SQL Developer Data Modeler / APEX Integration

– Architectural Software Development

Marc de Oliveira, Simplify Systems

Introduction

This paper is concerned with Architectural Software Development – meaning software development where the applications used by the end users are directly connected to the enterprise architecture. In this case the connection will be between Oracle's SQL Developer Data Modeler (used for enterprise architecture modeling) and APEX (used for developing the end user application).

Architectural Software Development

The term Architectural Software Development (ASD) covers any development method where high level business model information directly impacts the behavior of the applications used by end users. A simple example of this would be that updating the definition of a business term, such as ENROLLMENT TIME, would automatically be reflected in the online help text of the applications. Another example would be to map the menu system of an application to a section of the business process model.

The value of ASD is to make sure that changes to the Enterprise Architecture are implemented in the tools of the business, to avoid that models and documentation diverge from the actual applications used throughout the business.

Even though it is probably impossible to completely link all artifacts of an Enterprise Architecture with executable code of an application, interconnecting models and code as much as possible will help keep the different elements synchronized, and ensure that the business model is consistent with how the business is actually run.

ASD does not prescribe which specific elements of the Enterprise Architecture that should enforce particular behavior in the end user applications but in this paper I will describe a number of integrations I have implemented between SQL Developer Data Modeler and APEX on a recent development project I have participated in.

SQL Developer Data Modeler

Oracle's SQL Developer Data Modeler (SDDM) is primarily a tool for Data Modeling but it also supports diagramming of Process Models and simple documentation of some business information, such as documents, persons and locations.

The processes and business information are only supported on the conceptual level, while data is supported on conceptual (in SDDM called logical), design (in SDDM called relational), and physical levels.

So, Data Modeling is clearly the main focus of the tool but I will cover some of the other areas, too, as they can be very helpful when developing and maintaining complex application systems.

The easiest connection between SDDM and APEX is probably through the Reporting Schema of SDDM. The Reporting Schema is a database schema of 77 tables that mirrors most of the content of the SDDM models, allowing users to build PL/SQL routines based on the SDDM models.

The Reporting Schema is maintained by the Export function of SDDM, like this:

File → Export → To Reporting Schema (then select a database connection and click OK)

Every export stores a new version of the current models in the Reporting Schema tables. I have described the details about how the SDDM Reporting Schema is structured in the article “SQL Developer Data Modeler 3.0 Under The Covers: The Reporting Schema" that you will find on my web site: www.SimplifySys.com.

Integrating SDDM with APEX

Currently, there are only very few actual ASD tools, so it is up to system analysts

and system developers to find ways to integrate their modeling tools with their development tools.

Here are some examples of integrating SDDM with APEX to extract application help text from both an SDDM information model and a SDDM process model, creating a dynamic menu and breadcrumb from an SDDM process model, and finally an example about integrating SDDM with the project management features of APEX to specify development tasks, estimate development time and follow development progress.

Synchronizing SDDM with APEX

Even though it may be possible to extract the necessary ASD information directly from the SDDM Reporting Schema tables I recommend that a small set of ASD tables are maintained separately.

It will add an extra step to the synchronization process but it is well worth it for the following reasons:

- 1) You can structure the ASD tables to accommodate your development process, making it as easy as possible to extract the necessary SDDM information for APEX
- 2) You will be less vulnerable to changes in the SDDM Reporting Schema, as those changes only will have to be handled in one place (the synchronization code). All APEX code (that look in the ASD tables) will not see the changes to the Reporting Schema
- 3) It allows you to integrate APEX with multiple EA tools as if they had one common repository

In our case we only needed the following four ASD tables to store the SDDM information needed for the APEX integration covered by this article:

- MODEL_TABLES: Storing information about the database tables and their related entities in SDDM. Note, that each table could implement multiple entities if you have modeled entity subtypes in SDDM, ie entities PERSON and ORGNIZATION may both be implemented by the table PARTIES.
- MODEL_COLUMNS: Storing information about the database columns and their related entity attributes, as well as attribute descriptions, default values, and information about the relationships to other entities/tables that that the columns may represent. The relationships can be managed in this table because we know that every primary key (and, hence, every foreign key) consist of a single column.
- MODEL_PROCESSES: Storing information about the process hierarchy of SDDM. The leaf processes are, for the most part, to be implemented as APEX modules, while the composite processes are to be implemented as menus. Module is not really an APEX term. I define it as the set of APEX pages that covers the functionality of a SDDM process.
- MODEL_DOMAINS: Storing the SDDM domains, so that APEX can inherit the same domains and valid values as SDDM without having to maintain them twice.

The PL/SQL of synchronizing SDDM with APEX

The main pl/sql procedure for synchronizing the SDDM Reporting Schema with the four model tables described above must first delete any existing data from the tables and then populate each of the model tables with data from the corresponding SDDM Reporting Scheme tables.

As the SDDM Reporting Schema tables may contain multiple designs each in many versions, we need to extract SDDM data about the latest version of a specific design. The identification of a specific version of a design is stored in the column DMRS_DESIGNS.DESIGN_OVID. This is all explained in more detail in my article “SQL Developer Data Modeler 3.0 Under The Covers: The Reporting Schema” from 2011.

The resulting pl/sql could look like this:

```
procedure sync_model(P_DESIGN_NAME in DMRS_DESIGNS.DESIGN_NAME%TYPE) is
begin
  delete_old_model;
  for DESIGN in
   (select DESIGN_OVID
    from DMRS_DESIGNS DESIGN1
    where
      DESIGN1.DESIGN_NAME    = P_DESIGN_NAME and
      DESIGN1.DATE_PUBLISHED =
        (select max(DESIGN2.DATE_PUBLISHED)
         from   DMRS_DESIGNS DESIGN2
         where  DESIGN2.DESIGN_NAME = P_DESIGN_NAME))
  loop
    populate_model_proceses(DESIGN.DESIGN_OVID);
    populate_model_tables(DESIGN.DESIGN_OVID);
    populate_model_columns(DESIGN.DESIGN_OVID);
    populate_model_domains(DESIGN.DESIGN_OVID);
  end loop;
end;
```

The PL/SQL of procedure delete_old_model

The procedure delete_old_model is used to delete the current data in the model tables to make room for a new set of data from the SDDM Reporting Schema tables. Even though the SDDM Reporting Schema tables can hold multiple versions of models, we are only interested in the most recent version in the model tables for APEX. That is why this procedure simply deletes all data from the model tables, like this:

```
procedure delete_old_model is
begin
  EXECUTE IMMEDIATE 'truncate TABLE model_processes';
  EXECUTE IMMEDIATE 'truncate TABLE model_columns';
  EXECUTE IMMEDIATE 'truncate TABLE model_tables';
  EXECUTE IMMEDIATE 'truncate TABLE model_domains';
end;
```

The PL/SQL of procedure populate_model_processes

The procedure populate_model_processes is used to transfer all necessary process

model information, to be used by APEX, to the model_processes table.

These are the data that we want to transfer about the SDDM processes:

- PROCESS ID: To be able to keep the SDDM processes synchronized with APEX modules we need to store the SDDM process id.
- PARENT PROCESS ID: The parent process id is a foreign key to the parent process of a process. This is how we know about the process hierarchy. Ex, processes without a parent process id are the top processes of the process model.
- PROCESS NAME: The process name is the understandable identification of a process. We will synchronize the SDDM process names with the APEX module names. I will also define an APEX Feature for each SDDM Process.
- COMMENTS: I have used the comments of an SDDM process as the description of the process. The description will be used both as part of the APEX Feature description and as help text for the corresponding APEX pages.
- FOOTNOTES: I have used the footnotes of SDDM for short structured information about the processes that are both informational to end users, and also understandable by the system to control specific functionality in the application. An example of a footnote could be: “The context of the module is: Academic Term.”. I will explain this in more detail later.
- SPECIFICATION: I have used the notes of the SDDM process for my technical specifications of the corresponding APEX modules. This text is not intended for end users but for the APEX developers that have to implement the SDDM processes in APEX.
- PROCESS NUMBER: The process number indicates the position of a process within the process hierarchy, ex: 3.5.1
- PROCESS MODE: The process mode is an SDDM setting that indicates how a given process is to be implemented. Valid process modes are Unknown (default), Manual, Batch, and Interactive. I suggest that you use “Interactive” for processes that should be implemented by APEX, “Batch” for processes that should be implemented by pl/sql (these could in turn be used by an APEX module), and “Manual” for processes that are not going to be automated.
- PROCESS TYPE: The leaf processes are assigned the process type “Primitive”, while parent processes have the type “Composite”. This information can be used to differentiate APEX screens (Primitive) from menu items (Composite).

Getting all this information into the MODEL_PROCESSES table can be done like this:

```
Procedure populate_model_processes(
  P_DESIGN_OVID in DMRS_DESIGNS.DESIGN_NAME%TYPE)
is
begin
  INSERT INTO model_processes
    (object_id,
     parent_process_id,
     process_name,
     comments,
     footnote,
```

```
                    specification,
                    process_number,
                    process_mode,
                    process_type)
                SELECT
                  p.process_id,
                p.parent_process_id,
    p.process_name,
    (SELECT text
     FROM   dmrs_large_text t
     WHERE  t.object_id   = p.process_id
     AND    t.design_ovid = p.design_ovid
     AND    t.type        = 'Comments'),
    (SELECT text
     FROM   dmrs_large_text t
     WHERE  t.object_id   = p.process_id
     AND    t.design_ovid = p.design_ovid
     AND    t.type        = 'Footnote'),
    (SELECT text
     FROM   dmrs_large_text t
     WHERE  t.object_id   = p.process_id
     AND    t.design_ovid = p.design_ovid
     AND    t.type        = 'Note'),
    p.process_number,
    p.process_mode,
    p.process_type
  FROM   dmrs_processes p
  WHERE  p.design_ovid = P_DESIGN_OVID;
end;
```

The PL/SQL of the procedure populate_model_tables

I have merged a few SDDM Reporting Schema tables into the MODEL_TABLES table. This is the data I am interested in for my APEX application:

- TABLE ID: As for processes I need the SDDM table id as a hook back into the SDDM model.
- TABLE NAME: Obviously, the table name is the key to identify the table in question on the database.
- ENTITY NAME: To be able to connect business terms with the database tables used by the APEX application we need to collect the mappings of entities to tables. This means that the table MODEL_TABLES actually contains records matching the entities of the logical models, while the table names describe how the entities have been matched to the relational (and physical) model.
- PREFERRED ABBREVIATION: I have used the preferred abbreviation for storing the plural version of the entity names as SDDM 3.0 does not have this as part of its repository. I need the plural version of every entity name to create natural language descriptions of each foreign key in the database, such as “Each site may be addressed in by one or more comment submissions”.
- IS MAIN ENTITY: Marking the top level super type entity of each table allows the APEX application to use a business term to describe a database table in user help etc.
- MAIN SCREEN: If an entity of the SDDM model has a primary management screen, I think it is of value to specify it in the description of that entity. It could be that the data of the entity UNIVERSITY is usually managed by the APEX screen “University Management”. This information will make it possible to automate a lot of useful inter application navigation. I use the built-in function reg-

exp_substr to extract the APEX screen name from the entity comment.

Getting all this information into the MODEL_TABLES table can be done like this:

```
Procedure populate_model_tables(
  P_DESIGN_OVID in DMRS_DESIGNS.DESIGN_NAME%TYPE)
is
begin
  FOR t IN (
    SELECT
      dt.object_id,
      dt.table_name,
      de.entity_name,
      de.preferred_abbreviation,
      (SELECT text
       FROM   dmrs_large_text t
       WHERE  t.object_id   = de.object_id
       AND    t.design_ovid = P_DESIGN_OVID
       AND    t.type        = 'Comments') comnts,
      decode(de.supertypeentity_name, null, 'Y', 'N') is_main_entity
    FROM
      dmrs_tables   dt,
      dmrs_mappings dm,
      dmrs_entities de
    WHERE  dt.design_ovid          = p_design_ovid
    AND    dm.relational_object_id = dt.object_id
    AND    dm.design_ovid          = dt.design_ovid
    AND    de.object_id            = dm.logical_object_id
    AND    de.design_ovid          = dm.design_ovid)
  LOOP
    INSERT INTO model_tables
      (object_id,
       table_name,
       entity_name,
       preferred_abbreviation,
       comments,
       is_main_entity,
       primary_management_screen)
    VALUES
      (t.object_id,
       t.table_name,
       t.entity_name,
       t.preferred_abbreviation,
       t.comnts,
       t.is_main_entity,
       substr(regexp_substr(t.comnts, 'Main screen: ([^.]+)'),14));
  END LOOP;
end;
```

The PL/SQL of the procedure populate_model_columns

I have merged even more SDDM Reporting Schema tables into the MODEL_COLUMNS table. This has to do with relevant information coming from many areas of the SDDM model, such as logical attributes, relational columns, domains, and relationships.

This is the data I am interested in for my APEX application:

- COLUMN ID: As for processes and tables I need the SDDM column id as a hook back into the SDDM model.
- COLUMN NAME: Obviously, the column name is the key to identify the table column in question on the database.

- ATTRIBUTE NAME: The attribute name is the business term that the end users relate to the column. It should be used in labels, help text etc.

RELATION INFORMATION: To be able to create natural language descriptions of foreign key columns (such as “Each university may be rated through one and only one star rating”), I need a few elements of the related relationships, such as the source and target entity names, the source and target relationship labels, cardinality information, and optionality information.

- DESCRIPTION: The description of each attribute is used for the online help of the corresponding APEX items.
- SEQUENCE: The attribute sequence numbers should be reused to suggest the sequence of items on APEX screens. Even though APEX per default suggests the item sequences matching the column sequence of the column tables, this sequence could be wrong as columns that are added to the system later, often are added to the tables using ALTER TABLE ADD statements which just adds the columns as the last one.
- DEFAULT VALUE: The current version of APEX (4.0.2) has issues with managing default values – especially dates – so I need the default values from SDDM to be able to manage the handling of default values on the APEX screens.

Getting all this information into the MODEL_COLUMNS table can be done like this:

```
Procedure populate_model_columns(
  P_DESIGN_OVID in DMRS_DESIGNS.DESIGN_NAME%TYPE)
is
begin
  FOR c IN (
    SELECT
      dc.object_id,
      dc.column_name,
      da.attribute_name,
      dc.table_name,
      (SELECT text
       FROM datamodel.dmrs_large_text t
       WHERE
         t.object_id   = da.object_id  AND
         t.design_ovid = P_DESIGN_OVID AND
         t.type        = 'Comments') comnts,
      dr.target_label,
      dr.source_label,
      dr.source_entity_name,
      dr.target_entity_name,
      dr.sourceto_target_cardinality,
      dr.targetto_source_cardinality,
      dr.source_optional,
      dr.target_optional,
      da.sequence,
      NVL(dc.default_value, dom.default_value) default_value
    FROM
      dmrs_columns dc,
      dmrs_models dm,
      dmrs_mappings dma,
      dmrs_attributes da,
      dmrs_relationships dr,
      dmrs_domains dom
    WHERE dm.design_ovid                 = d.design_ovid
      AND dm.model_type                  = 'Relational'
      AND dm.model_id                    = dc.model_id
      AND dm.model_ovid                  = dc.model_ovid
      AND dma.relational_object_id (+)   = dc.object_id
```

```
      AND dma.relational_object_ovid (+) = dc.ovid
      AND da.object_id (+)               = dma.logical_object_id
      AND da.ovid (+)                    = dma.logical_object_ovid
      AND da.relationship_id             = dr.object_id (+)
      AND da.relationship_ovid           = dr.ovid (+)
      AND dc.domain_id                   = dom.domain_id (+)
      AND dc.domain_ovid                 = dom.ovid (+)
    ORDER BY dc.sequence)
  LOOP
    INSERT INTO model_columns
     (object_id,
      table_name,
      column_name,
      attribute_name,
      target_label,
      source_label,
      source_entity_name,
      target_entity_name,
      sourceto_target_cardinality,
     targetto_target_cardinality,
     source_optional,
     target_optional,
     comments,
     sequence,
     default_value)
   VALUES
    (c.object_id,
     c.table_name,
     UPPER(c.column_name),
     c.attribute_name,
     c.target_label,
     c.source_label,
     c.source_entity_name,
     c.target_entity_name,
     c.sourceto_target_cardinality,
     c.targetto_source_cardinality,
     c.source_optional,
     c.target_optional,
     c.comnts,
     c.sequence,
     c.default_value);
  END LOOP;
end;
```

The PL/SQL of the procedure populate_model_domains

Finally, I need the domain definitions with all the valid values of each domain from the SDDM Reporting Schema tables so I can define List of Values in APEX based on the SDDM domains instead of having to manage the valid values in multiple places.

This is the data I am interested in for my APEX application:

- DOMAIN ID: I need the SDDM domain id as a hook back into the SDDM model.
- DOMAIN NAME: The domain names is what I will refer to in my APEX List of Values as the domain id is not as readable.
- VALUE: Obviously, I need the actual valid values of each domain.
- SHORT DESCRIPTION: For list of values I will use the short value description as the display value in the APEX List of Values.
- SEQUENCE: Even though you cannot maintain the sequence of SDDM 3.0 valid values (hopefully this will be added at some point) they are exported in the sequence that you create them, hence, the APEX List of Values should be

ordered by the value sequence.

Getting this information into the MODEL_DOMAINS table can be done like this:

```
                Procedure populate_model_domains(
  P_DESIGN_OVID in DMRS_DESIGNS.DESIGN_NAME%TYPE)
is
begin
  INSERT INTO model_domains
   (domain_id,
    domain_name,
    value,
    short_description,
    sequence)
  SELECT
    do.domain_id,
    do.domain_name,
    v.sequence,
    v.value,
    v.short_description
  FROM
    dmrs_domains    do,
    dmrs_domain_avt v
  WHERE  do.domain_id   = v.domain_id
    AND  v.design_ovid  = do.design_ovid
    AND  do.design_ovid = d.design_ovid;
end;
```

With these four ASD tables in place we can start looking at providing automatic functionality to our APEX applications. That is what the rest of this article is about.

Extracting Help from SDDM Models

The SDDM models contain a lot of high level business descriptions of value to application users. All descriptions of business terms related to individual application screens should be made available on every APEX screen, as well as descriptions of each individual screen item (these should mostly be based on attributes from the data model). On top of this, the description of each business process should be able to bring end users an understanding of the purpose of the screens that they are working with.

Let us take a look at how this kind of integration can be automated using the ASD tables we have defined and populated in the previous section.

Providing ASD Item Help in APEX

APEX comes with built-in item help based on the database column comments stored in the data dictionary, but the implementation has a big problem in that it only reads the column comments once. Any changes to the column comments are not reflected in the APEX user interface, and we want our APEX applications to be alive in the sense that when the business changes its definition of business terms these should automatically be applied to the end user applications.

To take over the APEX item help we have created an item help template that uses JavaScript to look up the MODEL_COLUMNS table at run-time to fetch the current comment defined for the related attribute in SDDM.

We will not go into details about this solution but instead look at how we implemented an automatic module help screen in APEX.

Providing ASD Module Help in APEX

When it comes to module help, we wish to include the process descriptions as well as the entity descriptions of entities (business terms) related to the module.

This can be managed by creating a region plug-in that can be added to all modules, with the following parameters:

- MAIN ENTITY: A business term that should be explained as part of the help text for the current APEX module. This could also be a list of business terms/entities not necessarily used directly by the APEX module. The main idea about the entities on this list would be that they explain issues and terms that the end user needs to understand to be able to understand the purpose of the APEX module. In this example we just handle a single entity name.
- HELP TEXT: A section of help text that should be added to process description. As default this text is empty as the SDDM process description should describe the module completely. But in case some implementation specific explanation need be included, developers can add it in this parameter.
- MODULE TYPE: A select list of standard APEX page structures that requires different types of help text, such as Interactive Report, Form, or Tabular Form. In this case we only have the following two types of modules: 1) An Interactive Report with a Form, and 2) A Master detail form with a Form and a Tabular Form.

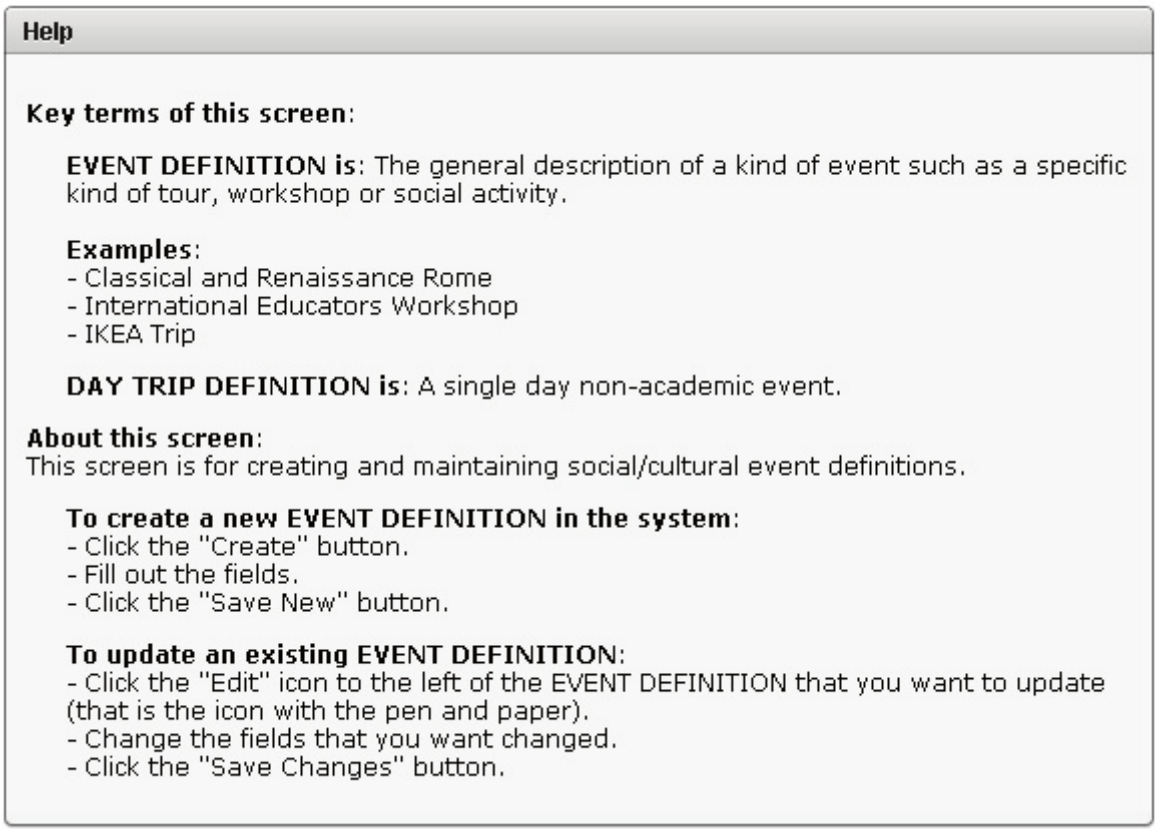
A region plug-in with these three parameters can automate the work of building context sensitive help, like this:

```
function HelpRegion (
  p_region              in apex_plugin.t_region,
  p_plugin              in apex_plugin.t_plugin,
  p_is_printer_friendly in boolean )
  return apex_plugin.t_region_render_result
is
  vHelp         varchar2(4000);
  vEntity       p_region.attribute_01%type := p_region.attribute_01;
  vHelpText     p_region.attribute_02%type := p_region.attribute_02;
  vModuleType   p_region.attribute_03%type := p_region.attribute_03;
begin
  return(
    BusinessTermDescriptions(vEntity) ||
    SpecificProcessHelp(vProcess) ||
    vHelpText ||
    StandardProcessHelp(vEntity,vModuleType));
end;
```

I suggest that you structure the help text like this:

First, I wish to describe one (or more) business terms that the end user have to know about to understand the process description. Then comes the actual process description and some specific technical description of the APEX implementation. Finally, I add some standard description about the specific kind of screen which the user is looking at.

Depending on how the functions BusinessTermDescription, SpecificProcessHelp, and StandardProcessHelp are coded, the resulting APEX help region could look like as shown on Illustration 1.



In the following pl/sql I have removed the HTML tag processing to unclutter the code.

Business Term Description

The function BusinessTermDescription returns the headline 'Key terms of this screen:' followed by the entity name, the string ' is: ', and the entity comment matching the parameter (p_entity). I extract the entity name and comment in a loop, so that the function easily can be extended to handle sets of entities instead of just one:

```
function BusinessTermDescription(p_entity in varchar2)
  return varchar2
is
  vReturnValue MODEL_TABLES.COMMENTS%TYPE;
begin
  vReturnValue:= 'Key terms of this screen:' || chr(10);
  for entities in
    (select entity_name, comments
     from   model_tables
     where  entity_name = p_entity)
  loop
    vReturnValue:= vReturnValue ||
                   entities.entity_name || ' is: ' || entities.comments;
  end loop;
  return(vReturnValue);
end;
```

Specific Process Help

The function SpecificProcessHelp can extract the current APEX page name from the APEX API, so because I match the SDDM process name with the APEX page names, it can be called without a process name. When called without a parameter the function looks up the page name in the APEX API view APEX_APPLICATION_PAGES using nv('APP_ID') and nv('APP_PAGE_ID') to fetch the current page page id. Using the current page name as the SDDM process name, we can look up the process description in our MODEL_PROCESSES ASD table.

Here are the details:

```
function SpecificProcessHelp(
  p_process in varchar2 default null)
  return varchar2
is
  vProcess
    apex_application_pages.page_name%type;
  vHelp        model_processes.comments%type;
  vReturnValue varchar2(4000);
begin
  if p_process is null
  then
    select page_name
    into   vProcess
    from
      apex_application_pages p,
      apex_applications      a
    where
      p.page_id        = nv('APP_PAGE_ID') and
      a.application_id = nv('APP_ID')      and
      p.application_id = a.application_id;
  end if;

  vReturnValue:= 'About this screen: ';
  begin
    select comments
    into   vHelp
    from   model_processes
    where  lower(process_name) = lower(vProcess);
    return(vReturnValue || vHelp);
  exception
    when no_data_found
    then
      vReturnValue:=
        vReturnValue ||
        'The description of the screen "' ||
        vProcess || '" is missing.';
  end;

  return(vReturnValue);
end;
```

Standard Process Help

The StandardProcessHelp is a function for generating the part of the help text that is almost identical on every page. It is concerned with the standard functionality of your APEX screens, rather than the areas that are specific to each screen. This would usually be about how to create a new record, how to update a record or how

to delete a record, but it could also include help text about how to view audit information, how to export data, etc.

To make it a little less obvious that the text is quite static, you can add parameters to the function that will allow you to adapt the text to the context. The name of the entity that corresponds to the driving table of the APEX screen can be used to rephrase a sentence like “This is how you create a new RECORD”, so it becomes “This is how you create a new EVENT DEFINITION”.

To the user it will look like the text was written specifically to the current screen, while you will be able to maintain the standard process help text in one place, and have your updates shown on all your APEX screens.

To make this work, you will have to define and follow a strict development standard that will result in a systematic work flow for end users. Of course you may work with a few different screen types but try to minimize the amount of different screen architectures. This will make it easier for end users to learn to use the application, and it will make it easier for developers to build and maintain it.

Here is an example of how you can implement a StandardProcessHelp function:

```
function StandardProcessHelp(p_entity in varchar2,
                             p_module_type in varchar2)
  return varchar2
is
  vReturnValue varchar2(4000);
begin
  if p_module_type = 'Report with Form'
  then
    vReturnValue:=
      'To create a new ' || p_entity || ' in the system:' ||
      ' - Click the "Create" button.' ||
      ' - Fill out the fields.' ||
      ' - Click the "Save New" button.' ||
      'To update an existing ' || p_entity || ':');
      ' - Click the "Edit" icon to the left of the ' || Entity ||
      ' that you want to update (the icon with the pen and paper).' ||
      ' - Change the fields that you want changed.' ||
      ' - Click the "Save Changes" button.';
  elsif p_module_type = 'Tabular Form'
  then
    vReturnValue:=
      'To create a new ' || p_entity || ' in the system:' ||
      ' - Click the "Add Row" button.' ||
      ' - Fill out the fields.' ||
      ' - Click the "Save" button.' ||
      'To update an existing ' || p_entity || ':' ||
      ' - Change the fields that you want changed.' ||
      ' - Click the "Save" button.' ||
      'To delete one or more existing ' || p_entity || 's:' ||
      ' - Mark the check-boxes on the rows that you want to delete.' ||
      ' - Click the "Save" button.';
  end if;
  return(vReturnValue);
end;
```

Extracting Menu and Breadcrumb from SDDM Models

If you wish to make your software development even more architectural, you can choose to map your process model to your application menu.

In the previous section I described how to create an APEX help region matching every leaf process. In the same way you can map all parent processes to an APEX menu item to be used in both the menu system and for an application breadcrumb.

When mapping the process model to your menu system, you will have to be even more careful about how you model your business, as even more people will be affected by your modeling decisions. But, still, all the extra resources you put into building the correct model will pay off as work you will not have to do when developing the application.

Also, when your menu is architectural, every time business changes require you to change the menu system, you will automatically update your process models at the same time (as that is where your menu system comes from). This will help you keep your model synchronized with the business as it evolves.

Providing an ASD Menu in APEX

There are many ways to display menus in an HTML application. I will not go into a discussion of menu design but just show how the menu structure can be extracted for any visual interface.

The main issue is that the process names must match the corresponding APEX pages that implement the processes. If you have that in place, then you just need to join the ASD table, MODEL_PROCESSES, with the APEX API view, APEX_APPLICATION_PAGES, using “connect by” to extract the processes in the hierarchical order.

The process numbering of SDDM is a little tricky to sort after, as each level can be both one and two digits. Consider these two process numbers: '1.1.9' and '1.1.10'. In a regular “order by” '1.1.10' would be considered lower than '1.1.9', so we have to calculate an “order by” value for each process number.

Presuming that we never have more than 99 processes on a single level (that would make a very ugly process model!), the function could look like this:

```
create or replace function process_number_value(
  p_process_number in varchar2) return number
is
  v_value      number:= 0;
  v_process_no varchar2(20);
begin
  v_process_no:= p_process_number || '.';
  for v_level in 1..10
  loop
    v_value:=
      100 * v_value +
      to_number(nvl(substr(v_process_no,1,instr(v_process_no,'.')-1),0));
    v_process_no:= substr(v_process_no, instr(v_process_no,'.')+1);
  end loop;
  return(v_value);
end;
```

As mentioned previously, processes that are marked as Batch Processes or Manual Processes should not show up as menu items.

The items that you need to create a hierarchical menu are:

The hierarchical level: To visually be able to show where in the menu hierarchy the current menu item is located.

The SDDM process name: To show as the menu item text that end users can click to navigate to the corresponding APEX page.

The SDDM process number: If you wish to refer to the menu item's corresponding position in the SDDM process model.

The APEX page id: To be able to link the menu item to the corresponding APEX page.

These can be extracted with a not too complicated select statement like this one:

```
SELECT
  LEVEL,
  sddm.process_name,
  sddm.process_number,
  apex.page_id
FROM
  model_processes sddm,
  apex_application_pages apex
WHERE
  apex.page_name        =      sddm.process_name and
  sddm.process_mode     NOT IN ('Batch','Manual') and
  apex.application_id   =      nv('APP_ID')
START WITH
  sddm.process_name     =      p_process_name
CONNECT BY PRIOR
  sddm.object_id        =      sddm.parent_process_id
ORDER SIBLINGS BY
  process_number_value(sddm.process_number);
```

A very simple way to loop through these menu items and display the hierarchical structure in HTML, could be like this:

```
LOOP
  IF item.level > 1
  THEN
    IF item.level > v_last_level
    THEN
      v_menu:= v_menu || '<ul>' ||
        create_menu_item(
          item.process_name, item.process_number, item.page_id);
    ELSIF item.level = v_last_level
    THEN
      v_menu:= v_menu ||
        create_menu_item(
          item.process_name, item.process_number, item.page_id);
    ELSIF item.level < v_last_level
    THEN
      v_menu:= v_menu || '</ul>' ||
        create_menu_item(
          item.process_name, item.process_number, item.page_id);
    END IF;
    v_last_level := item.level;
  END IF;
END LOOP;
```

The create_menu_item function could as simple as this:

```
FUNCTION create_menu_item(
  p_pname   IN model_processes.process_name%TYPE,
  p_pnumber IN model_processes.process_number%TYPE,
```

```
  p_pid     IN apex_application_pages.page_id%TYPE)
  RETURN varchar2 IS
BEGIN
  return('<li>' ||
         '<a title="' || p_pnumber || ' ' || p_pname || '"' ||
         '   href="' || get_apex_url(p_pid) || '" >' ||
         p_pname ||
         '</a>' ||
         '</li>');
END;
```

But you could also extend it with nice CSS classes, and variations for visually highlighting the currently chosen menu item.

Providing an ASD Breadcrumb in APEX

A breadcrumb line is very much like a menu, except that the menu shows the process hierarchy from some point in the hierarchy tree and down to all the leaves, while a breadcrumb line shows the process hierarchy from some point and up to the root process.

The breadcrumb line is simpler to display as it only has one entry per hierarchy level, but the select to extract the breadcrumb path is basically the same as for extracting a menu.

The only difference is that the “connect by prior” statement must be reversed for SQL to move towards the root rather than towards the leaves, and the “order by” clause can be omitted as there is only one matching record per level:

```
CONNECT BY PRIOR
  sddm.parent_process_id = sddm.object_id;
```

Extracting Project Management Information to APEX

A different area of ASD is to relate the SDDM process model to the development management features of APEX.

Every modeled process should somehow be implemented (except for the manual processes) as some kind of APEX Feature, so an obvious choice would be to map the SDDM processes to the APEX Feature.

So I have extended my ASD synchronization procedure to also create an APEX Feature for each non-manual SDDM process using the APEX API procedure wwv_flow_team_api.create_feature, like this:

```
for proc in (
  select
    sddm.object_id,
    sddm.process_name,
    sddm.comments,
    sddm.process_number,
    sddm.specification,
    sddm.process_mode
  from
    model_processes sddm
  where
    sddm.process_mode != 'Manual')
loop
  wwv_flow_team_api.create_feature(
    p_feature_name              => proc.process_number||' '||
                                   proc.process_name,
```

```
      p_application_id            => current_application,
      p_module                    => proc.object_id,
      p_publishable_description   => proc.comments,
      p_description               => proc.specification,
      p_estimated_effort_in_hours => 12);
end loop;
```

We link the SDDM processes to the APEX features using the process OBJECT_ID, which we store with the corresponding APEX Feature's MODULE attribute.

Of course this will work fine the first time but we need our synchronization to be able to also update the features once they have been created and changes happen to the SDDM process model afterward (such as changes to the name or description of the processes).

Unfortunately, APEX does not have a wwv_flow_team_api.update_feature procedure, so I had to break the rules and use the good old UPDATE statement of SQL instead. Be aware that this is unsupported by Oracle, and you will have to do it at your own risk.

For each iteration of the main loop (shown above) we have to check if a matching APEX feature already exists, like this:

```
select
  CASE count(1)
    WHEN 1
    THEN 'Y'
    ELSE 'N'
  END feature_exist
into
  v_feature_exist
from
  wwv_flow_features f
where
  f.module = proc.object_id;

if v_feature_exist = 'Y'
then
  update wwv_flow_features
  set
    feature_name              = proc.process_number||' '||
                                  proc.process_name,
    FEATURE_DESC              = proc.specification,
    PUBLISHABLE_DESCRIPTION   = proc.comments
  where
    module                    = proc.object_id;
else
  /* Insert a new feature */
end if;
```

The above code is a little simplified but I believe you get an idea of how to synchronize the parts of your business model with your project management system.

Instead of just assigning a default hour estimate to each APEX Feature, you could calculate a more reliable estimate based on factors, such as number of entities managed by each process, leaf processes being more complex than parent (menu) processes, etc.

You have to setup a strategy for which attributes of the APEX features that should be controlled by SDDM and which that should be managed in APEX. The point is to manage your information where it is most generic (where it can be reused most) and

avoid having to manage the same information more than once. You get the idea.

Individual developers should be able update APEX Feature information like feature assignment to developers and feature completion status, while business analyst can maintain the process/feature specifications in SDDM and synchronize it into the APEX Feature description attribute.

Using APEX Features for Project Management

In our latest project we assigned a complexity level and a size to each feature and calculated an estimated effort in hours by multiplying the complexity factor with the size factor, giving results between 1 hour and 30 hours. We can probably work with even more factors and more advance calculations to improve our estimates.

The project estimate then becomes the sum of all the individual feature estimates (you should probably also add estimates for other project elements such as developing general library code, templates etc, but let us leave that out for simplicity).

```
function project_estimate(
  p_application_name in apex_applications.application_name%type)
  return number
is
  v_estimate number:= 0;
begin
  select sum(feature.ESTIMATED_EFFORT_IN_HOURS)
  into   v_estimate
  from
    wwv_flow_features feature,
    apex_applications application
  where
    feature.application_id       = application.application_id and
    application.application_name = p_application_name;

  return(v_estimate);
end;
```

The project progress can be calculated by summing the product of the progress percentage and the estimate of each feature like this:

```
function project_progress(
  p_application_name in apex_applications.application_name%type)
  return number
is
  v_progress number:= 0;
begin
  select sum(feature.ESTIMATED_EFFORT_IN_HOURS * feature.FEATURE_STATUS /
100)
  into   v_progress
  from
    wwv_flow_features feature,
    apex_applications application
  where
    feature.application_id       = application.application_id and
    application.application_name = p_application_name;

  return(v_progress);
end;
```

Depending on the project governance, you can calculate how many workdays it will

take to complete the project with a given number of developers (project_estimate('project X') / hours_per_workday / no_of_developers), or how many developers you will need to complete the project on a given date (project_estimate('project X') / hours_per_workday / no_of_workdays).

You can also calculate the expected progress at any given time (hours_per_workday * no_of_developers * no_of_completed_workdays) and compare it to the actual progress (project_progress).

APEX ships with some simple project management screens. With a few extra hand built screens you will be able to publish all the key performance indicators needed for managing the project without the need for managing anything in a separate tool, like MS Project. This means that ASD will help you to always keep your estimates and progress calculations “up to the minute”.

Looking into the future: Business Rules

Traditionally, data modeling and process modeling have been considered the most important disciplines in system/business analysis but to me business rules should be considered on an equal level, with tools that can diagram and store business rules in a structured way.

Everything that can be expressed and stored declaratively as business rules will both improve data quality (as some business rules can be implemented as database constraints – example: every site of type ROOM must be located within a site of type BUILDING), and simplify process modeling (as business rules that are traditionally expressed as complicated process steps can be extracted from the process models – example: persons that are married, own their own home, and have a fixed income have a HIGH loan rating).

An ASD methodology would expect business rules to be mapped to things like constraints, triggers, functions etc. Hopefully, modeling tools such as SDDM will evolve to also cover this important part of ASD.

Conclusion

I hope to have given you an impression about the value of Architectural Software Development methods, and given you some new ideas about reuse for your next application development project.

I wish to thank the entire development team at DIS for a great cooperative effort on the DIMS project where everybody were open to think outside the box, and invent a whole range of advanced tools and features to automate and reuse code in new ways that ensure high quality in documentation, coding, and end user interfaces. I have to specifically state a special thank you to Martin Nielsen of MBN Data for his valuable insight into the APEX internal data structure and API that made it possible to reach this high level of integration between SQL Developer Data Modeler and Application Express.

If you have any questions or views about ASD and the areas I have covered in this article, I shall be happy to hear from you.

Send your feedback to Marc(at)SimplifySys.com.