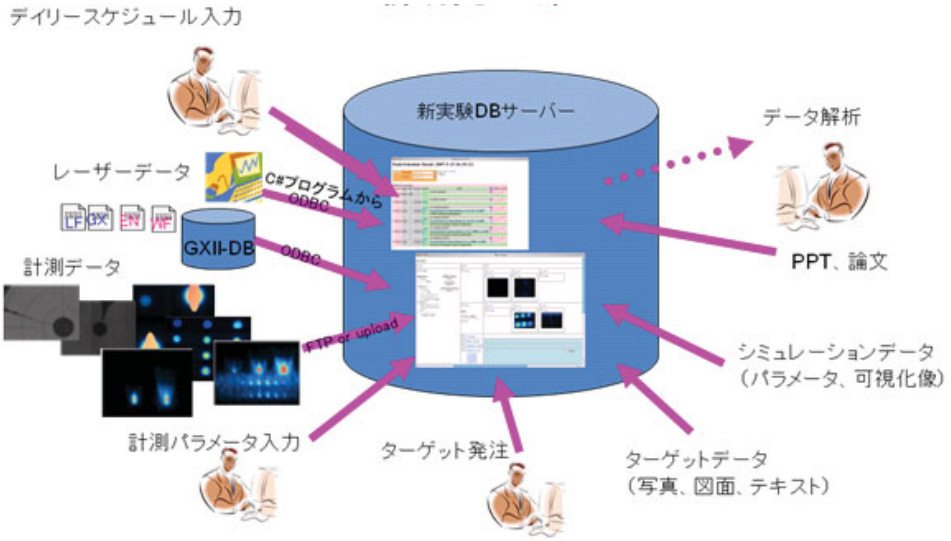
概要

大阪大学レーザー研では大型レーザー装置を主軸とした全国共同利用研究が行われており、円滑な実験とデータ管理のため、汎用的でセキュリティの高い DB システムが不可欠となっている。このため、XML-DB および Web-UI、定型作業の自動化機能等を内包した研究用 Web システムである RCM System Software を導入し、構築を進めている。同システムは高機能な反面、データ利用だけを目的としたユーザーにとっては操作が複雑で難しく感じられる側面もある。これを解消するため、Ajax と PHP を用いてライブラリを構築し、シンプルかつ応用可能な UI を実装したので、これについて報告する。

1　はじめに

大阪大学レーザーエネルギー学研究センター（以下、「レーザー研」）では高強度レーザー装置「激光 XII 号」を用いた様々な全国共同利用実験が行われている。激光 XII 号は最高出力 20kJ のガラスレーザーであり、12 本のビームを有す。

激光 XII 号実験においては図 1 に示すように様々なデータが発生する。それを統合的に管理し、利用できるようにするためにはデータベース（以下、「DB」）が不可欠である。



- 実験関係者、研究グループ、実験シリーズデータ
- 実験スケジュール（日時、レーザー条件、集光条件、ターゲット条件等）
- レーザーデータ（波長、目標・計測エネルギー、時間波形等）
- 計測データ（X 線フレーミングカメラ、ストリークカメラ、ピンホールカメラ等）
- 燃料ターゲット
- シミュレーションデータ（パラメータ、可視化像等）

図 1.　DB に格納されるデータのイメージ

DB システムとしては関係データベース（RDB）が一般的であるが、近年は XML のツリー構造をそのまま

格納する XML データベース（XML-DB）も普及してきている。XML の特徴として拡張性が高いことが挙げられる。スキーマを完全に定義せずに運用を開始できるため、試行錯誤を繰り返しながら基盤整備を進めるような用途にはメリットが大きい。

センター内・外を問わず、様々な人間が DB システムを利用するため、高いセキュリティとデータ保護技術が要求される。データの種類ごとにアクセス権を設定し、不用意にデータを公開しないようなポリシーも必要である。

このようなニーズに応えるため、レーザー研では XML-DB および Web-UI、定型作業の自動化機能等を内包した研究用 Web システムであるキャトルアイ・サイエンス社の RCM System Software[1]（以下、「RCM」）を導入した。同システムの動作イメージを図 2 に示す。

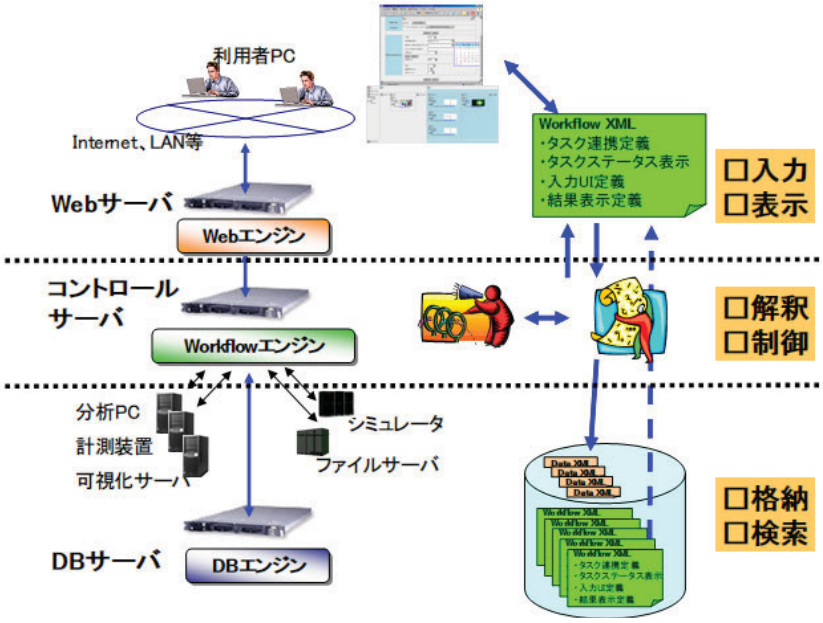


図 2.　RCM の動作イメージ

2009 年度より RCM を用いた実験データベースの構築を開始し、2010 年 1 月より運用を開始している。本実験 DB システムを「SEDNA」（**S**ystem of **E**xperimental **D**atabase for **NA**tional user's laser facility）と呼ぶ。また、本稿で紹介する独自の UI（User Interface）を「SEDNA-UI」と呼ぶ。

2　RCM の Template と簡易 UI、XML Viewer

RCM ではデータ構造だけでなく、自動実行用スクリプトの記述にも XML を用いる。この自動実行用スクリプトを「Workflow Template」（以下、「Template」）と呼ぶ。前述の通り RCM は Web-UI を内蔵しており、Template 内に特定の属性を記述することによって「簡易 UI」と呼ばれる簡易的な HTML フォームを表示することが可能である。また、Template で選択したデータの表示部分には「XML Viewer」と呼ばれる専用の XML ビューアーが搭載されている。

2.1　簡易 UI の問題点

簡易 UI の例として、実験スケジュールの入力用 Template のものを図 3 に示す。

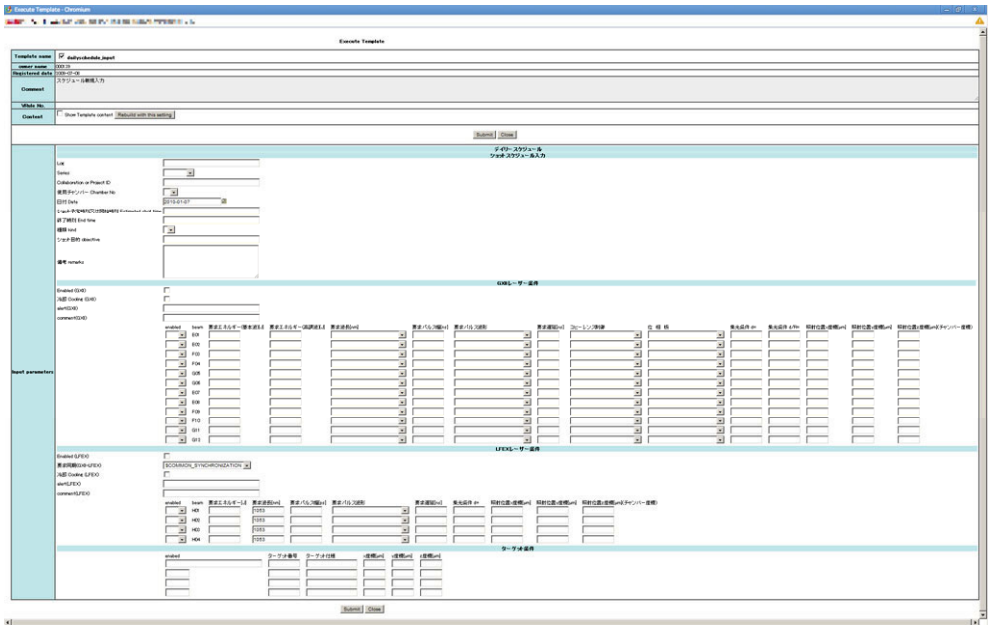


図 3.　簡易 UI の一例

実験スケジュールではレーザービームごとの条件を入力する必要があり、250 個以上のデータ点数となるため、図のように大量のフォーム要素が羅列される。これでは、単に入力が面倒なだけでなく、入力すべき項目が判別しづらく、ミスが起こりやすい。入力を補助する機能が少なく、それをユーザーが実装することもできない。また、送信時に確認が表示されないため、誤ったデータをアップロードしてしまい、混乱を招く可能性がある。

SEDNA は RCM を使い慣れないユーザーも利用するため、ミスが生じにくくユーザーフレンドリーな UI を実装する必要がある。SEDNA-UI では下記の 4 点を重視している。

1. そもそも不適切なデータを入力できないようにする
2. 入力ミスを検証・通知できるようにする
3. すでに入力したデータを活用できるようにする
4. ユーザーに再チェックを促す

2.2　XML Viewer の問題点

実験スケジュールの 1 データを XML Viewer で表示させたものを図 4 に示す。

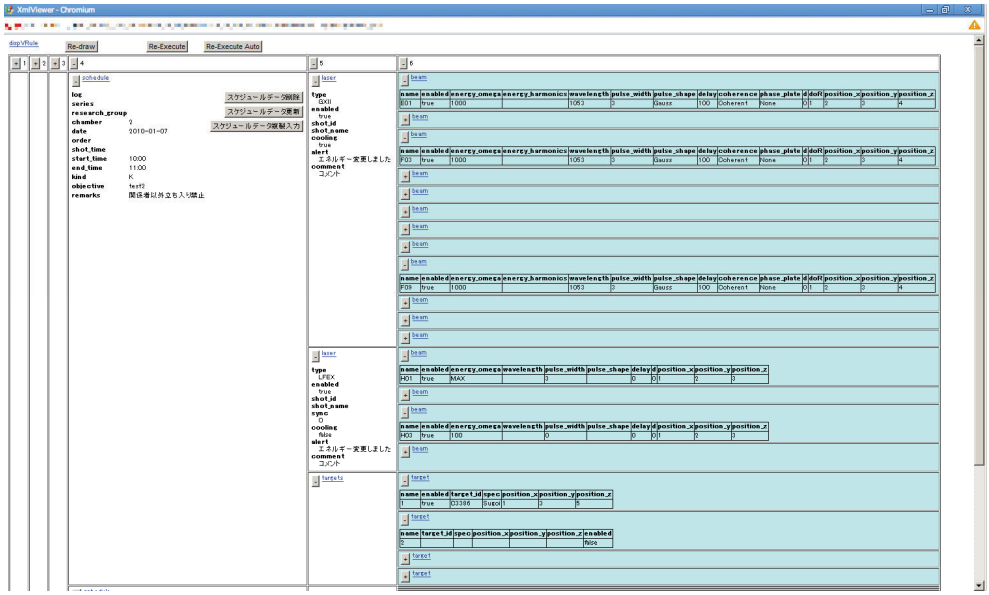


図 4.　XML Viewer の表示例

XML Viewer はツリーの展開・縮小、画像データの表示など XML ツリーを表示するビューアーとしては非常によくできており、汎用性が高い。また、タグ名のリンクからタグの編集・削除ができたり、任意のボタンを配置して別の Template を呼び出したりと、便利な機能が実装されている。

しかし、図のように大量のデータが格納された XML ツリーを表示する必要がある場合には、XML Viewer 上での視認性と操作性が問題となってくる。このような場合には、Template 上で XSLT を用いて XML データを HTML に変換して表示など、視認性と操作性を確保する為の対応が必要となる。

3　設計

前章で述べた通り、RCM の簡易 UI と XML Viewer が提供する入力と出力の実装は汎用的である反面、データの種類や量によっては操作性が悪い。これを改善するためには RCM とは別に UI を作成し、それを介して RCM を操作する必要がある。

まずすべての実験関係者が目にする実験スケジュールの編集や表示に関し、前述の問題点を解決したユーザーフレンドリーな UI を実装した。またこれに並行して、同様な UI を容易に実装できるようライブラリを構築した。本稿ではこのうち編集系の UI の開発に関して述べる。表示系に関しても、独自の HTML 表示を実装したが、本稿では割愛する。

3.1　設計方針

ページ表示は保守性の観点からすべて HTML+CSS をベースとする。要件は下記の通りである。

- 入力が必要な項目のみを表示すること
- 特定の入力項目に対し、入力制限をかけることができること（数値、英数字など）
- 特定の入力項目に対し、頻繁に使用される文字列を候補として表示すること
- レーザーの条件はコピー&ペーストなどで入力を容易にすること
- 過去のスケジュールを引用して新しいスケジュールを作成できること
- ライブラリ化して再利用できるようにすること

3.2　開発言語

SEDNA-UI で用いるプログラミング言語を図 5 に示す。いずれの言語も万能ではないため、それぞれを補い合うように連携させるのがよい。

	RCM Template	JavaScript/Ajax	PHP: Hypertext Preprocessor
動作環境	RCM 内部	クライアント側	サーバー側
構文	XML ベース	C/Java ライク	C ライク
メリット	○データベースとの通信を意識することなく、セキュアにデータアクセスができる ○スクリプト実行のログが詳細に残るため、デバッグに便利である	○HTMLを画面遷移なしに直接書き換えることができる ○非同期通信（ブラウザの画面遷移を伴わない通信）ができる ○リッチなインタフェースが実現できる（プログレスバーなど）	○習得が比較的容易 ○標準機能が豊富 ○DBとの親和性が高い ○クロスドメイン通信が可能
デメリット	△XMLのため文字数が多い △専用の開発環境がない	△別ドメインへの通信（クロスドメイン通信）ができない △ブラウザごとに実装・動作が異なる場合がある △ブラウザのリロードボタンを押すと変数や状態がリセットされてしまう	△ブラウザ・ページの動的制御はできない
SEDNA-UI での役割	☆DB操作（データの選択、追加、更新、削除）	☆ログイン情報（セッション）の管理 ☆非同期通信（PHP経由Ajax） ☆状態（データ）のフォーム要素への復元 ☆入力支援等のリッチUIの提供	☆HTMLの生成 ☆クロスドメイン通信

図 5.　使用する言語の比較

RCM の場合、SQL や XPath を記述してデータに直接アクセスすることはできない。そのため、外部から DB 操作するための Template を記述し、専用の操作用 URL に対して REST（REST:Representational State Transfer、HTTP 通信を用いて指定した URI のリソースを得ること）で通信しなければならない。よって Template はこの DB アクセスの根幹に用いる。

JavaScript に関しては近年、非同期通信部分が Ajax（**A**synchronous **Ja**vaScript + **X**ML）として確立され、JavaScript を用いた動的制御技術の代名詞となっている。大きなデメリットであったブラウザ間の差異に関しても、各種の Ajax ライブラリが整備されたことで大きく軽減された。SEDNA-UI では Ajax ライブラリの一つである prototype.js を利用している。動的処理を活かし、JavaScript をメインとして用いる。セキュリティ上の問題からクロスドメイン通信（別ホストへの通信）はできない仕様となっている。

PHP は機能やライブラリの豊富さが長所である。基本となる HTML の生成と JavaScript では不可能な事前処理やクロスドメイン通信を行う（後述）。図 6 は言語間の連携イメージである。

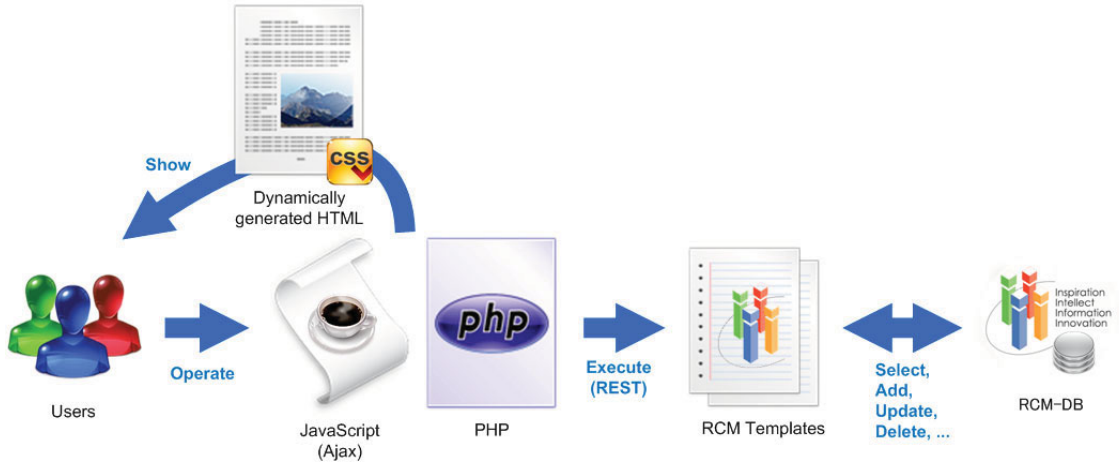


図 6.　言語間の連携

4　開発のポイント

4.1　クロスドメイン通信

REST には Ajax を用いるが、前述の通り JavaScript は基本的にクロスドメイン通信に対応しない。SEDNA-UI は RCM とは別のサーバーに実装するため、JavaScript だけでは実現できない。そこで PHP をプロキシとして用い、JavaScript からのリクエストを RCM に転送し、RCM からのレスポンスを JavaScript に返すようにした。

図 7 に PHP プロキシの実装例を示す。fopen メソッドによって指定した URL のデータが取得できる。PHP で OpenSSL が有効になっていれば HTTPS の URL からも同様に受信できる。この PHP スクリプトに対して、Ajax から URL を指定してリクエストを投げてやればよい。

```
function HttpPost($url, $data, $optional_headers = null) {
   $params = array('http' => array('method' => 'POST', 'content' => $data));
   if ($optional_headers !== null) {
      $params['http']['header'] = $optional_headers;
   }
   $ctx = stream_context_create($params);
   $fp = @fopen($url, 'rb', false, $ctx);
   if (!$fp) {
      return "ERROR: Login Failed.";
   }
   $response = @stream_get_contents($fp);
   if ($response === false) {
      return "ERROR: Data Unavailable";
   }
   return $response;
}

$url = $_POST["url"]; // URL を保存
unset($_POST["url"]); // $_POST から url を抜いておく
header('Content-type: text/xml; charset=UTF-8'); // Content-type の出力
$data = http_build_query($_POST); // クエリ文字列の生成
$response = HttpPost($url, $data); // 送信→応答が $response に返る
echo mb_convert_encoding($response, "EUC-JP", "UTF-8"); // エンコードの変更
```

図 7.　プロキシスクリプト実装例（PHP）

これにより、表面的にはあたかも JavaScript と RCM が直接通信しているように扱うことができる。ただし、プロキシとして実装した PHP スクリプトは SEDNA-UI 以外からのリクエストも受け付けてしまうため、攻撃の踏み台とされる可能性がある。URL を公開しない、実行元を制限するなどの対処が必要である。

4.2　必要な入力項目のみを表示

「そもそも不適切なデータを入力できないようにする」ためにはユーザーに入力項目が見えない状態にするのが最良である。実験スケジュール編集画面では図 8 のように種別によってモードを切り替え、必要な入力項目のみが表示されるようにした。

prototype.js では Element クラスに拡張メソッドが実装されているため、$('id').show() や $('id').hide() で表示・非表示が可能である。ただし、あらかじめ非表示にしておきたい要素の場合、CSS で display:none に設定されているものは show メソッドで表示できないので、window.onload イベントで hide メソッドを使って非表示にしておく。

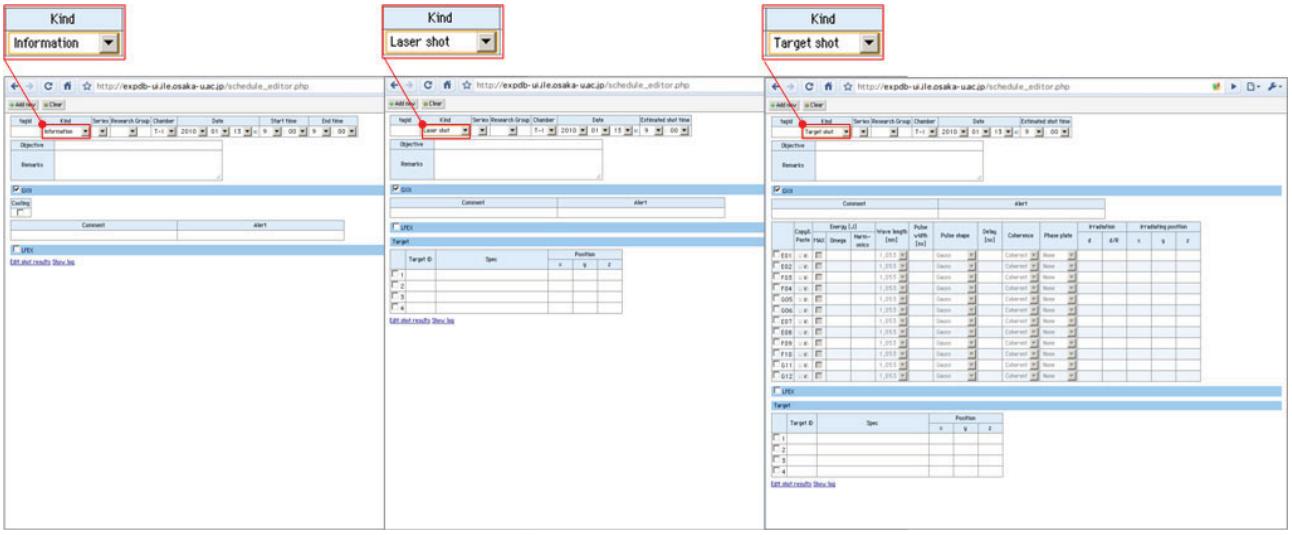


図 8.　種別による入力項目の切り替え

4.3　入力制限（TextBoxRestriction）

テキストボックス（<input type="text"/>）へのキー入力を特定の文字に制限できるように TextBoxRestriction 機能を実装した。これにより不本意な型のデータ入力を抑制できる。

入力を許可する文字に関しては図 9 のようにテキストボックスのクラス名（class 属性）と正規表現のセットによって定義できる。

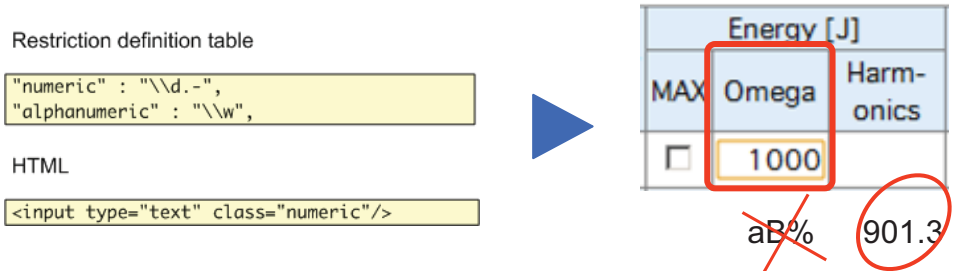


図 9.　TextBoxRestriction

あとはテキストボックスの HTML に class 属性を記述すれば、このテキストボックスは英数字しか入力できなくなる。念のためキー入力だけでなくフォーカスを失った際にも内容を検証することで、コピー&ペーストによる誤入力も抑止している。不適切な内容が検知された場合は、自動的に修正あるいはダイアログで通知することができる。

4.4　入力支援（Suggest）

テキストボックスにフォーカスが移ると既定の選択肢が表示される Suggest 機能を実装した（図 10）。頻出文字列のスピーディな入力や表現の統一が図れる。テキストボックスの id と一致するタグを別ファイルの XML に記述するだけで候補が提示される。

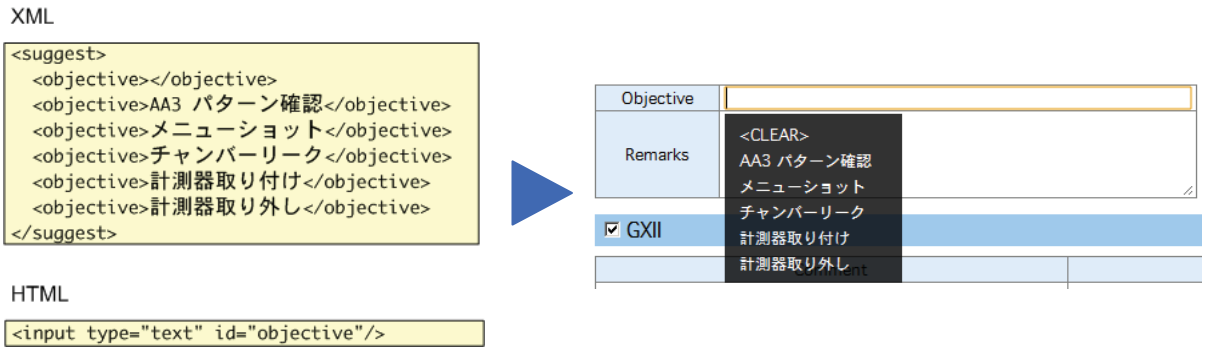


図 10. Suggest

4.5 コピー&ペーストの実装

入力項目が増えるほどユーザーの負担は増大するため、同じようなデータは簡単に入力できるほうがよい。今回はもっとも入力項目の多いレーザー条件(項目は 1 ビームにつき 14 個、全ビーム分では 168 個にも及ぶ。)をコピー&ペーストできるようにした。図 11 に示すように、専用のボタンで実行できる。

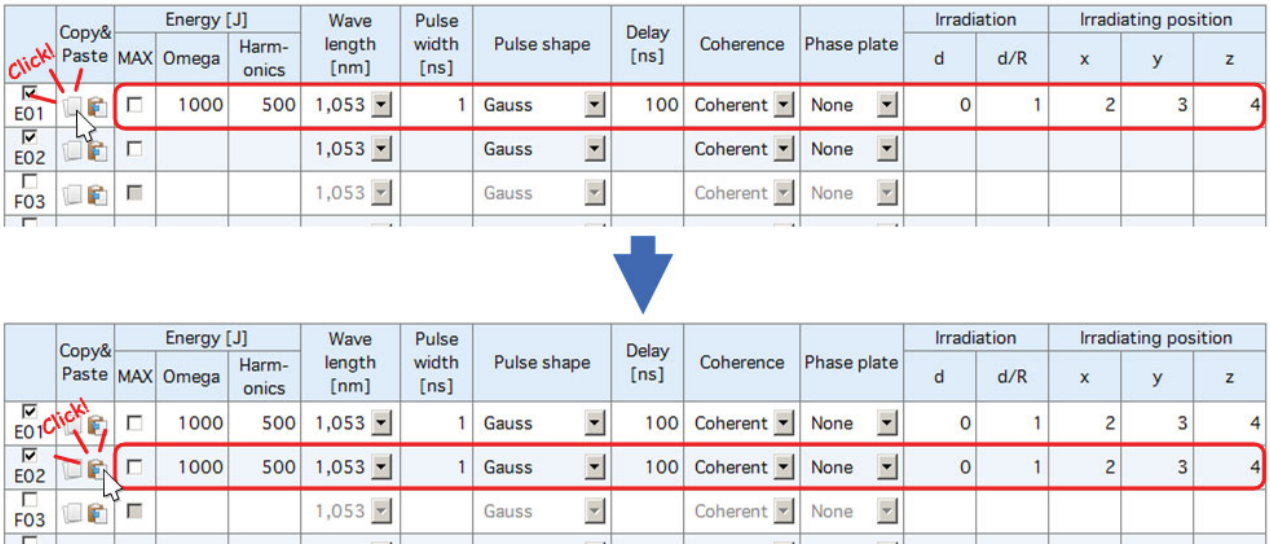


図 11. レーザー条件のコピー&ペースト

JavaScript はローカルマシンのクリップボードにアクセスできないため、クリップボードは使用していない。

```
var setting = new Array(); // クリップボード代わりの配列
setting['energy'] = $("energy").value; // コピー
$("energy ").value = setting['energy']; // 貼り付け
```

図 12. JavaScript によるコピー&ペーストの実装

図 12 に示すようにクリップボードの代わりとなる配列を用意し、コピーボタンが押されたら配列に代入、貼り付けボタンが押されたらそこから復元、というように独自に実装してある。このため Ctrl+C などのショートカットキーは利用できないが、簡単に実装できる上、他の作業に影響を与えずコピー&ペーストが行えるメリットがある。

4.6　既存データのロード・復元

データを編集・更新する際や似たようなデータを入力する際は、一度データをロードし、フォームに復元しなければならない。

RCM に保存されているデータは XML 形式であるが、JavaScript では直接 XML を扱いにくいため、JSON（JavaScript Object Notation）形式に変換している。JSON はその軽量さと汎用性からデータ通信で多く用いられるようになってきたデータ記述言語である。特に開発者にとってはなじみのある構文と文字数の少なさから歓迎されることが多い。

変換に用いているメソッドを図 13 に示す。この実装では Ajax による非同期通信の応答に対して使用するため、引数の ajax には Ajax.Request などで得られる XMLHttpResponse オブジェクトを渡すようにしてある。戻り値は JSON 形式の文字列である。JavaScript では eval が使用できるので、var params = eval("(" + json + ")"); のようにパースしてやればよい（カッコ "(" ")" を付加しないと処理されないので注意）。

```
function XMLToJSON(ajax) {
  if (window.ActiveXObject) {
    var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
    xmlDoc.async = false;
    xmlDoc.loadXML(ajax.responseText);
  } else if (window.DOMParser) {
    var xmlDoc = new DOMParser().parseFromString(ajax.responseText, "application/xml");
  } else {
    return;
  }
  var loopParse = function(obj) {
    var res = {}, cacheTag = {};
    var ob = {}, att = obj.attributes;
    if (att != null && att.length != 0) {
      for (var a = 0, lenA = att.length; a < lenA; a++) {
        ob[att[a].nodeName.toLowerCase()] = att[a].nodeValue;
      }
      res._attr = ob;
    }
    if (obj.childNodes.length > 0) {
      for (var i = 0, len = obj.childNodes.length; i < len; i++) {
        var ch = obj.childNodes[i];
        if (ch.nodeType == 3) {
          if (ch.nodeValue.replace(/[\s|\t|\n]/g, "") == "" || ch.nodeValue == null) continue;
          else return ch.nodeValue;
        }
        else if (ch.nodeType == 1) {
          (ch.tagName in cacheTag) ? cacheTag[ch.tagName].push(arguments.callee(ch)) : cacheTag[ch.tagName] =
            [arguments.callee(ch)];
        }
      }
    } else {
      return "";
    }
    for (var p in cacheTag) {
      (cacheTag[p].constructor == Array && cacheTag[p].length == 1) ? res[p] = cacheTag[p][0] : res[p] = cacheTag[p];
    }
    return res;
  }
  return loopParse(xmlDoc);
}
```

図 13. XML→JSON メソッドの実装例

フォーム要素への復元はすべての要素を逐一記述すると冗長なので、XML のタグ名とフォーム要素の id を一致させ、図 14 のように for〜in で回してやればよい。JavaScript の for〜in は他の言語と異なり、要素ではなくキーを返すので注意が必要である。また、ブラウザによってキー以外のプロパティが含まれるので、id の null チェックが不可欠である。

```
for (id in params) {
  var value = params[id];
  var obj = $(id);
  if (obj == null) continue;

  if (value == "true") { obj.checked = true; }
  else if (value == "false") { obj.checked = false; }
  else { obj.value = value; }
}
```

図 14. フォーム要素へのデータ復元

4.7　ライブラリ化

本章で紹介した機能はすべてライブラリとしてまとめ、再利用可能なコードにした（図 15）。ページごとのスクリプトはこれとは別ファイルで作成している。ライブラリ群のファイルサイズは 200KB 弱（prototype.js 含む）である。これによって重複コードが抑制でき、UI 実装コストの削減が期待できる。

図 15. ライブラリ群の構成

5　まとめ

本稿では実験 DB を操作する Ajax を利用した UI の実装に関して述べた。幾度かの動作テストとデバッグを経て、2010 年 1 月より「SEDNA-UI」の運用を開始している。間違いが少なく入力できるため、評価は良好である。入力制限などの機能実装に際して、有用なテクニックを今後はライブラリ内の整理とともに、他のデータに関しても同様に UI 実装を行う。

ユーザーに優しい Web-UI の実装は手軽ではない。いかにライブラリ化を進めようと、結局のところ JavaScript（Ajax）や PHP だけでなく、HTML や XML、CSS、XSLT といった多様な言語を習得しなければならず、とても「お手軽に」とは言い難い。

とはいえ、実験データに限らず、データアクセスには UI が欠かせない。開発者はユーザーにわかりやすく、間違えにくく、簡便なインターフェースを提供する努力を惜しんではならないと思う。

謝辞

最後に本システムの検討、開発に関して惜しみない協力をいただいた全共データベースワーキングの皆様、株式会社キャトルアイ・サイエンスの皆様、ならびに、ご支援いただいた国立情報学研究所の e-science 研究分野振興を支援する CSI 委託事業に謝意を表します。

参考文献

[1]　RCM System Software, http://www.i4s.co.jp/rcm/rcmabs.html