徹底！ＤＲＹな社内業務！

（株）北見コンピューター・ビジネス

■ 執筆者Ｐｒｏｆｉｌｅ ■

谷本　晃一

2007 年　（株）北見コンピューター・ビジネス入社
システム部配属
2009 年　現在　本社システム部 システム開発課
主任プロジェクトリーダー

■ 論文要旨 ■

弊社では、社内業務を改善するべく、従来から行っていたメールベースの作業報告やExcel にて行っていた工数管理を、DRY(Don't Repeat Yourself)原則に基づき RDB(Relational Database)を使用した WEB システムにより、一元管理するようにした。WEB システム開発にはアジャイルな開発と、社内の技術者レベルの向上を狙って、WEB アプリケーションの開発フレームワークである Ruby On Rails による開発を行った。また社内の機密情報を扱うため、PKIによるセキュリティ対策も同時に行った。これらのシステムを開発し、運用することで、従来から行っていた何度も同じようなことを入力していた社内業務を一度入力するだけであとはシステムからその情報を引き出すという形に移行できた。これにより社内業務にかかる工数の削減や、社内技術者のスキルアップを同時に行うことができた。

■ 論文目次 ■

■ 図表一覧 ■

１．はじめに

１．１　当社の概要

株式会社北見コンピューター・ビジネス（以下、当社）は、平成 9 年 5 月に北海道北見市を拠点として設立した。平成 18 年 4 月には北海道札幌市に支店を開設し、現在（平成 21 年 4 月時点）は、従業員 25 名の会社である。

当社では、システム開発をはじめ、パッケージソフト及びハードウェアの販売、ネットワーク事業、テクニカルサポート、システム運用保守などトータルサービスを展開し、富士通パートナー認定及びマイクロソフトパートナー認定を受け、さまざまな業種のユーザーへそのサービスを提供している。

また顧客満足度向上を目指した CS 活動への取り組みや個人情報保護の観点から、個人情報保護方針を定め、平成 17 年 8 月に日本情報処理開発協会(JIPDEC)より「プライバシーマーク」を取得し、情報処理サービス業としてのコンプライアンス経営にも取り組んでいる。

近年では、社内の技術レベルを促進することや社員の意欲向上を狙った有志によるシステム開発も行っている。今回紹介するシステムについてもこの活動の中で行ったものであり、仮想化の技術やアジャイルなソフトウェア開発で近年脚光を浴びている　Ruby On Rails などの新技術を取り入れて開発したものである。

１．２　社内システム開発の背景とシステムの特徴

今回、この社内システムを開発することとなった背景としては、最近のトレンド技術を使ったシステム開発をしてみたいという、私自身の〝欲望〟のようなものが大きい。また、これからのシステム開発を担う若年層の社員達に、今自分たちが開発しているシステム以外にも、こんな技術もあるということに興味を持ってもらいたい、さらに欲を言うなら興味を持った上で、システム開発に協力してほしいといった思いもあった。そのようなことを考えている中で、どのようなシステムを作ろうかと試行錯誤していたところ、せっかくなら社内で役に立つシステムを構築しようと考え、冗長な社内業務へと白羽の矢が立ったわけである。当社では、社内業務として日々の作業報告を E メールにて行っているほか、月に一度、それをまとめた報告書のようなものを Excel シートで作成し、それを印刷し提出している。つまり毎日作業報告を行っているにも関わらず、それをまとめるようなことも月に一度行わなければならない。このような冗長な社内業務を改善したのが今回構築した社内システムである。

２．社内業務の問題点とその改善方法

２．１　社内業務の問題点

当社では、従来より社員の日常業務を把握するため管理者に対して、毎日作業報告を E メールにより送信すること、また月に一度、勤怠管理面において勤怠管理表と作業報告書というものを作成し、提出することが義務付けられている。

日々の作業報告の E メール及び月に一度の勤怠管理表と作業報告書にて報告する内容を表 1 に記す。

	デイリー	マンスリー	
報告媒体	Eメール	勤務管理表(Excelファイル)	作業報告書(Excelファイル)
報告事項	始業時間	始業時間	作業案件毎の作業内容と、その工数
	終業時間	終業時間	
	作業時間合計	作業時間合計	
	残業時間	残業時間	
	作業案件毎の作業内容と、その工数		

表１　デイリー／マンスリーにおける報告事項の比較

上記に示した通り、E メールの内容を一ヶ月分まとめたものが、月に一度の報告となっており、既に報告しているものをもう一度報告という図 1 に示すような業務フローになっている。このような冗長な業務フローは「DRY(※1)ではない」。これを改善しようと考えた。ここが、このシステムの出発点である。

※1：「Don't Repeat Yourself」の略で、「同じことを繰り返さないでください」の意。

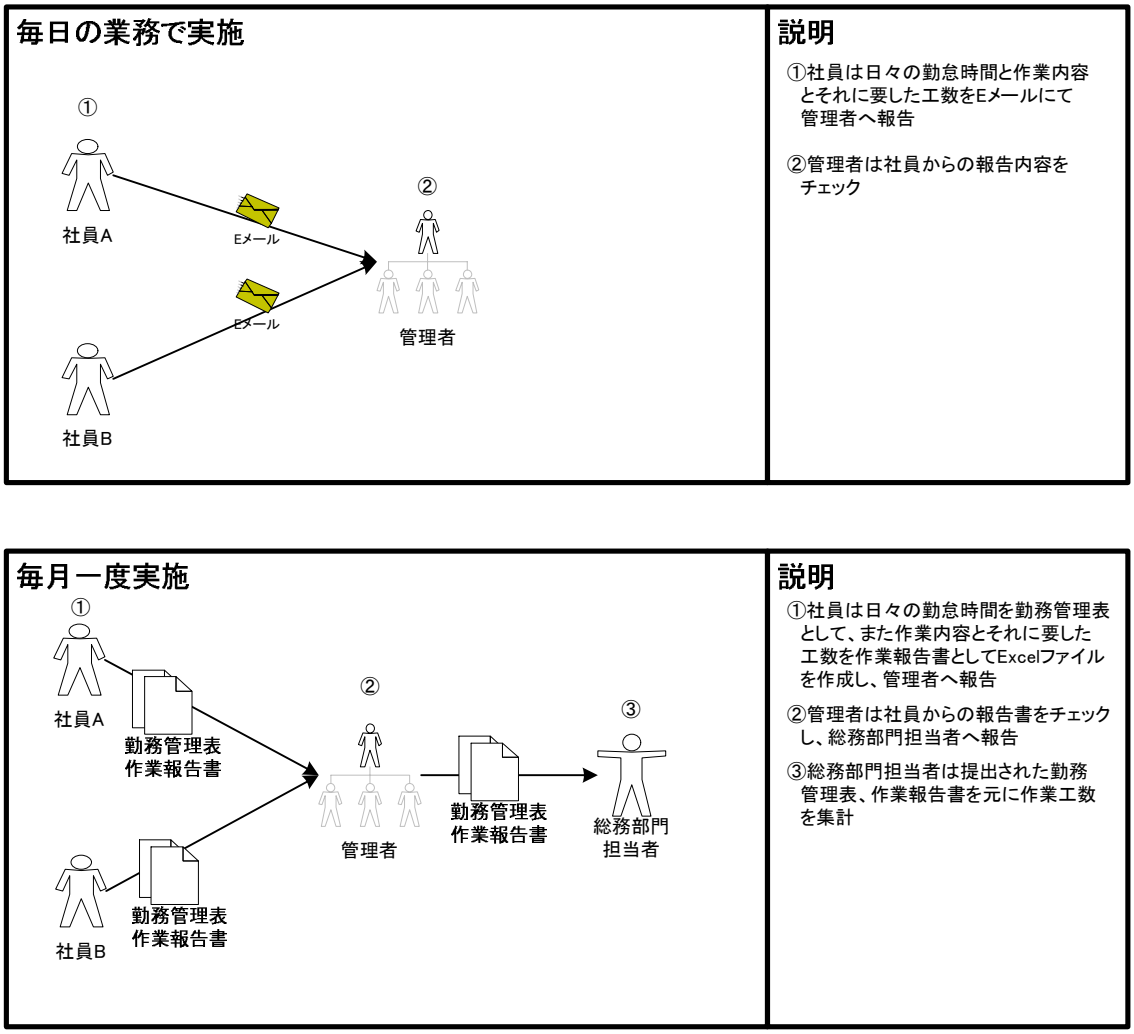


図１　改善前の社内業務フロー

２．２　社内業務の改善方法

今回構築しようと考えたこのシステムの使命は、前述した通り、「同じことを何度もしない」ことだ。つまりユーザーが情報を入力するのは初回のみであり、あとはシステムに登録されたデータを抽出して、メール送信やデータチェック、さらにはデータ集計をできることが求められる。以下に本システムの要求仕様を記す。

① ユーザーは毎日データを入力することが可能である。入力する項目は以下である。
- 始業時間
- 終業時間
- 作業時間合計
- 残業時間
- 作業案件毎の作業内容と、その工数

② システムはユーザーが入力したデータを元にＥメールを作業報告として送信することが可能である。

③ ユーザーは月に一度、勤務管理表、及び作業報告書を出力し、ユーザーの管理者への審査依頼が可能である。

④ システムはユーザーからの審査依頼を元にユーザーの管理者へ審査依頼があったことをＥメールで通知することが可能である。

⑤ 管理者はシステムからの審査依頼のＥメールをもとに勤務管理表、及び作業報告書の審査が可能である。

⑥ 管理者は審査済みの勤務管理表、及び作業報告書を総務部門担当者への承認依頼が可能である。

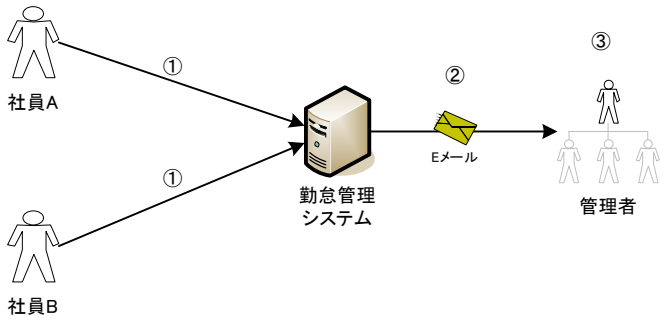
⑦ システムは管理者からの承認依頼を元に総務部門担当者へ承認依頼があったことをＥメールで通知することが可能である。

⑧ 総務部門担当者はシステムからの承認依頼のＥメールをもとに勤務管理表、及び作業報告書の承認が可能である。

⑨ ユーザーが入力した情報を集計することが可能である。

上記の要求仕様を満たした、社内業務フローを図２に記す。

毎日の業務で実施



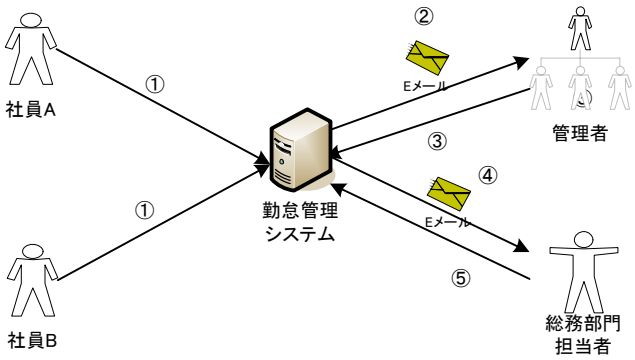
説明

①社員は日々の勤怠時間と作業内容とそれに要した工数をシステムへ登録

②システムは登録された内容をEメールに変換して、管理者へ送信

③管理者はシステムから送信されたメールをチェック

毎月一度実施



説明

①社員は日々入力したデータを元にシステムに審査依頼

②システムは社員からの審査依頼を元に管理者へ審査依頼メールを送信

③管理者はシステムからのEメールを元に社員の勤務状況を確認しシステムに承認依頼

④システムは管理者からの承認依頼を元に総務部門担当者へ承認依頼メールを送信

⑤総務部門担当者は承認依頼メールを元に勤怠状況を承認し、全ての社員の承認が完了したところで、システムへ工数集計を要求し、結果取得

図２　改善後の社内業務フロー

３．システム構築

３．１　システム開発

本章では、今回構築したシステムの開発手順、手法やシステムのインフラ構築について紹介する。

３．１．１　システム開発のインフラ

まず、システムを開発する上で必要となってくるインフラ部分について紹介する。まず複数人でのシステム開発には必須のソース管理システムには SVN(Sub Version)を使用した。社内サーバに公開リポジトリを設置し、通信には SSL を使用し社外にいるメンバーでも開発への参加を可能にした。

また、最近のシステム開発では、ほぼ必須となってきている BTS(Bug Tracking System)についてだが、今回のシステム開発で使用したのは Ruby On Rails 製の Redmine(http://redmine.jp/)というシステムである。このシステムでは、 機能やバグ単位でチケットを発行し、そのチケット毎に進捗を確認できる。また wiki や掲示板などもあり、情報共有をここで行うことが可能となる。

これらのインフラを業務で使用して、今回の開発を行った。冒頭にも記したが、今回の開発は自己啓発によるものであり、誰かがプロジェクト管理をするというような方式はとっていない。各メンバーが機能やバグを BTS 上で提起し、それを BTS 上で討議していくという形式で作業を行った。

３．１．２　Ruby On Rails による開発

今回の WEB システム開発で使用した WEB アプリケーション開発フレームワークである Ruby On Rails が、最近の Web システムの開発で脚光を浴びているのは、もはや言うまでもない。インターネットなどではよく「１０分でできる Rails アプリ」のようなものが紹介されているが、それほどに開発効率は高い。今回のシステム開発は冒頭に記したように、有志によるシステム開発であり、あまり工数をかけられない。このため、アジャイルなソフトウェア開発が可能といわれる Ruby On Rails を使用して開発することにした。しかし、当社における、プログラム言語 Ruby に対する有識者は１、２名程度でその他の社員については C、Java、VB などの技術者が多い。このような背景からシステム開発と並行して、Ruby の学習をしていくことが必要であった。

（１）Ruby の特徴

Ruby という言語は、まつもとゆきひろ氏が開発した言語であり、その歴史はまだ１０年程度の新しい言語だ。その特色を簡単に紹介する。

（ｉ）ダックタイピング

まず一つ目として Ruby は C や Java で行う変数を定義したときに型を指定する必要はない。ダックタイピングという考え方であり、そのオブジェクトの振る舞い（メソッド）をみて、そのオブジェクトがなにかがわかるという考え方である。それをアヒルを例にした

ものだ。つまり「“ガー”と鳴く鳥がそこにいたのなら、それはアヒル」なのだ。

（ⅱ）クロージャ

クロージャは C 言語における関数ポインタに似たものであるが、その実、構文や変数のスコープの考え方が全く異なる。以下にクロージャを使用したプログラムの例を記す。

```
array = [1,2,3]
array.each do |element|
  puts element
end

[結果]
1
2
3
```

上記では array オブジェクトの each メソッドに対して do～end 間のブロックを渡しており each メソッド内では array オブジェクトの全要素に対して、do～end のブロックを実行している。このようにコードが非常に直観的であると感じないだろうか。「配列の各要素に対してそれを出力してください」という、思ったことがそのままコードになっていると感じるのは私だけだろうか。

（ⅲ）オープンクラス

最後はオープンクラスという考え方だ。オープンクラスとは、クラスが動的に変化していくというものである。Java の考え方ではクラスは静的なものであり、クラスにメソッドを動的に追加していくことはできない。しかし、Ruby ではそれが可能である。あるクラスに途中からメソッドを増やすこともできるし、クラスではなくインスタンスにも増やすことができる。この考え方は非常に強力であり、メタプログラミングに最適だ。メタプログラミングとはあたかも「プログラムがプログラムを書く」ような振る舞いをするものである。簡単な例をあげると Ruby には attr_accesor というメソッドが存在する。このメソッドはそのクラス内のインスタンス変数への writer および reader メソッドを追加するというものである。　例にあげると Foo クラスのインスタンス変数 foo に対する　writer および reader メソッドを考えた場合、以下のようになる。

```
class Foo
  def foo=(argv)
    @foo = argv
  end

  def foo
    @foo
  end
end
```

これを以下のように置き換えることが可能だ。

```
class Foo
  attr_accesor :foo
end
```

上記のように attr_accesor メソッドをコールしたことでそのクラス内にメソッドが二個追加されることになる。

このように、Ruby の特色は当社の開発者が扱ってきた他の言語のそれとは若干異なった性質を持っており、開発する上で、Ruby に関する知識を向上させていくことも一つの課題であった。このため、社内で勉強会等を実施することで知識の向上を図るほか、先に紹介した BTS 上の掲示板で Ruby に関する討議の場を設け、そこで初心者の質問に有識者が回答したり、より良いコードの書き方などを討議したり、と有効活用することで当社の技術者のスキルの底上げを行った。

（2）ライブラリ

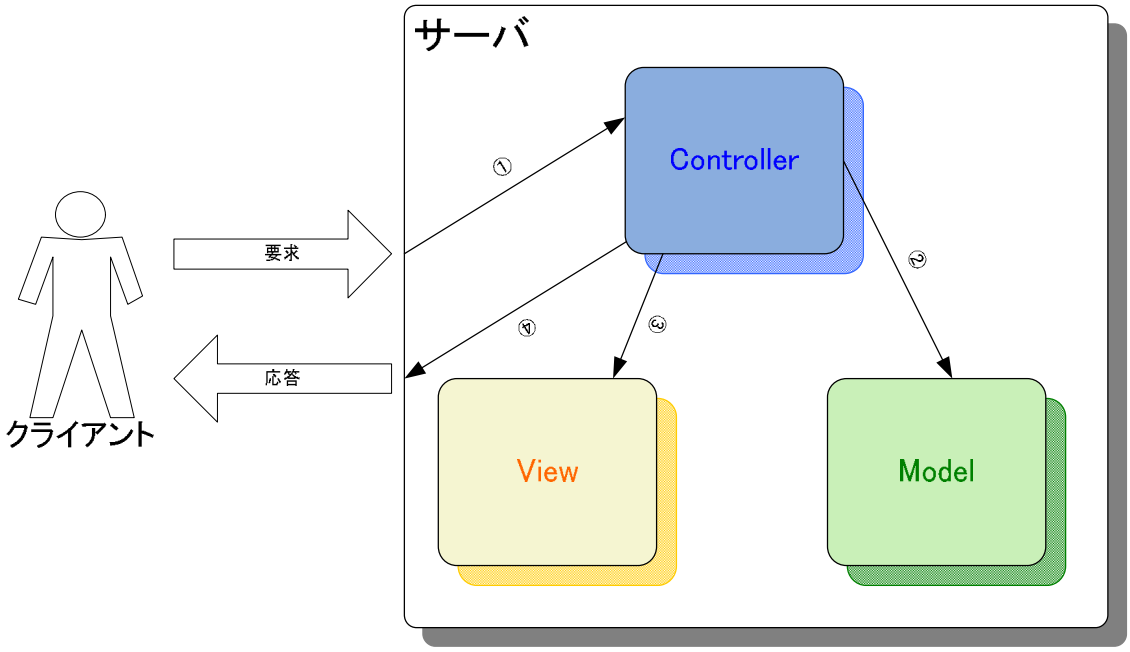
システム開発における、工数を短縮する手段としてライブラリを使用することが一般的である。Ruby 自体にも組み込みライブラリが多く含まれているが、Ruby On Rails も RDB へのアクセスに使用する ActiveRecord や、E-Mail を使用するための ActionMailer 等のライブラリを統合したようなフレームワークとなっている。今回は開発効率をより向上させるため、以下のようなライブラリを使用している。

- restful_authentication
- spreadsheet

restful_authentication はライブラリというよりは Ruby On Rails のプラグインであるが、これはユーザー認証およびユーザー毎のセッション管理を行うことができるものである。これを使用することでユーザー認証、セッション管理の開発を容易に行うことが可能だ。また spreadsheet は ruby から Excel ファイルを操作することが可能なライブラリである。これはユーザーが入力したデータを Excel ファイルとして出力する際に使用している。

（3）MVC モデル設計

Ruby On Rails では MVC モデルに基づいた設計となっている。MVC モデルとは Model と View と Controller を示す。概要としては図 3 に示すように、Model がデータであり、それを表示する View があり、View と Model の間に Controller がある。ユーザーからの要求をまず Controller が受け取り、必要な情報を Model から取得し、結果を View を使用して返すというものである。



①クライアントからの要求を受け付ける。
②クライアントの要求に応じたデータをModelから取得する。
③取得したデータからViewを使って、応答データを加工する。
④応答データを返却する。

図３ MVC モデル概要

Ruby On Rails においては、ディレクトリ構成がそのまま MVC モデルを示している。たとえば User というモデルについて考えた時、基本的には以下のようなファイル構成となる。

app/models/user.rb.....Model
app/view/users/edit.html.erb.....編集用の View
app/view/users/index.html.erb.....一覧表示用の View
app/view/users/new.html.erb.....新規登録用の View
app/view/users/show.html.erb.....表示用の View
app/controllers/users_controller.rb.....Controller

たとえばユーザー一覧の画面を取得するとき、クライアントは以下のような URL に対して Get 要求を行う。

http://example.co.jp/users/

この時 Rails アプリケーションではまず、controller の index メソッドを呼ぶ。Index メソッドでは Model からユーザーの一覧を取得し、index の view を使用してクライアントへ応答を返す。以下に実装例を用いて説明する。

controllers/users_controller.rb

```
class UsersController < ActionController::Base
  def index
    @users = User.all
    respond_to do |format|
      format.html # index.html.erb
      format.xml  { render :xml => @users }
    end
  end
end
```

上記の index メソッドがコールされたとき User.all によって User モデルに登録されたすべての User の配列情報が”users に設定され、それを index.html.erb に渡している。

models/user.rb

```
class User < ActiveRecord::Base
end
```

モデルは ActiveRecord を継承しており、モデルの属性へのアクセス用のメソッドは定義する必要はない。

views/users/index.html.erb

```
<table>
  <tr>
    <th>Name</th>
  </tr>

<% @users.each do |user| %>
  <tr>
    <td><%=h user.name %></td>
  </tr>
<% end %>
</table>
```

上記のように view は html へ ruby コードを埋め込む書式で記述する。上記の例ではテーブルにユーザー名の一覧を表示するような HTML を作成してクライアントにその結果を返す。

このように MVC モデルが体系化されておりコーディングが非常にしやすい。この構成さえわかってしまえば、実装は非常に単純明快である。今回のシステム開発においては必要となる Model とその関連は図 4 に記した通りであり、これらで示す Model 全てにおいて、view,controller を実装した。

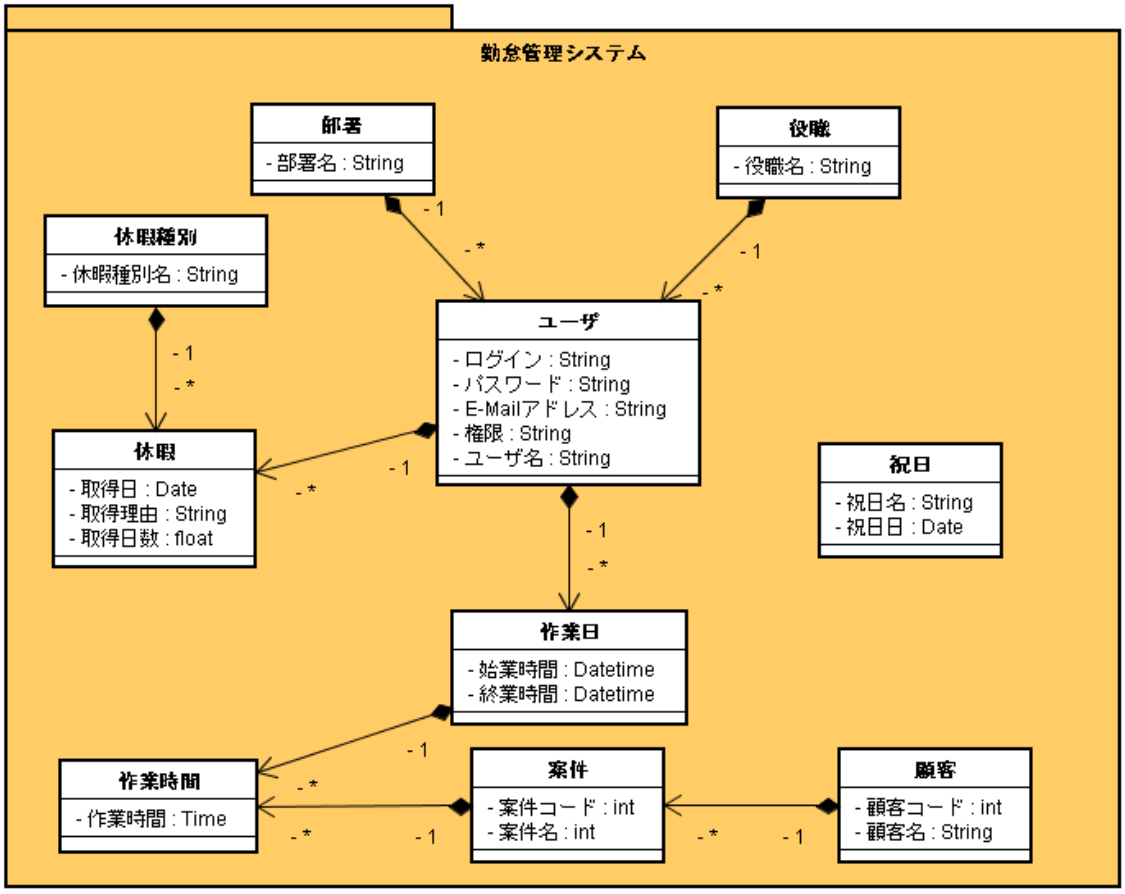


図 4 勤怠管理システムにおけるモデル図

（4）RDB のセキュリティ

RDB を使用した WEB アプリケーションの脆弱性としてよくあげられるのは SQL インジェクションである。Ruby On Rails では RDB を扱うライブラリ ActiveRecord が内部的にサニタイジングを行ってくれるため、基本的にユーザーが考慮する必要はない。ただし、以下のように構文の使用方法を誤ると、これが適用されないため、コーディングする際には注意が必要だ。

サニタイジング行われない実装例

```
User.find(:all , :conditions => “name =” + params[:name] )
```

サニタイジング行われる実装例

```
User.find(:all , :conditions => [“name = ?” + params[:name]])
User.find(:all , :conditions => [“name = :name”, { :name => params[:name] }])
User.find_ all_ by_ name( params[:name] )
```

（5）外部データの取り込み

本システムでは祝日情報を取得する必要があった。祝日は固定の日付のものもあれば、第 3 週の月曜日のようなものも存在する。また祝日が追加や変更されたりもするため、動

的に変更できるような３．１．２　（３）で示したようなモデルが必要だった。しかし、これを全て手作業で入力していくのは手間である。このため、本システムでは外部の Google の祝日情報を取り込むことにした。Google では URL にパラメータ（開始日、終了日、取得形式など）を設定して HTTP GET メソッドで要求すると指定したファイル形式で、そのカレンダー情報を取得することができる。今回は ATOM 形式で要求することで、その結果を取得し、XML パーサで解析して、祝日情報を取得し RDB に登録する。これによって、手作業による祝日登録作業は不要となった。

３．２　インフラ構築

今回のシステムを構築するに当たり、そのインフラ構築について本章にて記述する。システムインフラ、セキュリティインフラについてそれぞれの構築手段を記述する。

３．２．１　システムインフラ

まず、今回のシステムを導入するべくサーバを構築しなければならなかった。サーバを構築するにはハードが必要だが、近年では仮想化という技術が革新的に発展してきており、今回のシステムを構築するにあたり、この技術を取り入れることとした。仮想化のメリットは何と言ってもコストダウンだろう。既存のハードの稼働率が低い場合にはうってつけの技術といえる。当社で管理しているサーバマシンのうち業務用に使用している WEB サーバが存在する。このマシンの稼働率は全社員 25 名程度が業務で使用する程度で、ハード的には遊んでいる容量が多い。このような状態にあるサーバを遊ばせておくのは、最近流行りの「エコ」の観点からいっても非常に不経済である。このような背景があって、今回の仮想化技術の導入である。仮想化技術は MicroSoft 社の Virtual Machine や　VMware 社の VmwarePlayer が有名だが、今回は後者の VmwarePlayer を使用した。このようにハードを新設せずに仮想環境を使用することでコストダウンを実現することが可能となった。

VmwarePlayer 上で動作させる OS は Ubuntu Linux のサーバ版を使用した。その他のサーバアプリケーションとして WEB サーバやメールサーバ等が必要となるが、以下の表に使用したアプリケーションのバージョンを記す。

アプリケーション	バージョン	備考
OS（Ubuntu サーバ版）	8.04	
Apache	2.2.8	WEB サーバ用
Passenger	2.2.2	Ruby On Rails アプリを apache 上で動作させるためのアプリケーション
Postfix	2.5.1	メール送信サーバ用
Mysql	5.0.51a	データベース
OpenSSH	4.7p1	SSH サーバ
OpenSSL	0.9.8g	証明書作成に使用

表２　サーバアプリケーション一覧

３．２．２　セキュリティインフラ

今回のシステムでは、社員が作業中の案件にかかわる情報、つまりは顧客情報を扱うことになっている。このような社外秘の情報を WEB アプリケーションで使用する場合、暗号化などのセキュリティ技術は必要不可欠であるといえるだろう。WEB アプリケーションでの暗号化及びサーバ/クライアント認証をすべてサポートしているプロトコルとして SSL/TLS があり、当然のごとくこれをシステムに導入した。SSL/TLS は PKI(Public Key Infrastructure) という技術の元に成り立つプロトコルであり、第三者における認証が行われる。ここでいう第三者とは認証局（以下 CA とする）である。本システムでは、CA を設置し、サーバ証明書、クライアント証明書の作成、それぞれの証明書に対する CA からの署名を行い、サーバ、クライアントに対して、それらを配布し使用することでサーバ、クライアント認証による「なりすまし」の防止、データ暗号化による「盗聴」の防止を行うことが可能となった。 また、WEB サーバについてはパケットフィルタリングによるネットワークセキュリティの強化も実施した。

（1）PKI

当社にて設立した CA はシステムを利用するのは社員のみであり、社員の認証のみを目的としているため、プライベート CA として設立している。プライベート CA であるため、パブリック CA で制定しなければならない CP（証明書ポリシ）や CPS（証明書運用規定）は策定していない。今回、証明書を作成するにあたっては OpenSSL を使用している。

まず CA の秘密鍵および公開鍵証明書を作成した。次にサーバを認証するためにサーバの秘密鍵、公開鍵証明書を作成し、証明書は CA の秘密鍵によるディジタル署名を付加したディジタル証明書を作成した。このディジタル署名を CA の発行している証明書で複合したものを検証することで第三者による認証が可能となる。サーバ同様にクライアントに対しても秘密鍵、公開鍵証明書を発行する。クライアントへの秘密鍵、公開鍵証明書の配布にはファイル形式を PKCS#12 規格を使用してパスワードによるセキュリティをかけて配布を行った。パスワードについては証明書を配布するとは別の方法でユーザーに通知することでよりセキュリティを高めている。

図 5 に当社の本システムにおける PKI のセキュリティモデルを記す。

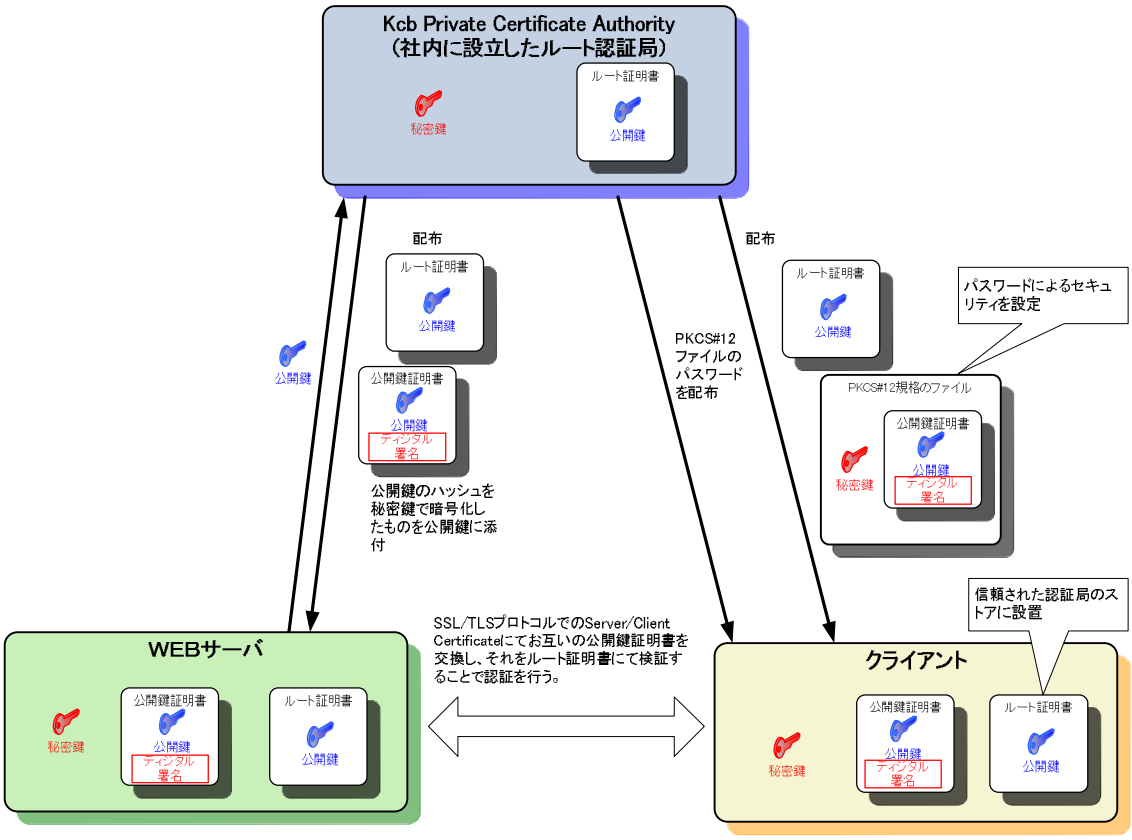
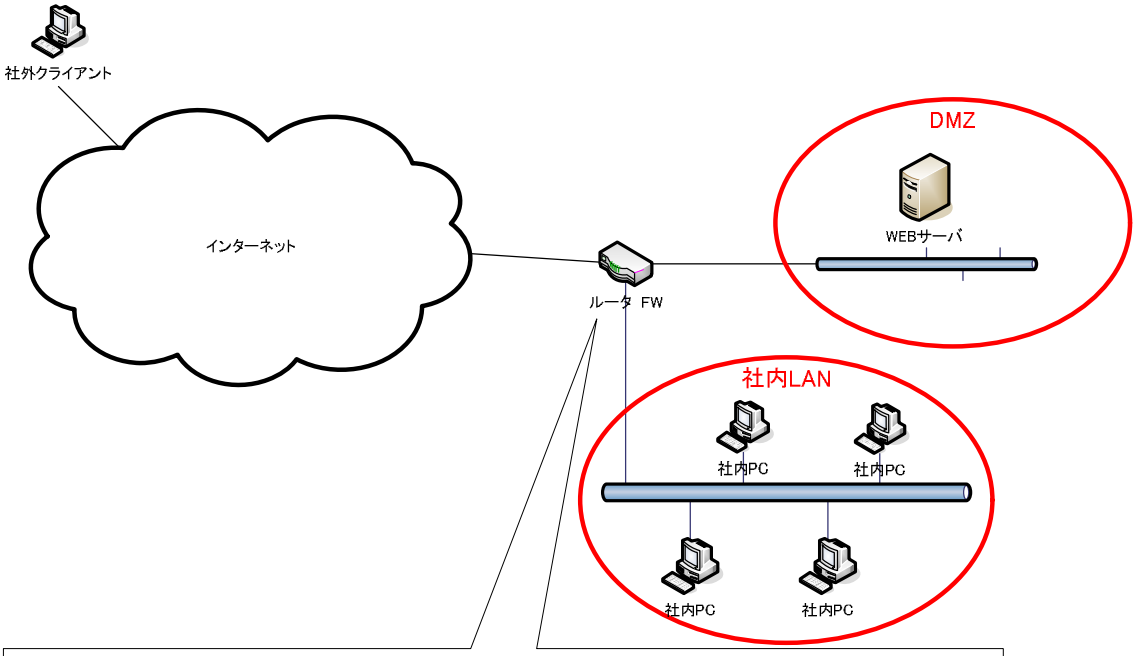


図５　勤怠管理システムにおける PKI セキュリティモデル

（２）ネットワークセキュリティ

今回設立した WEB サーバのネットワークセキュリティについて紹介する。基本的にネットワーク上のセキュリティ対策としては古くからファイアウォールによるものがある。ファイアウォールについては、アクセスコントロールリスト（以下 ACL）を事前にユーザーが設定するスタティックパケットフィルタ型、ネットワークセッションにより一時的なフィルタリングを行うダイナミックパケットフィルタ型、さらにはギリシャの Check Point 社が開発したダイナミックパケットフィルタリングに加えてアプリケーション毎の通信フロー状態などによるフィルタも行うステートフルインスペクション型などがあり、後者にいくほど、そのセキュリティは高くなるが、その分ファイアウォール自体のコストは高くなる。

今回当社におけるファイアウォールではコスト面からスタティックパケットフィルタ型を適用している。このときファイアウォールに設定した ACL について図 6 に記す。



FWのACLは以下のように設定

優先順位 高 ↑ 低

No.	送信元	送信先	アクション	備考
1	社内LAN:*	DMZ:7	許可	ICMPメンテナンス用(echo-request/echo-reply のみ)
2	社内LAN:*	DMZ:22	許可	sshメンテナンス用
3	社内LAN:*	DMZ:443	許可	SSL/TLS用
4	インターネット:*	DMZ:443	許可	SSL/TLS用
5	社内LAN:*	DMZ:*	拒否	基本的にはすべて拒否
6	インターネット:*	DMZ:*	拒否	基本的にはすべて拒否
7	インターネット:*	社内LAN:*	拒否	基本的にはすべて拒否

図６　WEB サーバネットワーク構成

４．導入効果

４．１　社内業務の改善結果

本システムを導入したことによる、効果を考察する。まず工数面に関してだが、一般社員については、日々のデータ入力については、いままで実施していた工数と変わらない。ただし、月に一度の報告提出にかかる工数分が変わってくる。これにかかっていた工数を一人当たり 0.5h 程度として一ヶ月当たり 0.5×20 = 10.0 h 程度の工数短縮である。これ以外に、総務部門で実施していた集計作業が大幅に短縮される。これが約 6.0 h 程度の工数短縮として一ヶ月あたりの工数短縮は 2 人日程度となる。年間で考えた場合には 1 人月程度となり、効果大とまでは言えないが、ある程度の工数短縮である。

次の効果として、今回入力データをシステム側でチェックを行うことにしたことによる入力ミスが格段に減ったことだ。これにより管理者のチェック時間、さらには各担当者の手戻りによる再提出時間も減少していると思われる。

最後の効果としてはコスト面というよりは、社員の精神面であるが同じ作業を何度も行うという理不尽な要求からのストレスを感じなくてすむようになった。

４．２　社員の意欲・スキル向上

今回のシステムは冒頭にも述べたとおり、有志によるシステム開発である。私自信がまずシステムを提案し、自由参加制で４名程度による共同開発を行った。今回の開発においては、Ruby の実装スキルを得る以外にもセキュリティを考慮したシステムインフラの構築による PKI の技術やネットワークインフラの技術など多岐にわたる技術を必要とした。PKI のインフラ構築では、OpenSSL のコマンド実行をするだけで秘密鍵や証明書が簡単に作成されてしまい、その中で実際になにをやっているのかがわからないブラックボックスのような部分も存在していたが、このような不明点は BTS 内の課題としてあげたり勉強会を開いたりすることで解消していった。ネットワークのパケットフィルタリングの仕組みについても同様に設定だけは WEB 画面から簡単にできてしまうが、その中身の TCP/IP の基礎技術については別途勉強していくことで、その中身までを理解するように努めた。

このような活動をメンバ全員で実施し、情報共有していくことでメンバ全員の大幅なスキルアップができた。また、自発的に開発をすること、そしてお互いによりよいものを作ろうという意識の元に検討しあって開発を行うことで開発意欲も非常に高まった。今回のシステム開発にて得た知識やスキルは次のシステム開発でも確実に役立つものであり、次のシステム開発が楽しみである。

今回のシステム開発にて、得た効果としては社内業務の効率化よりは、こちらのスキル向上と意欲向上が大きかったと私は感じている。

５．今後の課題

まず今回のシステムについてだが、社内の勤怠管理にのみ焦点を当てて、開発したものである。今回の開発によってインフラの構築は、ほとんどできたので今後は勤怠管理以外にも機能追加をしていきたいと考えている。勤怠管理以外にも、その他の社内手続きのシステム化（出張申請など）やスケジュール管理など、最終的には統合的なグループウェアのようなものに成長させていきたいと考えている。

システムインフラ面においては、上記に記した通り、今回の開発で概ね完了しているが、今後は技術的な興味から冗長化や、ネットワークセキュリティの強化（DOS 攻撃への対策など）も行っていきたいと考えている。

また、導入効果で述べたような社員の意欲向上、スキル向上も今後も常に課題としていきたい。今回は有志による開発であったため、基本的に自由参加で開発を行った。このため、意欲が向上するのは必然であったと言えるかもしれない。今回開発に参加しなかった、他の社員についても、今回のシステムを見て興味を持つなり、これに触発されて別のフレームワークや言語を使用した違うシステムを開発するなど、技術者の社員全員がお互いに技術力をぶつけあって、お互いが成長していけるようなスキルアップモデルができると非常にうれしい。そのような体制が確立できるように、現在は有志によって行っている活動を社内の教育の一環として、取り込んでいくことが今後の課題である。

６．おわりに

今回紹介したシステムは、規模的には非常に小さく簡単なシステム構築の事例である。

冒頭にも述べたとおり、今回のシステム開発は社内業務の改善をするとともに、私自身や当社の社員のスキルアップを目指して、開発したものである。

社内業務に限った話ではないが、業務で同じことを繰り返していないか？DRY な業務ができているか？これを常に考え改善していくことで DRY な業務を確立していくべく今後も業務改善活動を行っていきたい。

また、スキルアップの面においては、常に新しい技術が日々産み出されるなか、我々技術者はそれらの技術に着目していかなければならない。そうすることが自信のスキルアップにつながるし、自信がスキルアップするということは、それだけ今後の業務遂行が楽になるということである。

DRY な業務もスキルアップも結局のところ、最終的には工数の削減につながり、その結果、余った時間を、他のことに使えるようになる。それを家族のために使ってもよいし、自分の趣味に使ってもよい。そのために、日々業務改善およびスキルアップを意識的に実施していきたい。