Beam me up, Scotty!
Rails for the “Enterprise”

Charles Johnson
Director, IS Applications
Centerstone

Rick Bradley
Project Manager
Centerstone
What Enterprise?
What Enterprise?

- Enterprise system?
What Enterprise?

• Enterprise system?

• Enterprise software?
What Enterprise?

• Enterprise system?

• Enterprise software?

• Enterprise platform?
Centerstone Enterprise
Centerstone Enterprise

• 25 Counties
Centerstone Enterprise

- 25 Counties
- 150 Locations
Centerstone Enterprise

- 25 Counties
- 40,000 Clients
- 150 Locations
Centerstone Enterprise

- 25 Counties
- 40,000 Clients
- 150 Locations
- 1,100 Staff
Centerstone Enterprise

- 25 Counties
- 40,000 Clients
- Comprehensive Electronic Record

- 150 Locations
- 1,100 Staff
Centerstone Enterprise

Requirements

Dimension
Centerstone Enterprise

Requirements

Available & Reliable

Dimension
Centerstone Enterprise

Requirements

Available & Reliable

Dimension

• All the time
Centerstone Enterprise

Requirements

Available & Reliable

Dimension

• All the time

Accessible
Centerstone Enterprise

Requirements

Available & Reliable

Dimension

• All the time

Accessible

• Everywhere
Centerstone Enterprise

Requirements

Available & Reliable

Accessible

Scalable

Dimension

• All the time

• Everywhere
Centerstone Enterprise

Requirements

Available & Reliable

Accessible

Scalable

Dimension

• All the time

• Everywhere

• 300 to 9000 users
<table>
<thead>
<tr>
<th>Requirements</th>
<th>Dimension</th>
</tr>
</thead>
</table>

Centerstone Enterprise
Centerstone Enterprise

Requirements

Maintenance

Dimension

Maintenance
Centerstone Enterprise

Requirements

Maintenance

Dimension

• Invisible
Centerstone Enterprise

Requirements

Maintenance

Dimension

• Invisible

Management
Centerstone Enterprise

Requirements

Maintenance

Management

Dimension

• Invisible

• Central
Centerstone Enterprise

Requirements

- Maintenance
- Management
- Agnostic

Dimension

- Invisible
- Central
Centerstone Enterprise

Requirements

- Maintenance
- Management
- Agnostic

Dimension

- Invisible
- Central
- Hardware
<table>
<thead>
<tr>
<th>Requirements</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintenance</td>
<td>• Invisible</td>
</tr>
<tr>
<td>Management</td>
<td>• Central</td>
</tr>
<tr>
<td>Agnostic</td>
<td>• Hardware</td>
</tr>
<tr>
<td></td>
<td>• Database</td>
</tr>
</tbody>
</table>
“Enterprise”
“Big”
Bureaucratic
SOA
Vendor-driven definition -- the true source of the term "Enterprise".
Java, XML, RDF

... EJB3.0, J2EE, JUnit

... Hibernate + XDoclet

... n-phase commit

... RMI, XML-RPC, WSDL

... JMX, JAXL, SASL, CORBA

... WOTA, FIXN, JOKE
Aggressive!

100 Million Dollars Pissed Away...
What do you want to do?
- Go for 220!
- Hire more consultants?
- Give Up...

-- with sincerest apologies to the creators of “Bunk and Rambling”...
Inverted

We don’t use the Internet to track down a market for our product.

We have a fixed market for whom we build a new product.
Existing

Unlike “green field” applications, “Enterprise” applications fight the ills of the organization at least as much as technical hurdles.
Stubborn

Changing business processes (even horrid ones) can be very difficult.
Rick’s First Rule of Enterpriseyness:

Any sufficiently large system left at rest in any sufficiently @#$%!-ed up environment will decay into a stack of electronic post-it notes.
Start
small team
isolated offices
open source
open minds
Big Design Up Front
non-profit
for-profit
“collaborator”
web + DB experience
Java + Oracle
support from the Penthouse
widely distributed "customers"
camaraderie
bad legacy habits...
- no unit tests, no functional tests, no automated tests, no automated build.
Goal
agility

- small milestones,
- story-driven design,
- refactoring,
- pairing, few meetings, ...
in some ways we have to be careful not to end up BDUF. This can be meshed with "Getting Real" and TDD/BDD if done carefully. "Shared Domain Language" is critical. Avoid "too much talk, too little code."
solid testing
reproducibility

continous integration,
one-word builds,
automated deployments,
automated upgrades and
rollbacks, nightly data
conversions, etc.
scalability & portability

“Share Nothing” horizontal scalability, lightweight software, fast commodity hardware; database agnosticism, browser agnosticism.
+++!!!

launch bonus: a basket full of psychotropics.
legacy system

250,000+ lines of embedded “thedaywtf”-compliant Oracle PLSQL code
java system

JBoss, Struts, JSP, Hibernate, EJB3 (draft), JUnit, Ant, CruiseControl, ...
rails transition

Major healthcare application switches from J2EE to Rails

Posted by admin October 11, 2005 @ 09:56PM

Rick Bradley shares a great case study on how his team replaced a partial J2EE solution that wasn't moving the team forward fast enough with Rails. Result? A 20:1 reduction in the amount of code needed to solve the problem.

And this is not Yet Another Blog, or even those luxury todo lists we do at 37signals, but a healthcare application that has to play in the regulated world of HIPAA, Sarbanes-Oxley, drug trial requirements, and all that other heavy-duty joy.

Rails takes another step deeper into The Enterprise.

rails transition

Java:
10361 lines of Java code
1143 lines of JSP
8082 lines of XML
1267 lines of build configuration
-----------------------------------------
20853 TOTAL lines of stuff

Rails:
494 lines of code (rake stats: 386 LOC)
254 lines of RHTML
75 lines of configuration
0 lines of build configuration
-----------------------------------------
823 TOTAL lines of stuff

20+ : 1 code reduction
25+ : 1 “stuff” reduction
rails transition

8:1 reduction in books
<table>
<thead>
<tr>
<th>Name</th>
<th>Lines</th>
<th>LOC</th>
<th>Classes</th>
<th>Methods</th>
<th>M/C</th>
<th>LOC/M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helpers</td>
<td>385</td>
<td>227</td>
<td>0</td>
<td>29</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Controllers</td>
<td>989</td>
<td>593</td>
<td>19</td>
<td>64</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Components</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Functional tests</td>
<td>1110</td>
<td>815</td>
<td>31</td>
<td>117</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Models</td>
<td>1418</td>
<td>508</td>
<td>56</td>
<td>47</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>Unit tests</td>
<td>2218</td>
<td>1407</td>
<td>49</td>
<td>213</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Libraries</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Integration tests</td>
<td>42</td>
<td>33</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>6162</td>
<td>3583</td>
<td>157</td>
<td>475</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Code LOC: 1328  Test LOC: 2255  Code to Test Ratio: 1:1.7
domain
driven
design
analysis patterns

party, accountability, observation, phenomenon, measurement, plan, action, specification, time ranges, ...
try to burn a little time here... talk about how finding “Analysis Patterns” was such a useful discovery, having gotten to meet Martin was a real boon to us, even exchanged some emails. Keeping Martin Fowler at the core of the system ended up being really beneficial to us.
database
“agnosticism”

...any database, as long as it’s Oracle or Postgres

to paraphrase Henry Ford: ...; Actually we may support SQLServer. We simply need to have an ACID db, with standard SQL support, with support for constraints and triggers sufficient to perform auditing.
CREATE FUNCTION plpgsql_call_handler() RETURNS language_handler AS
    '$libdir/plpgsql' LANGUAGE C;

CREATE FUNCTION plpgsql_validator(oid) RETURNS void AS
    '$libdir/plpgsql' LANGUAGE C;

CREATE TRUSTED PROCEDURAL LANGUAGE plpgsql
    HANDLER plpgsql_call_handler
    VALIDATOR plpgsql_validator;

CREATE OR REPLACE FUNCTION get_auth_party() returns integer
    AS '/usr/local/lib/audit', 'get_auth_party'
    LANGUAGE C STRICT;

CREATE OR REPLACE FUNCTION set_auth_party(integer) returns integer
    AS '/usr/local/lib/audit', 'set_auth_party'
    LANGUAGE C STRICT;

create or replace function set_audit_context(p_userid varchar) returns varchar
    as $set_audit_context$
begin
    perform set_auth_party(p_userid);
    return '';
end;

$set_audit_context$ language plpgsql;

create or replace function audit_info() returns varchar as
$audit_info$
declare
    result varchar(200);
begin
    result :=
        '---
        'os_timestamp': |to_char(localtimestamp, 'YYYY-MM-DD HH24:MI')|chr(10)|
        'party_id': |get_auth_party()|chr(10)|
Copyright (c) 2005 Centerstone, All rights reserved.

This program is free software; you can redistribute it and/or modify
it under the terms of the Centerstone Public License as published by
the Centerstone; either version 1 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
Centerstone Public License for more details.

You should have received a copy of the Centerstone Public License
along with this program; if not, write to Centerstone, P.O. Box 40406
Nashville, TN 37204-0406

_filters added to this controller will be run for all controllers in the application.
Likewise, all the methods added will be available for all controllers.

class ApplicationController < ActionController::Base
  include AuthenticatedSystem

before_filter :login_required, :except => [:login]
before_filter :set_defaults, :except => [:login]

def rescue_action_in_public(exception) ...

protected

def set_defaults
  Party.login_user_id = self.current_login.party.id
  ActiveRecord::Base.connection_audit_context = self.current_authenticated_login.party.id
  session[:context] ||= HashWithIndifferentAccess.new
end
end
searchable_models plugin provides the ‘searchable’ DSL as well as a search controller, and a helper to let you put simple searches anywhere for any class.
# A PhenomenonType is a concept about which ChartNote, Measurement, and CategoryObservations can be recorded with a ChartElement for a Client. The PhenomenonType collection forms a Directed Acyclic Graph (DAG) of subtypes and supertypes. An observation of the presence of a subtype implies the presence of the supertypes. An observation of the absence of a supertype implies the absence of the subtypes.

class PhenomenonType < ActiveRecord::Base
  acts_as_dag :in => :subtypes, :out => :supertypes
  validates_uniqueness_of :title
  validates_presence_of :title
end
module ActiveRecord
module Acts #:nodoc:
module DAG #:nodoc:
def self.append_features(base)
super
base.extend(ClassMethods)
end
end

# This act provides the capabilities for a class to behave as a directed acyclic graph.
# The class that has this specified needs to have a <model>_edges tables defined.
module ClassMethods
def acts_as_dag(options = {})
  configuration = {
    :edges_table => "#{self.table_name}_edges",
    :in => :in,
    :out => :out,
  }
  configuration.update(options) if options.is_a?(Hash)
  configuration
end
  class_eval <<-EOV
    include ActiveRecord::Acts::DAG::InstanceMethods
    has_and_belongs_to_many :#{configuration[:in]}, :join_table => "#{configuration[:edges_table]}",
    :foreign_key => 'destination_id',
    :association_foreign_key => 'source_id',
    :class_name => 'self.name'
  end
  has_and_belongs_to_many :#{configuration[:out]}, :join_table => "#{configuration[:edges_table]}",
    :foreign_key => 'source_id',
    :association_foreign_key => 'destination_id',
    :class_name => 'self.name'
EOV
  include ActiveRecord::Acts::DAG::InstanceMethods
end
Stories...

- Using Stories
- Client Summary: Current Meds
- Client Summary: Dosage for Meds
- Client Summary: Name for Meds
- Client Summary: Sources for Meds
- Client Summary Diagnosis
- Client Summary: Client Demographics
- Client Summary: Progress Note Listing
- Client Summary: Medication Allergy List
- Client Summary: Lab List
- Client Summary: Observation Graph
- VNS Note
- Client Creation should warn when duplicate info appears
- Protocol suggests observations to monitor
- Med Note: Client Demographics
- Med Note: Identifying Data
- Med Note: Chief Complaint
- Alter diagnosis from note (or eval)
- Med Note: Chief Complaint (Subjective)
- Alter Medications via the Progress Note
Form mocking
Form mocking

Form Sets
Intake (clinical view -- additions only)
20 nodes view
Intake (Support Staff view)
62 nodes view
Mental Status Exam

- Mood
  - Beer Intake:
    - Way too much
    - Tipple
    - Plenty
    - Not Nearly Enough
    - Bone Dry
  - Caffeine Intake: Amped
  - Hookin' up?:
    - Word
    - Straight up.
    - Cruisin'
    - Nada, hombre.

- Crib
ActiveRecord magic
auditing

Level 0 - No auditing.

Level 1 - Read-only. Audit creation.

Level 2 - No deletion. Audit creation, changes.

Level 3 - Archive all changes and deletions.
# A Credential is the authorization by a Panel for a Staff to deliver services
# at a given Location for a period of time. It is contingent on the Staff maintaining # a certain licensure.
create_table :credentials, :audit => 2, :options => 'tablespace billdata' do |t|
  t.column :licensure_id, :integer
  t.column :staff_id, :integer
  t.column :panel_id, :integer  
  t.column :location_id, :integer
  t.column :begin_date, :datetime
  t.column :end_date, :datetime
  t.column :created, :string, :limit => 200
  t.column :updated, :string, :limit => 200
end

# A Rate is the price a payer will pay for a given service (identified by service) # to be delivered.
create_table :rates, :audit => 2, :options => 'tablespace billdata' do |t|
  t.column :service_code_id, :integer
  t.column :payer_id, :integer
  t.column :begin_date, :datetime
  t.column :end_date, :datetime
  t.column :created, :string, :limit => 200
  t.column :updated, :string, :limit => 200
end

# Actual delivered services. An Activity performed for a Client, by a Staff, at # Location, with a time reference.
create_table :services, :audit => 2, :options => 'tablespace billdata' do |t|
  t.column :activity_id, :integer
  t.column :location_id, :integer
  t.column :staff_id, :integer
  t.column :client_id, :integer
  t.column :begin_date, :datetime
  t.column :end_date, :datetime
end
foreign key
extensions
acts_as_date_range
USPS

county, state, ZIP models
CD-ROM import scripts
Pitfalls
Refusing to use good code simply because it wasn’t written by us. With Ruby sometimes it / is/ faster to write it oneself than use a plugin. Fortunately, as the plugin space has matured this has become less of an issue for us.
feeling compelled to write a bit of functionality. When someone else, two weeks later, releases a plugin for the same thing, feeling compelled to adopt it. Also, overturning homegrown methodologies to do things The Rails Way. Contrast: moving from homegrown Login to Acts_as_authenticated-generated homegrown login with moving from SQL table scripts to Migrations.
In implementing Fowler’s “Party” and “Accountability” patterns, we attempted to use Class Table Inheritance with Rails. Getting inherited behavior, much less cross-database, was a nightmare.
parties in class table inheritance
parties in single table inheritance

Downsides: we still have report writers, outside processes working against database. Makes it difficult to allow access to some identifying data (for example, staff data) without giving access to client data. We were planning on using exports to reporting database anyway, but this requires us to think more carefully about views, etc., to control reporting access.

Attempts at CTI failed -- while possible to get the database semantics correct, synchronizing the separate tables (esp. efficiently, was a nightmare -- esp.
In a number of areas we were able to identify just whose blood was on the bleeding edge.
performance issues, primary key names, performance, adding/removing constraints, testing with constraints, synonyms, Windows-only/Oracle-only date problem, etc.

Nearly all the oracle people you find want to show you how cool they are by writing a bunch of non-portable Oracle crap, half of which doesn’t work half as well as half of them think. As for standards-compliance and ease of use, Oracle blows.
here's a fun one: oracle 10g, with rails running on Windows, can’t store dates prior to the UNIX(!) epoch.
monkey-patching
if $\$.grep(%r,oci8.rb,).length > 0
  STDERR << "Patching Oracle columns() performance (see: http://dev.rubyonrails.org/ticket/3210)\n"
  require 'oci8'

module ActiveRecord
  module ConnectionAdapters
    class OCIAdapter
      def columns(table_name, name = nil) #:nodoc:
        table_info = @connection.describe_any(table_name)

        table_cols = %Q{
          select column_name, data_type, data_default, nullable,
          decode(data_type, 'NUMBER', data_precision,
          'VARCHAR2', data_length,
          null) as length,
          decode(data_type, 'NUMBER', data_scale, null) as scale
          from all_tab_columns
          where owner    = '#{table_info.schema}'
          and  table_name = '#{table_info.name}'
        }

        select_all(table_cols, name).map do |row|
          row['data_default'].sub!(\'^\(.*\)\".*$/', '\1') if row['data_default']
          OCIColumn.new(
            oci_downcase(row['column_name']),
            row['data_default'],
            row['data_type'],
            row['length'],
            row['scale'],
            row['nullable'] == 'Y'
          )
        end
      end
    end
  end
end
when we wrote a ticket, sent in a patch, we dealt directly (and immediately) with the maintainers of those functional areas. When the patch landed and we pulled edge or a new release, we just removed our monkey-patch.
Now, instead of monkey-patching, we refactor fixes to plugins, which are a clean point of change and which can be easily removed when patches land in trunk.
Maximum Rate of Change
Less than the maximal rate of change? The obvious risk would be that the change by the target date would be less than ideal. The real risk is that without an aggressive change rate, NO real change would take place.
Faster than maximal? You risk confusing the team members, failing to provide enough stability to allow progress. Worse, you may awaken politically entrenched opponents who will cause harm to the project.
Big Design Up Front
Oracle

Big Design Up Front

“deployments”

goodage

time
Big Design Up Front

“deployments”

for-profit partner
Oracle
Java
cvs
Big Design Up Front
SQL schemae
"deployments"
Java
for-profit partner
time
suckage
goodage
Oracle
Java
cvs
Postgres
JBoss
SQL schemae
Big Design Up Front
"deployments"
for-profit partner
suckage

Oracle

Postgres

svn

SQL schemae

Big Design Up Front

JBoss

“deployments”

Java

Hibernate

for-profit partner
Oracle

Java

svn

Postgres

Ant

JBoss

SQLite schemae

Big Design Up Front

“deployments”

Java

Hibernate

for-profit partner
Oracle
Java
svn
trac
Postgres
IRC
continuous integration
domain driven design
analysis patterns
Pair-on-demand
assigned tickets
SQL schemae
Big Design Up Front
JBoss
Ant
CruiseControl
“deployments”
Java
EJB3.0
Hibernate
for-profit partner
Oracle
Java
svn
trac
Postgres
Ant
continuous integration
SVN
trac
domain driven design
unit tests
IRC
auto deployments
pair-on-demand
analysis patterns
assigned tickets
SQL schemas
Big Design Up Front
EJB3.0
JUnit, etc.
Hibernate
CruiseControl
for-profit partner
Oracle
Java
svn
trac
public svn
domain driven design
unit tests
Postgres
IRC
auto deployments
pair-on-demand

analysis patterns
continuous integration
svn
trac
public svn
domain driven design
unit tests
Postgres
IRC
auto deployments
pair-on-demand

assigned tickets
SQL schemae
Big Design Up Front
JBoss
Ant
JUnit, etc.
CruiseControl
Java
EJB3.0
Hibernate

for-profit partner

goodage
suckage

time
Oracle

suckage

goodage

Rails

analysis patterns
Rake
continuous integration

nightly conversions

svn
trac
public svn
domain driven design
unit tests

Postgres
IRC
AJAX
continuous builder

auto deployments
public IRC

pair-on-demand

assigned tickets

SQL schemae

Big Design Up Front

continuous integration

nightly conversions

assigned tickets

SQL schemae

Big Design Up Front
Oracle

suckage

Rails
  analysis patterns
  Rake
  continuous integration
  nightly conversions

svn
  trac
  public svn

Postgres
  IRC
  AJAX

unit tests
  auto deployments
  public IRC
  pair-on-demand

Big Design Up Front

assigned tickets

SQL schemae

Big Design Up Front

CTI

lose technical lead
Oracle

suckage

suckage
goodage

time

Rails

analysis patterns

Rake

continuous integration

migrations

switchtower

getting more “Real”

Rails

频频

trac

public svn

domain driven design

unit tests

Postgres

IRC

AJAX

auto deployments

public IRC

pair-on-demand

Oracle

assigned tickets

1,000 meetings

public SVN

public trac

public trac

public trac

public trac

public trac

public trac

public trac

public trac
Insights
You really have to do things the Rails Way. All the little opinionated choices turn out to be right, whether they seem important or not. We fought pluralization for a long time. When we finally turned it on so many little headaches disappeared. When we were able to get onto Migrations (too early to start there) so many big headaches disappeared. We still keep foreign keys, auditing, various constraints, which are a big headache -- but at least we can have them in plugins and not have to think about them again.
Back in the fall I'd go to Healthcare Technology conferences and I'd mention to people that we were doing our system in Ruby on Rails and they'd say, ‘Ruby on Rails...What's that?’
Back in the fall I'd go to Healthcare Technology conferences and I'd mention to people that we were doing our system in Ruby on Rails and they'd say, ‘Ruby on Rails... What's that?’

Now when I go to conferences they say, “Nobody is using Ruby on Rails.”