

# Chain of Command Design Pattern & BI Publisher

Miroslav Samoilenko

October 2009

## Chain of Command Design Pattern & BI Publisher

### Statement of the Problem

Consider the following simple task. The client is running Oracle eBusiness Suite release 11i or 12, and needs to print customer statements or AR dunning letters in a nice looking format with pictures, diagrams, hyperlinks, and fine print.

Definitely, we will be using BI Publisher to convert XML produced by the customer statement or dunning letter report into whatever format our client desires. The key question is: how to apply BI Publisher to customer statement output since customer statement is a spawned process which, as of time of this article, does not allow for BI Publisher template application!

### Chain of Command Design Pattern

The fundamental problem which we need to resolve is performing an action when a concurrent program completes, i.e. once customer statement is generated we need to apply BI Publisher template. We can rephrase this task saying that customer statement report cannot produce the desired result and needs to delegate action to another process. It sounds exactly as the chain of command design pattern (see for example,

[http://en.wikipedia.org/wiki/Chain-of-responsibility\\_pattern](http://en.wikipedia.org/wiki/Chain-of-responsibility_pattern)).

There are several implementations of design patterns available in Oracle eBusiness Suite out of the box. The most popular one is a request set. You can chain requests together to be executed in parallel or sequentially to complete desired process. Oracle Lease Management Lease Billing request set is a perfect example of such chain. However, since customer statement process cannot be submitted from the standard report submission (SRS) form, we cannot use request sets to print customer statements.

Another implementation of a chain of command related to concurrent requests is the post processor. You can associate post processor actions with a concurrent request. Action execution depends on the completion status of the concurrent request, i.e. you can define one action when concurrent request finishes successfully and different action for failures. The best known post processor is a printing action. The drawback of this approach is that the post processor actions must be defined before the concurrent request is picked up by the concurrent manager for processing.

The other example is a user procedure. We can instruct the user to execute 'XML Report Publisher' concurrent program to convert output of the customer statement from XML into PDF. This implementation of the chain of command design pattern will definitely work, but is very error prone. The user may just forget to do it.

The approach which we would like to discuss here is building the chain of command implementation based on business events.

### Chain of Command Controller

We will use Oracle eBusiness Suite business events and business event subscriptions as the chain of command controller mechanism. That is whenever a concurrent program completes, we raise a

business event; any action which we want to perform after the completion of the concurrent request are subscribed to the business event.

The business events can be raised from a database trigger against FND\_CONCURRENT\_REQUESTS table, or from a periodic concurrent program (listener) which raises business events for all concurrent requests that completed their executing since the last execution of the listener.

To control the number of business events triggered, we can use a naming convention for business events. If we want a business event to be triggered by completion of a concurrent program, we register a business event with the name format:

<prefix>.<concurrent\_program\_name>

For example

premierotec.conc.ARXSGPO

Triggering of the business event can be accomplished by the following procedure

```

procedure raise_business_event(p_request_id IN NUMBER, x_return_status OUT VAR-
CHAR2) is
  l_business_event_name VARCHAR2(200);
  l_parameter_list      wf_parameter_list_t;
begin
  x_return_status := 'S';
  begin
    -- get name of the business event for the concurrent program, provided it exists
    select we.name
    into l_business_event_name
    from wf_events we
    , fnd_concurrent_requests fcr
    , fnd_concurrent_programs fcp
    where fcr.program_application_id = fcp.application_id
    and fcr.concurrent_program_id = fcp.concurrent_program_id
    and we.name = fnd_progile.value('PREMIERTEC_COC_PREFIX') || '.' || fcp.concur-
rent_program_name
    and fcr.request_id = p_request_id;
  exception when others then
    return;
  end;
  -- raise business event with request_id as the event parameter
  wf_event.AddParameterToList('REQUEST_ID', p_request_id,l_parameter_list);
  Wf_Event.RAISE(p_event_name => l_business_event_name,
                p_event_key => l_business_event_name || p_request_id,
                p_parameters => l_parameter_list);
  exception when others then
    x_return_status := 'E';
  end;

```

In this procedure we use profile option PREMIERTEC\_COC\_PREFIX which contains the prefix for the business event as specified in the naming convention.

As mentioned before, this procedure can be inserted into a database trigger on FND\_CONCURRENT\_REQUEST table when PHASE\_CODE changes to 'C', or in a periodic concurrent program that executes it for all concurrent requests completed since the last run.

### Chained Processes

The next task is to subscribe an action to the business event. We can subscribe a workflow to be triggered, or a PL/SQL procedure. In this example, we will subscribe a PL/SQL procedure which submits another concurrent request and apply the BI publisher template to the output of the customer statement report.

The sample handler provided below reads the requests ID from the event parameters, retrieves the BI publisher template information and submits the XML Report Publisher concurrent program.

```

FUNCTION process_event( p_subscription_guid In RAW
                      , p_event IN OUT NOCOPY WF_EVENT_T) RETURN VARCHAR2 IS
  l_request_id  NUMBER;
  l_user_id     NUMBER;
  l_resp_id     NUMBER;
  l_application_id NUMBER;
  l_security_gr_id NUMBER;
  l_template_app_id NUMBER;
  l_locale     VARCHAR2(64);
  l_template_type_code VARCHAR2(64);
  l_template_code VARCHAR2(64);
begin
  -- read request_id from the event parameters
  l_request_id := p_event.GetValueForParameter('REQUEST_ID');
  -- read information from the request and associated BI Publisher template
  -- we assume there is only one template and the output is PDF
  select xt.template_code
    , fcr.requested_by
    , fcr.responsibility_id
    , fcr.responsibility_application_id
    , fcr.security_group_id
    , xt.application_id
    , xt.default_language || decode(xt.default_territory, '00', null, '-' || xt.default_terri-
  tory) locale
    , xt.template_type_code
    , xt.template_code
  into l_template_code
    , l_user_id
    , l_resp_id
    , l_application_id
    , l_security_gr_id
    , l_template_app_id
    , l_locale
    , l_template_type_code
    , l_template_code
  from fnd_concurrent_requests fcr
    , fnd_concurrent_programs fcp

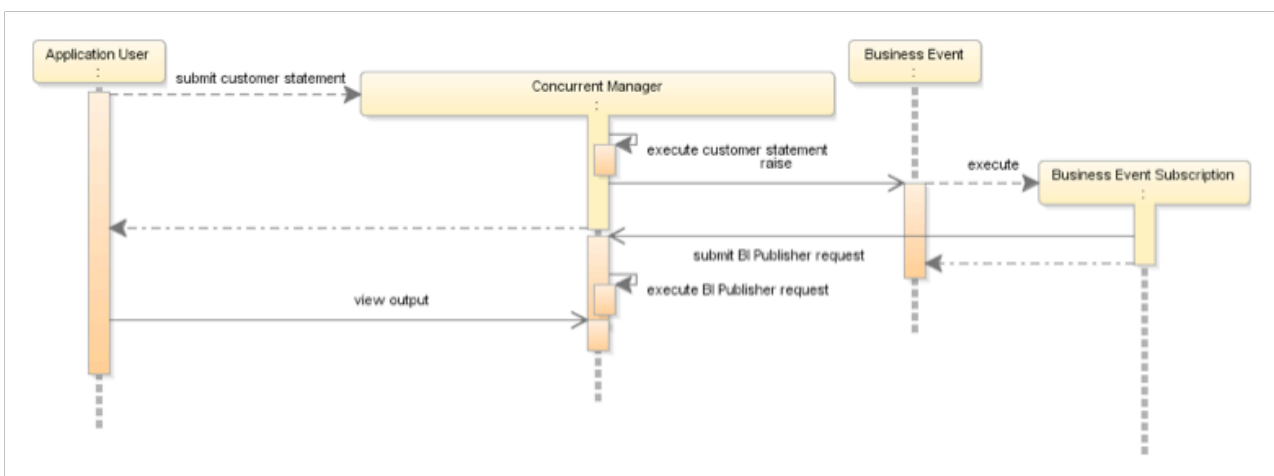
```

```

, fnd_application fa
, xdo_templates_b xt
where fcr.program_application_id = fcp.application_id
and fcr.concurrent_program_id = fcp.concurrent_program_id
and fcp.application_id = fa.application_id
and xt.data_source_code = fcp.concurrent_program_name
and xt.ds_app_short_name = fa.application_short_name
and fcr.status_code = 'C'
and fcr.request_id = l_request_id;
-- set up application context so that the user who submitted the original request
-- can see the chained request
fnd_global.apps_initialize(l_user_id,l_resp_id,l_application_id, l_security_gr_id);
-- submit BI Publisher print process
l_request_id := FND_REQUEST.submit_request( application => 'XDO'
, program => 'XDOREPPB'
, argument1 => to_char(l_request_id)
, argument2 => l_template_code
, argument3 => l_template_app_id
, argument4 => l_locale
, argument5 => 'N'
, argument6 => l_template_type_code
, argument7 => 'PDF');
if l_request_id = 0 then
return 'ERROR';
end if;
Return 'SUCCESS';
exception when others then
return 'ERROR';
end;
```

The example above is greatly simplified. We only handle successfully completed requests, assume that there is only one BI publisher template associated with the customer program, and the output type is hardcoded to PDF.

User Procedure



This is how the process now works for the application user.

The user submits generation of customer statements to the concurrent manager. Concurrent manager executes the request and produces customer statements as an XML document. The chain of command implementation raises a business upon completion of customer statement generation request. Subscriber to this even submits XML Publisher Print program to the concurrent manager. Concurrent manager executes the request and its output is available to the application user in a PDF format.

## Summary

Submission of concurrent programs from the event handlers ensures that we are reusing the same approach to chain the processes. It is easy to maintain and understand how customer statements and dunning letters are printed, if it is implemented the same fashion.

The chain of command implementation is not tied to the customer statements generation process alone. We use it to FTP outputs of concurrent requests to remote machines, to upload files from desktop into Oracle eBusiness Suite (which is the topic for my next white paper) and a multitude of other needs