OpenText™

# OpenText™ Process Suite Platform Architecture

Learn how the platform enables customers to improve their business operations

This whitepaper provides an introduction to the OpenText Process Suite Platform (Process Platform) architecture and explains how Process Platform uniquely enables customers to improve their business operations. It is intended for solution architects and other technical people who want to obtain a thorough understanding of the technical aspects of Process Platform. This paper dives into the technical capabilities of Process Platform and how these capabilities are provided.

OPENTEXT™

# Table of Contents

# Architecture Vision and Goals

At OpenText, our mission is to help our customers improve their business operations with world-class, process oriented software that they can use to change and innovate the way they do business with greater speed and flexibility.

This mission is translated into the following set of architecture goals:

| | |
|---|---|
| **INTEGRATED PLATFORM** | Results in simplified installation and maintenance, thus reducing total cost of ownership. |
| **BROWSER-BASED ACCESS FOR ALL USERS, INCLUDING ADMINISTRATORS AND APPLICATION DESIGNERS** | Allows any user, inside or outside the company, to work with Process Platform with just a browser. |
| **APPLICATION DEVELOPMENT FOR TECHNICAL AND NON-TECHNICAL USERS** | Bridges the gap between business and IT by enabling non-technical users to participate in the development process. |
| **INTEGRATED IN ENTERPRISE INFORMATION MANAGEMENT (EIM) SUITE** | Enables creation of EIM solutions that leverage products like OpenText™ Content Server, OpenText™ Media Management and many other products from the OpenText portfolio. |
| **STANDARDS COMPLIANCE** | Enables easy integration, thus reducing total cost of ownership. |
| **EXTENSIBLE ENVIRONMENT** | Drives total cost of ownership down. |
| **INTERNET AND INTRANET DEPLOYABILITY** | The same platform can be used for cloud computing and on-premise. |
| **LINEAR SCALABILITY** | Enables use of commodity hardware, thus keeping total cost of ownership low. |
| **HIGH AVAILABILITY** | Provides high availability for business critical systems. |
| **MULTITENANCY** | Enables cloud computing. |

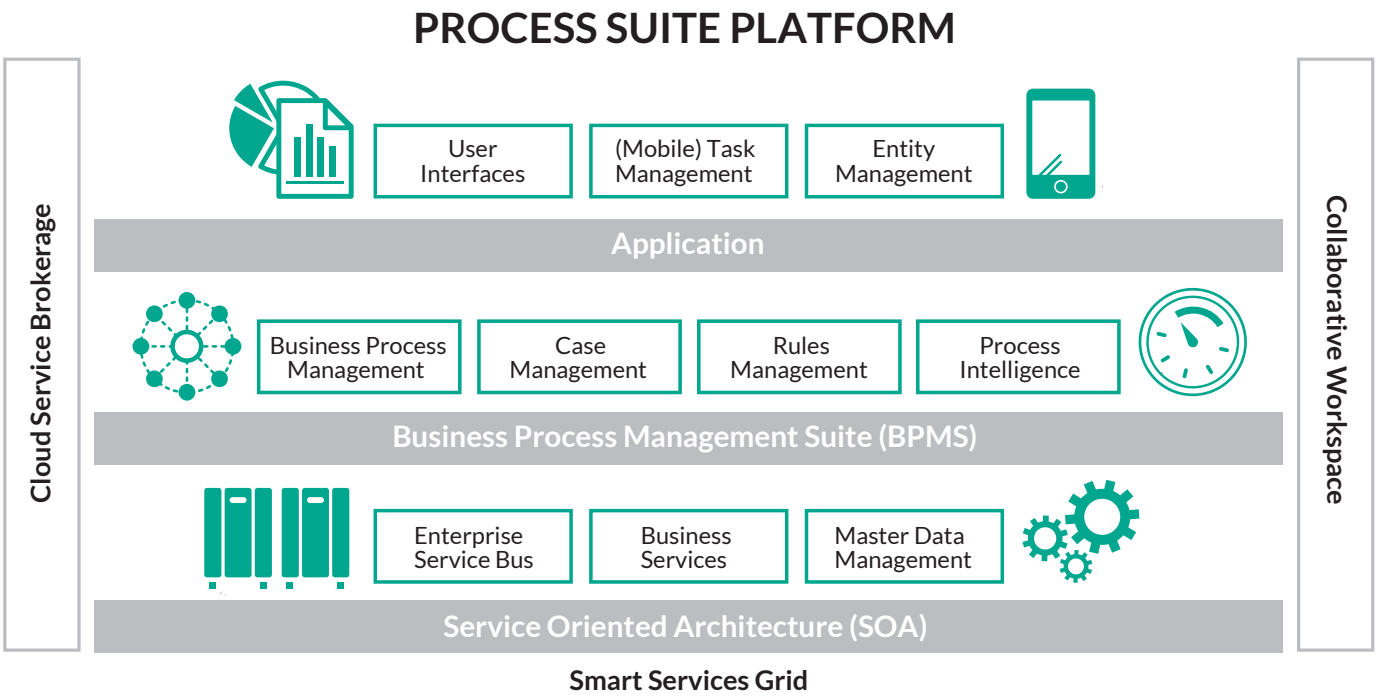
# Process Platform Overview

Process Platform delivers the power and flexibility to digitize, automate and integrate processes across functions, systems, machines and clouds. These processes can be structured or unstructured giving you ultimate control to optimize your business' performance and expand its reach. Process Platform has been built as a single product. All features are based on one technology, thus simplifying the experience. The diagram on this page provides an overview of the platform components.

OpenText is unique in that it has designed a single platform capable of bridging three different process management worlds: **Low-code application development**, **Case and Process Management** and **SOA-based Integration** (see the three platform layers in the previous diagram). Most processes that span a business are at times structured (process) and at other times more ad hoc (case). Being able to manage processes as they go from structured to case and back again supports the reality of business. Companies benefit from the simplicity of a single system to run their businesses with the flexibility to do so as they need. Additionally, OpenText™ Cloud Service Brokerage is built on top of Process Platform to provide automated provisioning and metering of applications for the cloud.

This architecture whitepaper focuses on Process Platform.

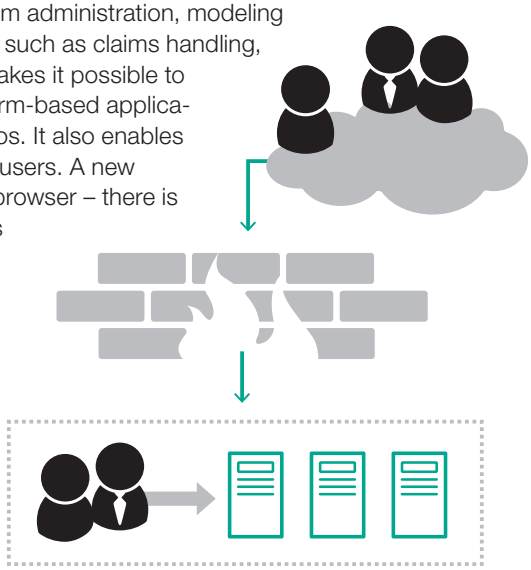
A key objective of Process Platform is to simplify the development process with a low-code development option. Business users can participate in model-driven application development with an approach that is more intuitive for them. We call this approach "information-driven design" and it is an alternative way of thinking about the process. Business users can start with the information they want to manage vs. the process flow. This a different approach to the traditional process-centric development, where platforms enable business people to participate by contributing models that programmers take as input. Process Platform takes a radically different approach – the model is t*he application*, not merely an input to a programmer. To enable that, Process Platform application development is mostly model driven.



The Process Platform modeling environment is built as an application on top of the Process Platform runtime environment. This delivers the following benefits:

- **Scalable, robust, and secure.** All platform runtime features, such as scalability, high availability, and security directly contribute to the design-time environment.
- **Testable.** No need to install another product to enable testing.
- **Available everywhere.** Every Process Platform installation comes with built-in design capability.

The full functionality of Process Platform is available through a variety of completely browser-based user interfaces. Be it system administration, modeling of applications, or an end user application such as claims handling, all interactions are browser-based. This makes it possible to deploy Process Platform, as well as platform-based applications, in both Internet and intranet scenarios. It also enables quick and hassle-free involvement of new users. A new participant in a project needs only a web browser – there is no need to install anything locally. Process Platform provides support for the industry's most popular browsers, Chrome™, Microsoft® Edge, Firefox®, Microsoft® Internet Explorer and Safari®.



The design-time environment is based on the platform runtime environment. This however, does not imply that the application needs to run in the environment where it was designed. Standard development practice is to employ a DTAP setup (Development, Testing, Acceptance, and Production[1]). Different environments are used for the different phases of a software development cycle. The platform has provisions to package and deploy applications to facilitate this approach.

The Internet deployability of Process Platform enables two scenarios:

- **Cloud computing.** In the same way that the OpenText Cloud Service Brokerage solution is built as an application on top of Process Platform, any cloud application can leverage the Internet and multitenancy features of Process Platform.
- **Classic B2B or B2C.** Classic business-to-business and business-to-consumer applications can be built using Process Platform. Such scenarios are usually not multitenant; the application is hosted on-premise at the owning company, but exposed to the Internet.

Process Platform provides the following basic features:

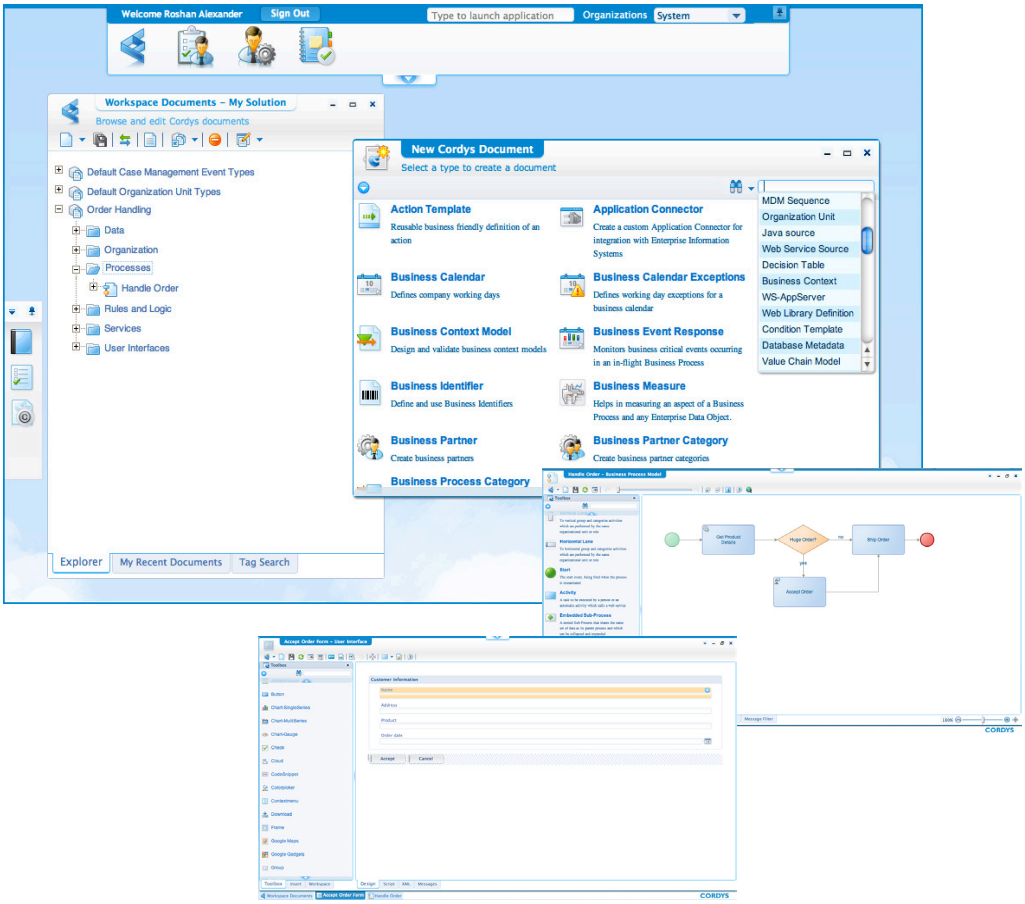
- **High availability.** Mission critical applications must always be available. Process Platform can be deployed on a network of systems, ensuring there is no single point of failure.
- **Scalability.** Enterprises handle thousands of business processes a day. The platform is built for this task. It scales vertically (scale up), as well as horizontally (scale out). Horizontal scalability is accomplished with just commodity hardware.
- **Multitenancy.** Cloud computing scenarios demand multiple organizations, called tenants, to share the same infrastructure. Multitenancy is a basic feature of Process Platform that can also be useful in some on-premise scenarios.
- **Security.** With cybercrime being very common, it is crucial to harness the system appropriately. Process Platform has an advanced set of security measures, including access control lists, auditing, encryption, and sandboxing.
- **Service orientation.** Service-oriented architecture[2] is the predominant design principle for modern enterprise systems. Service orientation belongs to the very core of the platform. All interactions are done through services.

# Design-Time Architecture

This chapter drills a little deeper into the design-time architecture of Process Platform. It introduces the model-driven approach and then describes the highlights of the integrated metamodel, the approach to team development, and the anatomy of a modeler. A closing section provides an overview of the standard facilities of the Process Platform Collaborative Workspace (CWS).

## Model driven

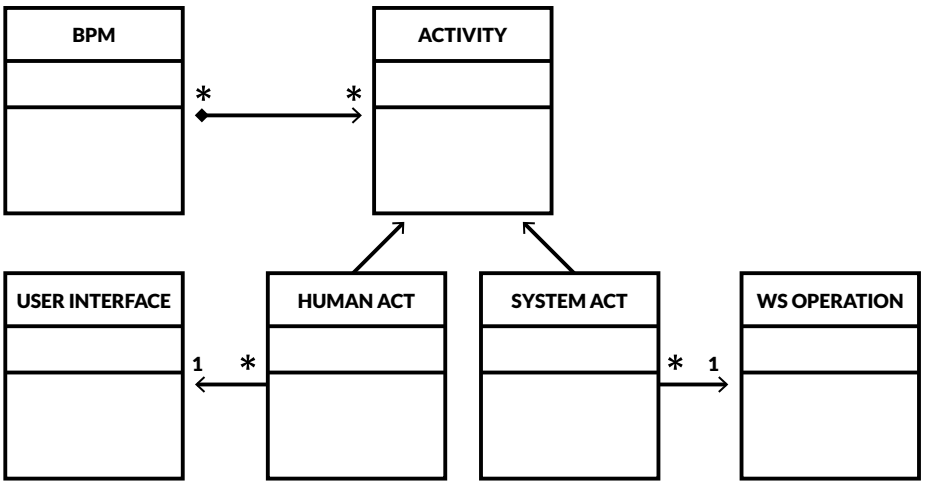
Process Platform takes a model driven approach to application development. A key principle is "What you model is what you execute." All modeling activities are done in the Collaborative Workspace (CWS), a browser-based integrated modeling environment allowing definition of all kinds of models: business process, user interface, entity, WSDL, and so forth. Model developers use a browser to create new models or optimize existing ones. Most models are represented graphically, featuring a responsive, rich user interface.




## Integrated metamodel

The models are all based on a single integrated metamodel. This model not only captures the structure, but also information on the (graphical) visualization, constraints, the building and packaging procedure, and so forth. A key objective of CWS is to guard consistency. An application that successfully passed the build step should be deployable. This is particularly important during refactoring. Renaming a model should not cause an inconsistency where the referrer holds on to the old name while the referee has a new one. Each model has an associated Java class that takes care of the runtime behavior inside the CWS service. The XML document representing the models is stored in a relational database through XDS (for more information see Repository).

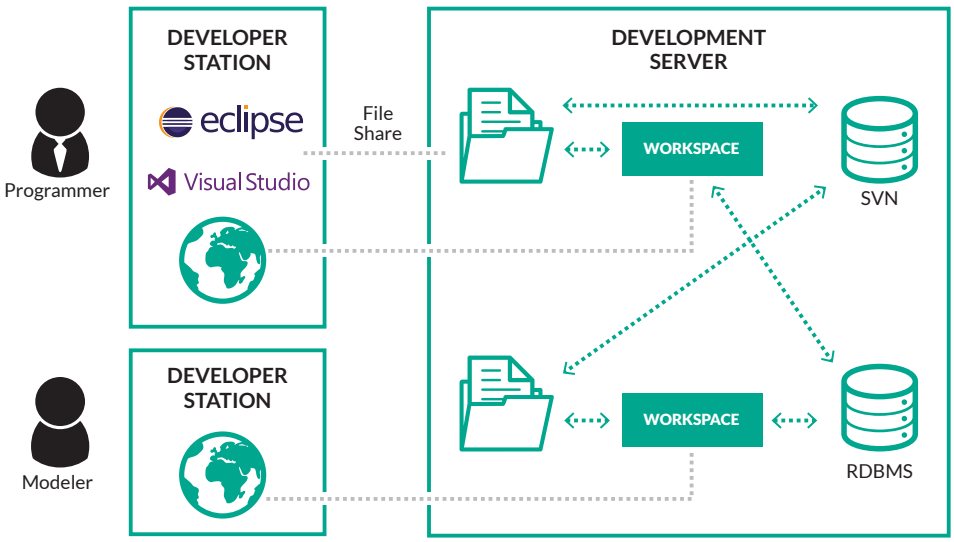
A simplified view of a part of the metamodel looks like this:




### Team development scenario

A business application of reasonable size requires a team of people to model it. Development teams will want to use a Software Configuration Management (SCM)[3] product to keep track of revisions of the models and code. To leverage "best of breed" Software Configuration Management tools, CWS has been designed in such a way that SCM products can be plugged into it. One plug-in is available out of the box that supports Subversion[4]. The SCM interaction is implemented through what is called the file synchronizer. This facility keeps a file system directory and a CWS workspace in sync. The standard SCM features, such as check-in and check-out, are available to all users as part of the CWS browser interface.

Some team members might be programmers, using Eclipse™[5] or Microsoft® Visual Studio®[6]. Such file-based Integrated Development Environments (IDEs) are also supported through the file synchronizer of CWS.



### Anatomy of a modeler

Process Platform comes out of the box with several modelers for Business Processes, User Interfaces, Entities, and so forth. This section looks at the anatomy of a modeler and explains how to build one.

The first step is to define the class definition of the model (that is, the metamodel) along with its view definition. The view definition defines the user interface for this model. Both definitions are expressed in XML.

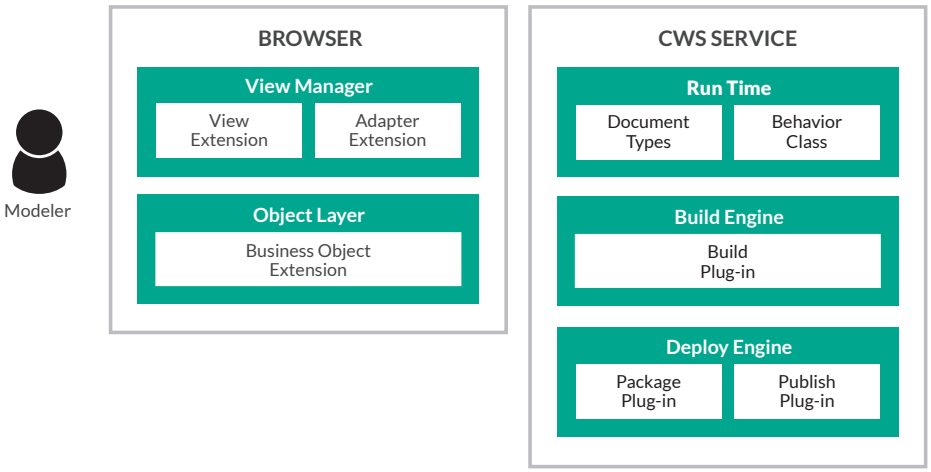
**Class definition**

```
<modelerdefinition
 xmlns="http://schemas.cordys.com/CWS/ModelerDefinition/1.0"
 version="1.0">
 <package
  namespace="com.cordys.xforms"
  version="1.0">
  <class
   name="XForm"
   description="XForm"
   displayName="XForm"
   displayNamePlural="XForms"
   id="{78817809-7334-4903-9300-086883918845}">
   <extend
    class="URLBasedForm"
    package="com.cordys.uiapplication"
   />
   <properties>
    <property
     name="qualifiedclassname"
     type="String" />
   </properties>
   <associations>
    <association
     role="Webservices"
     aggregation="none"
     class="WebserviceBindingOperation"
     package="com.cordys.wsdl"
     minOccurs="0"
     maxOccurs="*"/>
   </associations>
  </class>
 </package>
</modelerdefinition>
```

**View definition**

```
<viewDefinition
 xmlns="http://schemas.cordys.com/CWS/ViewDefinition/1.0">
 <icons
  package="com.cordys.bpm">
  <icon
   name="bpm"
   url="/cordys/wcp/theme/default/icon/document/businessprocess.png" />
  <icon
   name="bpmRuntimeIcon"
   url="/cordys/wcp/theme/default/icon/document/businessprocess_runtime.png" />
  <icon
   name="messageIcon"
   url="/cordys/cas/images/messagemap.gif" />
 </icons>
 <relatedClass
  class="BusinessProcess"
  package="com.cordys.bpm"
  icon="bpm">
  <viewPackage
   namespace="com.cordys.bpm.views.explorer">
   <treeView>
    <treeNode
     treeNodeAdapterClass="BusinessProcessTreeNodeAdapter"
     icon="bpm"
     expandable="true">
    </treeNode>
   </treeView>
  </viewPackage>
 </relatedClass>
</viewDefinition>
```

These two definitions form the input to a generator that produces classes such as view extension, adapter extension, build plug-in, and so forth to use in the front end and back end of CWS. The modeler developer adds the custom behavior as needed, and then builds the modeler. The following illustration shows where the various classes are used when running CWS.



### Standard CWS facilities

The CWS framework provides a set of standard facilities to all modelers:

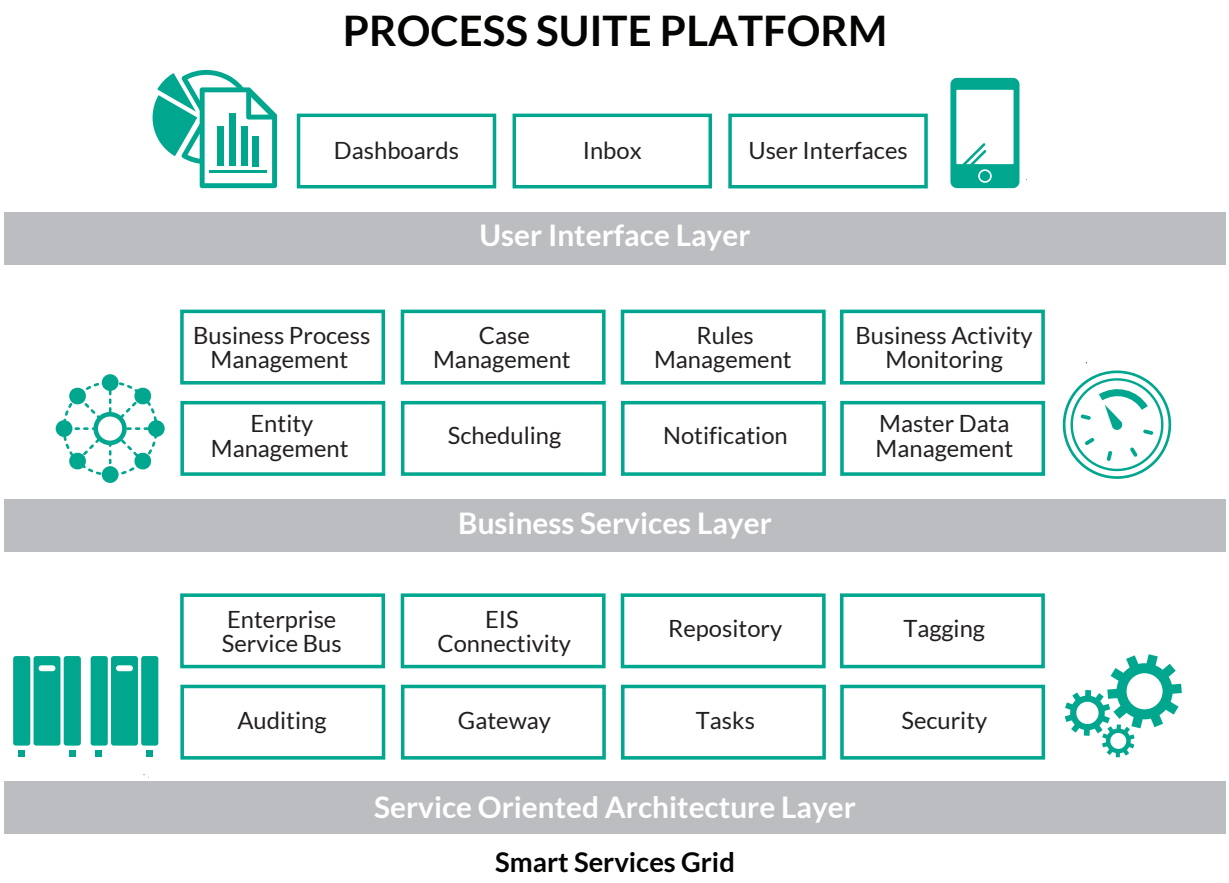
- **Where used.** Based on the associations between models, as expressed in the metamodel, CWS automatically provides "where used" overviews. This is an important tool during impact analysis and refactoring.
- **User interface.** Common look and feel across modelers is facilitated through an advanced UI framework.
- **File system synchronization.** The content of a CWS workspace can be synchronized to a file system directory, to enable interaction with SCM tools, and for use of file-based editors and IDEs. It also helps in easy exchange of workspace content.
- **Import/export.** A plug-in framework supports generic import/export of models. XPDL import/export ships out of the box. Other import/export plug-ins can be developed as needed.
- **Build.** The build engine takes care of building all changed documents. The extent of what is to be built is determined based on the associations between the models. This prevents unnecessary rebuilding of unchanged models. The actual build behavior is specifically implemented for each model.
- **Package.** The packaging framework takes the build output of all models of a project to create the deployable package (CAP).
- **Publish.** For testing purposes, models can be directly published to the development system. This is taken care of by publishing the framework. This framework also uses the associations to determine what needs to be published.
- **Tagging.** To enable easy look-up of models, it is possible to attach user defined tags to them. This is a standard facility of CWS.

# Runtime Architecture

At runtime, Process Platform comprises a set of web service containers connected through a SOA grid. This section provides a look at the logical view, followed by the deployment view, an explanation of Process Platform multitenancy and an overview of the runtime services.

### Logical view

A logical view of the Process Platform architecture has a strong resemblance with the illustration provided in the high-level overview of Process Platform, although there are a few differences that do justice to the technical aspects.




Each layer is described as follows:

1. **User Interface Layer**
   The User Interface Layer contains all the User Interface components, such as dashboards and the Inbox, but also the application user interfaces developed through Process Platform. These user interfaces are built on top of the business services as defined in the next layer.
2. **Business Services Layer**
   The Business Services Layer hosts services relevant to the business domain; notable examples being Business Process Management (BPM), Case Management, and Entity Management. These services are all built on top of the SOA layer.
3. **Service Oriented Architecture Layer**
   The Service Oriented Architecture Layer provides the fundamental SOA grid functionality used by the other layers.

Each component is described in more detail later in this section.

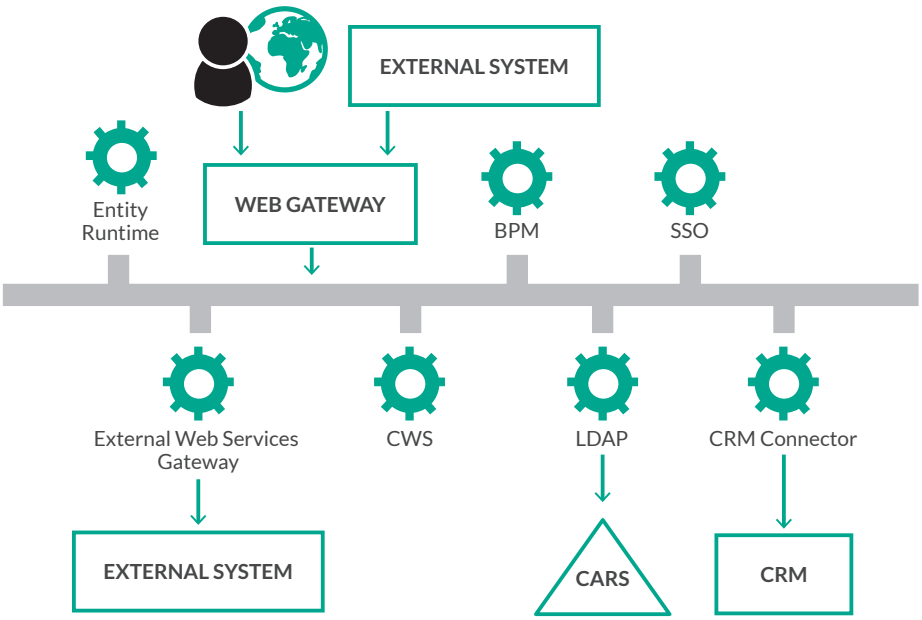
### Deployment view

All runtime components are linked through the Process Platform SOA Grid. The SOA grid provides three main facilities.

1. **Routing of SOAP messages**
   The SOA is entirely SOAP-based. The services deliver their XML messages to the Enterprise Service Bus (ESB). Based on the service registry, stored in an LDAP[7] directory called CARS, it knows the details of all the services. Given the required quality of service and whether to use a reliable transport, it chooses a channel and delivers the message to the recipient. The messages can be transferred over a variety of protocols, ranging from plain TCP/IP sockets to message queues.
2. **Load balancing**
   As the load increases, not everything can be handled on a single system, so multiple systems might run the same service, thus sharing the load. The ESB has pluggable load balancing algorithms to decide which service instance to address.
3. **Failover**
   If one of the service instances fails, load should immediately be moved to other instances and business should go on as usual. This is taken care of by the failover features of the ESB.

Details of these features are discussed in Enterprise Service Bus.

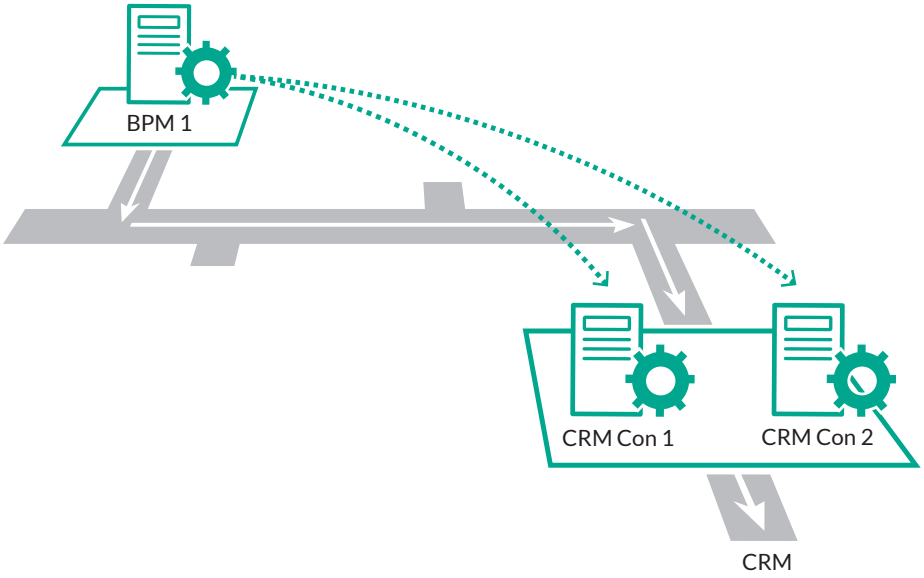
The following diagram provides a 'bus view' of Process Platform:




All participants on the bus are equal, so there is no central bus coordinator or single point of failure. The web gateway is depicted with a different icon because of its special role, it only acts as a bus client and does not have a role in requests between the peers. The service icons in the diagram represent a random set of the service groups in a standard Process Platform system. A *service group* denotes a conceptual service. The actual implementation is through a service container. To provide load balancing and fast failover one service group might be implemented through multiple service containers, most of the time running on different systems.

## DEPICTED SERVICES

- ***Entity Runtime.*** *This component hosts the entities with their building blocks. See Entity modeling and runtime.*
- ***Web gateway.*** *This actually is not a service on the bus but just a client. It provides an access point for users and external systems using the Process Platform web services.*
- ***External web services gateway.*** *This service, often called the UDDI connector, links external web services to the bus.*
- ***LDAP.*** *Process Platform uses an LDAP directory, called CARS, as the runtime repository for certain types of information. This repository is accessed through the LDAP service.*
- ***CRM connector.*** *The Customer Relationship Management (CRM) connector is an example of a connector to any Enterprise Information System (EIS) or legacy system. Using Process Platform, connectors can be developed for the various enterprise information systems in an organization. This connector bridges the proprietary protocol of the EIS with the bus.*

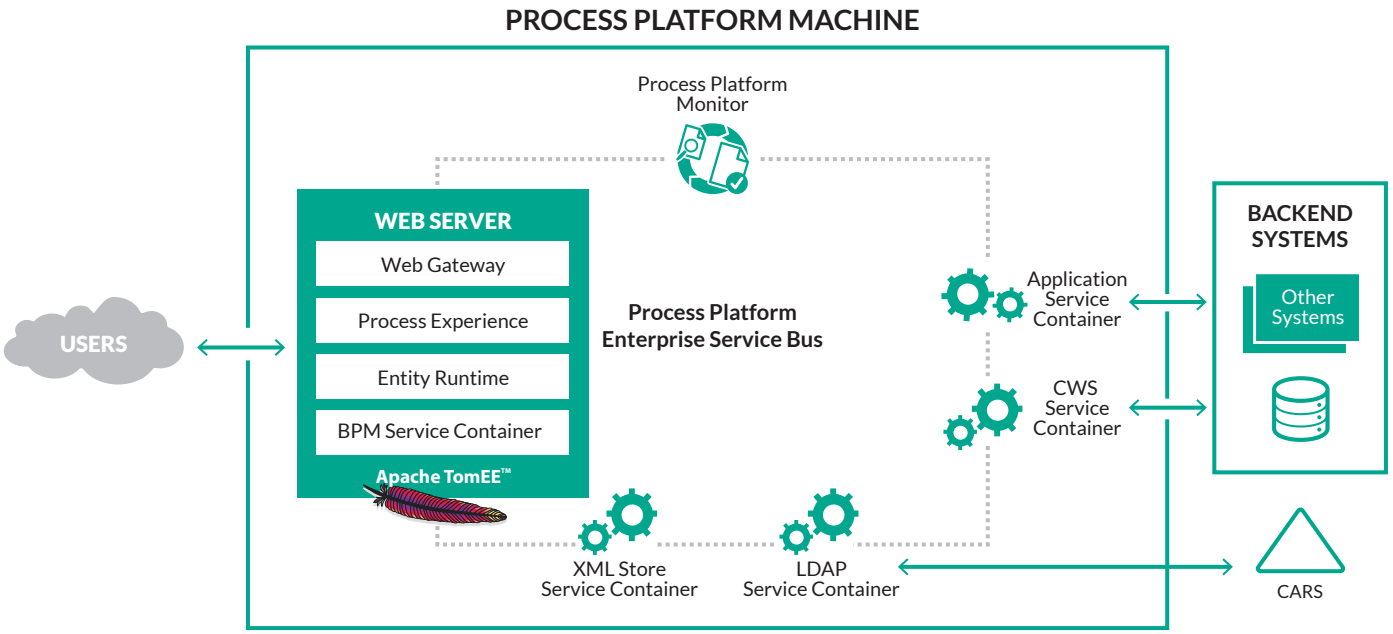



The previous illustration gave an abstract representation of the relationship between services. The following illustration provides a concrete view of the interaction between two services in an example deployment configuration of the BPM engine and a CRM connector. The BPM engine is deployed as a single service container on one system, whereas the CRM connector is deployed as two service containers, each running on its own system.

The BPM engine directly connects to one of the two CRM connectors to submit its request.

### Node view

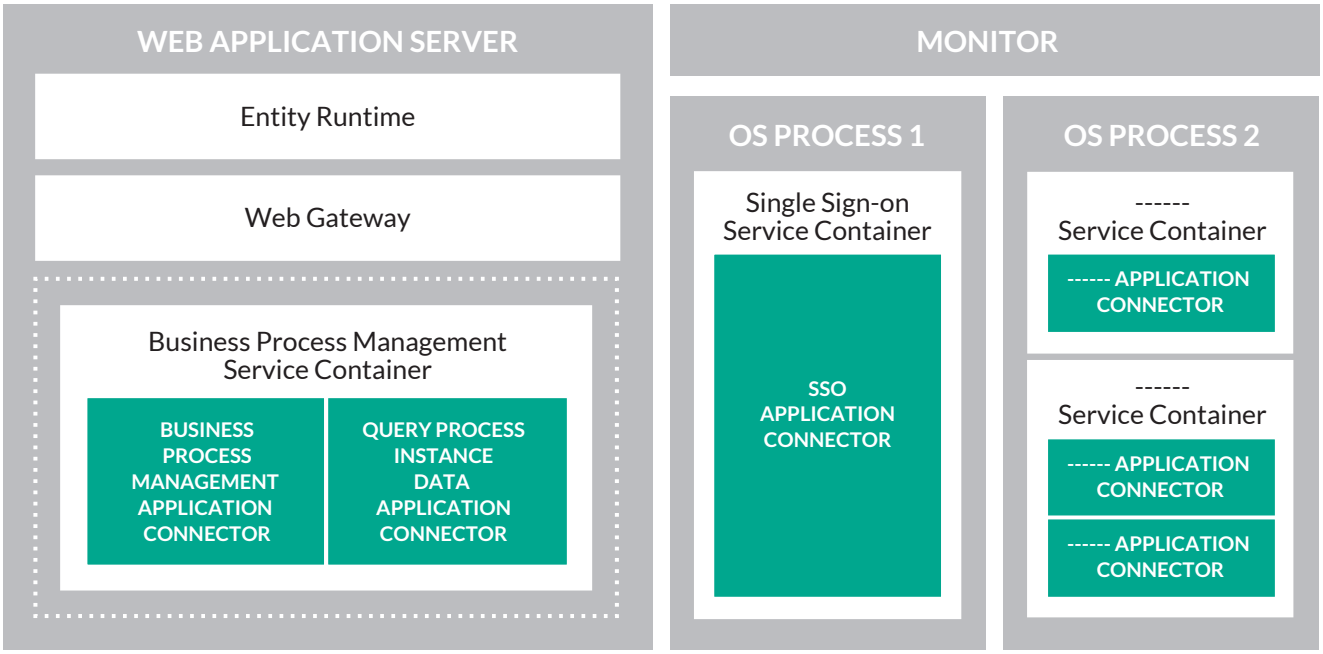
The following illustration shows schematic view of a typical Process Platform node:



The web application server (currently only Apache® TomEE® is supported) acts as a front end for the Process Platform node and it hosts a number of components:

- **Web applications.** A number of Process Platform components is implemented as Java EE web applications. The most notable example is Entity Runtime.
- **Service containers.** Traditionally, all service containers were hosted in processes managed by the so-called Monitor. More recent versions of Process Platform allow running service containers in the web application server, thus enabling a transaction scope that spans multiple web applications and service containers. The most notable example is the Case/BPM engine, which works in close cooperation with Entity Runtime, to support the Lifecycle building block.
- **Web Gateway.** All requests from browsers and external systems enter Process Platform through the web gateway.

Service containers are managed by the Process Platform Monitor. This is a Linux® daemon or Windows® service responsible for the lifecycle of the service containers, both inside and outside the web application server. For service containers, it is indifferent whether they run inside the web application server or outside it. Both types of processes provide an identical execution environment, as visualized in the following picture:



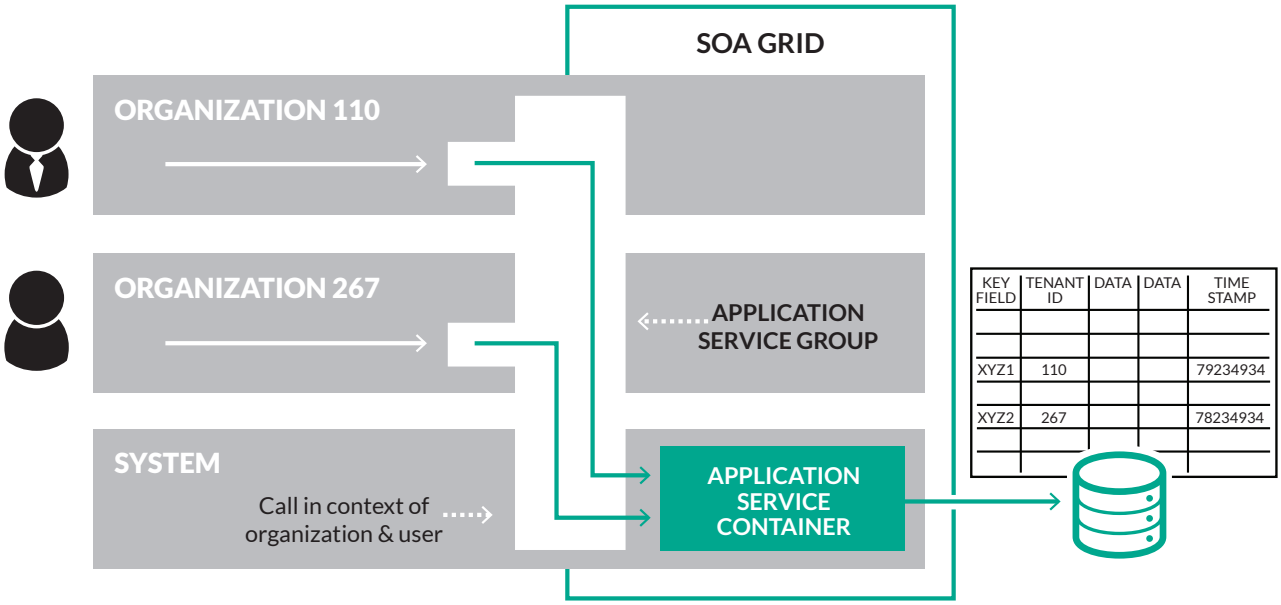

It is the administrator's choice to assign service container to operating system processes. Deciding factors include:

- **Memory consumption.** Colocating service containers reduces the memory footprint, but putting everything in a single process will make the footprint of that process large.
- **Fault isolation.** If for some reason a process crashes, it will take all service containers in it down.
- **Transaction scope.** Service containers within an OS process can invoke each other as part of a single transaction.
- **Efficiency.** Web service requests within an OS process are more efficient.

### Multitenancy

The Process Platform SOA Grid represents the physical or deployment aspect of the Process Platform architecture. The organization notion represents a logical concept in the Process Platform architecture. All functionality is invoked in the context of a user and the organization of that user. The organization is not really located on a particular node. All service containers, wherever located, can execute the functionality on behalf of a user in the context of that organization. So, if a user starts a user interface it will be in the context of an organization of which the user is a member. And if the user interface invokes a call on a service container, it will execute in the context of that organization and user. Before it is executed, the role of user is validated against the **Access Control List** of the service. If the user does not have the required authorization, the logic will not be executed. A single user can exist in multiple organizations and have different roles in each organization.

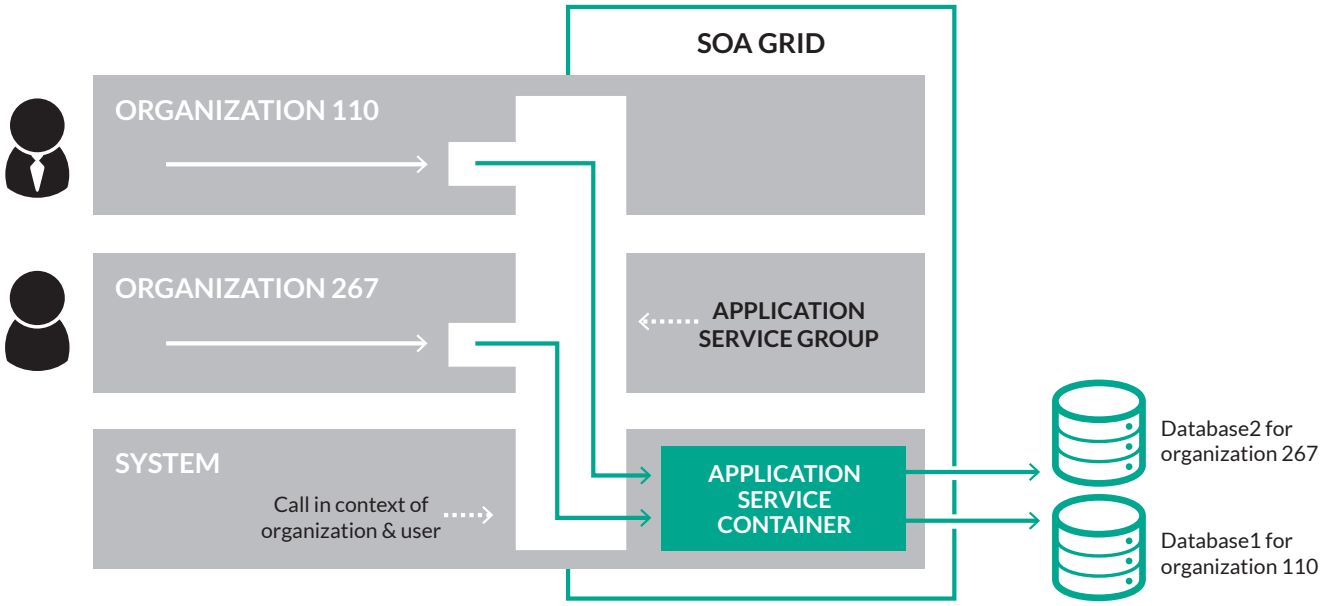
Service containers exist in the context of an organization. The functionality in a service container is exposed via a SOAP or REST interface. When a SOAP call is initiated, it is always done in the context of an organization. For efficiency and reuse, a common shared organization called **System** acts as a fall back mechanism for other organizations. If the service is not implemented in the current organization, the call is delegated to the service container in the System organization and executed there, but still in the context of the invoking user and organization.




All Process Platform service containers are organization aware and use multitenant data stores to persist their data. When an **Independent Software Vendor (ISV)** builds a new application, two options are available to deal with multitenancy in the datastore:

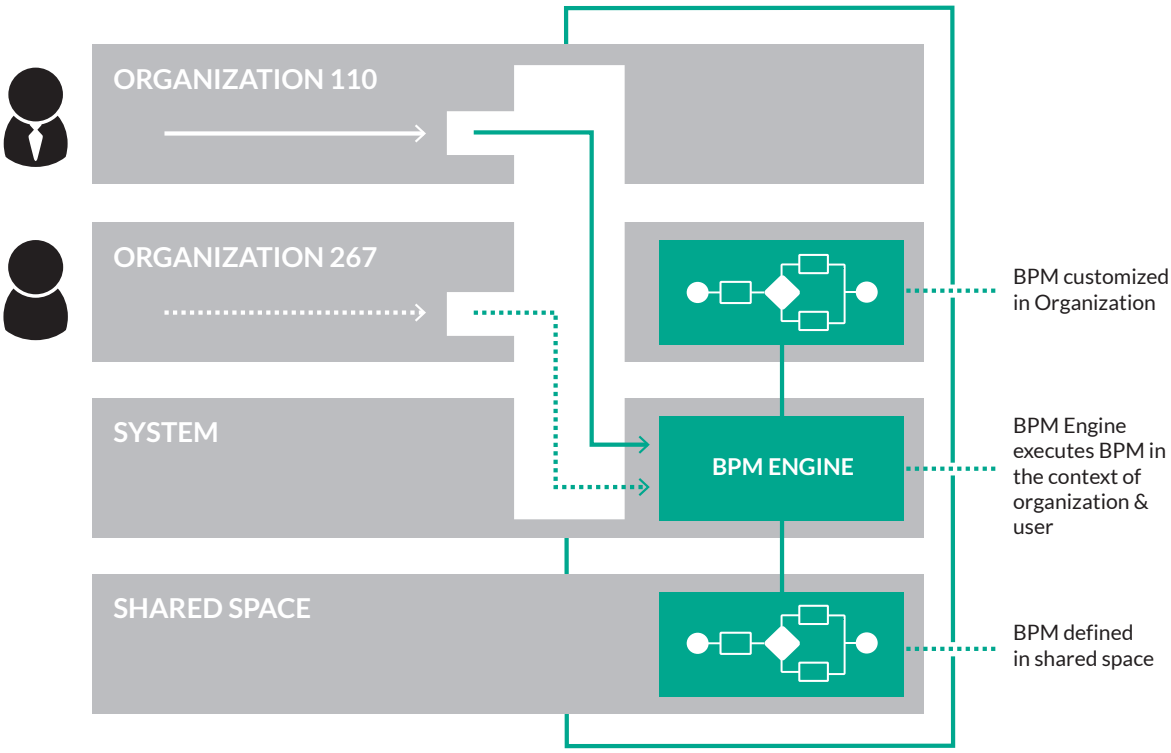
- Support multitenancy as part of the database schema, as depicted previously. The service container of the application can be deployed into the System organization to allow all organizations to use this new application.
- Provision a separate database (schema) for each tenant. Entity Runtime and WS-AppServer support per tenant database configuration, enabling segregation of tenant data in separate database schemas. This is depicted in the following diagram.

The content of the application (for example, the business processes and user interfaces) can be deployed for a single organization (or tenant), thus making it available within the context of only that organization. This is called organization level deployment. It typically happens when developing or customizing an application for just one organization.




Alternatively, an application can be deployed for all organizations. In that case, the application content is stored in the so-called Shared space. This space contains content common to all organizations. This is depicted in the following illustration. Although the Shared space is drawn in the same fashion as an organization, it is, technically speaking, not an organization. Content (for example, user interfaces and business processes) is always stored in the context of an organization, or the Shared space. Wherever a service container loads its content from, the context is always based on the organization of the user on whose behalf the logic is executed.

The following illustration provides an example where a business process of an application is customized for one organization. The user of "organization 110" will be using the business process defined in the Shared space, though it will be executed in the context of "organization 110." A user in the "organization 267" will use a customized version of this business process, stored in its own organization.



# Overview of Runtime Services

This section provides further details on each of the components depicted in the Logical view.

### User Interface Layer

The User Interface Layer contains all User Interface components, such as dashboards and the Inbox, but also the application user interfaces developed through Process Platform. These user interfaces are built on top of entities and business services as defined in the next layer.

### User Interfaces

The Process Platform supports User Interfaces of different kinds and technologies. This section describes the User Start Page, the User Interface Modeler, and the Process Experience technology.

#### USER START PAGE

The User Start Page is a generic and task-driven view of the product functionality presented to the user. This view shows a non-hierarchical navigational model with role-based set of tasks.

The key elements of the User Start Page are:

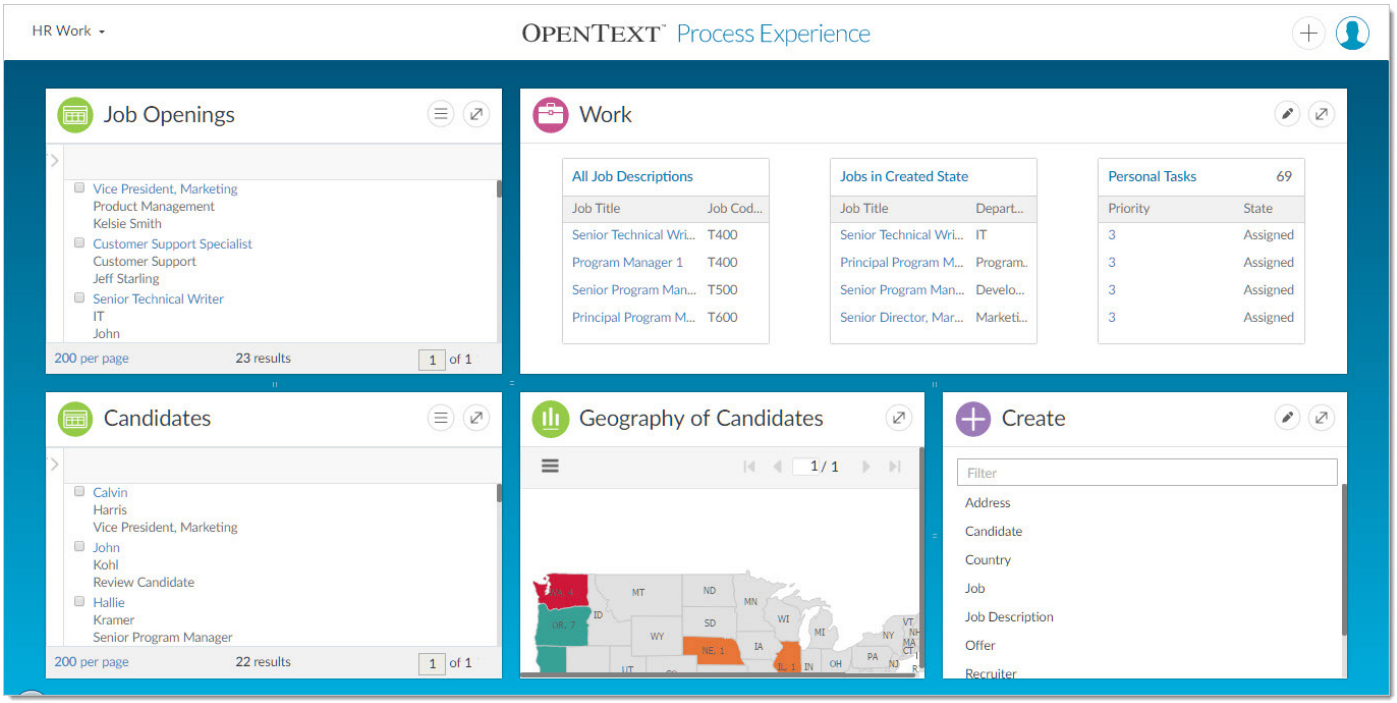
- A **Most Used Tasks** list that consists of the tasks and shortcuts available to a particular user based on the roles and permissions assigned and the usage pattern of the user.
  - This will be shown only as a list. Initially all the tasks will be listed.
  - The most frequently used tasks will move up the list with progressive usage.
  - The most frequently used tasks come with default categories, and are able to be 'Live Categorized' by the user.
  - Typical examples of a task would be user management, create a user interface, create a business process, and so forth.
- A **Most Recently Used** list that summarizes the last and latest used artifacts in the system.
  - This list will be specific to a user.
  - The initial view will become empty and the list of most recently used tasks will appear only as system artifacts are used.
  - The system artifacts listed here come with their default categories, and are able to be 'Live Categorized' by the user.
  - Typical artifacts can be files, forms, business process models, and so forth.
- An **Organization Switcher** enables users with accounts in different organizations to switch to another organization.

#### PROCESS EXPERIENCE

Process Experience structures user interfaces through layouts with different types of panels showing different information. For example, a layout can contain panels to show forms, lists, web content, and so forth. List panels can list entities of different origin, including entities from various business process management and content management systems. Through this, it is possible to design good looking and informative user interfaces that enable users to access disparate systems and applications using a consistent user interface.

*Layouts*

The user interface consists of layouts that organize a set of panels into a page to be presented to a user. There are two types of layouts: entity and home page. Entity layouts are used to display instances of a specific entity and may contain panels that display various aspects of the current entity. Home page layouts are launched with no entity and can only include panels that do not require an entity.




A layout can contain any number of panels arranged as specified at design time. If multiple panels are combined in a single zone in a layout, the layout manager presents tabs for a user to select a panel to activate.

Layouts subdivide screen real estate to present information. However, the amount of screen real estate available varies greatly, as the same layout can be accessed from various displays ranging from a desktop with a huge monitor to a mobile device with a small screen. While a modeling user could design separate layouts for different sized screens, this is tedious and difficult to maintain. To simplify this, the layout manager uses responsive design techniques to automatically rearrange the panels in a layout based on the available screen real estate. The layout is designed for the largest sized screen and the layout manager rearranges panels automatically for smaller screens.

*Lists*

The most common way for a user to interact with the system is by running a list. The function of a list is to filter and present a set of entities (see Entity modeling and runtime). Two classes of entities are supported: native entities, defined through Entity Modeling, and external entities, imported through an Enterprise Information System (EIS) connector. An EIS connector makes data from an EIS available in the form of entities. The growing set of EIS connectors includes connectors to OpenText legacy BPM products, thus enabling a unified user experience across multiple BPM systems.

*Forms*

Typically, when you choose to present entities to a user, you present each entity's properties in a form. The form can present any property from the entity and its related entities. This includes any property from any of its building blocks, not just the custom properties that you define. For example, if an object includes a Title building block that provides a Title property, that property can be added to a form.

A form consists of a set of form components. The types of form components follow:

- **Properties.** A property component presents one of the object's properties. Each type of property has a set of alternative presentations called formatters that can be used to display the property's value.
- **Containers.** Containers are used to add structure to a form. Examples of containers include the tab, stack, and general containers.
- **Decorations.** Decorations can be used to make the form more usable or attractive. Examples of decorations include text, image, and horizontal rules.

### USER INTERFACE MODELER

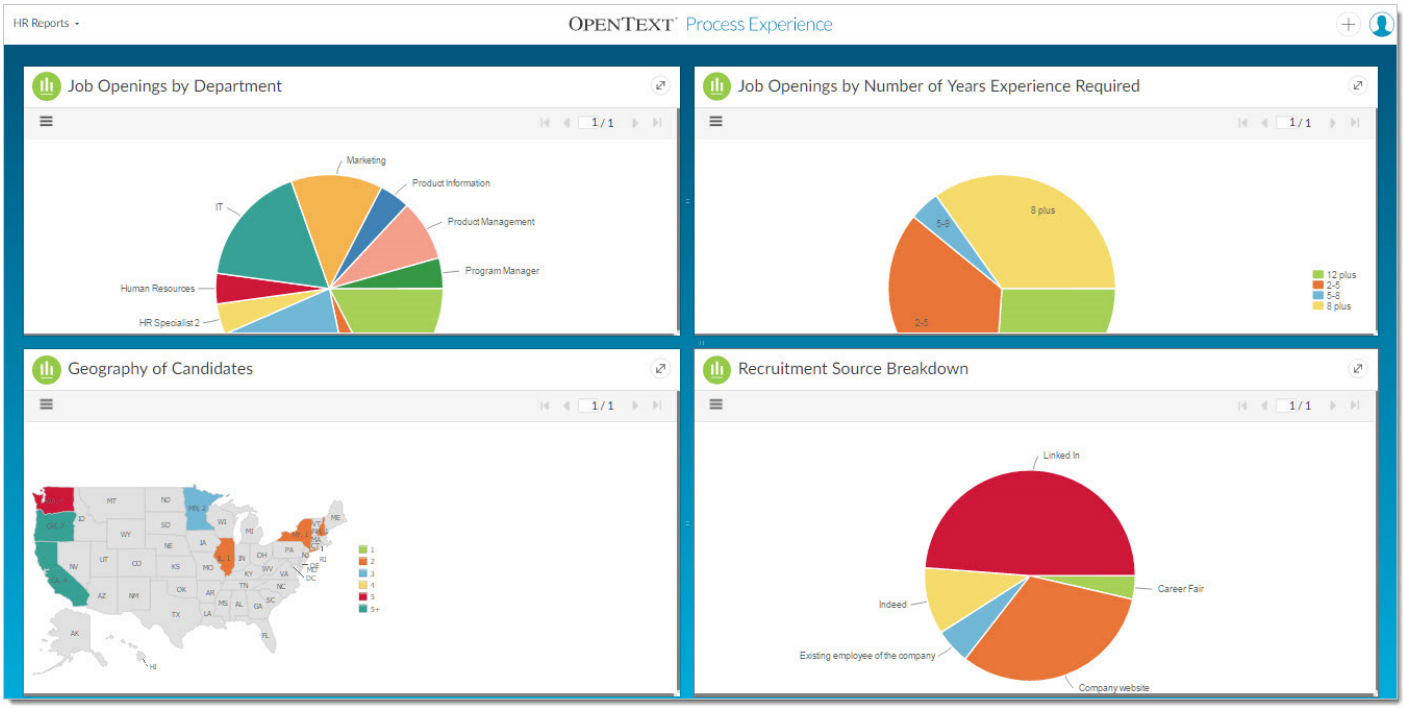
Besides the entity-based Process Experience User Interface (UI) technology, Process Platform also includes a UI modeler to build user interfaces on top of web services. With this modeler, developers can quickly create applications that use the full power of Process Platform, link directly to services running on the Process Platform SOA grid, and provide full transactional capability. The UI modeler features a WYSIWYG (What You See Is What You Get) approach and rich UI controls to streamline and simplify the process of converting ESB functionality into rich, intuitive, process-centric applications that run in a browser environment.

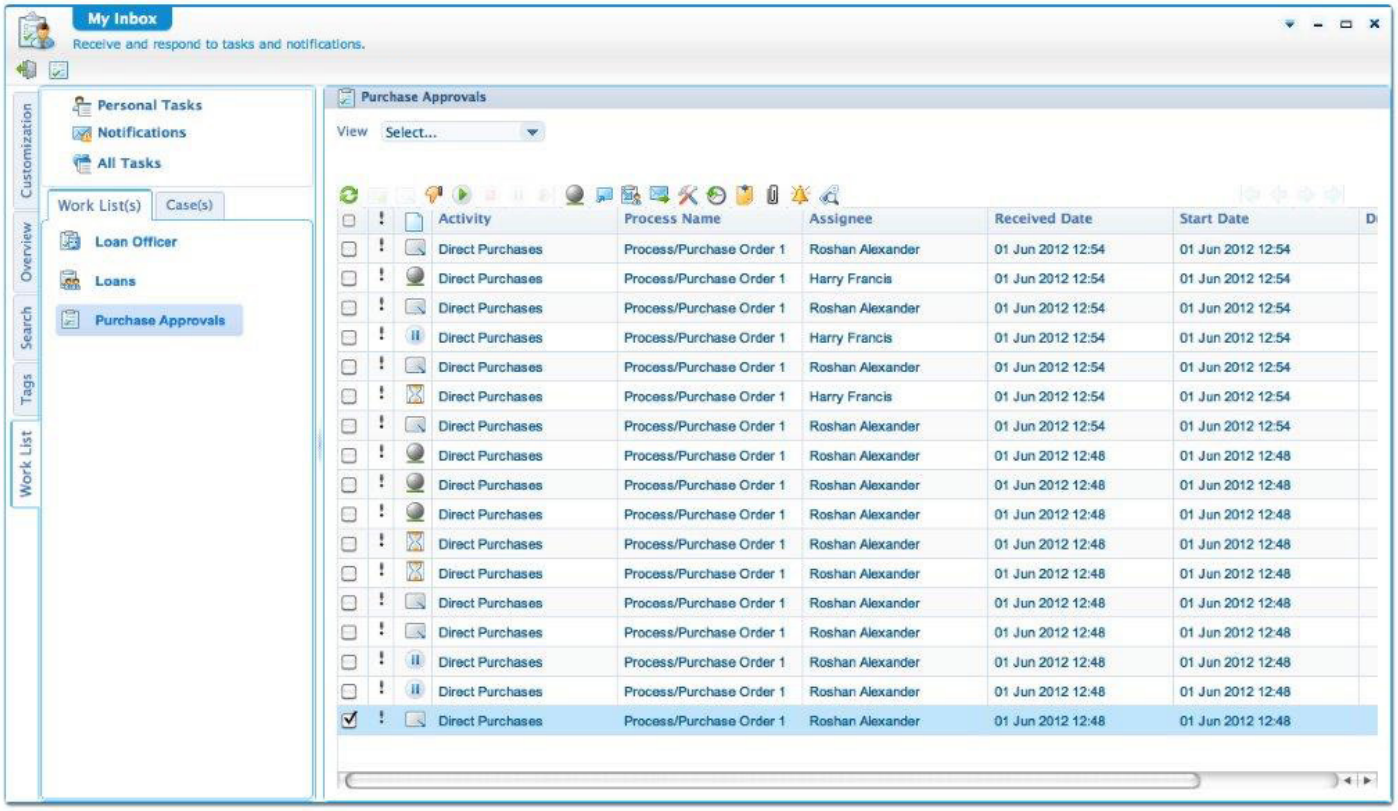
The forms are based on the XForms[9] standard, as defined by W3C. XForms is an XML format for the specification of a data processing model for XML data and user interface(s) for the XML data, such as web forms. Process Platform provides a rich set of UI controls, including basic elements such as check boxes and radio buttons, but also advanced controls such as AppPalette and Google maps™. The environment is extensible through the notion of composite controls. This technology allows building UI controls of any complexity and delivering them as reusable UI controls.

The forms are stored in the XMLstore and delivered through the XForms Service Container. One of the key responsibilities of this service container is to translate the form in the language applicable for the current user. Depending on the language, the form is rendered right-to-left[10], as required for languages such as Arabic and Hebrew, or left-to-right, as required for English and French.

#### Dashboards

Building a dashboard is as simple as building any user interface. Every Key Performance Indicator (KPI) or business measure defined in Business Activity Monitoring comes with a composite control, which provides a chart representation of the business indicator. These charts are placed on a user interface to build a rich and interactive dashboard.






**Inbox**

The most common user interface for Process Platform users is the **Inbox**. The Process Platform provides an advanced one, which is fast and highly configurable to suit everybody's needs.

Case Management integration:

- The Inbox has a seamless integration with the Case Management solution
- Overview of all the tasks to which the user has access, grouped by case model and case instance
- Case Task view providing a consolidated view of the active tasks for a particular case instance
- Complete Case dashboard enabling the user to
  - Plan follow-up activities
  - Raise events
  - Attach documents
  - Perform administrative activities (for example, suspend, resume, forward, delegate)
- Case history view

Customization possibilities in Inbox:

- Visual customization (personal and on role/team/work list level)
- Behavioral customization
- User preference for language and date format
- User-selectable business data
- Extensible for application developers through custom JavaScript code supporting events, such as *onBeforeCommit* and *onBeforeFollowup*

The Inbox interacts with the notification service on the back end to fetch work items and update statuses.

### Business Services Layer

The Business Services Layer hosts services relevant to the business domain; notable examples being Business Process Management (BPM), Case Management, and Entity Management. These services are all built on top of the SOA layer.

#### Business Process Management

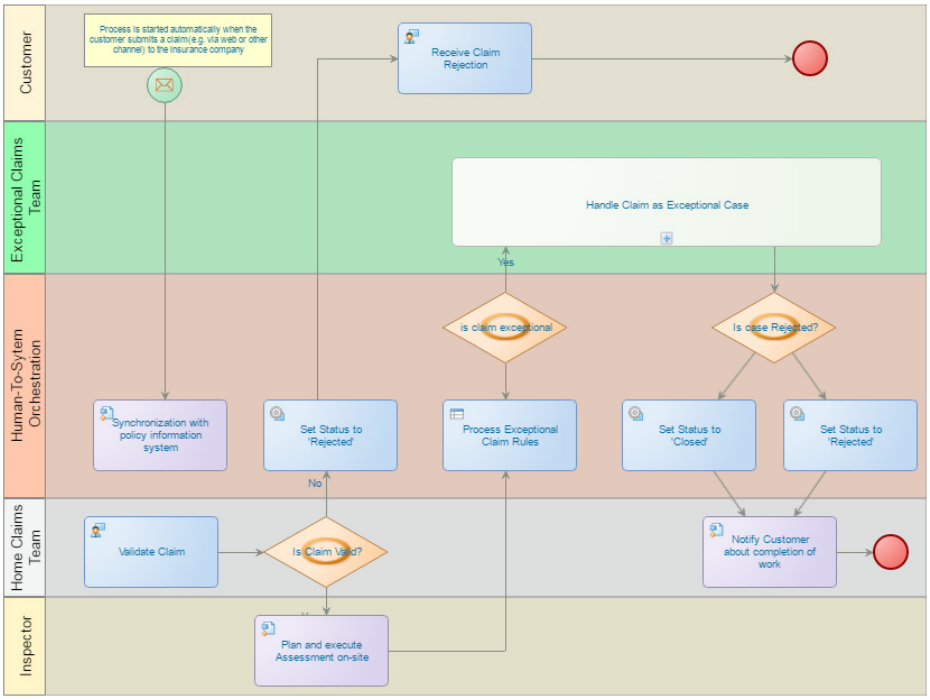
In Process Platform, the BPM engine is one of the most important components. Processes are defined as BPMN[11] compliant graphical models in CWS. The build step on such a model results in an XML definition that is interpreted by the BPM engine at runtime.

Some key characteristics of the BPM engine follow.

- **Supports short-lived and long-lived processes.** Short-lived processes do not involve user interaction and do not log progress information. This makes short-lived processes ideal to express fragments of logic. Long-lived processes log their progress in the Process Instance Manager (PIM) tables and can involve user interaction.
- **Fast and scalable.** If sufficiently powerful hardware is available, a single BPM engine can execute thousands of short-lived business processes per second. If a single system cannot handle the load, the BPM engine can be scaled out to multiple systems.
- **Part of Business Process Management service group.** In a default Process Platform deployment, the BPM engine is part of the Business Process Management service group, together with Case Management. To optimize performance, this service group can run the following services in an embedded mode:
  - Rule Engine
  - WS-AppServer
  - CoBOC
  - Data transformation
- **Process Instance Manager.** Execution of long-lived processes is tracked in the PIM tables. The PIM user interface provides access to this data.
- **Multitenant.** Like any Process Platform service, the process engine is multitenant enabled, so multiple tenants can have their own process definitions and collections of running processes.
- **WS-AppServer Integration.** High speed transactional processing can be accomplished by embedding WS-AppServer in the BPM service container. This allows short-lived processes to directly call WS-AppServer classes in a transactional manner.
- **Crash recovery.** The BPM engine restarts a crashed process from the last recovery point as captured in the PIM.
- **Reliable messaging.** When using a reliable message transport, the BPM engine coordinates the transactions on the PIM tables and the message oriented middleware to ensure a web service operation is executed once, including the handling of the service response.
- **Standard SCXML execution format.** The process models are complied to SCXML[12], a W3C standard.

Following is a screenshot of a real world BPMN flow in Process Platform that handles a long-living insurance claim across different departments and external parties. It is depicted as a yellow swim lane at the bottom representing the Inspector who reviews the incident on-site, per request of the insurance company.

This flow shows clearly that Process Platform orchestrates **human tasks** (via user interfaces) and **system tasks** (such as web services updating the claim status during the claim handling) in one process model.




The earlier mentioned Rule Engine and Data transformation are all invoked from this BPMN flow, not necessarily visible for the business end user, but visible for the functional administrator or process analyst.

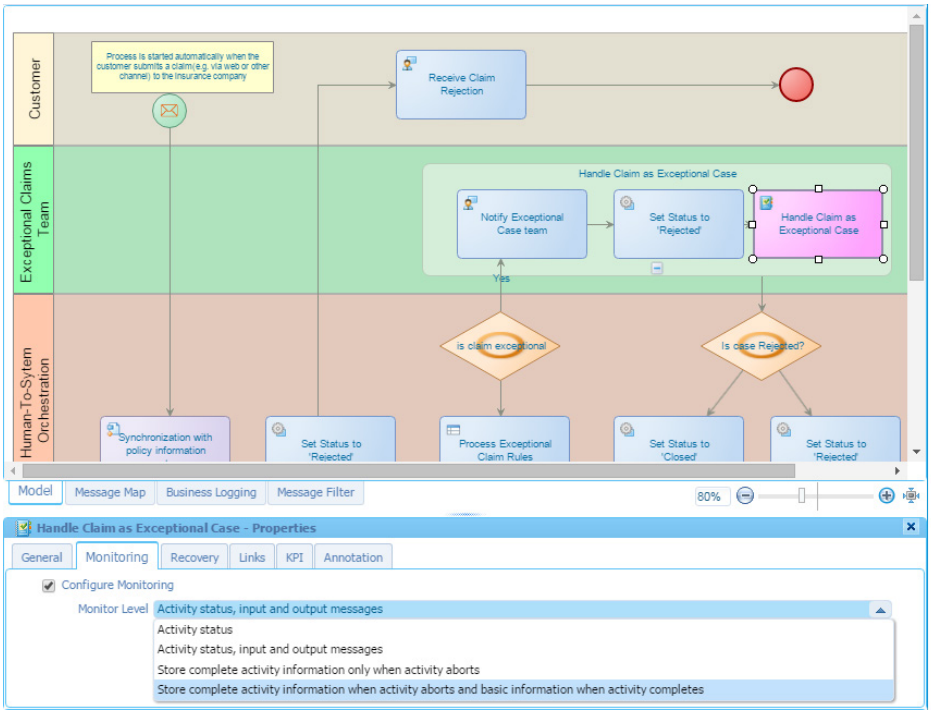
**Case Management**

BPMN is suitable for processes that are well defined and uniform. In day-to-day life, however, processes often vary on a case-by-case basis and the knowledge worker in the process wants to decide on the necessary next steps. This is the "sweet spot" for Case Management. At the core, a case process is a state machine. Modeling users can decide to express their case model as a set of interrelated states, but they can also opt for a simplified model with just one implicit state. The activities of a case model are related to each other through "follow-up relations." The build step of a case model produces SCXML that at runtime is interpreted by the state machine of Case Management. Cordys (now OpenText) is one of the submitters of the Case Management Model and Notation (CMMN) standard.

Some key characteristics of the Case engine follow.

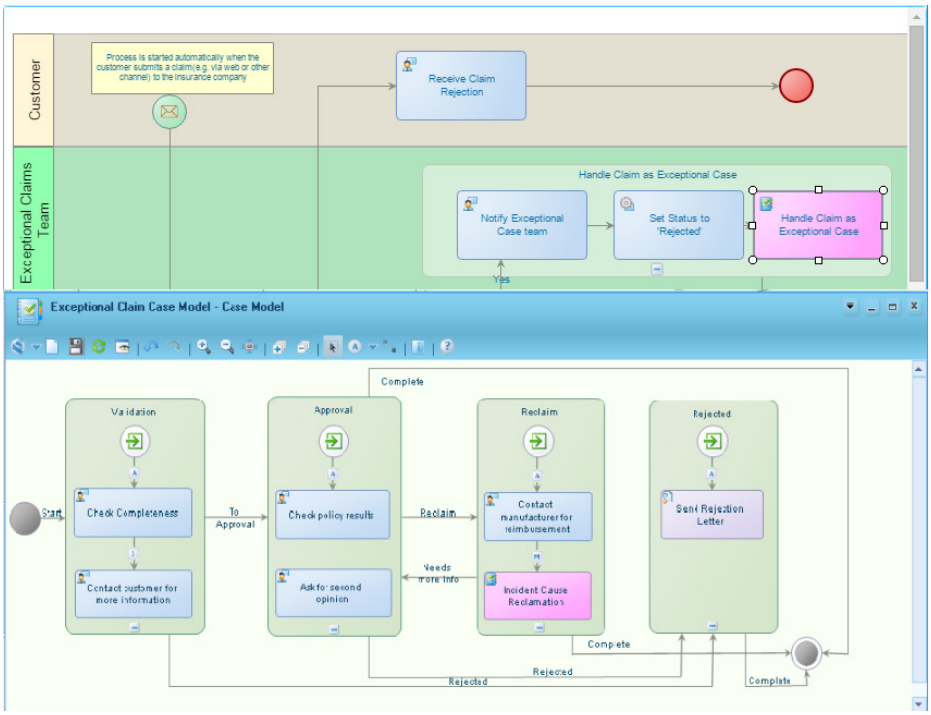
- **Fully state machine-based.** A case model is converted into a state model, including aspects such as follow-up relations, which might not seem state-related at first glance.
- **Standard SCXML execution format.** The state definition complies with the SCXML[12] standard as defined by W3C.
- **Case data management.** Case models define the data structure applicable for that case.
- **Case Instance Manager.** The progress of a particular case, as well as any changes to the case data, are tracked in the case instance tables. The Case Instance Manager provides access to this data.
- **Part of Business Process Management service group.** In a default Process Platform deployment, the case engine is part of the Business Process Management service group, together with the BPM engine.

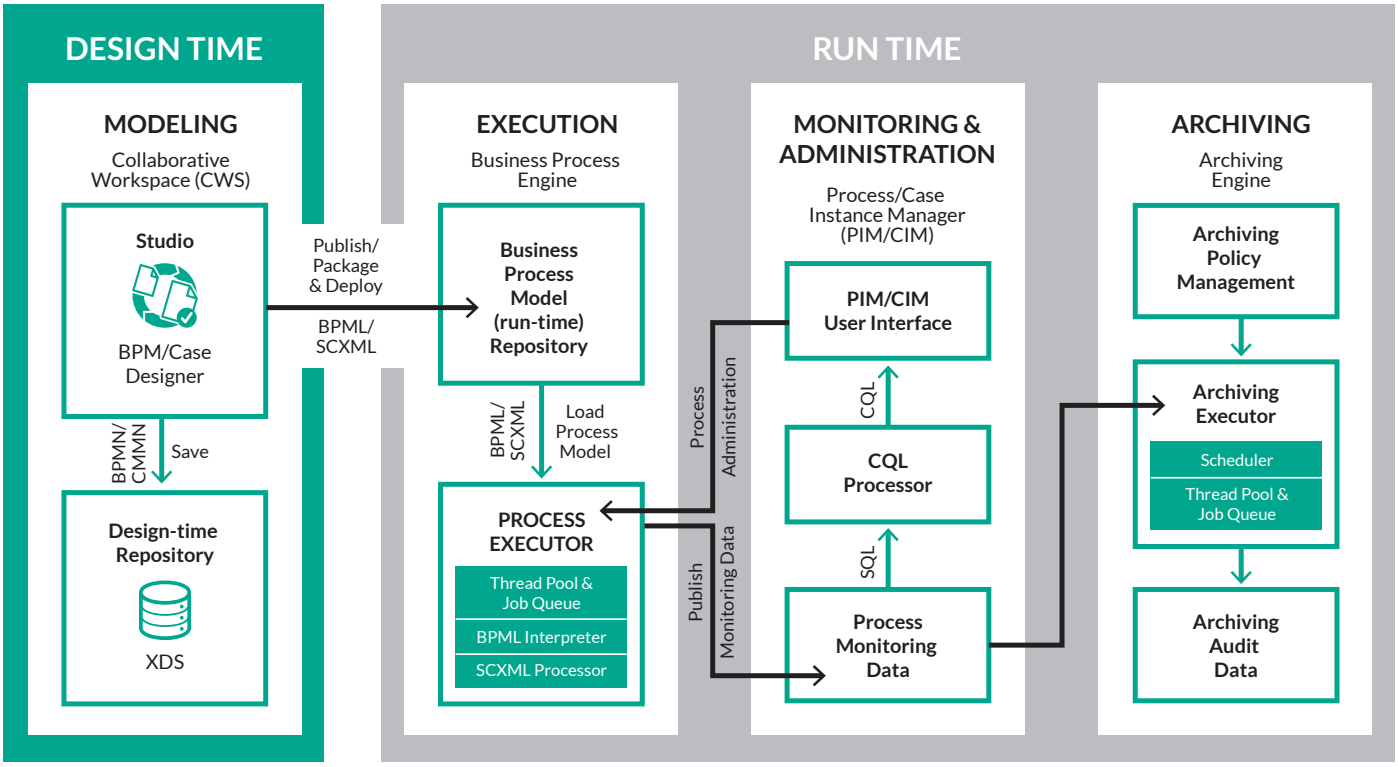
Following is a screenshot of the case model invoked from the main BPMN flow shown previously. It demonstrates how the BPMN flows and case models can invoke each other. This case model has four functional states and the arrows depict the state transitions that are often event-based.




If you compare the main BPMN flow given on the last page with the one given above, you can see that Process Platform has the option to *collapse* and *expand* groups of activities. The embedded sub-process labeled *Handle claim as an Exceptional Case* is expanded by the process designer into detail activities of which the last one, labeled *Handle Exceptional Case* invokes the case model.

The way this invocation works can even be configured via the properties boxes (using Tabs) allowing process designers to tune runtime behavior exactly to their needs.



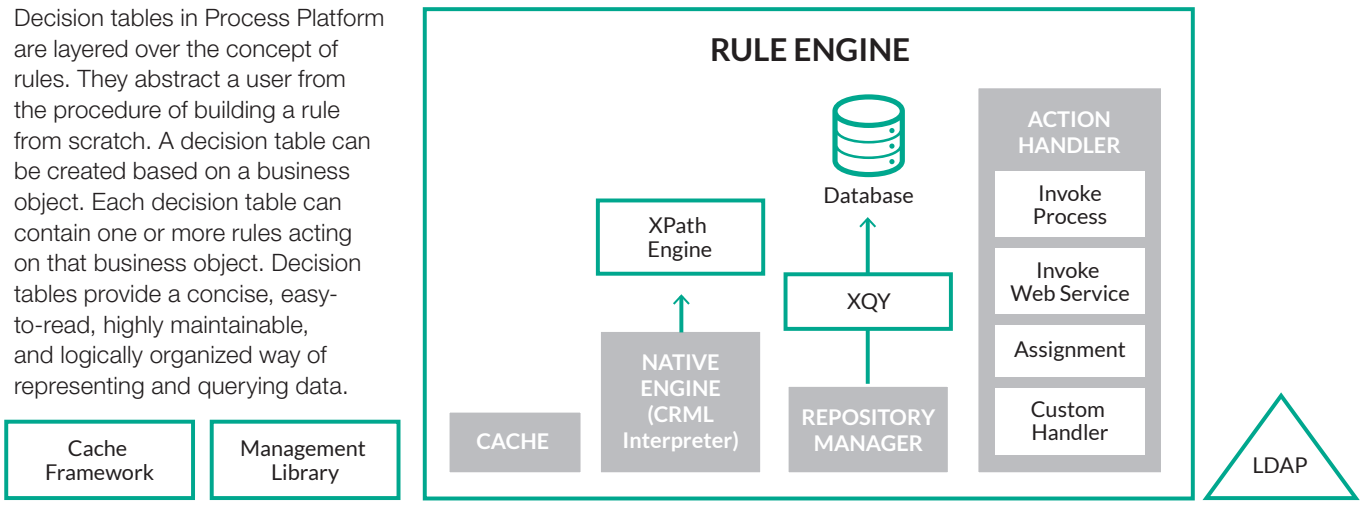



Case and BPM processes share a single runtime engine, for which the architecture is depicted above:

**Rules Management**

Process Platform offers a powerful, high performing rule engine for expressing business logic in the form of rules. Business logic (such as calculation of discount, calculation of customer rating, and so forth) is well suited for expressing in business rules, as such logic is subject to frequent changes. It should be easy for business users to modify and deploy business rules. Part of the logic of business processes is often expressed as business rules to make the business process less complex, more flexible and easier to maintain. The following picture depicts the rule engine architecture.

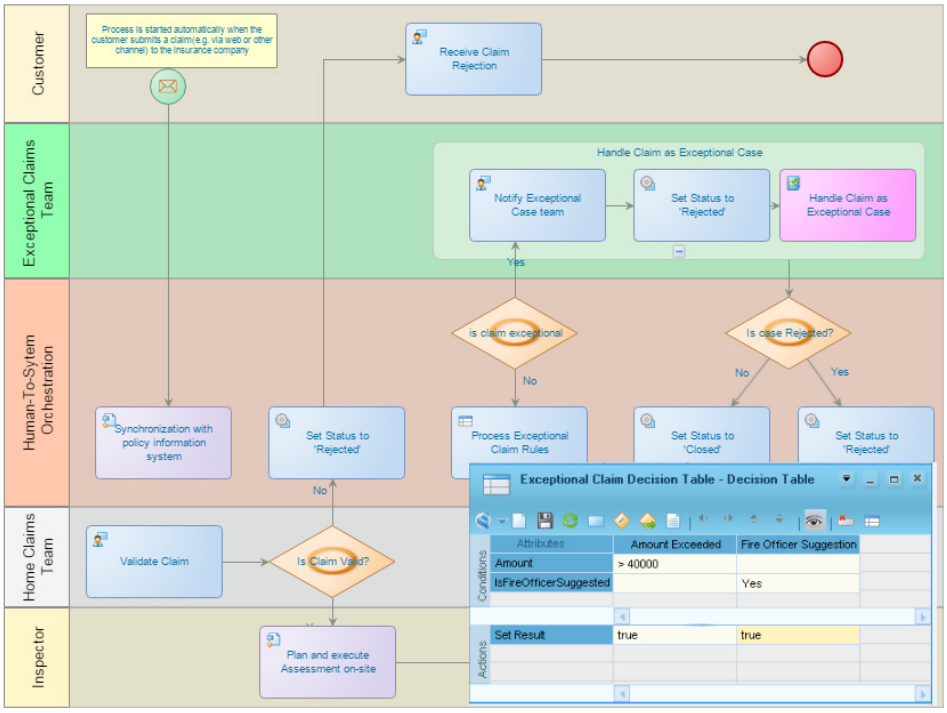
- The rule engine is well integrated with WS-AppServer. Applications should decide which logic should be part of Java code and which logic should be expressed as rules. The rule of thumb is that any logic that is very dynamic and changes frequently should be expressed as rules.
- Decision tables in Process Platform are layered over the concept of rules. They abstract a user from the procedure of building a rule from scratch. A decision table can be created based on a business object. Each decision table can contain one or more rules acting on that business object. Decision tables provide a concise, easy-to-read, highly maintainable, and logically organized way of representing and querying data.



- Decision tables can be used directly as an activity in a BPM process.
- Alternatively, you can generate a web service on top of a decision table, enabling it from all web services consumers.
- The Process Platform rule engine is available as a Java library, allowing any application to use it as an in-process Java library.

The following screenshot shows the decision table invoked from the main BPMN flow for the insurance claims handling example. The decision table is invoked as a web service from the BPMN flow, demonstrating the end-to-end web services based architecture of Process Platform.

The blue BPMN activity labeled *Process Exceptional Claim Rules* is in fact the invocation of the decision table, executing at runtime its *Conditions* and *Actions* (if-then-action rules).

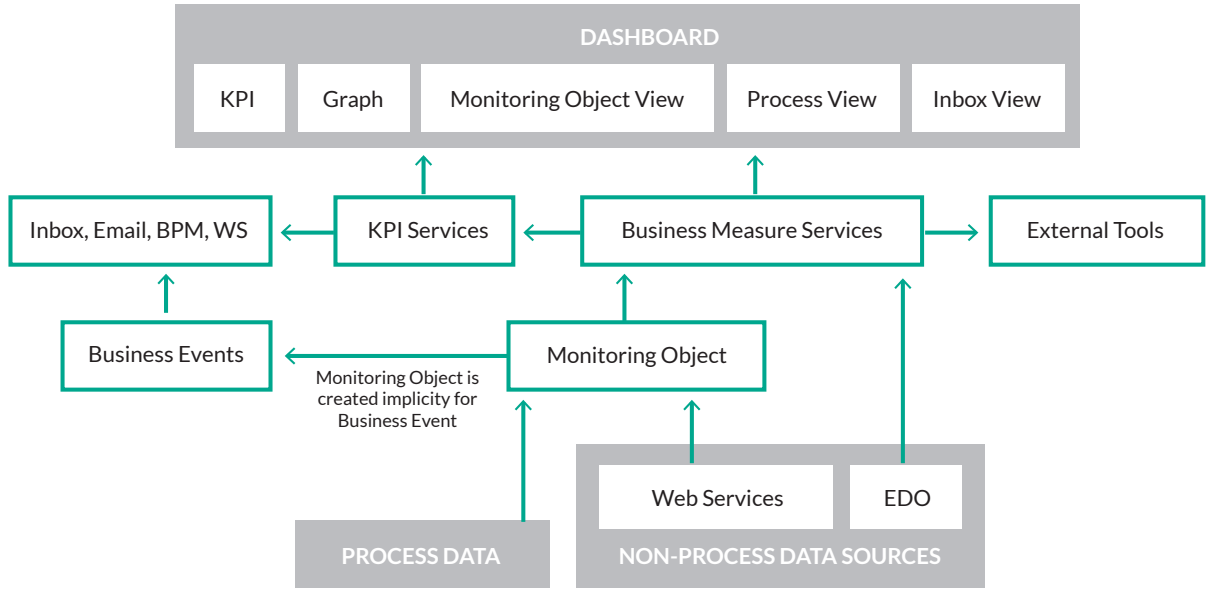



**Business Activity Monitoring**

Business Activity Monitoring (BAM) helps to monitor key business events for changes or trends indicating opportunities or problems on which business managers can take (corrective) action.

BAM provides real-time alerts and notifications of critical events and a centralized performance dashboard. It offers a drill-down analysis to discover trends, patterns, and bottlenecks.

In addition to monitoring business processes, BAM can also be used to monitor data coming from other systems using Master Data Management (MDM) or web services and present it to users via alerts or the dashboard.



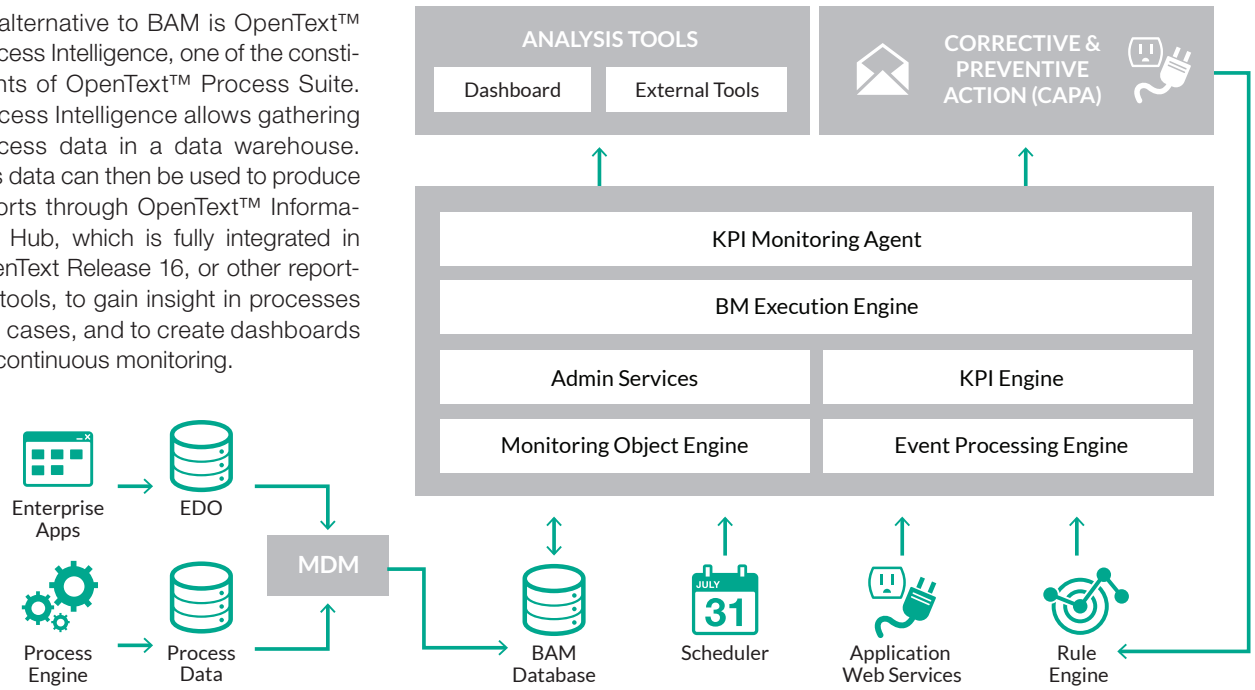

The above picture depicts the BAM architecture

The picture below depicts different aspects of the BAM runtime architecture.

Some explanation about the depicted components:

- The data for both EDO and processes is collected through MDM. The process data is picked up by identifying process events based on process state transitions in the database.
- The business measure web services are executed by fetching and executing corresponding SQL query from the BAM metadatabase.
- KPIs are monitored by the monitoring agent (scheduler) by invoking KPI web service with parameters specified in the KPI editor during design time.
- The BAM dashboard charts are built through the XForms designer, using FusionCharts graphical controls.

An alternative to BAM is OpenText™ Process Intelligence, one of the constituents of OpenText™ Process Suite. Process Intelligence allows gathering process data in a data warehouse. This data can then be used to produce reports through OpenText™ Information Hub, which is fully integrated in OpenText Release 16, or other reporting tools, to gain insight in processes and cases, and to create dashboards for continuous monitoring.



**Entity management**

Though Process Platform is strongly service oriented, with SOAP and WSDL as the common language between the different components, there is also a notion of entities. When building a user interface form, the user interface modeler recognizes certain patterns in the interface contract and based on that, knows that a certain entity (for example, an order) is dropped on the form. Process Platform supports entity management in two ways:

- Low-code, through entity modeling and runtime
- Developer-oriented, through WS-AppServer

Both variants are described in subsequent sections.

**ENTITY MODELING AND RUNTIME**

The entity model concept brings a more pervasive and complete notion of Entity to Process Platform, enabling an information first approach to application design and development. An Entity represents an identifiable notion in the business domain. Entity modeling takes a compositional approach - entities are constructed by adding building blocks until the desired result is achieved. The building blocks available include:

- **Properties -** various property types (such as Date, Integer, Text) are supported. An *Order* entity, for instance, typically has properties such as *deliveryDate (Date)* and *orderAmount (Integer)*.
- **Relationships -** entities can have child entities and maintain peer-to-peer relationships with other entities, supporting cardinality 1:1 and 1:n. An *Order* entity for instance would have zero or more *OrderLine* child entities and a n:1 relationship with *Customer*.
- **Rules -** business logic is added to an entity using declarative rules. Rules can be used to do validation, calculate values, control form behavior, and to start processes.
- **Actions -** custom actions can be added to an entity. The action can calculate and set the value of properties and start processes.. An *Order* entity for instance would have an action *plan*.
- **More -** there are many more building blocks available including File, Content List, Discussion, Security, Lifecycle, Mobile App, List, and Layout. An Order entity for instance would have a Security building block to manage the authorization on the entity, and a number of lists and layouts to represent the orders to the user. A Claim entity would have a Lifecycle building block to manage the states of the claim, a Discussion building block for threaded discussions, and a Content List to manage the documents.

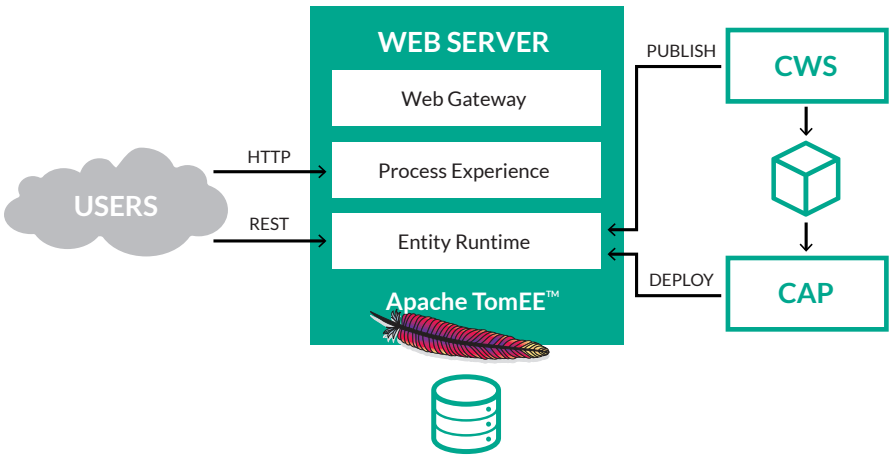
The entity modeler provides the option to expose SOAP web services on entities to read and manipulate them (by adding the Web Services building block). This enables exposing web services to external web service consumers (for example a .NET application) and using entities from models that currently only support SOAP web services. The SOAP request is routed to the Entity Runtime through the Application Server connector.

In the following sections, we zoom in on important building blocks and aspects of Entity modeling and runtime:

- Entity Runtime
- Lifecycle building block
- File and content list building block
- Mobile app building block

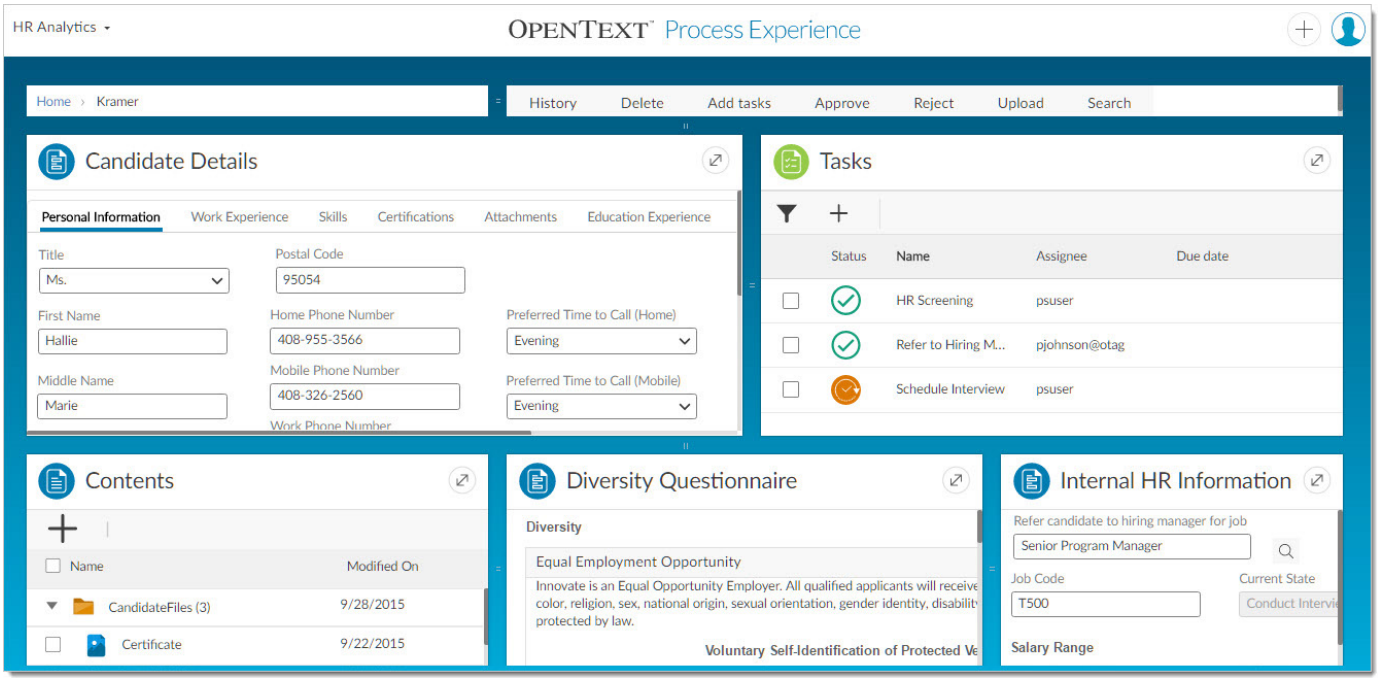
## ENTITY RUNTIME

The Entity Runtime is running as a web application inside the web application server, next to Process Experience, which provides the user interface functionality related to entities. As depicted in the following diagram, Entity Runtime exposes a set of RESTfull interfaces to its clients and it stores its data and metadata in a relational database. In multitenant scenarios, the relational database can be tenant specific or shared among tenants (while still maintaining strict data isolation). The most common scenario is to define the entities through the Entity Modeler and have the Entity Runtime create the database schema. In scenarios where the database schema already exists, it is possible to import it and create entities based on the existing database schema.




## LIFECYCLE BUILDING BLOCK

Entity modeling is a very powerful tool to define case-centric applications. Adding the Lifecycle building block to an entity brings all the power of the Case engine into the Entity world. It is now possible to define the states and actions of the case in the Lifecycle building block and have the data of the case (both structured data in the form of properties and related entities, and unstructured data in the form of documents) defined and managed as an entity. Dedicated UI panels enable showing task lists (based on the Task and Notification service) combined with document lists and dashboards in a uniform user experience.

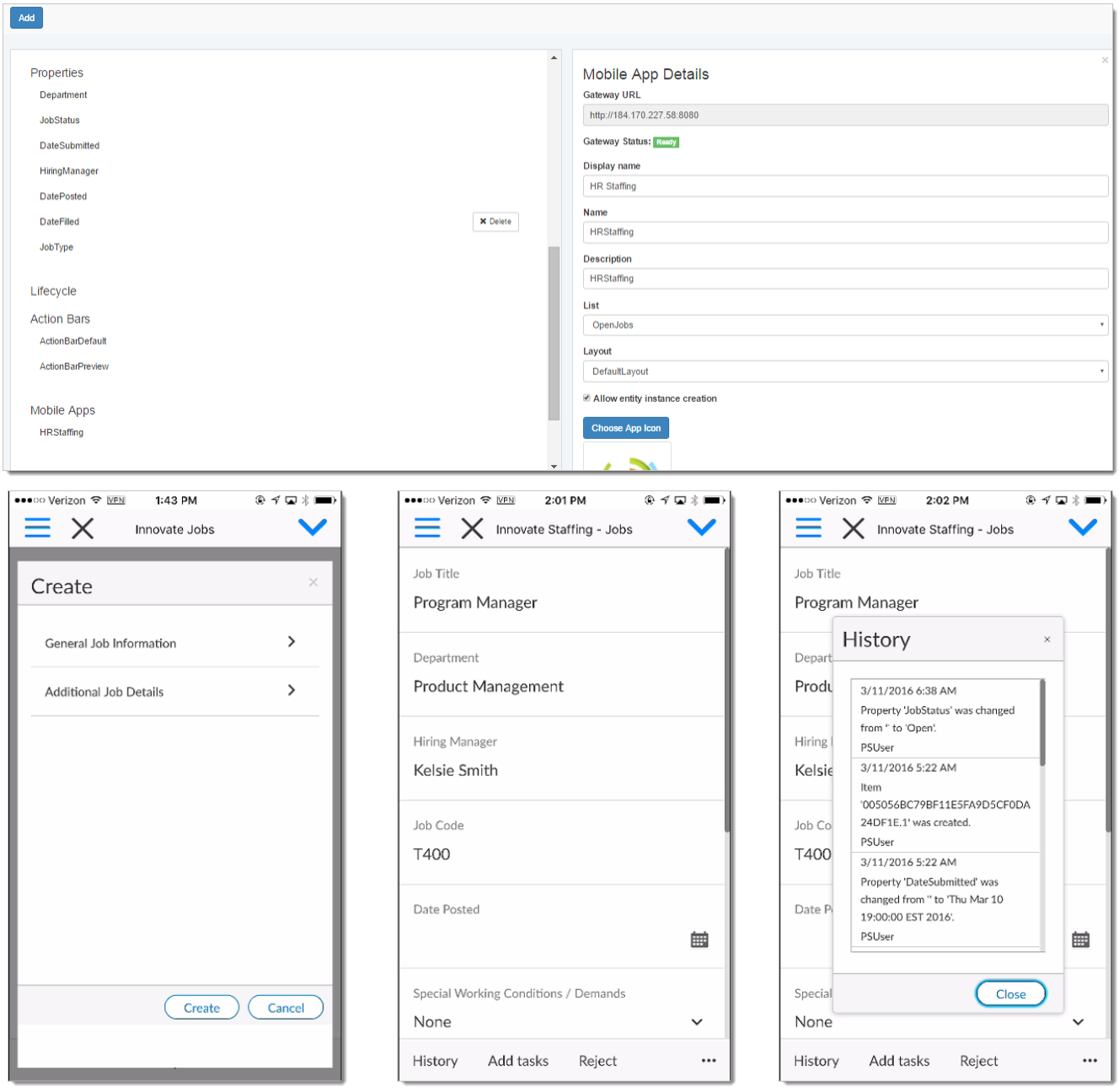


### FILE AND CONTENT LIST BUILDING BLOCK

Integrating entities with documents is common. If an entity needs just a single file, the File building block takes care of it. If multiple files are required, it can be done with the Content list building block. The latter one will internally add a child entity with, among others, a File building block on it. The panel showing the content list is featured in the previous screen shot, bottom left. The file building block uses the Document store to abstract from different types of content repositories and to provide a single point of configuration.

### MOBILE APP BUILDING BLOCK

Creating a mobile microapp through entity modeling is a matter of modeling the desired entity layout to show the entity, the list to show a collection of entities, and the mobile app building block. That suffices to create a mobile application that can be deployed through OpenText™ AppWorks.



### WS-AppServer

For a coding developer, WS-AppServer provides a way to define the application and UI logic in a simple and structured manner. WS-AppServer pulls out database metadata from a relational database and generates Java code. The generated code, along with the WS-AppServer framework, provides the application with the following features:

- Persistence is completely taken care of by the framework and there is no need to write code to perform CRUD[13] operations.
- Object Lifecycle and Transaction management is handled by the framework.
- Event-based programming makes coding easier.
- WS-AppServer has a large number of object lifecycle and transaction related events that are raised by the framework. Based on its specific needs, the application just needs to fill in what it needs to do when a certain event is raised.
- Dynamic business logic is supported via an integration with the rule engine.
  - Logic that is unlikely to change frequently is put inside the Java code.
  - Logic that changes frequently is defined as business rules, thus ensuring that business users can change the application behavior without having to bring in their developers to change the code.
- Application logic can be easily exposed through web services.
- Table specific classes can be aggregated into user defined classes, called custom classes, which map closely to the view the end user needs to see.
- Business logic of non-Process Platform systems can be extended by defining classes over entities of external systems and using the WS-AppServer events to handle the persistence and define the business logic.

### Scheduling

In a real-time business environment, the ability to trigger processes or applications based on specific system events or at a specific time is a critical requirement. Process Platform facilitates modeling of schedules that can trigger time-based actions, such as processes or web service invocations. Process Platform provides an intuitive UI-based schedule modeler, enabling modeling of different kinds of schedules that can be integrated into an application.

Schedules are generally of two kinds:

**One-time schedule.** These are executed only once

| TYPE OF SCHEDULE | DESCRIPTION |
|---|---|
| **RUN NOW** | Once created, this schedule is instantly executed |
| **DURATION** | This schedule can be set to execute after a specified duration |

**Repeating schedule.** These are triggered at specified periodic intervals, ranging from annually to hourly

| TYPE OF SCHEDULE | DESCRIPTION |
|---|---|
| **HOURLY** | Recur at a specified time every hour |
| **DAILY** | Recur at a specified time every day |
| **WEEKLY** | Recur at a specified time every week |
| **MONTHLY** | Recur at a specified day every month |
| **FIRST DAY OF MONTH** | Recur at a specified time on the first day of every month |
| **LAST DAY OF MONTH** | Recur at a specified time on the last day of every month |
| **FIRST WEEKDAY OF MONTH** | Recur at a specified time on the first weekday of every month |
| **LAST WEEKDAY OF MONTH** | Recur at a specified time on the last weekday of every month |
| **FORTNIGHTLY** | Recur at a specified time every 15th day |

At the scheduled interval, one of the following actions can be triggered:

- Invoke a web service by providing appropriate parameters.
- Instantiate a new business process by providing the process model name and the input message.
- Call back an already running business process instance by providing the instance ID.

### Task and Notification service

Every Process Platform user is assigned an Inbox that is used to send and receive tasks and notifications. The Inbox is configured for a given user profile and functions like a mailbox. Process Platform users can access tasks that are sent to their work lists, to their associated roles, to the teams they are part of, and to their personal task lists.

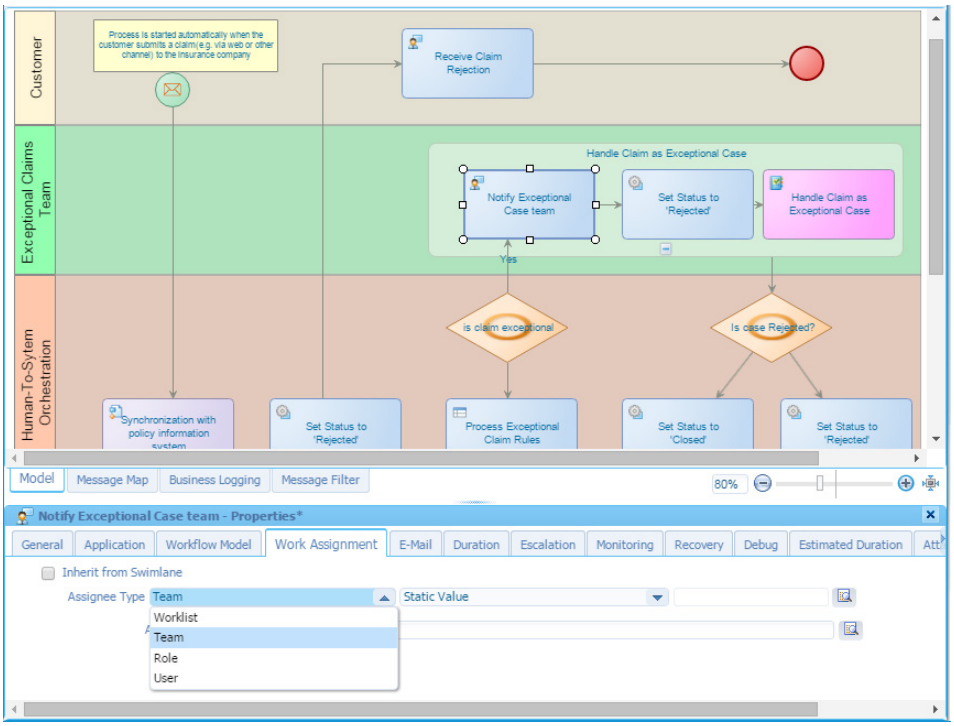
### Tasks

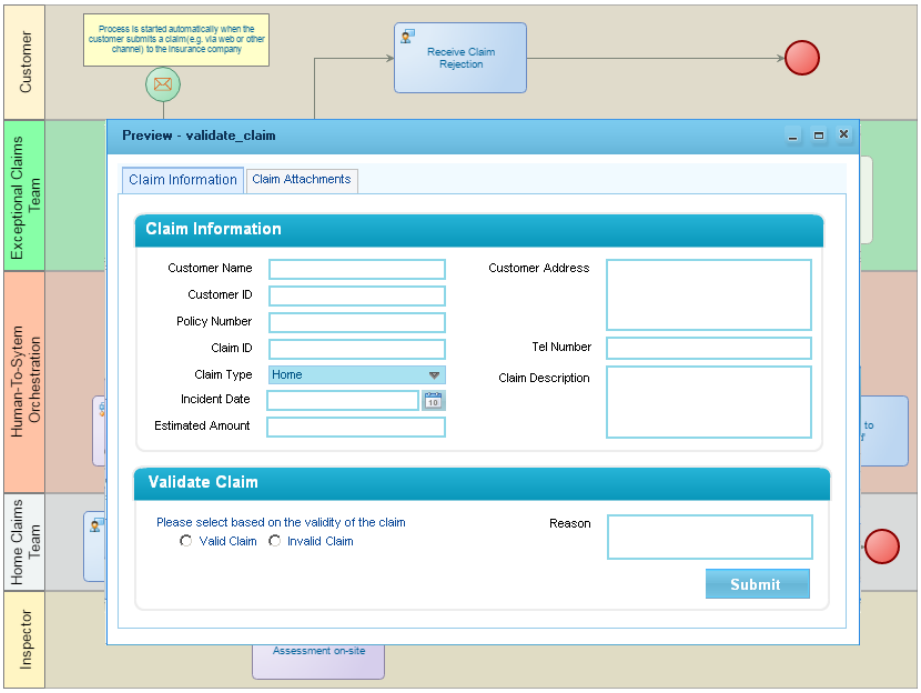
A task is an activity in a process that is to be executed by a human participant of the process. Users can either respond to the task, add a new action to the flow after the task, or do both. For example, when the stock of a particular item in a warehouse reaches a certain level, a replenish task is sent to the warehouse manager. On receiving the task, the warehouse manager opens it, fills in a purchase order form and forwards it to the purchase manager.

### Notifications

Notifications are used to convey event information to designated recipients or roles. For example, a notification can be used to send a message to the *Exceptional Claims Team* that there is a claim to be handled as an exceptional case.

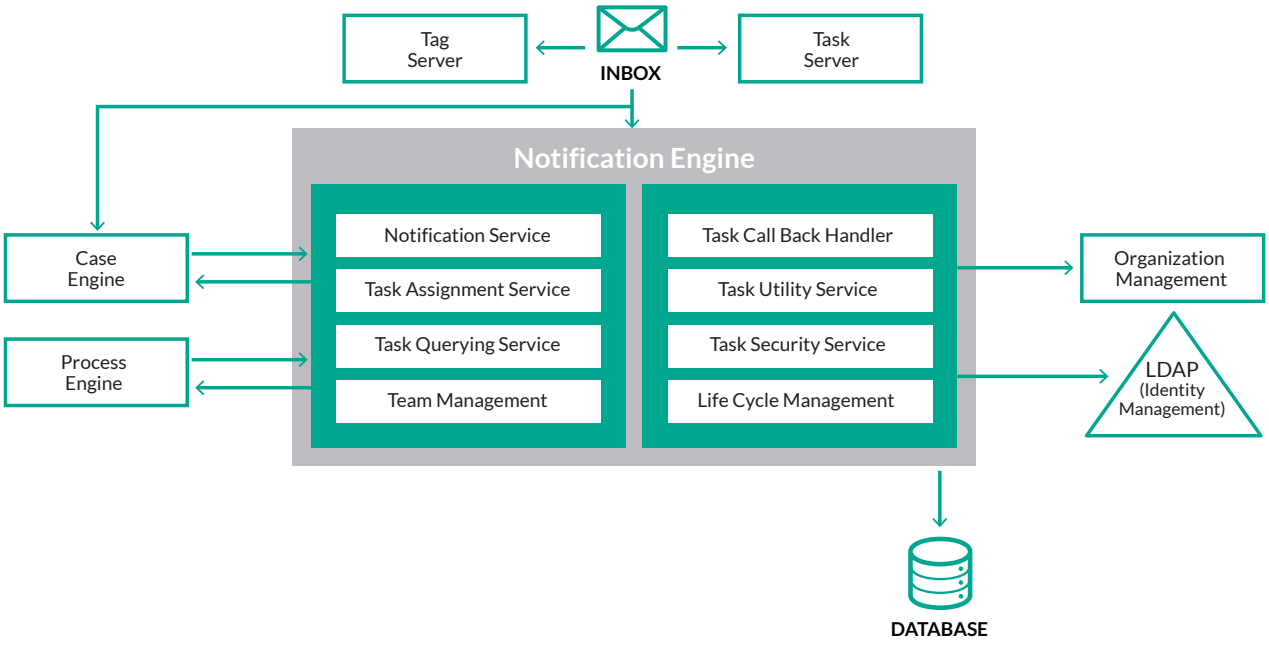
The following screenshot shows how the work assignment can be configured via the Process Platform properties tabbed dialog boxes.






The above screenshot shows the Process Platform User Interface used for one of the tasks drawn from the previously explained insurance claims BPMN process model. The user interface is shown in preview mode when clicking the user interface icon part of the Process Activity *Validate Claim* in the grey swim lane. This exemplifies the model-driven approach of the Process Platform: models for flows, cases, task user interfaces, and so forth.

A service container called Notification Service manages the lifecycle of both tasks and notifications. The following illustration shows the architecture of the Notification Service.



| COMPONENT | RESPONSIBILITY |
|---|---|
| INBOX | Facilitates knowledge workers to effectively and efficiently work with their tasks. |
| NOTIFICATION ENGINE | Responsible for task management. Interacts with the ecosystem and presents the details to Inbox. It is the core component of workflow. |
| PROCESS/CASE ENGINE | Interacts with the notification component to deliver the tasks for the human activities. Upon completion of the task, it takes the BPM or case to the next step. |
| ORGANIZATION MANAGEMENT | Responsible for team management. |
| IDENTITY MANAGEMENT | The user information and the roles are stored in LDAP. The notification engine interacts with LDAP to retrieve the user/role information. |
| TASK SERVER | Every UI in Process Platform is represented as a task. Inbox interacts with the task server to retrieve the URL of the user interface associated with a human task. |
| TAG SERVER | The Inbox interacts with the tag server to retrieve tasks associated with user defined tags. |
| DATABASE CONNECTIVITY AND PERSISTENCE STORE | All tasks and associated models are stored in a database. The notification engine uses the XQY database connectivity layer for persistence and retrieval of the task list content. |

- **Work list.** A work list is a container of the tasks processed in a workflow, case, or any other composite application and is displayed in the Process Platform Inbox. Teams are associated with a work list. Users who are part of the teams assigned to work list can view all the tasks assigned to their work list, the status of various tasks in their work list, and so on. Depending upon the skill set and requirements, users can claim tasks from the work list. Once the task is claimed, it appears in the personal tasks folder of the user, which contains all the tasks assigned to or claimed by the user.
- **Escalation support.** For tasks created from processes or cases that are not completed in the stipulated time, as defined at design time (either statically or dynamically through a variable), an escalation message is sent to the manager of the case worker or to the manager of the work list to which the task is delivered. It is also possible to transfer the task to some other user on escalation
- **Open.** It is not mandatory to have XForms as the user interface. It is also possible to reuse the existing UI pages, developed in ASP.NET or JSP, as a UI task in Process Platform for workflow integration and task delivery. It is possible to create an external user interface document and link the existing user interface developed in ASP.NET or in JSP by providing an interface contract in the form of XML schema. The JSP or ASP.NET pages need to be enhanced to consume the data from Process Platform and provide the data back to Process Platform, according to the XML schema provided.

### Master Data Management

Master Data Management (MDM) focuses on the core infrastructural needs of enterprise data integration. Process Platform separates infrastructural MDM needs from all that is domain-specific. This implies a data domain-agnostic approach to data integration. Users can apply MDM for integrating data of different data types: master, reference, and transactional data. In addition, within a given data type (for example, master data), MDM can be applied to harmonize data of different subject areas, such as Customers, Suppliers, Products, Locations, and so forth.

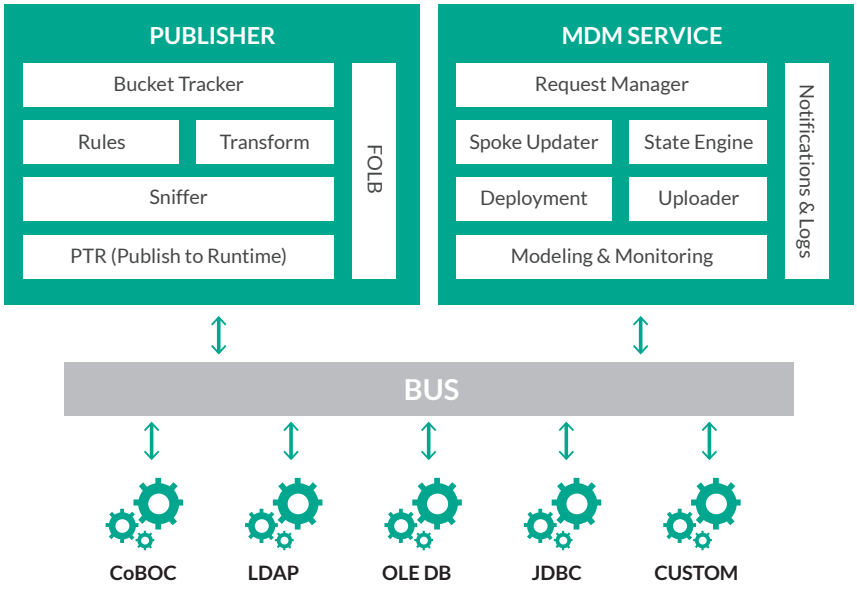
Highlights of the Process Platform MDM offering:

- Pluggable architecture (employing third party data quality tools, and so forth)
- Near real-time data synchronization
- Master data, business object lifecycle management
- Event driven object lifecycle management
- Strong workflow capabilities
- Works in publish - subscribe model
- Support for web services
- Supports all three MDM patterns (Registry, Coexistence and Transactional)

There are various ways in which organizations can use MDM to build and deploy trusted data hubs, according to the needs and goals of the enterprise. Following is a summary of what MDM can provide for the enterprise:

- Identify sources of master data
- Identify producers and consumers of master data
- Collect and analyze metadata of master data
- Appoint data stewards
- Implement a data governance program
- Develop a master data model
- Design and set up the infrastructure
- Generate and test the master data
- Modifying producing and consuming systems to integrate with the new MDM solution
- Implement maintenance process

**Architecture of the MDM runtime**



### Service Oriented Architecture Layer

The Service Oriented Architecture Layer provides the fundamental SOA grid functionality used by the other layers.

#### OpenText XML technology

From its inception, Process Platform uses XML as the core communication protocol format. The information exchange across the Process Platform components is in the form of XML, which leads to heavy XML processing. This necessitated the development of better implementations[14] of XML processing technologies, such as an XML parser and an XPath/XSLT processor.

The Process Platform XML parsing API has two flavors. The first flavor, NOM (Native Object Model), is a set of proprietary APIs, which is highly efficient but a bit complex in use. The second variant, coined DOM-over-NOM[15], is a standard DOM API layered over the NOM APIs, bringing the benefits of a convenient standards compliant API while keeping the overhead extremely low. Developers can choose their favorite API flavor and even switch from one to the other[16].

#### Enterprise Service Bus

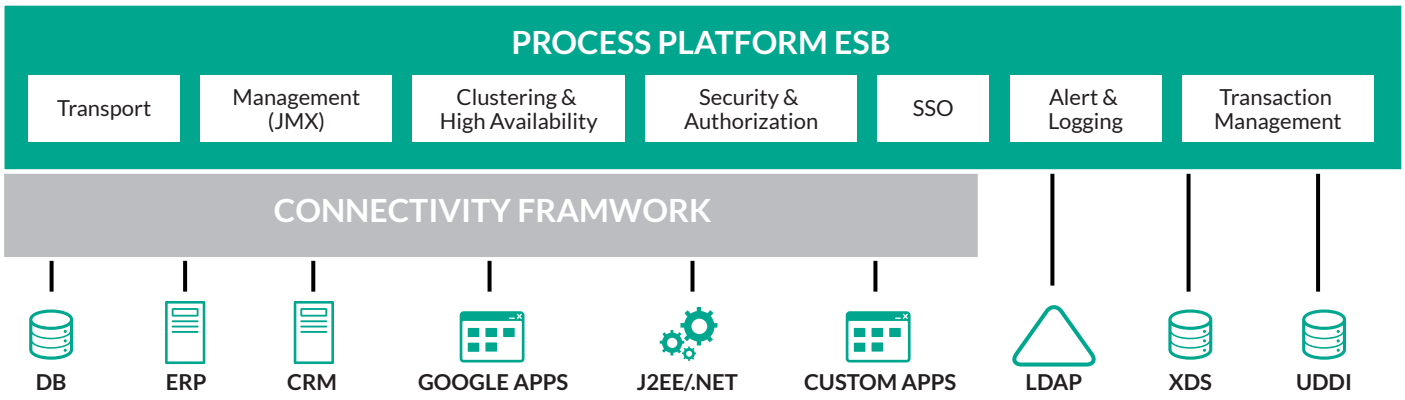
Process Platform provides an integrated suite of services to enable customers to develop, deploy, and manage business applications on a Service Oriented Architecture (SOA). Therefore, Process Platform has a broad view of middleware that forms the platform on which the other components are built and offered as services. These are then consumed and orchestrated by our customers in a process-driven approach for building their custom applications, web applications, and composite applications.

The center of this middleware is the enterprise-ready ESB, which is the enabler for SOA. This unique technology enables every service as a loosely coupled, distributed service on a bus, with the associated benefits of granular failover and scalability.

Traditional ESB implementations have emerged from **Message Oriented Middleware (MOM)** by adding web services and **Enterprise Application Integration (EAI)** on top of this existing MOM infrastructure. In a hub-and-spoke architecture, all back-end systems (spokes) rely on the hub to communicate with each other and any hub failure causes the entire integrated system to fail. In addition, any back-end systems can potentially overload the hub, making it necessary to augment the hub with additional computing power. The Process Platform ESB does not have a central hub, which eliminates the single point of failure and removes a common performance bottleneck. The Process Platform ESB uses bus as an architecture and peer-to-peer as the communication paradigm.

The enterprise service bus has emerged as the best practice to perform application connectivity and to decouple service consumers and service providers, and is the most tangible software infrastructure for SOA.

The following sections focus on some of these services, which are expected to be delivered by any modern day ESB, and how the Process Platform ESB provides them.



### STANDARDS COMPLIANCE

Process Platform uses universally accepted standards, such as **WSDL, XSD, XML and SOAP**. All consumers of the Process Platform web services can use the "design by contract" model for invoking these web services without having to worry about underlying implementations. Process Platform is also WS-I Basic Profile[17] compliant, so the services hosted on Process Platform are completely compatible with different platforms. Developers can base their functionality on the contract given by the WSDL and not worry about implementations.

### LOOSE COUPLING

All capabilities provided by Process Platform or built by customers using the platform are delivered to the end user as services, and from a technology perspective as web services. Web services use WSDL and XSD and provide loose coupling because they formalize the contract between the consumer and provider. The Process Platform messages follow the paradigm of stateful objects and stateless connections, so all invocations for the Process Platform services are completely decoupled and do not hold any information about the client's prior interactions.

### TRANSPORT TRANSPARENCY AND MULTIPROTOCOL SUPPORT

A critical capability of the ESB is to route service requests through a variety of communication protocols and to transform service requests from one protocol to the other where necessary. Enterprise applications typically involve talking to different individual applications that support different transport protocols, and the Process Platform ESB provides physical transport protocol bridging to allow communication between services using different transports. Process Platform provides a choice of configuring services on sockets, Microsoft Message Queuing (MSMQ), Java Message Service (JMS), and also offers the option to build custom transports. This gives flexibility to effectively integrate disparate systems and manage complex communications at the transport level.

### LOCATION TRANSPARENCY

The ESB acts as a layer of middleware through which a set of business services is made widely available, but the client is not aware of where the service is hosted, as it breaks the previously mentioned loose coupling strategy. All Process Platform services point to one physical address and the Process Platform ESB locates the service when it is invoked, providing a level of service virtualization and location transparency, so that if a node goes down or a service provider has to be moved, individual service clients do not need to be notified of the change.

### CLUSTERING - LOAD BALANCING AND HIGH AVAILABILITY

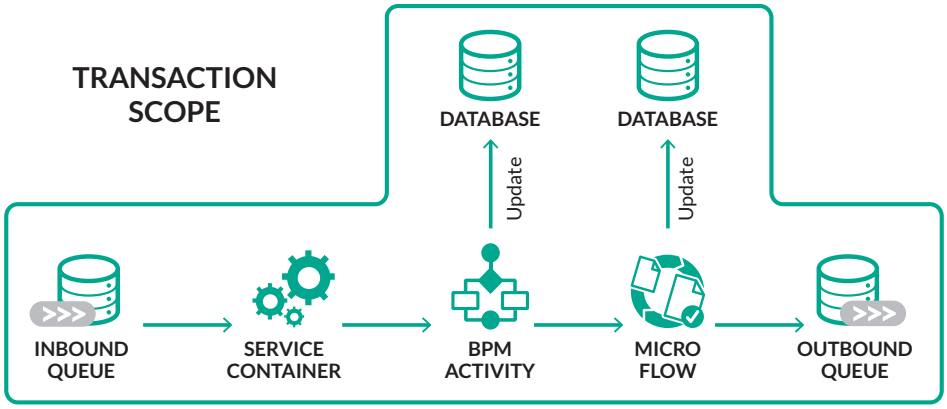
The ESB acts as the middleware to host all services. It is also suited to perform load balancing of service requests across services. The Process Platform services are configured under a logical entity called service group. Each service is hosted in a service container, which is the Java process providing the implementation. A service group can be implemented through more than one service container of the same configuration, each running on a different physical node. An administrator can choose the load balancing algorithm, such as simple failover or round robin for load balancing. Service containers can be added or removed on demand depending on the need. The Process Platform clustering leverages a reliable broadcasting technology called Gossip. Service containers broadcast their state and health information using the Gossip protocol[18].

### SECURITY INFRASTRUCTURE

As the ESB acts as the central mediator for service invocations, it is ideal for declaring and enforcing security. The Process Platform ESB provides a comprehensive security infrastructure for developers to create access control lists for specific roles/users on all important components, such as service groups, web services, and metadata, which are then enforced by the engine after authentication. Process Platform can integrate with any enterprise domain authentication systems or integrate with external SAML providers, and also provides the option of custom authentication.

### RELIABLE MESSAGING

Reliable messaging refers to the ability to queue service request messages and ensure guaranteed delivery of these messages to their destinations. It includes the ability to provide message delivery notification to the message sender/service requester. The ESB supports distributed transactions and the Process Platform components are configured for accepting requests on message queues. They can deliver and receive messages reliably, even in case of component, system, or network failures. The Process Platform reliable messaging uses distributed transactions through the XA[19] protocol and supports JMS and MSMQ message queues.

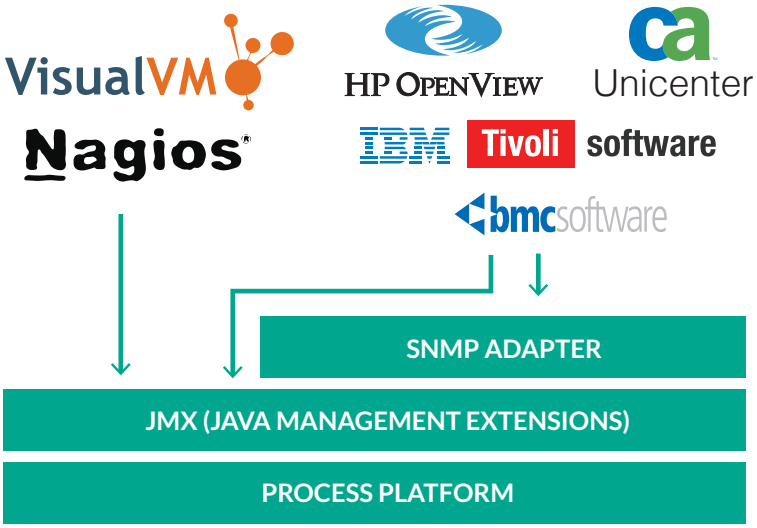



### MANAGEABILITY

It is important to monitor the health of the ESB and the services hosted on it. The ESB comes bundled with applications that provide comprehensive information about all the services, health of the system in a dashboard view, and the ability to drill down to specifics for any component. Process Platform has self-healing capabilities, where services raise alerts and can take corrective actions automatically. Process Platform services can be monitored using any JMX and SNMP compatible tools.

*Manageability using JMX*

Process Platform can be managed through the JMX APIs as exposed by the platform. JMX-capable tools can directly interact through JMX. Other tools can use an SNMP adapter.



### COMPOSITE APPLICATION LOGGING

The Process Platform ESB supports diagnostic logging, known as **Composite Application Logging (CAL)** for composite applications. The composite application log messages have contextual information, such as correlation ID and multilevel diagnostic contexts, which indicate the execution path of the application. The context is non-intrusively propagated across several layers based on the execution. This enables administrators to perform causality analysis of any incident. The log messages are optionally stored in a centralized database, thus providing an integrated view of the logs of a cluster of multiple nodes. The composite application log viewer provides drill down, correlative analysis, and filtering.

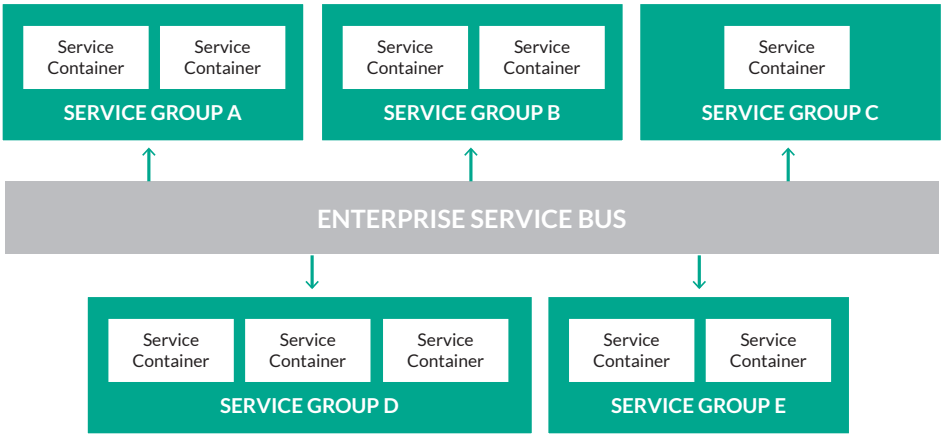
**Implementation aspects of the Process Platform ESB**

### RUNTIME NOTION

Web services are served and processed by service containers. A service container is a Java virtual machine, typically running as a physical process on the system.

Service containers are logically grouped as service groups. Service groups hold the necessary configuration information to identify the type of request and the routing. Requests are received through connection points configured on the service containers. Service containers can be configured on multiple nodes and each service container has multiple connection points, potentially with different transport protocols. Connection points are end points for services on the ESB.

Service groups provide the loose coupling aspects in terms of location transparency. Connection points enable the network protocol neutral message delivery. Service groups, service containers and connection points are configured in the Process Platform LDAP repository.




### MESSAGE ROUTING AND DELIVERY PARADIGMS

All the requests (web service invocations) are identified through the namespace. The namespace indicates the type of request that needs to be processed.

- Based on the request, the respective service group is first identified using the Process Platform LDAP repository.
- Then, based on the availability and routing policy, a specific service container is chosen for processing the request.
- After selecting the service container, the connection point information is used for choosing the transport protocol.
- Request is delivered to the service container for processing.

The Process Platform ESB supports both synchronous (request-reply) and asynchronous (fire-and-forget, request-callback) messaging paradigms.

### THE PROCESS PLATFORM MONITOR

The Process Platform Monitor is a special service container on the ESB. It acts a supervisor for all service containers configured on that particular node. Each node in the ESB cluster has its own monitor configured and running. It runs as a Linux daemon or Windows service and takes care of:

- Managing (start, stop, restart, reset) service containers (inside and outside the web application server, see Node view).
- Providing the necessary bootstrap configuration.
- Synchronization and broadcasting of status information of all service containers on the cluster.
- Synchronization and broadcasting of any problems faced by any service on that node.

A service container can be on one of three states:

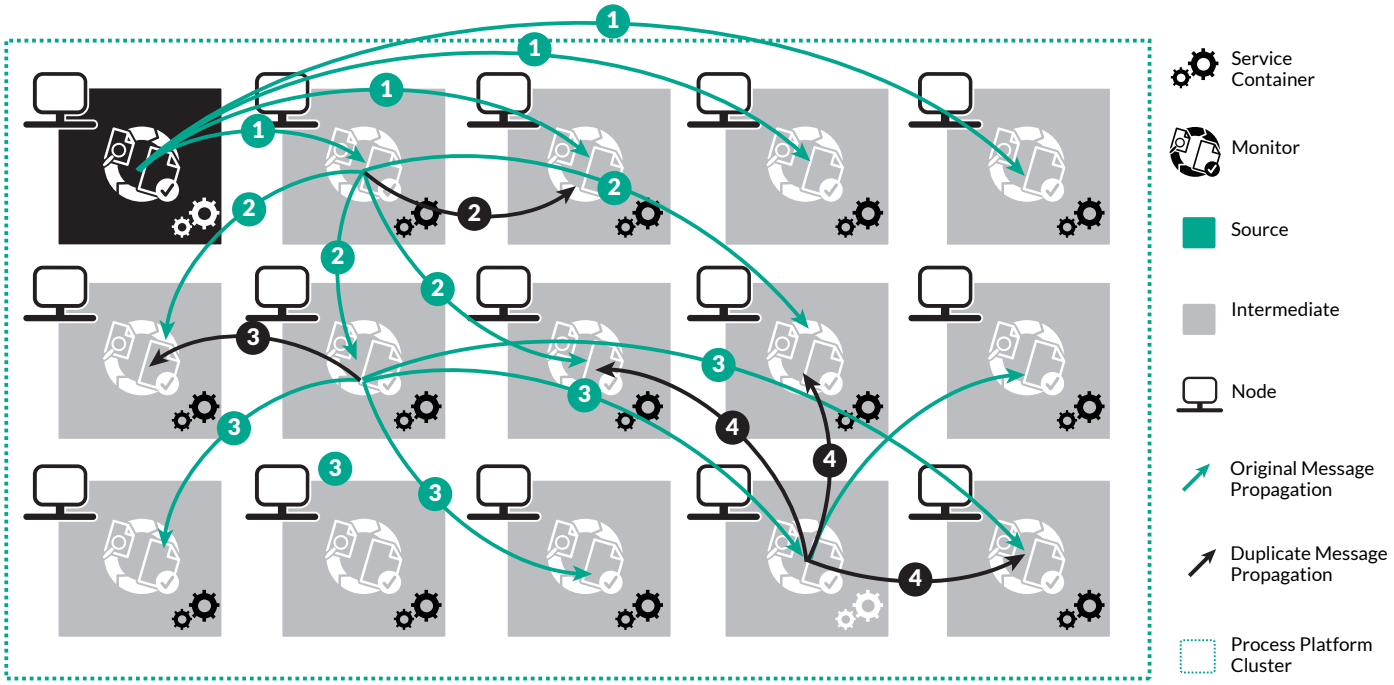
- **Stopped.** The service container is not loaded in memory.
- **Paused.** The service container is passive. If it receives a request, it will handle it, but it will not take initiatives itself. The Scheduler for example will handle a *CreateSchedule* request as normal, but when time passes by, it will not fire the scheduled action.
- **Started.** The service container operates normally.

An entire cluster can be paused and resumed through the System Resource Manager. Pausing a cluster is useful during upgrades but also for certain replicated setups. When starting or stopping a service container, the container always transitions through the Paused state.

### GOSSIP PROCESS PLATFORM CLUSTERING TECHNOLOGY

Process Platform is a distributed system[20] that uses multiple processes (Java virtual machines) spread across nodes. The Process Platform ESB uses routing and load-balancing techniques to distribute user requests to appropriate processors (called service containers). These activities require group communication to be included in the ESB, for which the Gossip protocol[18] is chosen.

The message propagation[21] in Gossip is depicted in the following illustration, where nodes selectively broadcast to a set of peers[22] and where duplicates are ignored.



Some of the use cases for this framework are:

1. **Distributed cache invalidation.** Most of the components (for example, LDAP, Rule, CoBOC) in Process Platform use caching for performance reasons. All these components use Gossip for cache invalidation. More information is available in this article[23].
2. **State registry.** State registry is an in-memory database of the state of the Process Platform cluster, built on Gossip. The Process Platform state registry provides the needed infrastructure for high availability. More information is available in this article[24].

**APPLICATION PACKAGING AND DEPLOYMENT**

Process Platform provides both web-based and command line-based facilities for managing the applications built using Process Platform.

Any composite application developed using Process Platform is packaged through a format called **CAP**. Packaging the application is part of the standard CWS facilities.

A major part of the platform is packaged as CAPs and deployed through the CAP deployer. The CAP format is well defined to manage dependencies, prerequisites, and cluster-level deployments for installation, upgrade, and uninstallation. CAP contains a manifest of all artifacts that are bundled. The manifest maintains the type information, identification, and a hash for each artifact.

- The type information is used to identify the needed deployment logic.
- The identification is twofold: a human-readable name, often the name of the CWS model, and a synthetic identifier that remains the same when the artifact is renamed.
- The hash helps to identify the change in the artifact across different versions of the application.

Using the manifest, CAP deployment provides the fine grained detail and impact caused by the application deployment. For example: The impact analysis section displays the list of artifacts affected and whether they are deployed at cluster level or node specific.

A package can be deployed in the shared space, available to all organizations, or to a specific subset of the organizations. Organization level deployment allows having different packages and package versions for different organizations. When building a package, it is possible to indicate whether the package can be deployed in the shared space, on organization level, or both.

Process Platform maintains an XDS-based repository called **CAP Registry** to register the details of all the applications deployed.

CAP provides a single step to install, upgrade, uninstall, and roll back any application across the cluster.

Optionally, CAPs can be digitally signed to build the chain of trust. The trusted entities are registered through the Security Admin Console. It is configurable to allow/disallow unsigned and tampered applications. This option establishes the trust factor for multitenant deployment over the cloud.

**Enterprise information system connectivity**

Every enterprise uses one or more business applications and IT systems to manage the business of the enterprise. Solutions developed with Process Platform nearly always integrate with existing Enterprise Information Systems (EIS). These applications and systems include ERP systems, Content Management systems, CRM systems, databases, and so forth. Process Platform provides a generic connectivity framework to connect to various systems and applications. Based on this framework, Process Platform and the community around Process Platform have developed a set of ready-made connectors for some of the most commonly used IT systems.

The Process Platform connectors act as an interface between the ESB layer and a specific application, system or technology. It provides a two-way communication channel. That is, requests or messages from the ESB layer are converted to a format that is understandable by the specific application or system and messages from the application are converted to SOAP requests on the ESB.

The application connector framework leverages the rich functionalities and features provided by Process Platform, which include:

- Logging
- JMX support
- Access control list (ACL)
- Problem Registry
- Localization support
- Transaction support
- Load balancing and failover support

The generic connectivity framework helps ISVs and application developers to develop specific connectors with minimum effort, and also enables third party connectors and adapters to be integrated with Process Platform.

The ready-made connectors provided by Process Platform and the community around it provide instant access to most widely used systems in an enterprise. Some of the out-of-the-box connectivity options provided by Process Platform are:

- **Email connector.** The email connector enables sending and receiving mails through IMAP or SMTP and POP3.
- **FTP connector.** The FTP connector enables uploading and downloading files to/from an FTP server.
- **HTTP connector.** The HTTP connector is commonly used to interact with RESTful web services. It supports both JSON and XML.
- **JMS connector.** The JMS connector enables integration with JMS-based services. It exposes web services to send and receive messages, and it allows defining a trigger to invoke a web service upon receipt of a message over a JMS queue. The latter feature is commonly used to execute a process upon receipt of a message.
- **SAP® connector.** The SAP connector provides connectivity to SAP back ends through RFC and BAPI.
- **Script connector.** The Script connector enables web services to be implemented in JavaScript.

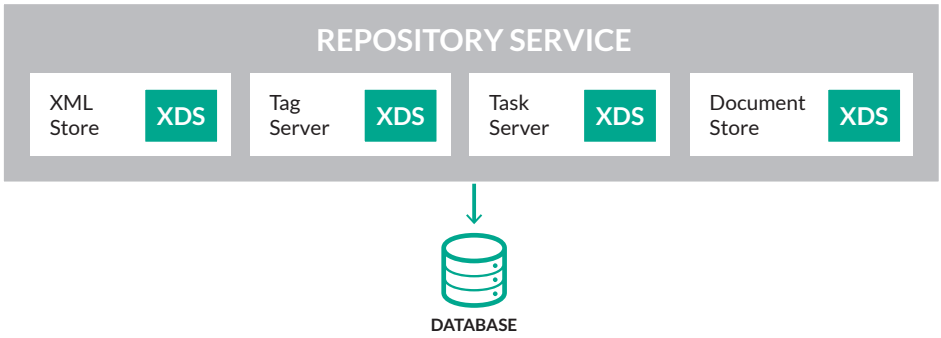
A full catalog of the available connectors is available here[25].

**Repository**

Process Platform consolidates storage of metadata in a single repository called XDS. This repository is the underlying store for the Process Platform Collaborative Workspace (CWS) but also stores tag definitions, tasks, business calendars, and so forth. XDS is RDBMS-based and meets the high availability requirements of Process Platform.

The Process Platform repository contains different kinds of data:

- **Application content developed in CWS.** Stored in XDS via CWS.
- **Data published from CWS.** Certain deployed models are stored in XDS (example business calendar, organization models, tasks, and so forth.)
- **Runtime data.** The platform and applications built on top of it can store metadata and definitions in the repository (for example, XMLStore, which uses XDS to store the content in the database).
- **Platform metadata.** Stored in repositories. Package installation details, for example, are stored in XDS.




The above diagram shows the architecture layer of the repository service.

**Document Store**

Documents are usually stored in a content repository. Process Platform provides Document Store as the facility to work with any content repository. Any document, regardless of the content, can be stored in these document stores. Documents can be added as attachments and be retrieved later. Documents can also be viewed, uploaded or downloaded.

Document Store enables you to plug in a content repository so that any existing content repository can be used. Process Platform can work with any content repository server that has exposed its repository API according to JSR 170[26] or CMIS[27] specification. Some repositories, such as Apache Jackrabbit™[28], provide a true implementation of JSR 170. Other repository servers exposing their repository service through a JSR 170 compliant wrapper can also be plugged into Process Platform.

The document store supports the following content repository types:

- Repository (based on XDS, not supported for production usage)
- Apache Jackrabbit
- OpenText Content Server
- OpenText™ Archive Center
- CMIS (any type of CMIS 1.1 compliant content repository)
- Custom plug-in implementation

To optimize memory consumption, a streaming-capable RESTful API exists. An integration with OpenText™ Brava!™ DWG Viewer enables viewing any type of document in the browser.

**Tagging**

Many social networking sites allow their users to tag[29] information. Tags are generally chosen informally and personally by the item's creator or by its viewer, depending on the system. Tagging is an easy way for end users to group or categorize their own tasks and artifacts. Tagging abstracts the storage approach (for example, hierarchical) and enables users to view the system in a more linear and self-managed approach.

The tagging service within Process Platform allows associating tags to any type of artifact. Process Platform currently uses it to associate tags to tasks on the User Start Page, tasks in the Inbox, and documents in the Process Platform Collaborative Workspace (CWS). The design is extensible and allows tagging application artifacts (for example, orders, products, employees) as well.

### Auditing

Auditing is a well-known process of tracking changes to an object of concern in the software world. Process Platform provides an auditing framework that is used internally by many of its components and can be used for auditing of application events as well.

Process Platform provides an auditing framework to define artifact types and provides needed abstraction to help developers audit the needed artifacts. Process Platform stores all the audit information in the Process Platform database configured at installation.

Process Platform provides auditing capabilities to audit critical actions:

- Starting and stopping of service containers
- Deployment of application packages, BPMs, business calendars, schedules, and so forth.
- Changes to LDAP and XMLStore
- Invocation of web services (this can be done on web gateway level and per service container)
- User login and logout

What's actually audited can be configured through filters. The administrator can view and search specific audit events based on:

- Artifact type or ID
- User who performed the action
- Date range
- Organization context
- Action performed on the audited artifact, e.g. deploy or undeploy

### Gateway

The Process Platform web gateway[30] is the HTTP interface of the Process Platform SOA Grid. It is meant to be a lightweight component, hence its functionality is very minimal. In brief, it is the HTTP end-point of all web services hosted by Process Platform. Besides that, it also supports some security features[31].

The primary functions of the web gateway are:

- Authentication (optional).
  - Integrated authentication, for example, Microsoft® Active Directory®.
  - Process Platform authentication, using Single Sign-On (SSO) service.
- Authorization: The set of accessible web services can be limited on web gateway level.
- SOAP request/response validation (optional).
  - Verify if request is according to the WSDL.
  - Verify if response is according to the WSDL.
- Translate HTTP requests into SOAP requests.
- Translate SOAP responses into HTTP responses.

The gateway runs in the web application server.

### User Interface (UI) tasks

UI applications in Process Platform are referred to as UI tasks. UI tasks facilitate fine grained authorizations on different elements of the UI.

The UI task modeling environment allows identifying different elements as task parts. The task parts may be linked to web service invocations, UI elements, or a combination of both. The UI tasks are assigned directly to users or through roles. During this assignment, fine grained authorizations can be granted by enabling or disabling different task parts. The assigned UI task is called a configured task. It takes care of updating all access controls for a particular user or role that are required to invoke the UI task.

For example: An administrator can allow certain users to start and stop service containers, and limit other users to only view the status of service container. Both groups will have the same UI with some options enabled or disabled. Enabling and disabling is done through configuration, not during the development time. Bypassing the user interface does not breach security, as the related back end authorizations are granted along with the UI permissions.

It is possible to define composite UI tasks. UI tasks can be linked to workflow tasks through the Inbox.

### Security

#### Authentication

Authentication is about establishing the identity of the user within the Process Platform system in a secure and trusted way.

Process Platform has multiple options for user authentication:

- Web server authentication
- Process Platform authentication
- SAML authentication
- OpenText™ Directory Services (OTDS) authentication

#### WEB SERVER AUTHENTICATION

With web server authentication, the responsibility of authenticating the user is put at the web server. The web server handles user authentication and passes the user identity on to Process Platform.

Web server authentication can be handled in different forms:

- Basic, Digest, Domain Authentication (NTLM)
- Certificate-based authentication.

Certificate-based authentication adds an extra level of security to the platform as Public Key Infrastructure (PKI) is used to identify the user. Each user has a unique certificate, which will be used for authentication in the web server. Only after the client certificate is validated is the user identity passed on to Process Platform.

#### PROCESS PLATFORM AUTHENTICATION

With Process Platform authentication, user authentication is performed by the Process Platform Single Sign-On service (SSO) instead of by the web server.

User authentication is based on a username and password and, after validating them, the SSO generates a signed SAML 1.1 assertion that states the user identity. This SAML 1.1 Assertion includes a SAML artifact that can be used as a reference to the SAML assertion.

With each web service request from the front end, this SAML artifact communicates in a web-browser session cookie. The web gateway looks up the corresponding SAML assertion for the given SAML artifact and uses this internally in Process Platform.

When a user logs on to Process Platform, the username and password are entered in a login form and sent to the Process Platform SSO service. The username and password, by default, are validated against the **Cordys Administration Repository Server (CARS)**.

Process Platform Authentication is an extensible system that allows custom login forms to be used in the front end. The Process Platform SSO service also has an extension mechanism, so that one can also authenticate against other sources (e.g. Microsoft Active Directory or a database).

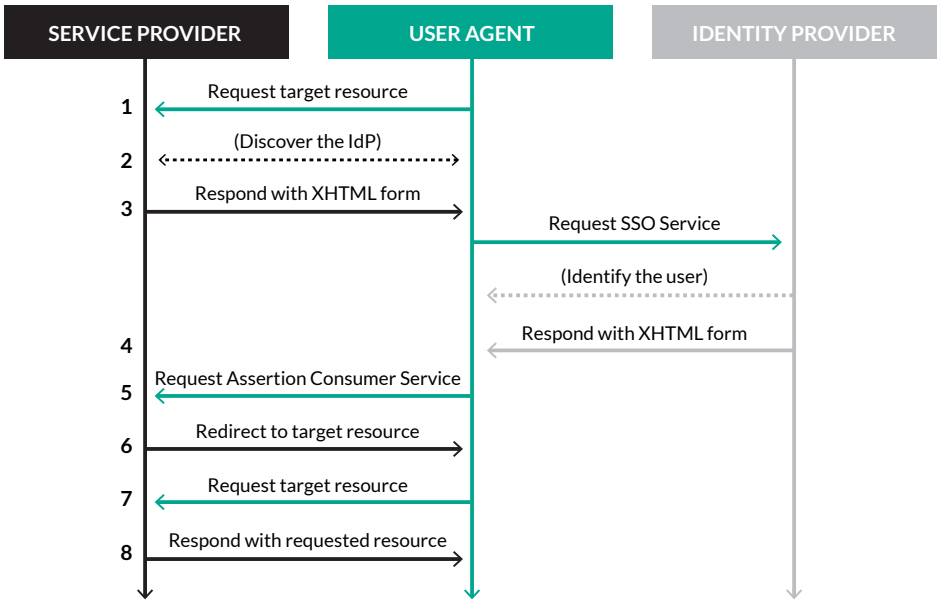
As complete user authentication is done by Process Platform, authentication in the web server needs to be disabled.

### SAML AUTHENTICATION

Another form of non-web server authentication is based on SAML. Here, authentication is relayed to an external service or **Identity Provider (IDP)**. Process Platform implements the SAML 2 Web browser SSO profile[32], which means the external IDP must support the SAML 2.0 standard.

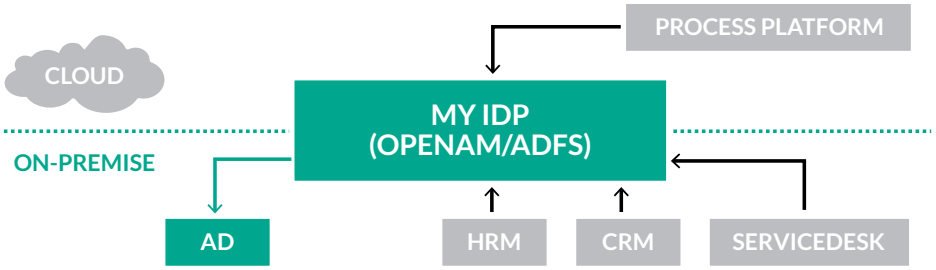
When Process Platform needs to authenticate the user, the browser is redirected in a separate window to the configured external IDP that handles the authentication. After the user is authenticated, the external IDP Posts a SAMLResponse back to the Process Platform SSO service. The SAMLResponse contains a signed SAML 2.0 Assertion that states the identity of the user. This external SAMLResponse is validated and matched with the available users in Process Platform. After validating the external SAML 2.0 Assertion, a new internal SAML 1.1 Assertion is generated. So the external SAML 2.0 Assertion is only used once for user authentication.

The flow between the browser, Service Provider and Identity Provider is given below. In this flow, Process Platform SSO is the Service provider and the User Agent is the user's browser.




The external authentication is based on a two-way trust configuration between Process Platform and the external IDP. The Security Admin Console in Process Platform is used to configure one part of this trust. See the Security Management space[33] for documentation and examples.

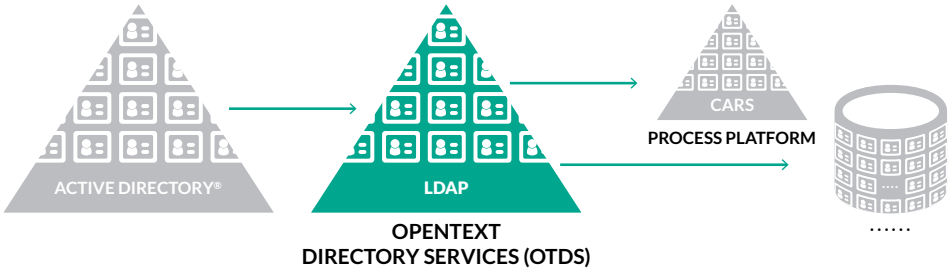
The external IDP might be cloud-based or on-premise. Direct communication between the external IDP and Process Platform is not required. The information exchange is done through browser redirects and automatic form submits only.



## OPENTEXT DIRECTORY SERVICES (OTDS) AUTHENTICATION

OpenText Directory Services (OTDS) is a product for user authentication and identity management across OpenText products. OTDS provides the user with Single Sign-On across various OpenText products. Users can be managed in OTDS and information about the identity of a user can be exchanged between products that support OTDS. OTDS supports various authentication methods, including Microsoft Active Directory and two-factor authentication. Given an authentication proof for a user in one product, OTDS can provide the authentication proof of that same user for another product.


OTDS supports synchronization of users, groups, and group memberships across the connected products. With this, users and groups can be managed centrally, in OTDS or its identity provider (for instance Microsoft Active Directory), across all OpenText products.




## ADVANCED AUTHENTICATION FEATURES

Advanced authentication features, such as password policies, strong authentication, One-Time-Passwords (OTP) are supported through the external IDP.

When an SSO user experience is needed, this can be implemented by configuring multiple applications with the same Identity Provider. When all involved applications share an Identity Provider, the user only needs to authenticate once at the Identity Provider and not on a per application basis.

### Authorization

Process Platform has a role-based access system. By assigning roles to users, they get the set of privileges as defined in the Access Control Lists (ACLs) of these roles. Roles can be composed of other roles.

## ROLES

There are two kinds of roles.

- A package role is defined during application development. It defines access to web services, data, and tasks. This is packaged as part of an application package and loaded to the Shared space. A call handling application could, for instance, introduce an Escalation Manager role. The following role types exist:
  - *Is functional:* indicates whether the role is to be shown in organization model.
  - *Is internal:* indicates it should be skipped in the organizational modeler and user manager.
  - *Is technical:* indicates it should be skipped in the organizational modeler but shown in the user manager.
- An organizational role is defined during runtime. It is created on an organization level and normally aggregates package roles. An organization could define the roles *Project Manager* and *Development Manager* and assign the application role *Escalation Manager* to both.

### ACCESS CONTROL LIST

The ACL contains a set of authorizations or permissions that are added to a role or user. Each time a user accesses a protected resource (e.g. a web service), an access control request is formulated and matched against the complete set of ACLs for this user.

### Web gateway security features

The sandboxing feature of the web gateway can be leveraged to enhance the security of an Process Platform system. It is implemented as part of the ISAPI extension and the Apache module running in the web server.

Sandboxing is a mechanism that restricts the SOAP requests and tells them when they can be executed on a web server. It is configured through the Management Console on the server.

# Conclusion

Process Platform offers a strong foundation to build business applications. This paper describes the design goals of the platform and offers an insight into the design-time architecture, as well as the runtime architecture of the platform.

We could only cover the highlights of the architecture here. To learn more, join the Process Platform Community at http://community.cordys.com. You will find many articles about various aspects of the platform, as well as an active community that can answer any question you might have.

# About OpenText

OpenText enables the digital world, creating a better way for organizations to work with information, on premises or in the cloud. For more information about OpenText (NASDAQ: OTEX, TSX: OTC) visit opentext.com.

**Connect with us:**

- OpenText CEO Mark Barrenechea's blog
- Twitter | LinkedIn | Facebook

# Applicable Standards

Process Platform plays in the domains of business process management, cloud computing, and web services. These domains are being standardized by standards bodies such as W3C, Oasis, and OMG, each producing numerous standards. The following table lists the most important standards supported by Process Platform.

* Partial support | ** From Process Platform 10.8 onward

| WEB SERVICES, XML, AND INTERNET STANDARDS | | |
|---|---|---|
| **CSS** | Style sheet language to describe the look and feel of an HTML document | 2.1 |
| **HTTP** | Hypertext Transfer Protocol is the method used to transfer information over the Internet | 1.1 |
| **HTTPS** | Combination of a normal HTTP interaction over an encrypted secure socket layer (SSL) or transport layer security (TLS) | 1.1 |
| **LDAP** | Lightweight Directory Access Protocol, a standard for organizing directory hierarchies and interfacing to directory servers | V3 |
| **SOAP** | A standard for exchanging XML-based messages over a computer network, normally using HTTP | 1.1 |
| **UDDI** | An XML-based protocol that provides a distributed directory that enables businesses to list themselves on the Internet and discover other services | V2 |
| **WSDL** | Web Services Description Language is the standard format for describing a web service | 1.1 |
| **XFORMS** | An XML format for the specification of user interfaces, specifically web forms | 1.1 |
| **XINCLUDE** | A W3C specification defining a general purpose inclusion mechanism for XML documents | |
| **XML** | XML provides a text-based means to describe and apply a tree-based structure to information | 1.0 |
| **XML-DOM** | Description of how an HTML or XML document is represented in an object-oriented fashion | 3.0 * |
| **XPATH** | Language that describes how to locate specific elements in an XML document | 1.0 |
| **XSD** | A way to describe and validate data in an XML environment | 1.0 |
| **XSLT** | Language for transforming XML documents into other XML documents | 1.0 |
| **SAML** | Security Assertion Markup Language is an XML standard for exchanging authentication and authorization data between security domains, that is, between an identity provider (a producer of assertions) and a service provider (a consumer of assertions). | 1.1 *<br>2.0 * |
| **WS-I BASIC PROFILE** | WS-I provides resources for Web services developers to create interoperable Web services and verify that their results are compliant with WS-I guidelines. Key WS-I deliverables include Profiles, Sample Applications and Testing Tools. | 1.1 |
| **WS-I BASIC SECURITY PROFILE** | Describes a set of security specification with the goal of creating interoperable web services. Process Platform does not support Kerberos and REL. | 1.0 * |
| **BPMN** | Business Process Model And Notation™ | 2.0 * |
| **CMMN** | Case Management Model And Notation™ | 1.0 * |
| **XPDL** | Standardized XML-based language for import/export of BPM | 2.0 |
| **CMIS** | Content Management Interoperability Services | 1.1 ** |
| **SECURITY STANDARDS** | | |
| **SSL / TLS** | Commonly-used protocol for managing the security of a message transmission on a network | 3.0/1.2 |
| **X.509** | A security standard that defines a certification process by which users are authenticated | V3 |
| **XML ENCRYPTION** | Specification that defines how to encrypt the content of an XML element | 1.0 |
| **XML SIGNATURE** | W3C recommendation that defines an XML syntax for digital signatures | 1.0 |
| **WS-SECURITY** | WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication | 1.1 * |
| **JAVA STANDARDS** | | |
| **JMS** | Java messaging service is the standard API for sending and receiving messages | 1.1 |
| **JMX** | Technology that supplies tools for managing and monitoring applications, system objects, devices, and so forth. | 1.0 |

# References


| | |
|---|---|
| 1 | http://en.wikipedia.org/wiki/Development,_testing,_acceptance_and_production |
| 2 | http://en.wikipedia.org/wiki/Service-oriented_architecture |
| 3 | http://en.wikipedia.org/wiki/Software_configuration_management |
| 4 | http://subversion.apache.org/ |
| 5 | http://www.eclipse.org/ |
| 6 | http://msdn.microsoft.com/en-us/vstudio/default.aspx |
| 7 | http://nl.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol |
| 8 | http://tomee.apache.org/ |
| 9 | https://www.w3.org/MarkUp/Forms/ |
| 10 | http://en.wikipedia.org/wiki/Right-to-left |
| 11 | http://www.bpmn.org/ |
| 12 | http://www.w3.org/TR/scxml/ |
| 13 | http://en.wikipedia.org/wiki/Create,_read,_update_and_delete |
| 14 | https://wiki.cordys.com/display/PI/2009/06/17/The+how+and+why+of+the+Cordys+XML+engine |
| 15 | https://wiki.cordys.com/display/PI/2009/05/22/Bringing+NOM+to+the+Java+Developer |
| 16 | https://wiki.cordys.com/display/PI/Using+DOM+APIs+Over+Cordys+NOM |
| 17 | http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile |
| 18 | http://en.wikipedia.org/wiki/Gossip_protocol |
| 19 | http://en.wikipedia.org/wiki/X/Open_XA |
| 20 | https://wiki.cordys.com/display/PI/2009/12/22/Guidelines+for+building+distributed+applications+and+frameworks |
| 21 | https://wiki.cordys.com/display/PI/2010/02/11/SSU+Framework+highlights+-+1 |
| 22 | https://wiki.cordys.com/display/PI/2010/03/19/SSU+-+Framework+highlights+-+2 |
| 23 | https://wiki.cordys.com/display/PI/2009/11/10/Distributed+cache+invalidation |
| 24 | https://wiki.cordys.com/display/PI/2009/12/01/State+registry+in+Cordys |
| 25 | https://wiki.cordys.com/display/dsc/Connector+Directory |
| 26 | http://jcp.org/en/jsr/detail?id=170 |
| 27 | http://docs.oasis-open.org/cmis/CMIS/v1.1/os/CMIS-v1.1-os.html |
| 28 | http://jackrabbit.apache.org/ |
| 29 | http://en.wikipedia.org/wiki/Tag_(metadata) |
| 30 | https://wiki.cordys.com/display/PI/2009/08/20/The+how+and+why+of+Cordys+HTTP+Gateway |
| 31 | https://wiki.cordys.com/display/PI/2009/09/10/Cordys+Web+Gateway+Security+-+Part+1 |
| 32 | http://en.wikipedia.org/wiki/SAML_2.0#Web_Browser_SSO_Profile |
| 33 | https://wiki.cordys.com/display/SecMan/Security |




Copyright © 2016 Open Text SA or Open Text ULC (in Canada). All rights reserved. Trademarks owned by Open Text SA or Open Text ULC (in Canada). (05/2016)04761EN