OpenText™

OpenText Cordys Architecture

Learn how the platform enables customers to improve their business operations

This whitepaper provides an introduction to OpenText Cordys architecture and explains how OpenText uniquely enables customers to improve their business operations. It is intended for solution architects and other technical people who wish to obtain a thorough understanding of the technical aspects of OpenText Cordys. The whitepaper touches upon the technical capabilities of OpenText Cordys and also gives insight in the way these capabilities are provided.

OPENTEXT™

Table of Contents

Architecture vision and goals

At OpenText, we see it as our mission to help our customers improve their business operations with world-class, process oriented software which allows them to change and innovate the way they do business with greater speed and flexibility. This mission is translated in 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	Enables any user, inside or outside the company, to work with the OpenText Cordys system 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
STANDARDS COMPLIANCE	Enables easy integration, thus reducing total cost of ownership
EXTENSIBLE ENVIRONMENT	Easy extensibility drives total cost of ownership down
INTERNET AND INTRANET DEPLOYABILITY	Same platform can be used for cloud computing and on premise
LINEAR SCALABILITY	Linear scalability on commodity hardware keeps total cost of ownership low
HIGH AVAILABILITY	Business critical systems demand high availability
MULTITENANCY	Cloud computing is a core feature of the platform

High-level overview of OpenText Cordys

OpenText Cordys is inherently integrated as it is built as a single product. All features are based on one technology. The picture below provides a generic overview of the platform capabilities.

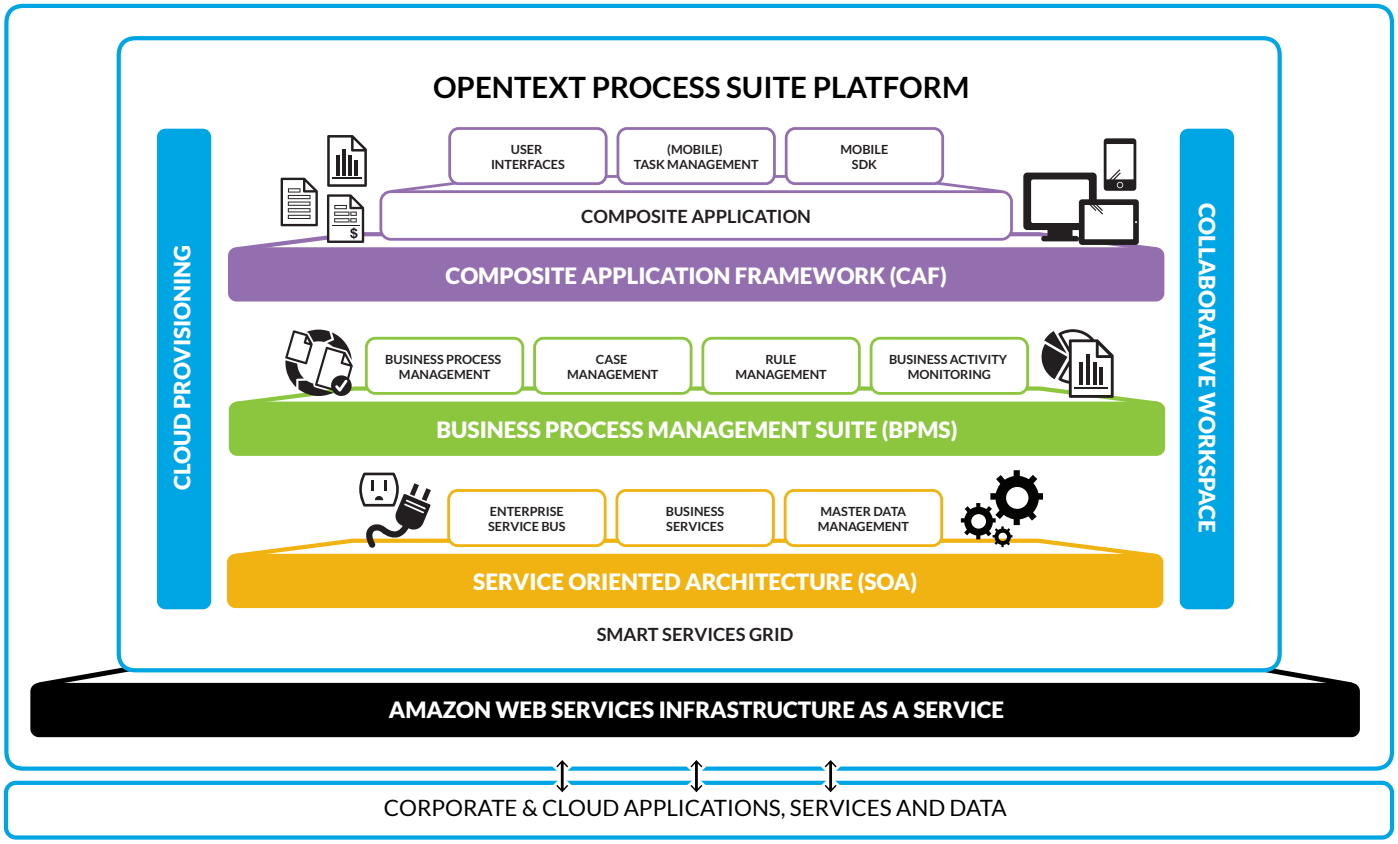
OpenText is unique in that it has designed a platform to bridge three seemingly disconnected worlds: SOA-based Integration, Business Process Management and Composite Application Development (see the three platform layers in the below picture). OpenText Cloud Provisioning is built on top of OpenText Cordys; it provides automated provisioning and metering of applications for the cloud.

In this architecture whitepaper, we will focus on OpenText Cordys.

A key objective of OpenText Cordys is to enable business users to participate in a model-driven application development. This has been done before: many platforms allow business people to participate through contribution of models that programmers take as input. OpenText Cordys takes a radically different approach: the model is the application, not merely an input to a programmer. To enable that, OpenText Cordys application development is mostly model driven.

The OpenText Cordys modeling environment is built as an application on top of the OpenText Cordys run-time environment. This delivers the following benefits:

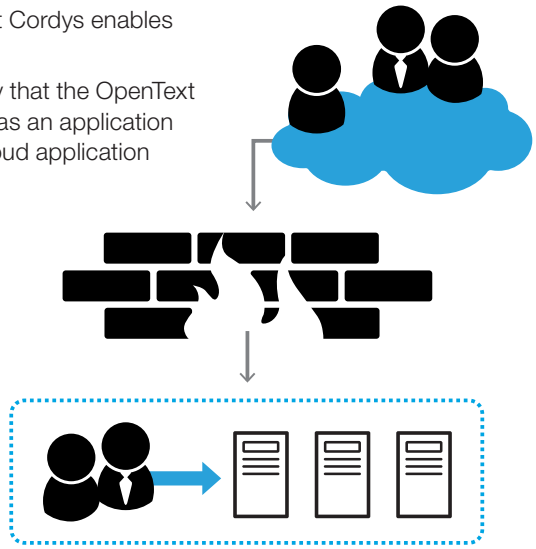
- **Scalable, robust, and secure:** All platform run-time features like scalability, high availability, and security directly contribute to the design-time environment
- **Testable:** No need to install another product to enable testing
- **Available everywhere:** Every OpenText Cordys installation comes with built-in design capability



The full functionality of OpenText Cordys 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 OpenText Cordys, as well as OpenText Cordys-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. OpenText Cordys provides support for the industry's most popular browsers, Chrome™, Firefox, Internet Explorer® and Safari®.

As mentioned above, the design-time environment is based on the platform run-time environment. This however does not imply that the application needs to run in the environment where it has been 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 OpenText Cordys enables two scenarios:



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

OpenText Cordys provides the following basic features:

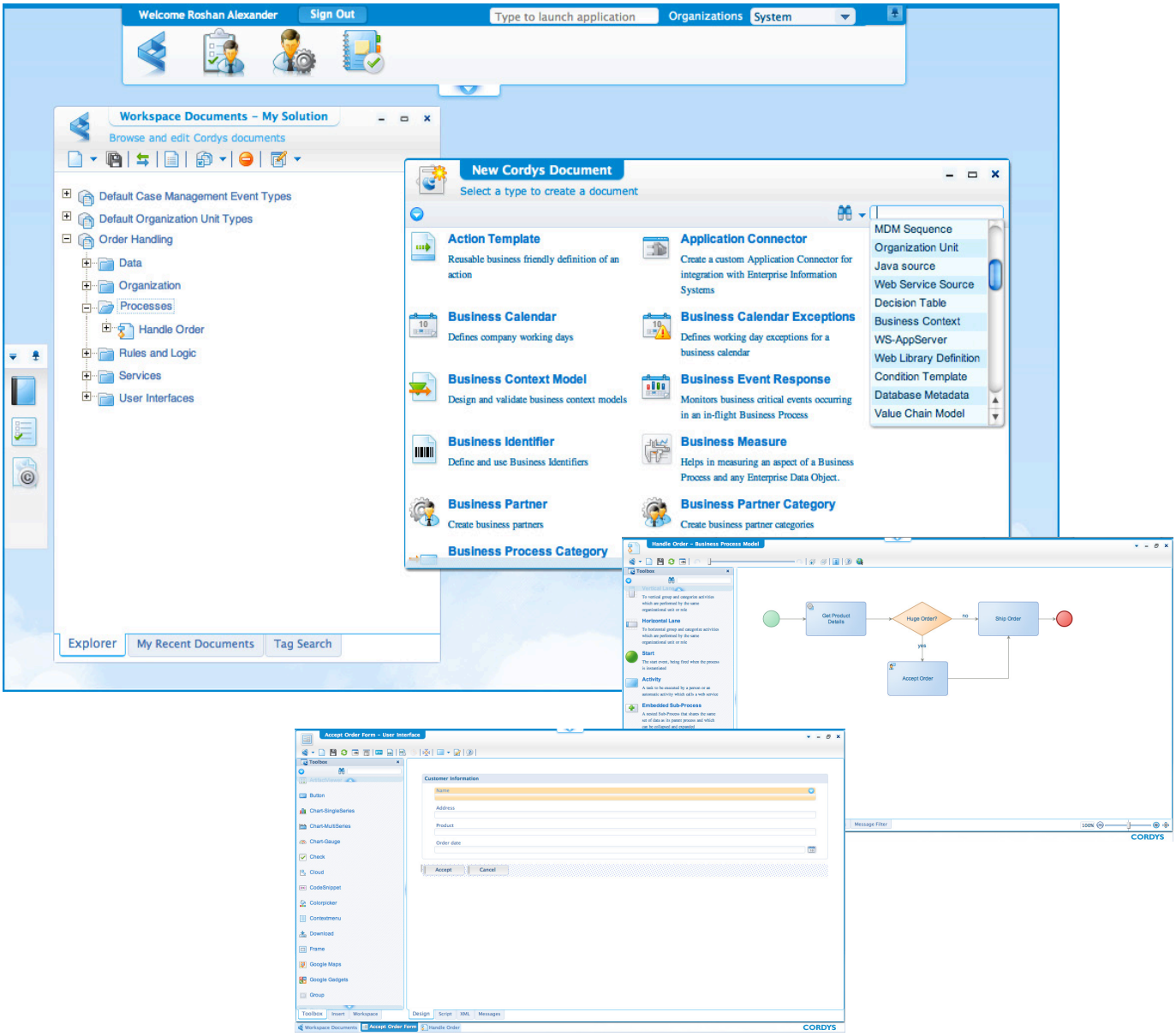
- **High availability:** Mission critical applications must be available always. The OpenText Cordys platform can be deployed on a network of systems, ensuring there is no single point of failure.
- **Scalability:** Enterprises today process thousands of business processes a day. The platform is built 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 OpenText Cordys that can also be useful in some on-premise scenarios.
- **Security:** Cybercrime being very common, it is of crucial importance to harness appropriately. OpenText Cordys 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.

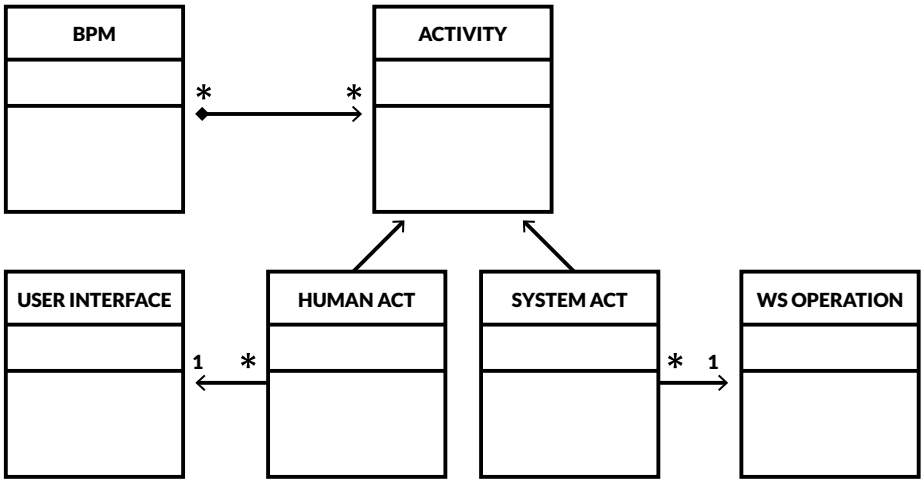
Design-time architecture

This chapter drills little deeper into the design-time architecture of OpenText Cordys. After a quick introduction to our model-driven approach, it 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 Collaborative Workspace (CWS).

Model driven

OpenText Cordys takes a model driven approach towards 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, application object, WSDL, etc. Model developers use a browser to create new models or optimize existing ones. Most models are represented through graphical models, featuring a responsive rich user interface.





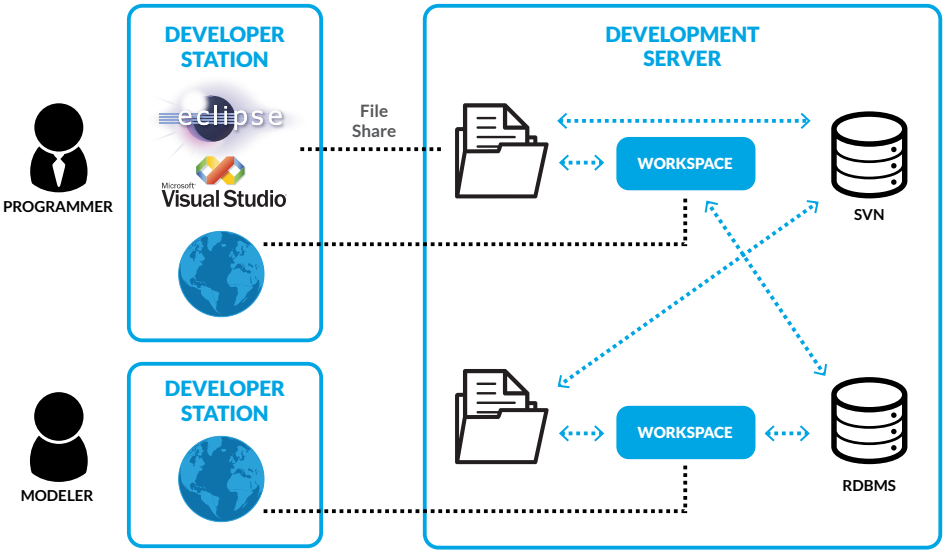
Integrated metamodel

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, etc. 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 a belonging Java class that takes care of the run-time behavior inside the CWS service. The XML document representing the models is stored in a relational database through XDS (for more information see Repository section).

A simplified view of a part of the metamodel looks like the above diagram.

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. As OpenText does not want to play in the Software Configuration Management market, 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 like 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

OpenText Cordys comes out-of-the-box with several modelers for Business Processes, User Interfaces, Organization Models, etc. All these modelers are built on an extensible, completely metamodel driven designer platform called CWS. In this section, we will look at the anatomy of a modeler, and explain how to build one.

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

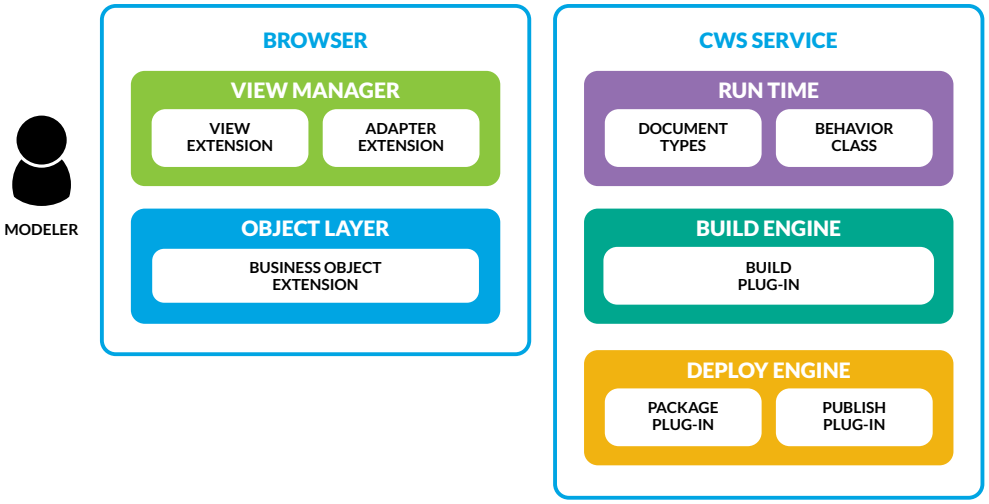
Class definition

```
<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="{78817809-7334-4903-93D0-0B6B83918B45}">
   <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 which produces classes like view extension, adapter extension, build plug-in, etc. 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 picture below visualizes where the various classes are used when running CWS.



Standard CWS facilities

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

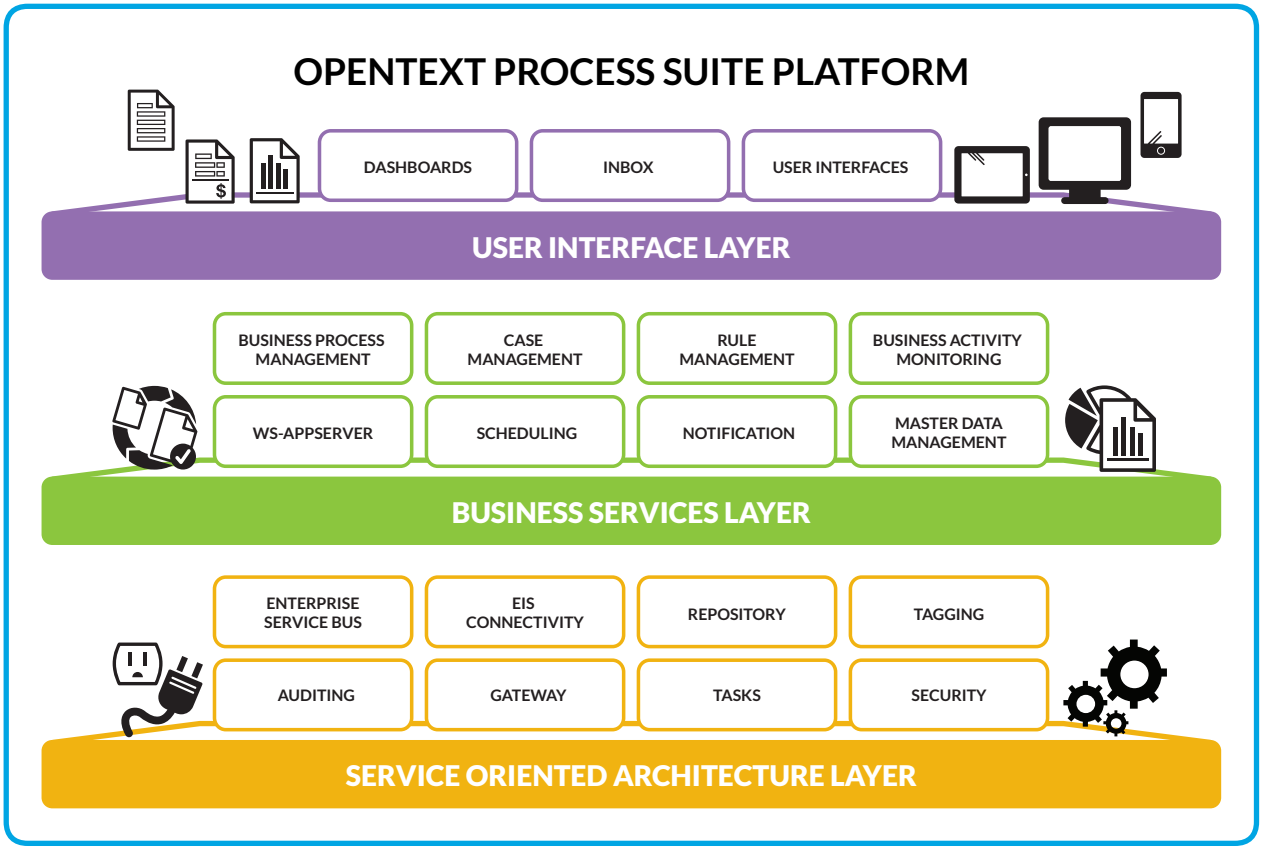
- **Where used:** Based on the associations between models, as expressed in the metamodel, CWS automatically provides “where used” overviews. This is an important tool during impact analysis and refactoring.
- **User interface:** Common look and feel across modelers is facilitated through an advance 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, 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 from unnecessarily rebuilding 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 Cordys Application Package (CAP).
- **Publish:** For testing purposes, models can directly be 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 lookup of models, it is possible to attach user defined tags to them. This is a standard facility of CWS.

Run-time architecture

At run time, OpenText Cordys is comprised of a set of web service containers, connected through a SOA grid. In this chapter we will first look at the logical view, followed by the deployment view, an explanation of OpenText Cordys multitenancy and an overview of the run-time services.

Logical view

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



We distinguish three layers:

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

Later in this chapter, we will look into each component in more detail.

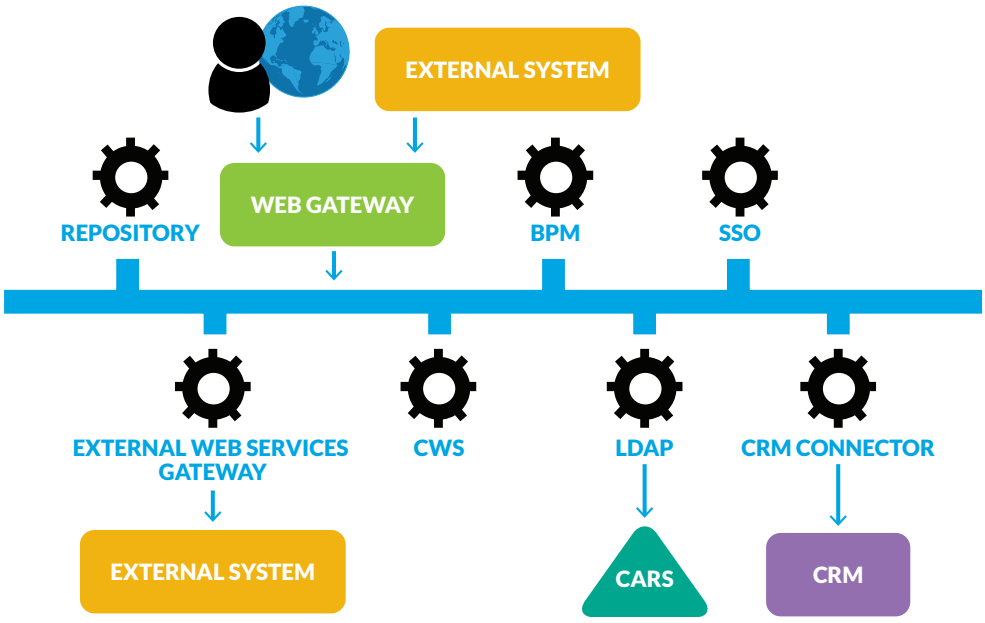
Deployment view

All run-time components are linked through the OpenText Cordys 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, whether or not 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; 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. **Fail over**
   In case 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 fail over features of the ESB.

Details of these features are discussed in the Enterprise Service Bus section later in this white paper.

Here is a diagram providing a 'bus view' on OpenText Cordys.



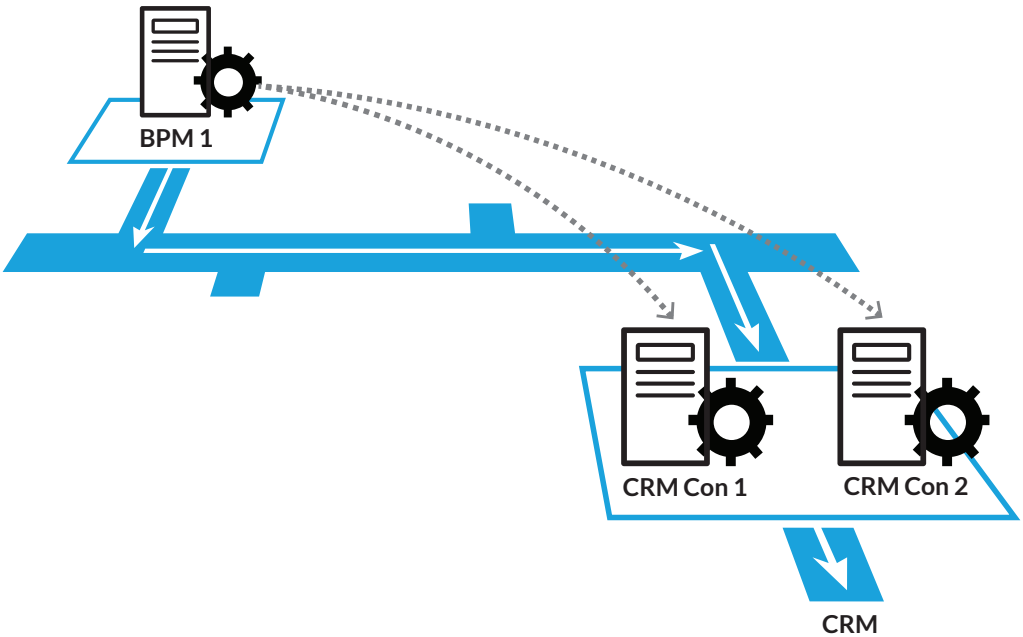
All participants on the 'bus' are equal, so there is no central 'bus coordinator' or 'single point of failure'. The web gateway is depicted with a different icon because of its special role: it only acts as a 'bus client'. It does not have a role in requests between the peers. The service icons in the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

A few words about some of the depicted services:

- **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 OpenText Cordys web services.
- **External web services gateway:** This service, often called the UDDI connector, links external web services to the bus.
- **LDAP:** OpenText Cordys uses an LDAP directory, called CARS, as run-time 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. The OpenText Cordys platform allows developing connectors specific for the various enterprise information systems in an organization. This connector bridges the proprietary protocol of the EIS with the bus.

The following picture clarifies the above explanation by depicting the interaction between 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 on running on its own system.

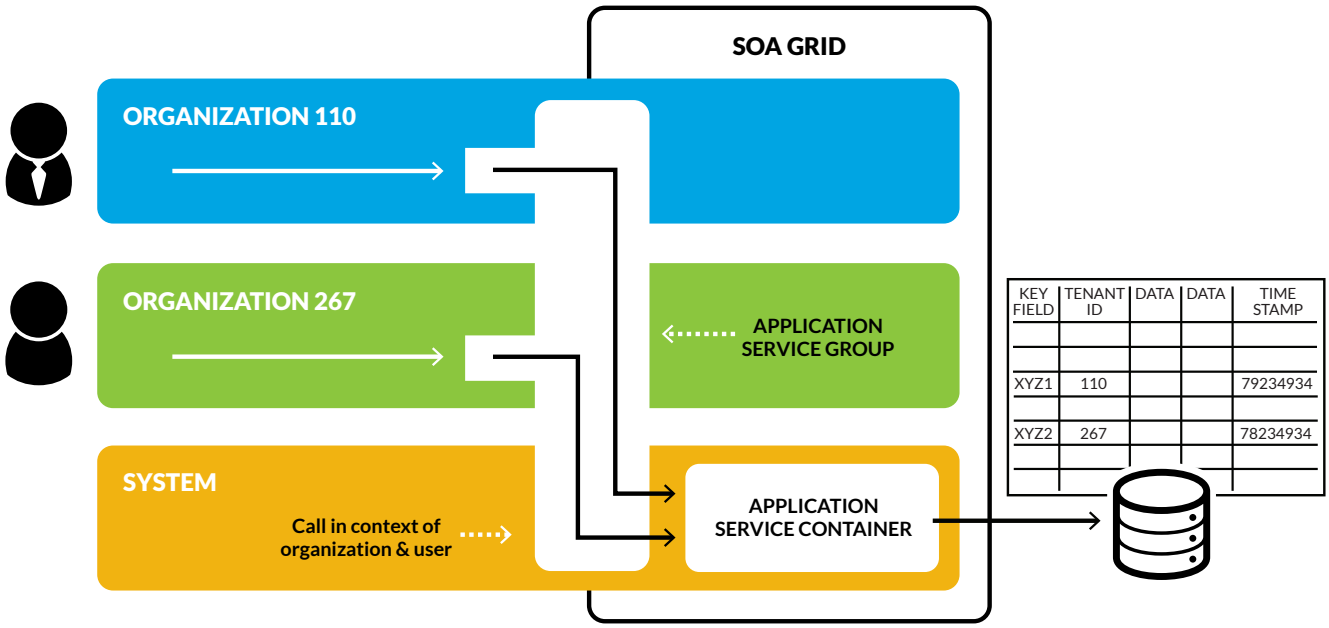
The BPM engine directly connects to one of the two CRM connectors to submit its request.



Multitenancy

The OpenText Cordys SOA Grid represents the physical or deployment aspect of the OpenText Cordys architecture. The organization concept represents a logical concept in the OpenText Cordys 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 given machine. 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 multiple organizations.

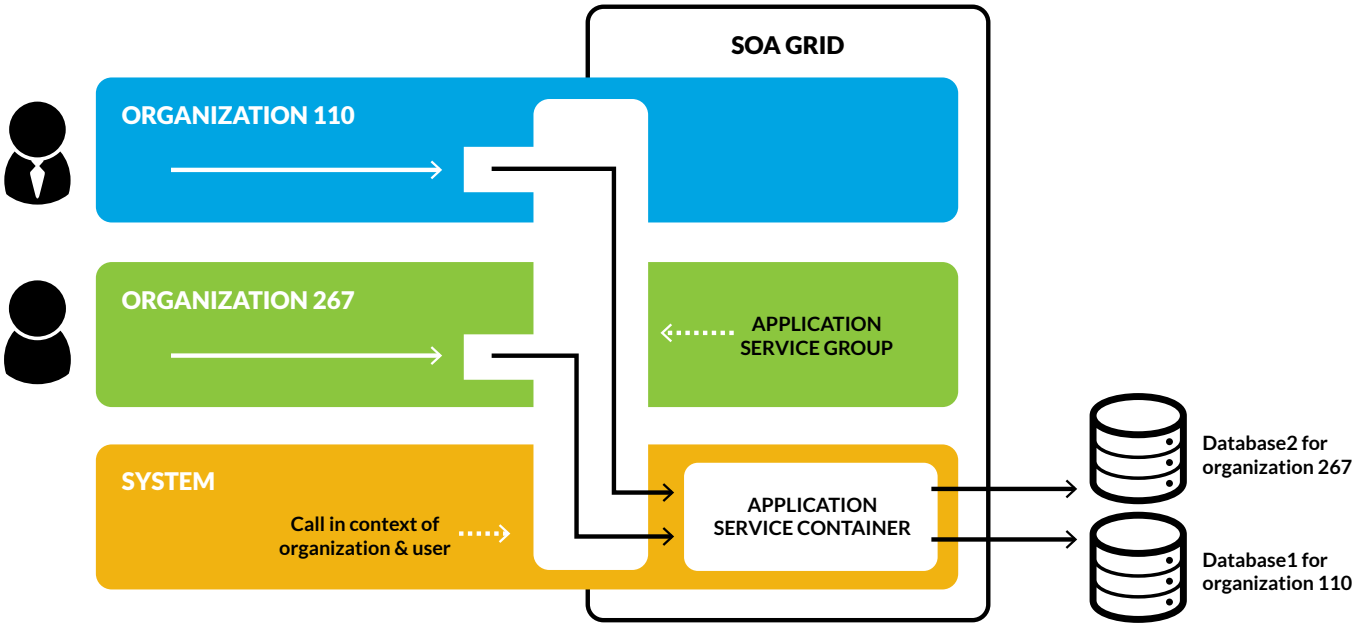
Service containers exist in the context of an organization. The functionality in a service container is exposed via a SOAP or REST interface. When a SOAP call is initiated, it is always done in the context of an organization. For efficiency and reuse, a common shared organization called System acts as 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 OpenText Cordys service containers are organization aware and use multitenant data stores to persist their data. When an Independent Software Vendor (ISV) builds a new application, they have two options to deal with multitenancy in their datastore:

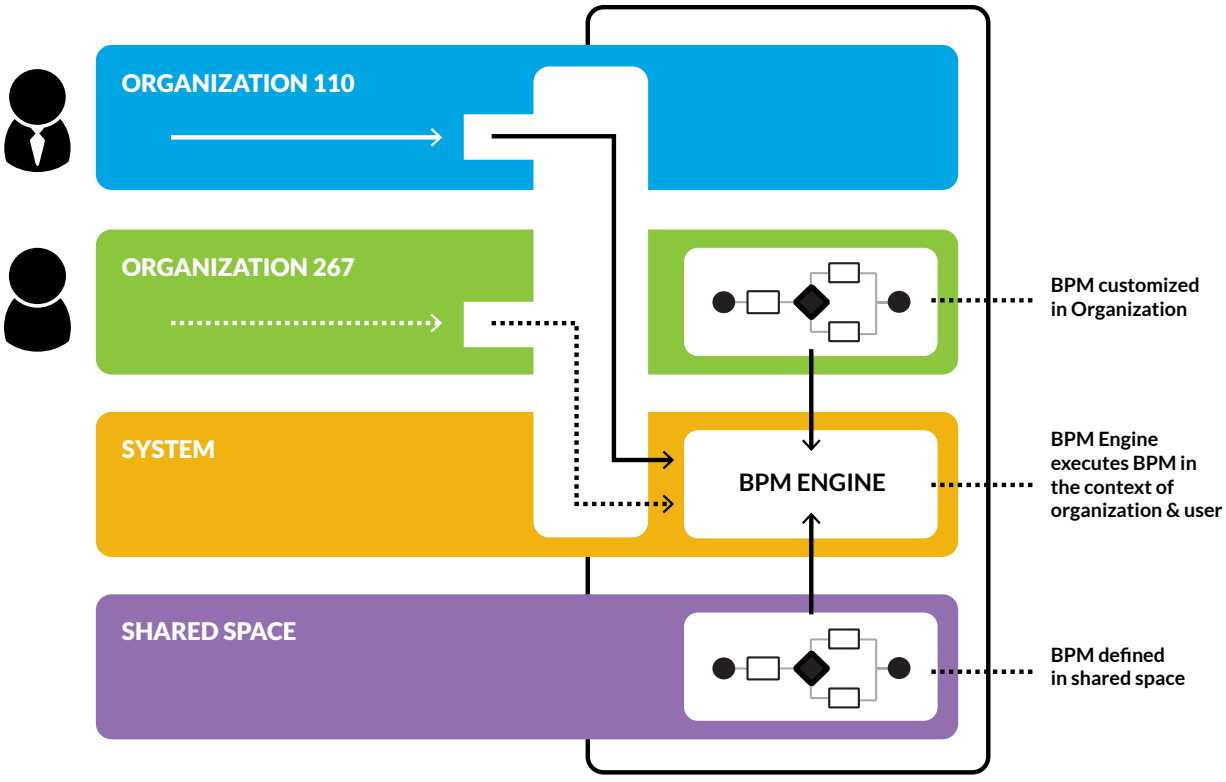
- Support multitenancy as part of the database schema, as depicted above. The service container of the application can be deployed into the System organization to allow all organizations to use this new application.
- Provision a separate database (schema) for each tenant. WS-AppServer supports per tenant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

The content of the application (e.g. 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 also 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 picture. Although the Shared space is drawn in the same fashion as an organization, it is, technically speaking, not an organization. Content (e.g. 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 below picture 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 run-time services

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

USER INTERFACE LAYER

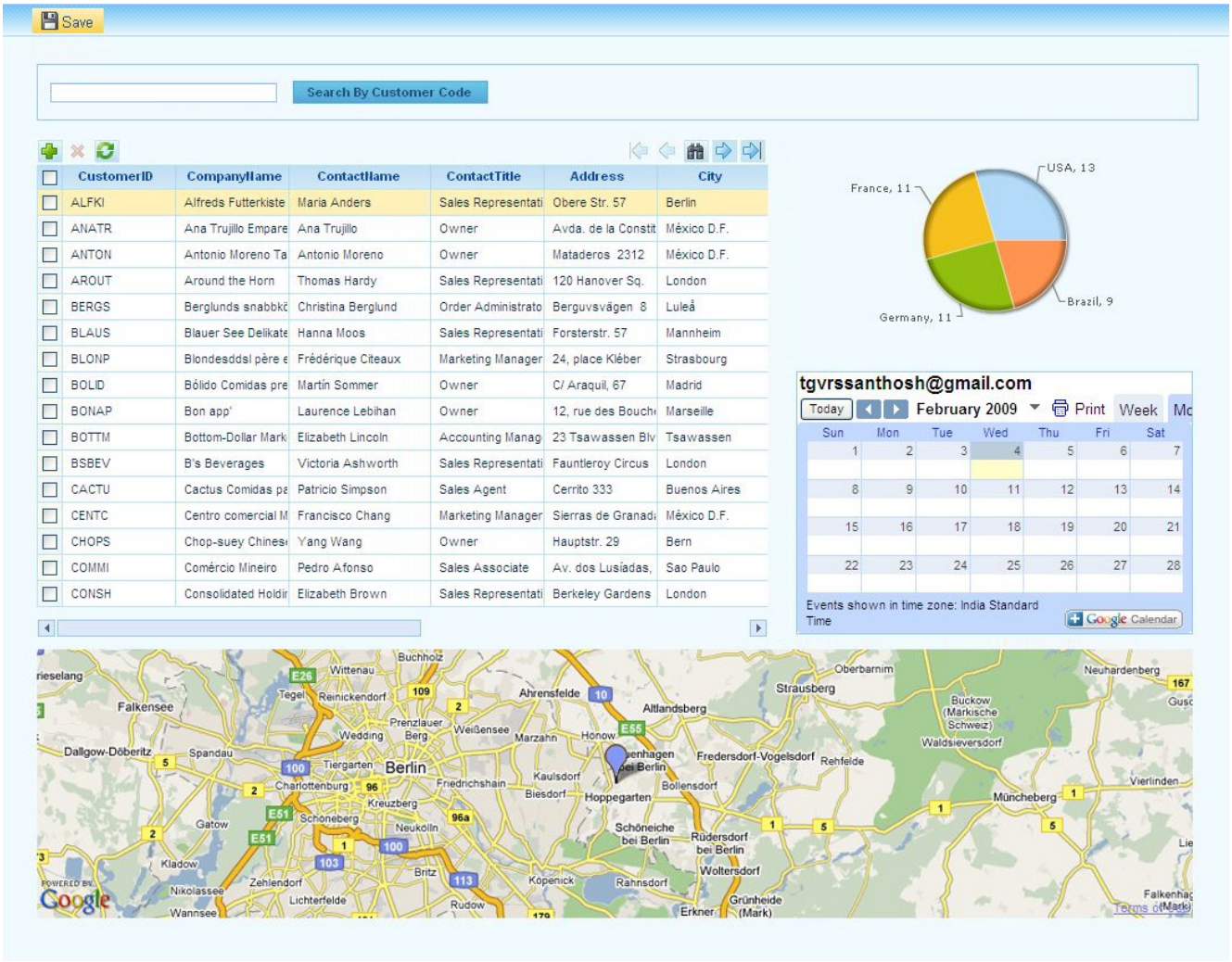
User Interfaces

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 the roles abstracted from view.

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, etc.
- 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 the files, the forms, business process models, etc.
- An **Organization Switcher** enables users with accounts in different organizations to switch to another organization.



User Interface Modeler

The OpenText Cordys User Interface Modeler makes it easy to design effective user interfaces for composite applications. With the UI modeler, power users can quickly create custom applications that leverage the full power of OpenText Cordys, link directly to services running on the OpenText Cordys ESB, and provide full transactional capability. The UI modeler features a WYSIWYG (What You See Is What You Get) approach and rich UI controls to dramatically streamline and simplify the process of converting ESB functionality into rich, intuitive, process-centric applications that run in a browser environment. The AJAX technology ensures a rich and responsive user experience.

The modeled user interfaces in OpenText Cordys are based on the XForms[8] 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. OpenText Cordys provides a rich set of UI controls including basic elements like check boxes and radio buttons, but also advanced controls like 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[9], as required for languages such as Arabic and Hebrew, or left-to-right, as required for English and French.

Dashboards

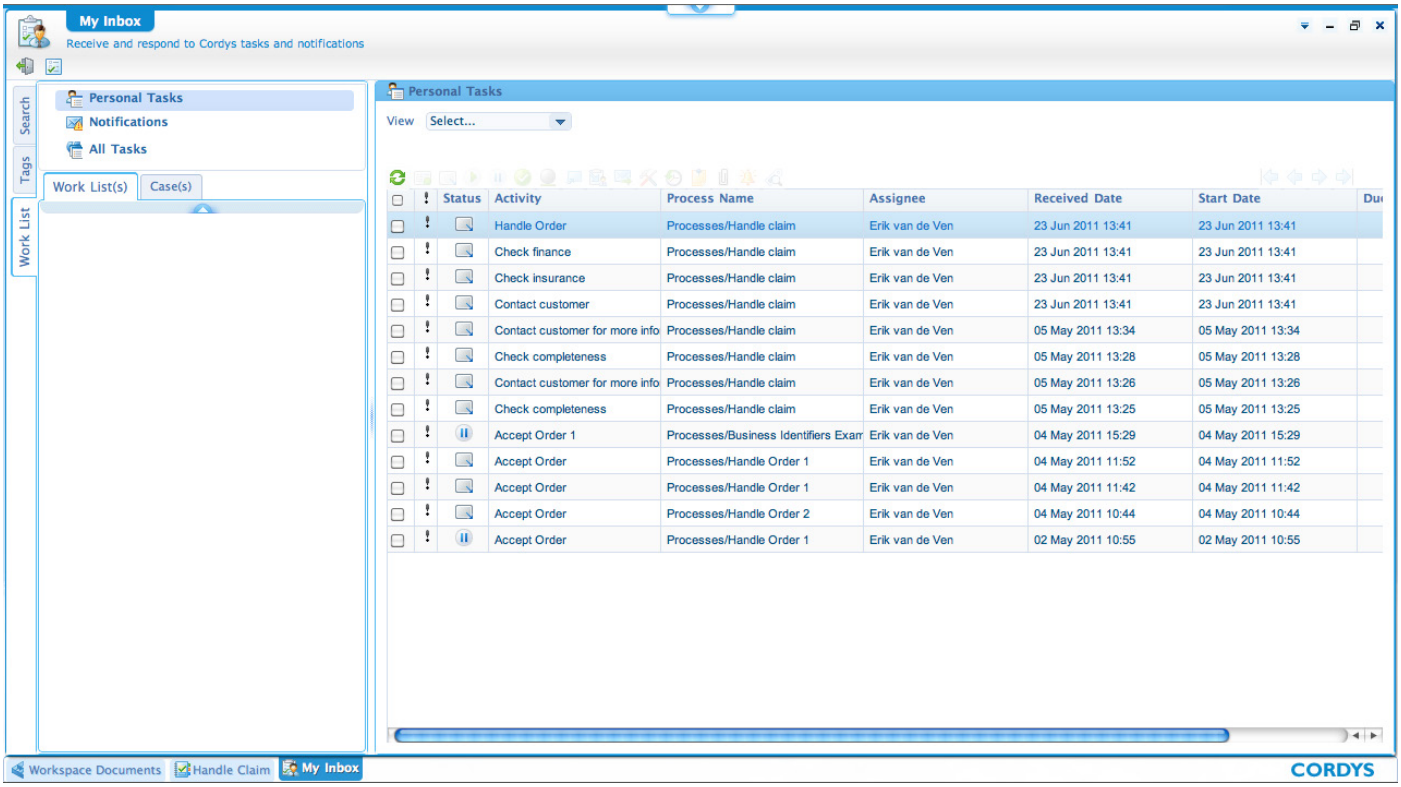
Building a dashboard is as simple as building any user interface. Every Key Performance Indicator (KPI) or business measure defined in Business Activity Monitoring comes with a composite control that provides a chart representation of the business indicator. These charts are placed on a user interface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

Inbox

The most common user interface for OpenText Cordys users is the Inbox. To facilitate this, an advanced user interface was developed that is fast and highly configurable to suit everybody's needs.

- **Case Management integration:**
  - The Inbox has a seamless integration with the Case Management solution.
  - A Case worker can view all the tasks for a particular case model, grouped by case instance, which he/she has access to work.
  - The Case Task view of the Inbox is a consolidate view of all the active tasks for a particular case instance.
  - This can be considered as a complete dashboard where the user can plan follow up activities, raise events, attach documents and also perform administrative activities like suspend, resume, forward, delegate, etc.
  - The case worker will be able to view the complete case history from the case Inbox.

- **Customization possibilities in Inbox:**
  - Inbox has customization possibilities in both 'representation' and also in 'behavior'.
  - The user can select/change language and date format.
  - The user can customize the list view of the Inbox to show the business data, so that it will be easy for the case worker to select the task that they want to work on
  - The "view customization" is possible at personal Inbox level and also at the role/team/work list level
  - Application developers can hook their custom JavaScript code via the customization hooks provided for different events like onBeforeCommit, onBeforeFollowup, etc,.



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

BUSINESS SERVICES LAYER

Business Process Management

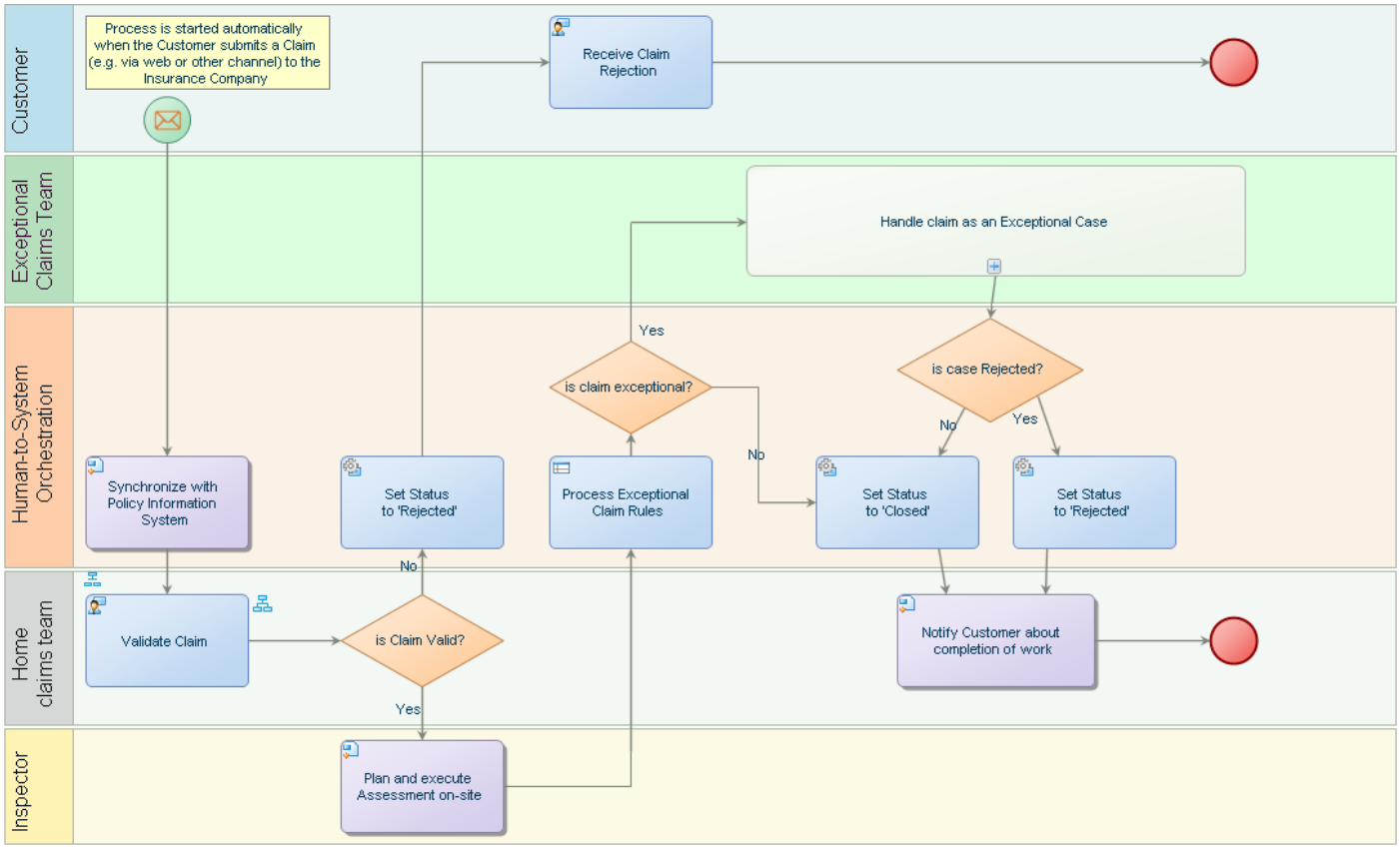
Given our focus on process oriented software, the BPM engine is one of the most important components in OpenText Cordys. Processes are defined as BPMN[10] compliant graphical models in CWS. The build step on such a model results in an XML definition which gets interpreted by the BPM engine at run time.

Some highlights:

- **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 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 OpenText Cordys deployment, the BPM engine is part of the 'Business Process Management' service group, together with Case Management. To optimize performance, this service group runs 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 Process Instance Manager (PIM) tables. The Process Instance Manager user interface provides access to this data.
- **Multitenant:** Like any OpenText Cordys 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 accomplished by embedding WS-AppServer in the BPM service container. This allows short-lived process 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 Process Instance Manager.
- **Reliable messaging:** When using a reliable message transport, the BPM engine will coordinate the transactions on the process instance manager tables and the message oriented middleware to ensure a web service operation is executed once, including the handling of the service response.

Following is a screenshot of a real world BPMN flow in OpenText Cordys 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 OpenText Cordys orchestrates **human tasks** (via user interfaces) and **system tasks** (e.g. 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.

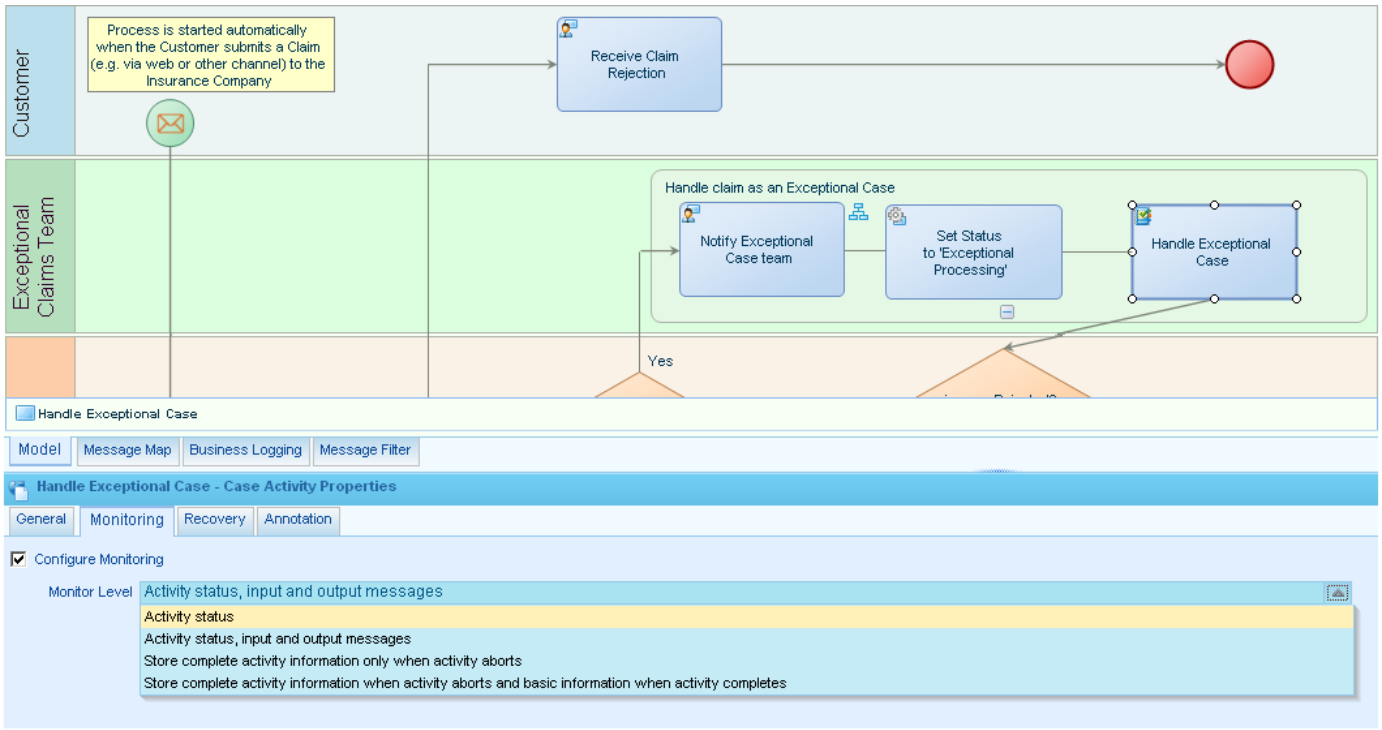
Case Management

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 next steps to be taken. This is the 'sweet spot' for Case Management. In 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 XML that at run time gets interpreted by the state machine of Case Management.

Some highlights:

- **Fully state machine based:** A case model is converted into a state model, including aspects like follow-up relations that might not seem state-related at first glance.
- **Standard SCXML format:** The state definition complies with the SCXML[11] 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 OpenText Cordys deployment, the case engine is part of the Business Process Management service group, together with the BPM engine.

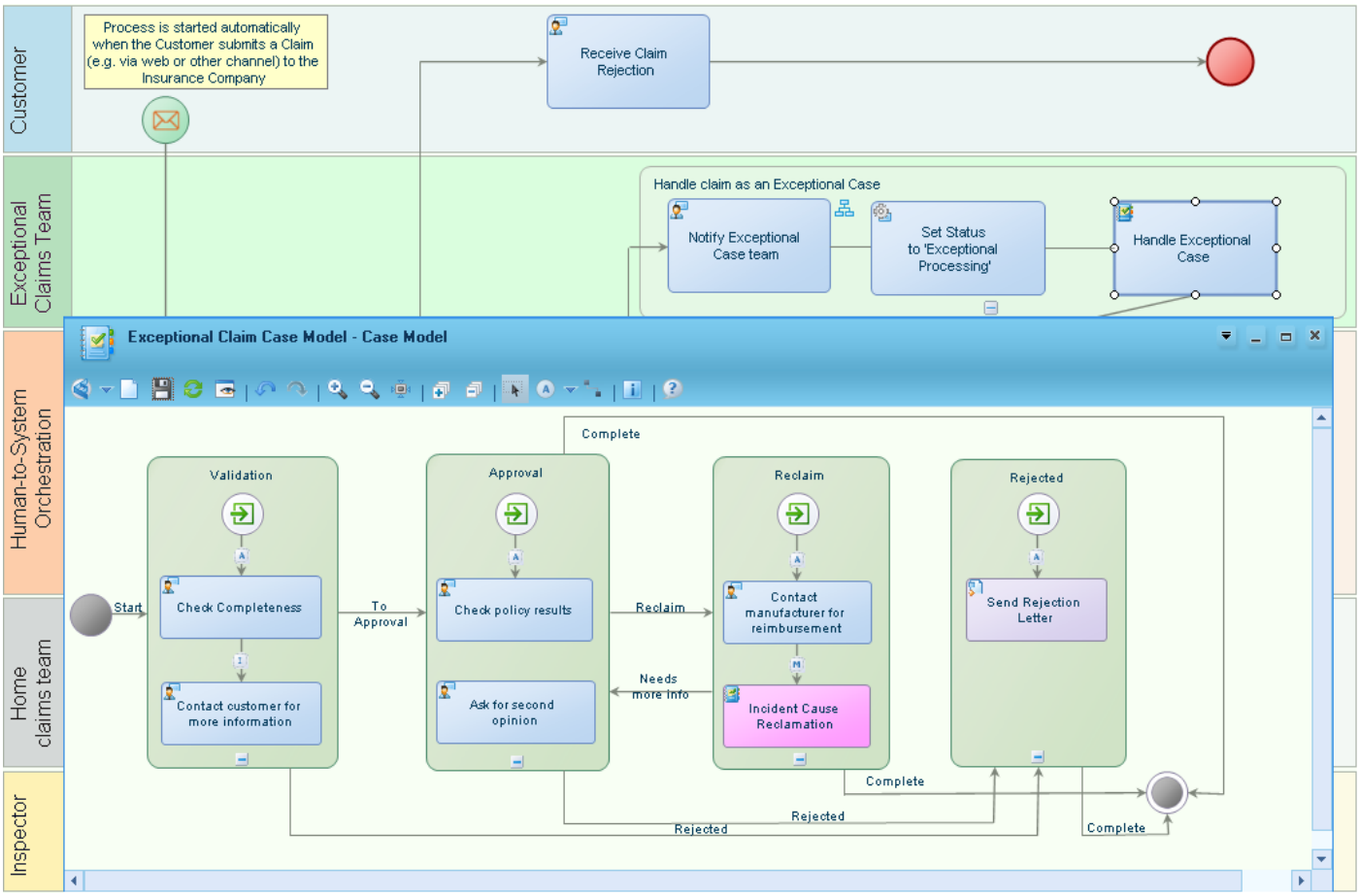
Below is a screenshot of the case model invoked from the main BPMN flow shown above. It demonstrates how the BPMN flows and case models can invoke each other. This case model has four functional states and the arrows depict the state transitions which are often event-based, trigger-based.



If you compare the main BPMN flow given above with the previous picture, you can see that OpenText Cordys has the option to collapse and expand groups of activities. The embedded subprocess in the above screenshot labeled Handle claim as an Exceptional Case is expanded by the process designer below into detail activities of which the last one, labeled Handle Exceptional Case invokes the case model.

As you can see, the way this invocation works can even be configured via the properties boxes (using Tabs) allowing process designers to tune run-time behavior exactly to their needs.

Below is the actual case model invoked from above highlighted BPM activity, showing the vice-versa integration of BPMN flows and case models:

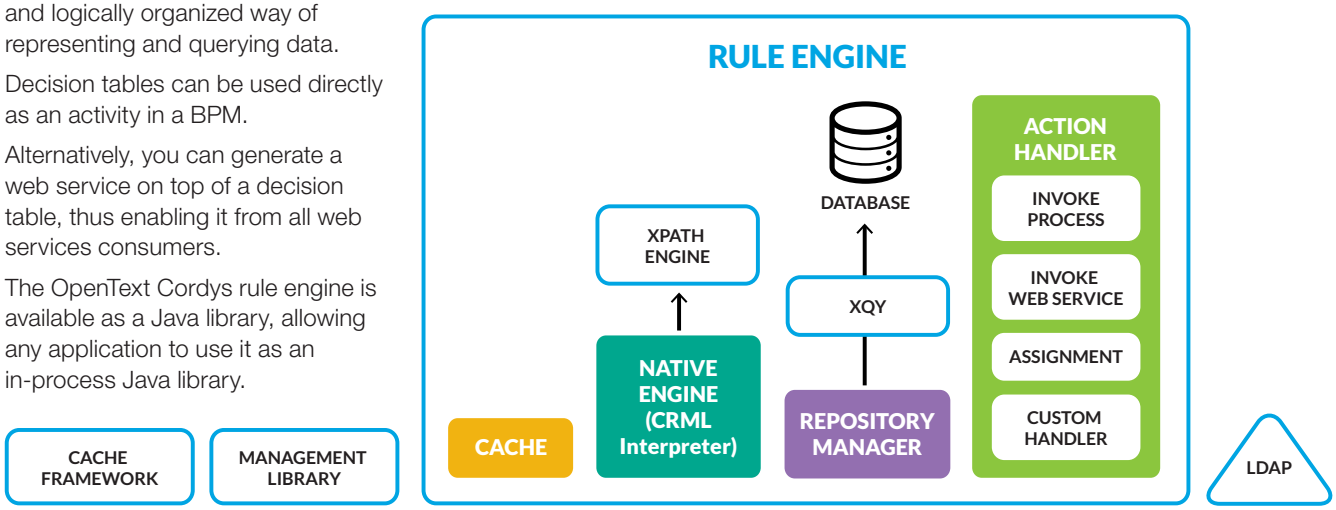


Rules Management

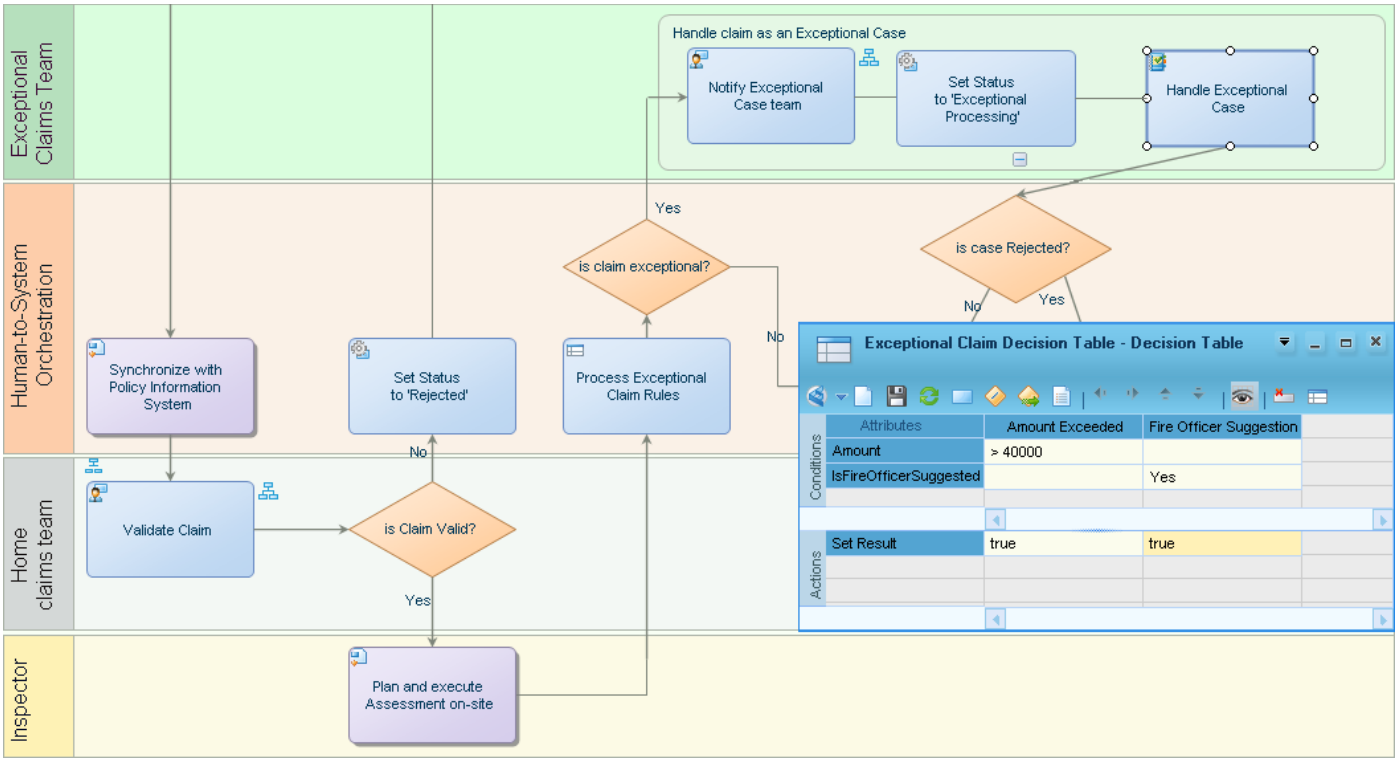
OpenText Cordys offers a powerful, high performing rule engine for expressing business logic in the form of rules. Business logic (e.g. calculation of discount, calculation of customer rating, etc.) is well suited for expressing in business rules, the reason being such logic is prone to frequent changes. Business rules should be easy to modify and deploy by business users. 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 below 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 which is very dynamic and changes frequently should be expressed as rules.
- Decision tables in OpenText Cordys 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.
- Alternatively, you can generate a web service on top of a decision table, thus enabling it from all web services consumers.
- The OpenText Cordys 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 OpenText Cordys.

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

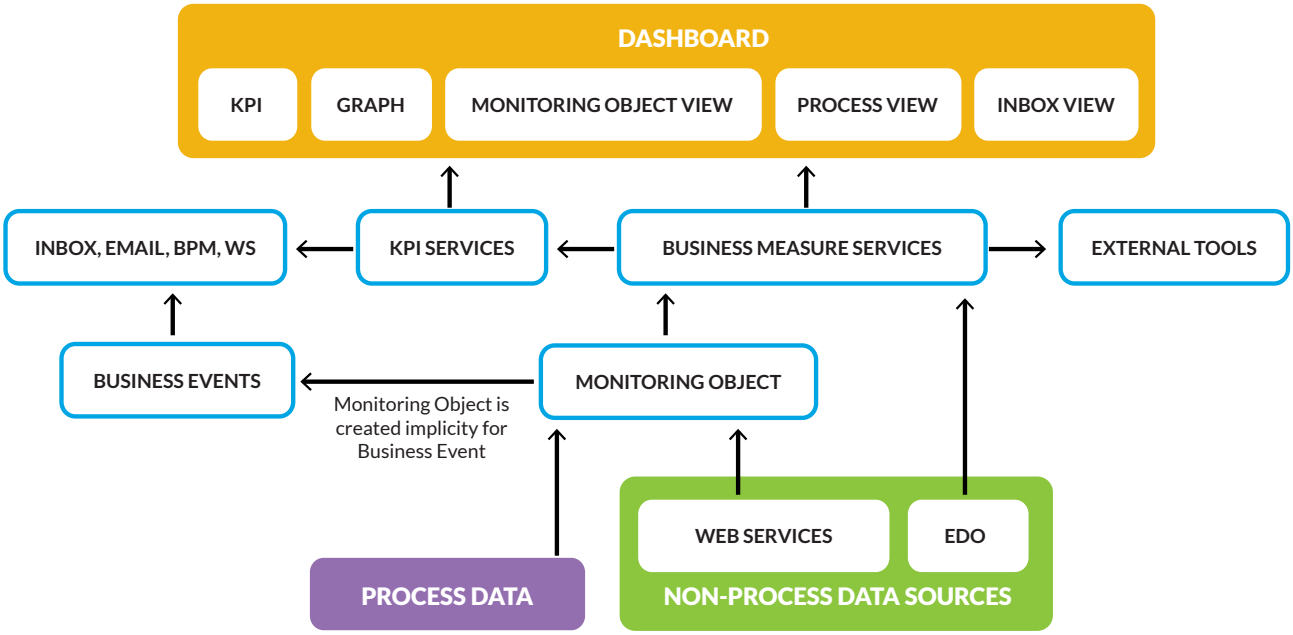
Business Activity Monitoring

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

OpenText Cordys 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.

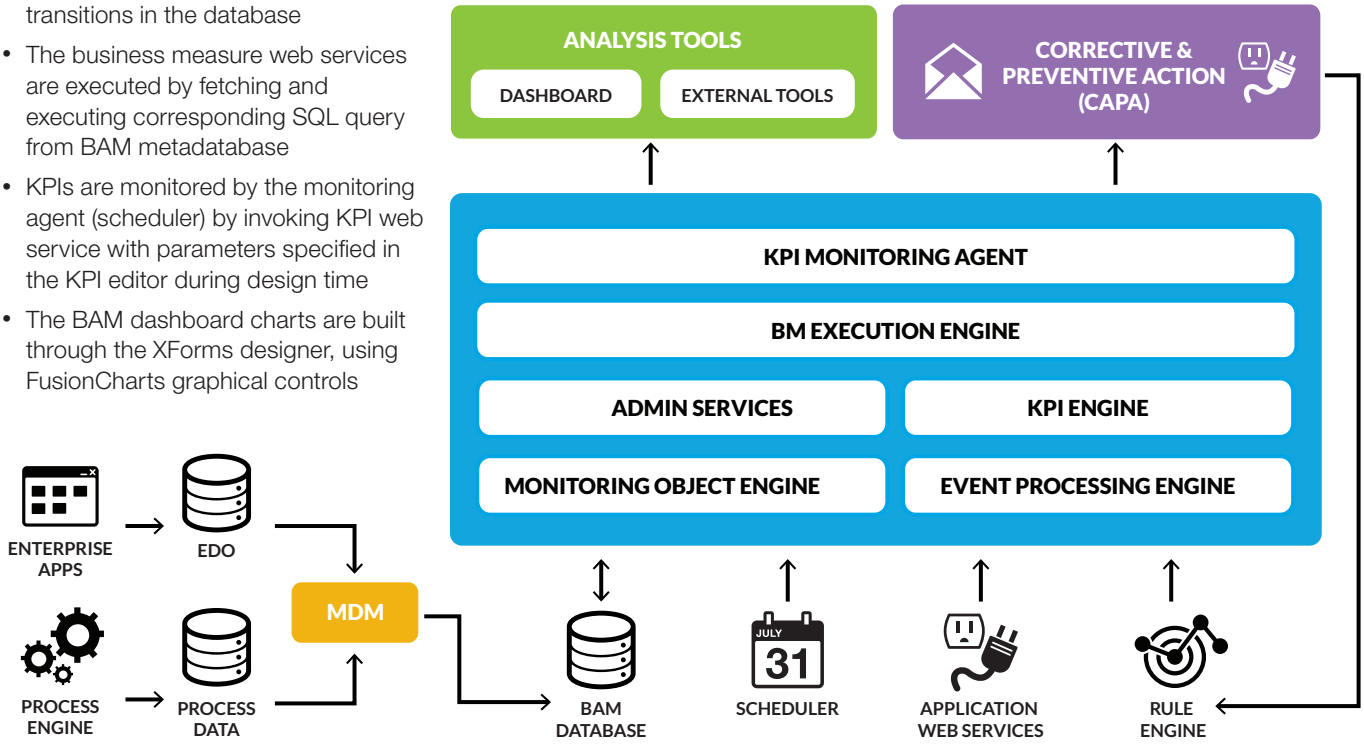
The following picture depicts the BAM architecture.



The picture below depicts different aspects of BAM run-time architecture.

Some explanation:

- 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 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



WS-AppServer

Next to processes and user interfaces, applications often leverage an application server to deliver business entities and accompanying logic. This is done through OpenText Cordys WS-AppServer; it enables the application to define its core application and UI logic in a simple and structured manner. OpenText Cordys WS-AppServer pulls out database metadata from a relation database and generates Java Code. The generated code along with the WS-AppServer framework provides the application with the following features:

- Rapid Application Development, because of a reduced need for coding due to the model driven approach
- Persistence is completely taken care of by the framework and there is no need to write code to perform CRUD[12] operations
- Transaction management is handled by the framework
- Object Life Cycle management is handled by the framework
- Event based programming model makes coding easier
- WS-AppServer has a large number of object life cycle and transaction related events which are raised by framework; based on its specific needs, the application just needs to fill in what it needs to do when a certain event is raised
- Support for changing dynamic business logic easily via integration with 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

- Out of the box support for exposing the application logic as web services
- Support for aggregating database specific classes into user defined classes, so called custom classes, that map closely to the 'view' the end user needs to see
- Support for extending business logic of non-Cordys systems by defining classes over entities of external systems and using the WS-AppServer events to handle the persistence and define the business logic
- Since WS-AppServer is a part of OpenText Cordys, many features that an application needs like authentication, authorization and scalability come for free, though they are not provided by the WS-AppServer component

These features ensure that the application has to focus only on its core logic and the rest is left to WS-AppServer and OpenText Cordys.

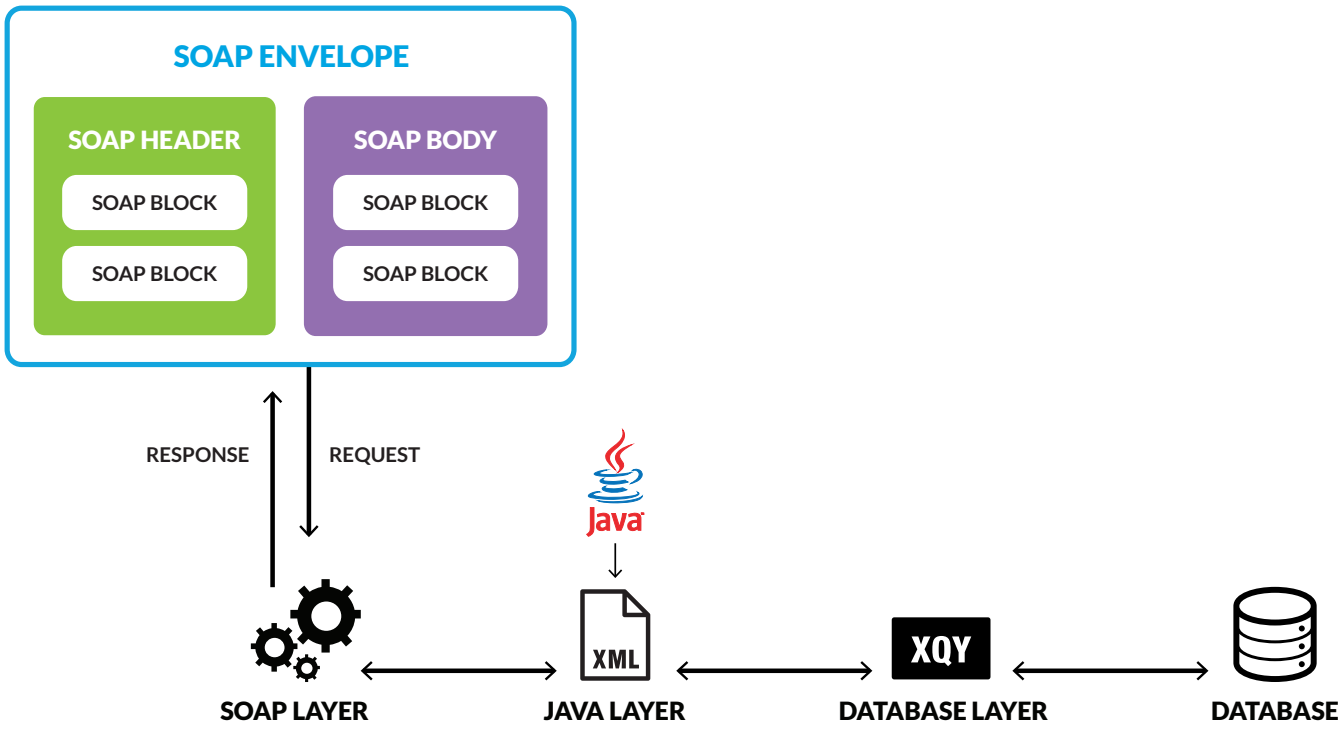
The WS-AppServer framework has three distinct layers:

1 **SOAP messaging layer**
   - The SOAP messaging layer manages the transaction
   - It processes the incoming SOAP request, creates the corresponding Java Objects, calls the setters to fill the object with the information coming from the SOAP request and then calls the desired operation on the object as specified in the SOAP request
   - Once the request processing is done, it creates the SOAP response from the Java Object

2 **Object layer**
   - This layer manages the object data as XML, this helps in talking to other OpenText Cordys components because XML is the common language in OpenText Cordys
   - This layer manages the object life cycle and calls the object life cycle events
   - See Business Objects as XML[13] for more information

3 **Persistence layer**
   - This layer persists the data in relational database, using the XQY layer of OpenText Cordys

The following picture depicts the layers of WS-AppServer:



Scheduling

In a real-time business environment, the ability to trigger processes or applications based on either specific system events or at a specific time is a critical requirement. OpenText Cordys facilitates modeling of schedules that can trigger time-based actions like processes or web service invocations. OpenText Cordys 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
RUNNOW	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	These schedules can be set to recur at a specified time every hour.
DAILY	These schedules can be set to recur at a specified time every day.
WEEKLY	These schedules can be set to recur at a specified time every week.
MONTHLY	These schedules can be set to recur at a specified day every month.
FIRST DAY OF MONTH	These schedules can be set to recur at a specified time on the first day of every month.
LAST DAY OF MONTH	These schedules can be set to recur at a specified time on the last day of every month.
FIRST WEEK DAY OF MONTH	These schedules can be set to recur at a specified time on the first week day of every month.
LAST WEEK DAY OF MONTH	These schedules can be set to recur at a specified time on the last week day of every month.
FORTNIGHTLY	These schedules can be set to 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

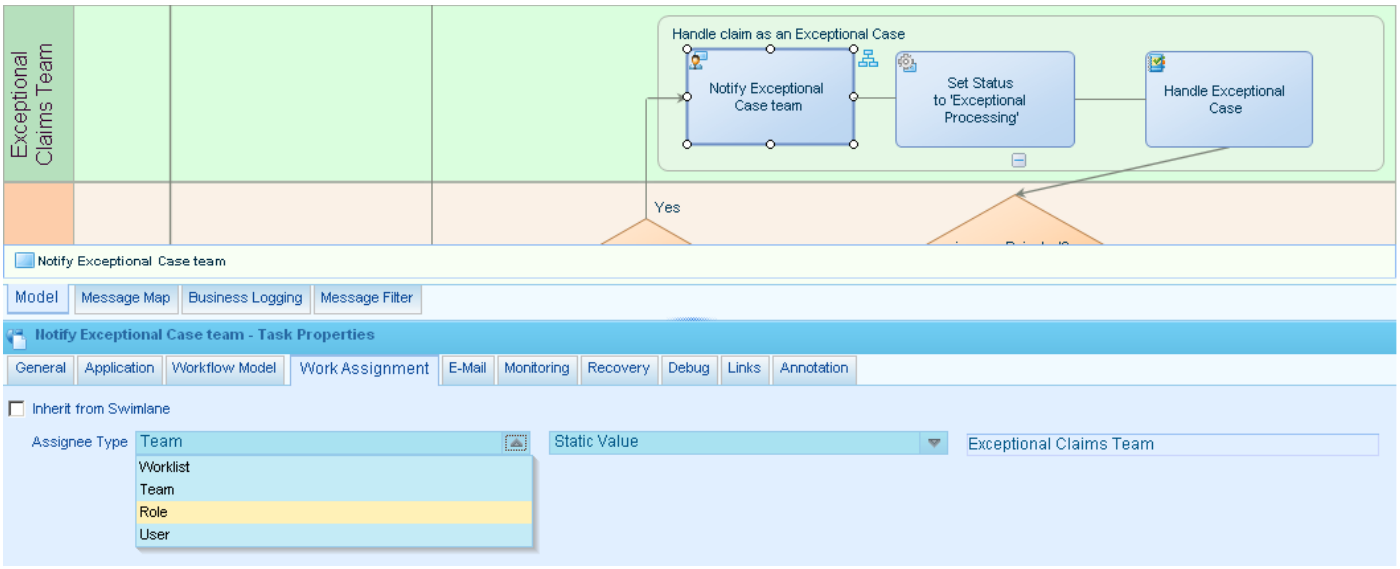
Every OpenText Cordys user is assigned an Inbox, which is used to send and receive tasks and notifications. The inbox is configured for a given user profile and functions like a mailbox. OpenText Cordys users can access tasks that are sent to their work lists, to their associated roles, to the teams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

Tasks

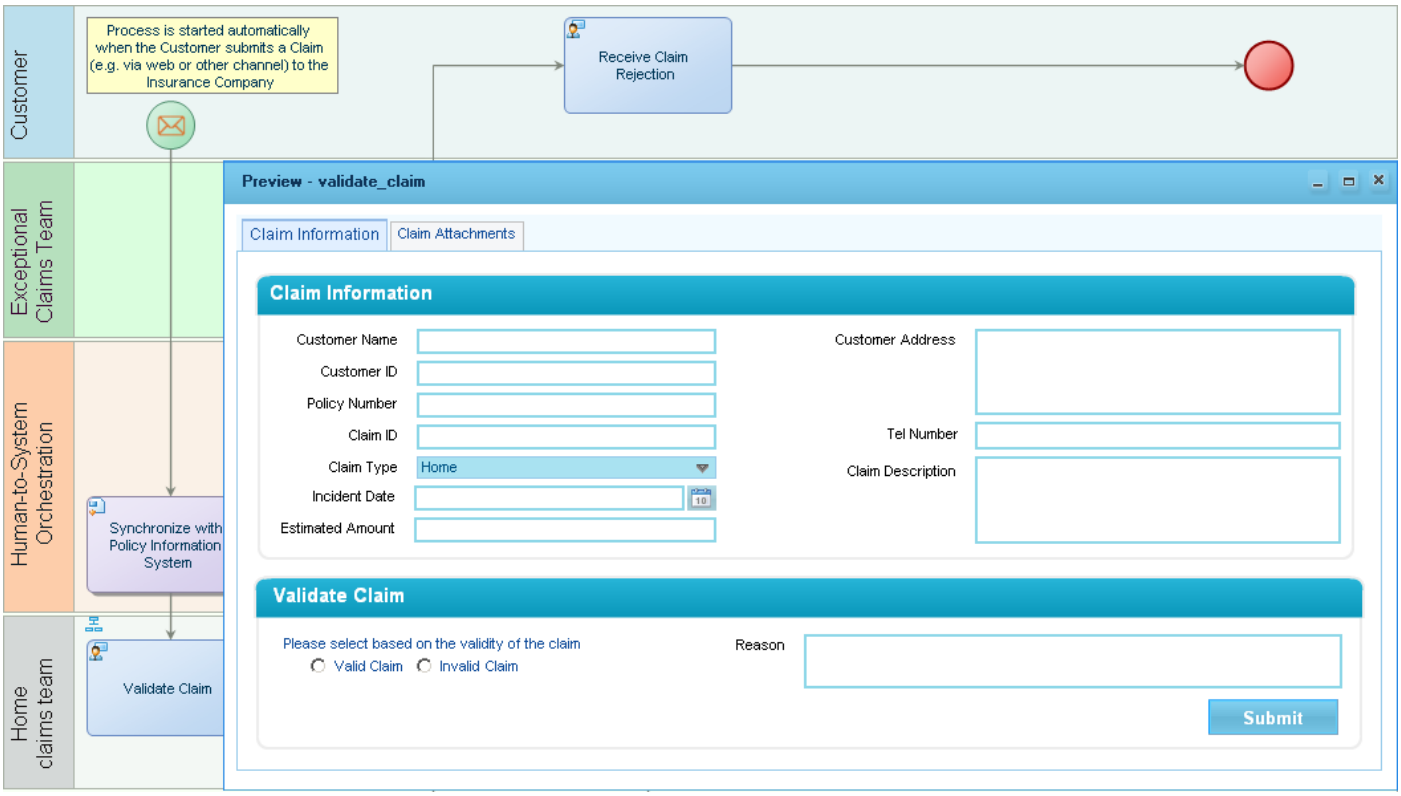
A task is an activity in a process that is to be executed by a human participant of the process. Users can either respond to the task, add a new action to the flow after the task, or do both. For example, when the stock of a particular item in a warehouse reaches a certain level, a replenish task is 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.

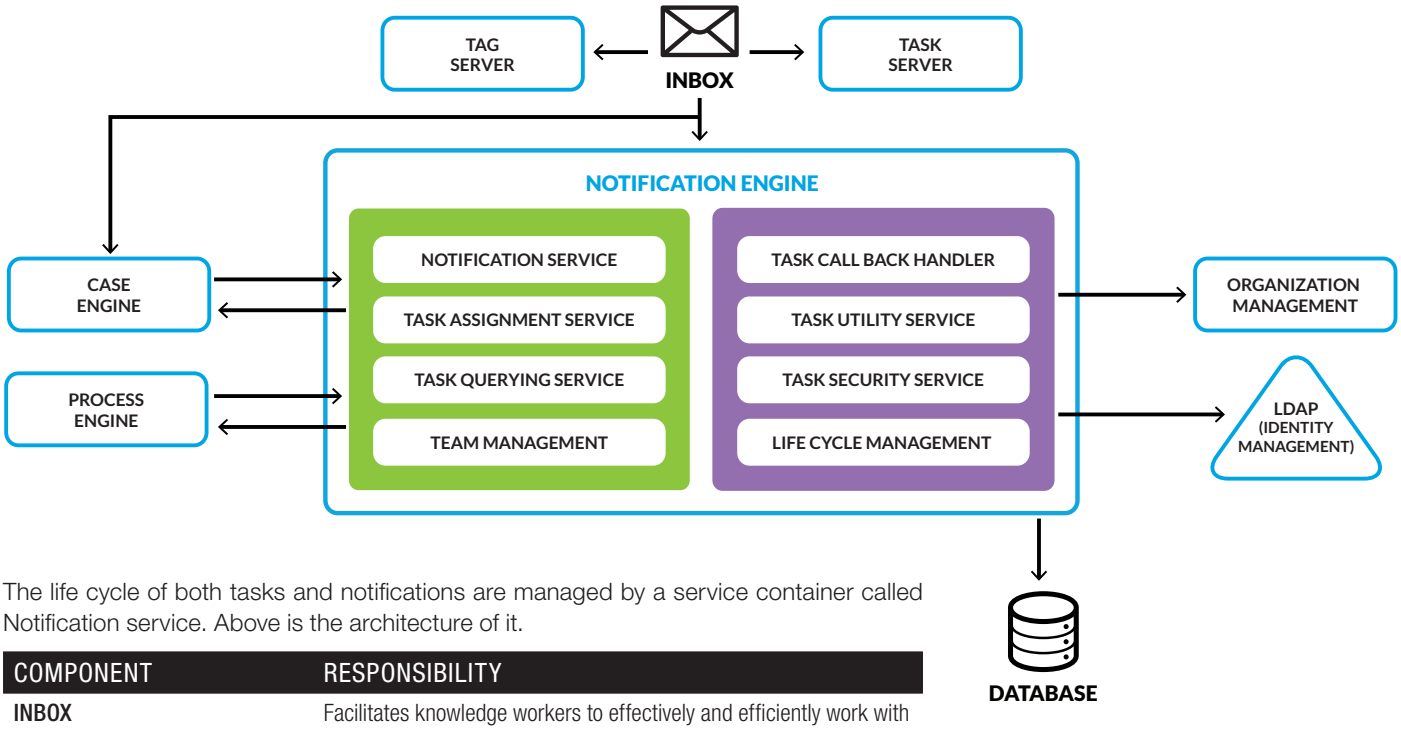
Notifications

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 OpenText Cordys properties tabbed dialogs.



The following is a screenshot of the OpenText Cordys User Interface used for one of the tasks drawn from the earlier 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. Again, an example of OpenText Cordys' model-driven approach: models for flows, for cases, for task user interfaces, etc.





The life cycle of both tasks and notifications are managed by a service container called Notification service. Above is the architecture of it.

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 the 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 OpenText Cordys 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 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 of persistence and retrieval of the task list content

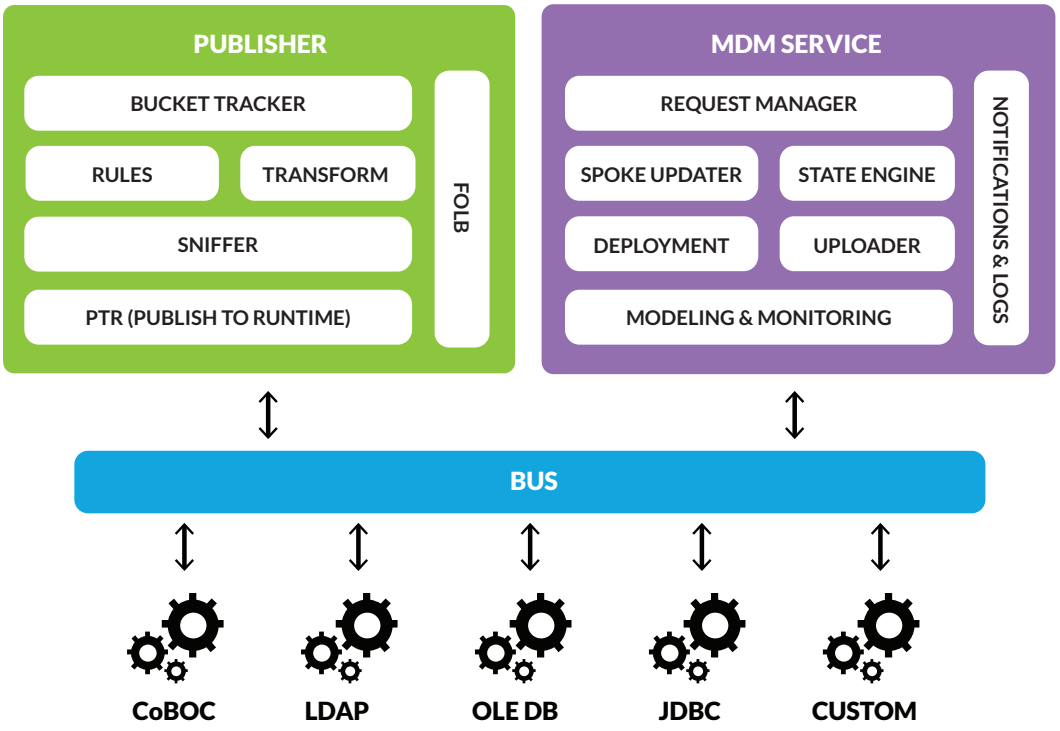
- **Work list:** A work list is a container of the tasks processed in a work flow, case, or any other composite application and is displayed in the OpenText Cordys Inbox. Teams are associated with a work list. Hence, users who are part of the teams assigned to work list can view all the tasks assigned 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 will be sent to the manager of the case worker or to the manager of the work list to which the task is delivered. There is also a possibility 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 OpenText Cordys for work flow 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 interface contract in the form of XML Schema. The JSP or ASP.NET pages need to be enhanced to consume the data from OpenText Cordys and provide the data back to OpenText Cordys, according to the XML schema provided.

Master Data Management

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

Architecture of the MDM run time



Highlights of the OpenText Cordys MDM offering:

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

Thus, there are a wide variety of ways in which organizations can utilize OpenText Cordys MDM to build and deploy trusted data hubs, according to the needs and goals of the enterprise. In a nutshell, here is 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 data a governance program
- Develop a master data model
- Design and setup the infrastructure
- Generate and test the master data
- Modifying producing and consuming systems to integrate with the new MDM solution.
- Implement maintenance process

SERVICE ORIENTED ARCHITECTURE LAYER

OpenText XML technology

From its inception, OpenText Cordys uses XML as the core communication protocol format. The information exchange across OpenText Cordys 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 like XML parser, XPath/XSLT processor.

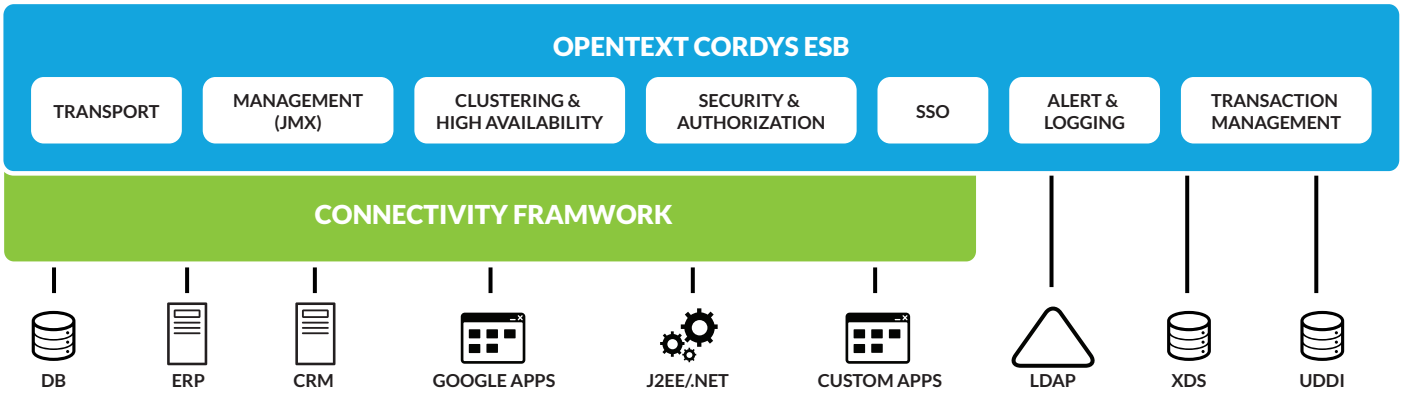
The OpenText Cordys XML parsing API has two flavors. The first flavor, NOM (Native Object Model), is a set of proprietary APIs, which is highly efficient but a bit complex in use. The second variant, coined DOM-over-NOM[15], is a standard DOM API layered over the NOM APIs, bringing the benefits of a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

Enterprise Service Bus

The OpenText strategy has always been to develop an integrated suite of services to enable customers to develop, deploy, and manage business applications on a Services Oriented Architecture (SOA). Therefore, OpenText Cordys has a very 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, their web applications, and their composite applications.

The center of this middleware is the enterprise ready OpenText Cordys Enterprise Service Bus (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 fail over 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 OpenText Cordys ESB does not have a central hub, which eliminates the single point of failure and removes a common performance bottleneck. The OpenText Cordys 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 today serving as the most tangible software infrastructure for SOA.

Let us focus on some of these services which are expected to be delivered by any modern day ESB, and how the OpenText Cordys ESB provides them.

Standards Compliance

OpenText Cordys uses universally accepted standards such as WSDL, XSD, XML and SOAP and all consumers of OpenText Cordys web services can use the "design by contract" model for invoking these web services without having to worry about underlying implementations. OpenText Cordys is also WS-I Basic Profile[17] compliant, so the services hosted on OpenText Cordys 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 OpenText Cordys 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. OpenText Cordys messages follow the paradigm of stateful objects and stateless connections, so all invocations for OpenText Cordys services are completely decoupled and do not hold any information about the clients prior interactions.

Transport transparency and Multiprotocol support

A critical ability of the ESB is to route service requests through a variety of communication protocols and to transform service requests from one protocol to another where necessary. Enterprise applications typically involve talking to different individual applications which support different transport protocols, and the OpenText Cordys ESB provides physical transport protocol bridging to allow communication between services using different transports. OpenText Cordys 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 are made widely available, but the client does not or should not be aware of the physical location at which the service is hosted as it breaks our previous loose coupling strategy. All OpenText Cordys services point to one physical address and the OpenText Cordys ESB locates the service when it is invoked, providing a level of service virtualization and location transparency so that if a machine 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

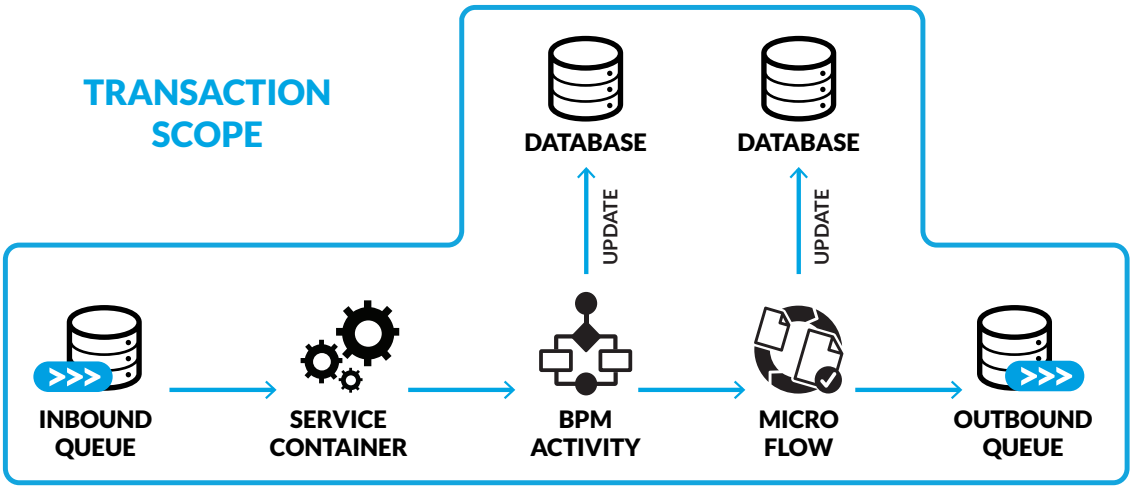
The ESB acts as the middleware to host all services. It is also well suited to perform load balancing of service requests across services. OpenText Cordys services are configured under a logical entity called service group, each service is hosted in a service container, which is the Java process providing the implementation. A service group can be implemented through more than one service container of same configuration, each running on a different physical machine. An administrator can choose the load balancing algorithm such as simple fail over or round robin for load balancing. Service containers can be added or removed on demand depending on the need. OpenText Cordys 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 OpenText Cordys 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. OpenText Cordys can integrate with any enterprise domain authentication systems, integrate with external SAML providers and also provide 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. OpenText Cordys ESB supports distributed transactions and OpenText Cordys 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. OpenText Cordys reliable messaging uses distributed transactions through the XA[19] protocol and supports JMS and MSMQ message queues.

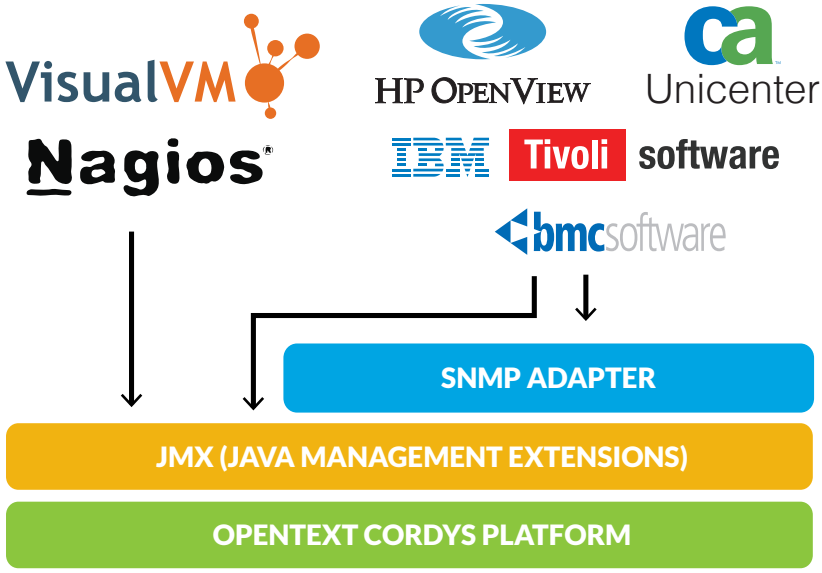


Manageability

It is important to monitor the health of the ESB and the services hosted on it. OpenText Cordys ESB comes bundled with applications which 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. OpenText Cordys has self-healing capabilities, where services raise alerts and can take corrective actions automatically. OpenText Cordys services can be monitored using any JMX and SNMP compatible tools.

Manageability using JMX

OpenText Cordys 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

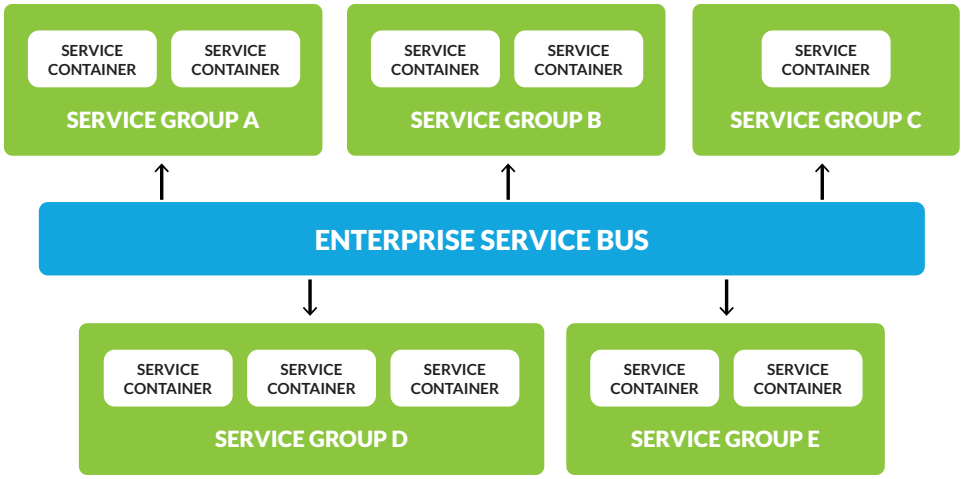
The OpenText Cordys ESB supports diagnostic logging, known as **Composite Application Logging (CAL)** for composite applications. The composite application log messages have contextual information like correlation ID and multilevel diagnostic contexts, which indicate the execution path of the application. 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 OpenText Cordys ESB

Run-time 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 machines 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 OpenText Cordys 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 OpenText Cordys 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 OpenText Cordys ESB supports both synchronous (request-reply) and asynchronous (fire-and-forget, request-callback) messaging paradigms.

OpenText Cordys Monitor

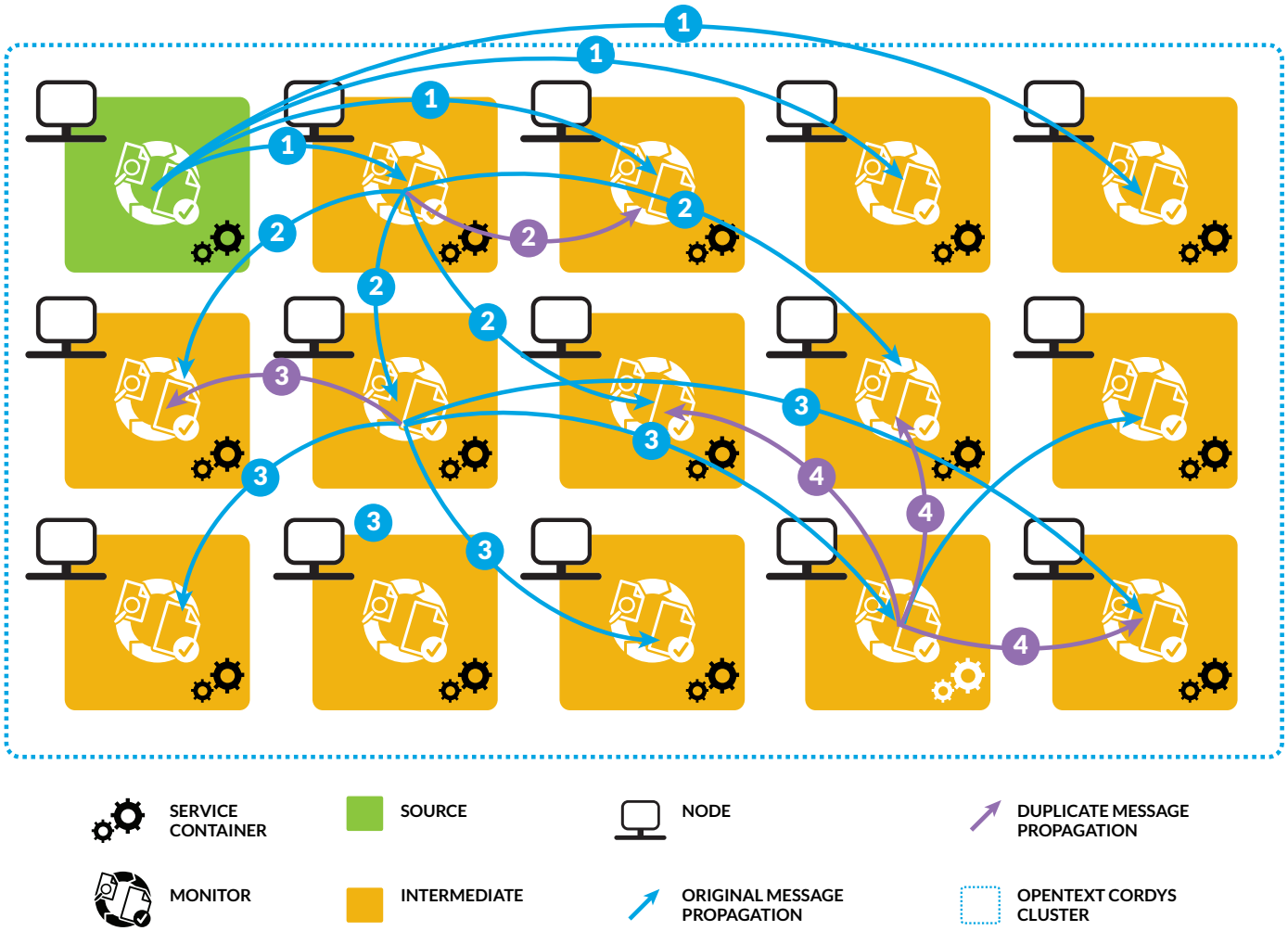
The OpenText Cordys Monitor is a special service container on the ESB. It acts a supervisor for all service containers configured on that particular node. Each node in the ESB cluster has its own OpenText Cordys monitor configured and running.

It runs as an OS-level system process and takes care of:

- Managing (start, stop, restart, reset) service containers
- 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

Gossip OpenText Cordys clustering technology

OpenText Cordys is a distributed system[20] that uses multiple processes (Java virtual machines) spread across machines. The OpenText Cordys 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[21] is chosen.



Some of the use cases for this framework are

1 **Distributed cache invalidation:** Most of the components (e.g. LDAP, Rule, CoBOC) in OpenText Cordys use caching for performance reasons. All these components use Gossip for cache invalidation. More information is available in this article[22].

2 **State registry:** State registry is an in-memory database of the state of the OpenText Cordys cluster, built on Gossip. The OpenText Cordys state registry provides the needed infrastructure for high-availability. More information is available in this article[23].

The message propagation[24] in Gossip is depicted in the picture given above, where nodes selectively broadcast to a set of peers[25] and where duplicates are ignored.

Application packaging and deployment

OpenText Cordys provides both web based and command line based facilities for managing the applications built using OpenText Cordys.

Any composite application developed using OpenText Cordys is packaged through a format called **Cordys Application Packaging (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.
- It ensures the uniqueness of artifact identification.
- 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 that are 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's possible to indicate to whether the package can be deployed on in the shared space, on organization level, or both.

OpenText Cordys 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 rollback any application across the cluster.

Optionally, CAPs can be digitally signed to build the chain of trust. The trusted entities are registered through 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 OpenText Cordys nearly always integrate with existing Enterprise Information Systems (EIS). These applications and systems include ERP systems, Content Management Systems, CRM systems, databases, etc. OpenText Cordys provides a generic connectivity framework to connect to various systems and applications. Based on this framework, OpenText Cordys and the community around OpenText Cordys have developed a set of ready-made connectors for some of the most commonly used IT systems.

OpenText Cordys 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 which 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 OpenText Cordys, which include:

1. Logging
2. JMX support
3. Access control list (ACL)
4. Problem Registry
5. Localization support
6. Transaction support
7. Load balancing and fail over support

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

The ready-made connectors provided by OpenText Cordys 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 OpenText Cordys are:

- **Database connector:** This is used to interact with database (RDBMS) systems using popular database technologies like JDBC and OLEDB. This layer acts as an XML to relational mapping layer and vice versa.
- **Email connector:** The email connector enables sending and receiving mails through IMAP or SMTP and POP3.
- **SAP connector:** SAP connector provides connectivity to SAP® back-ends through RFC and BAPI.
- **Script connector:** Script connector allows web services to be implemented in JavaScript.

A full catalog of the available connectors is available here[26].

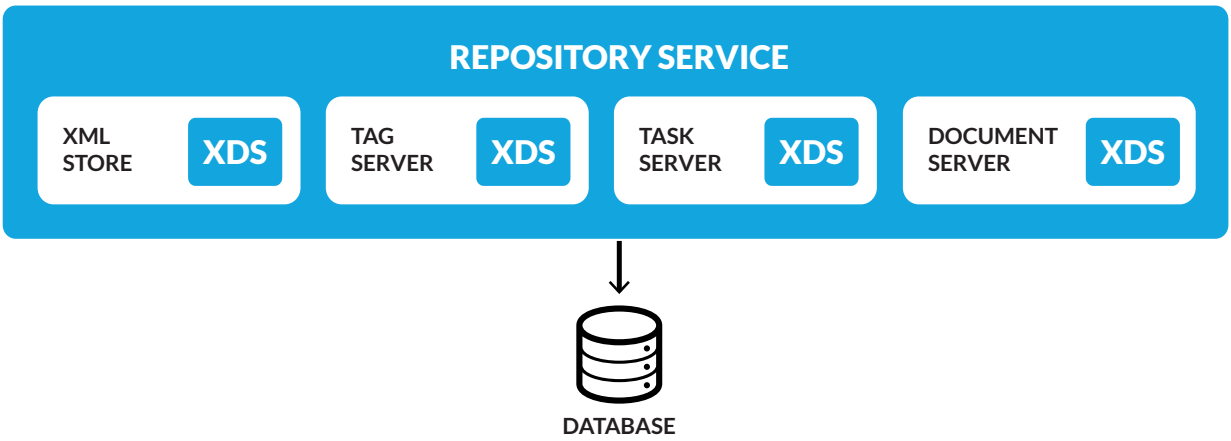
Repository

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

The OpenText Cordys repository is populated through the following ways:

- Application content developed in CWS–Stored in XDS via CWS.
- On publish of application content from CWS–Deployed models are stored in XDS (example business calendar, organization models, tasks, etc.)
- Run-time data–The platform and applications built on top of it can store metadata and definitions in the repository (e.g. 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.

Here is the architecture layer diagram of repository service:



Tagging

Many social networking sites nowadays allow their users to tag[27] the 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 the end user to group/categorize their own tasks and artifacts. Tagging abstracts the storage approach (e.g. hierarchical) and enables users to view the system in a more linear and self-managed approach.

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

Auditing

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

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

OpenText Cordys provides auditing capabilities to audit critical actions:

- Starting and stopping of service containers
- Deployment of application packages, BPMs, business calendars, schedules, etc.
- 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:

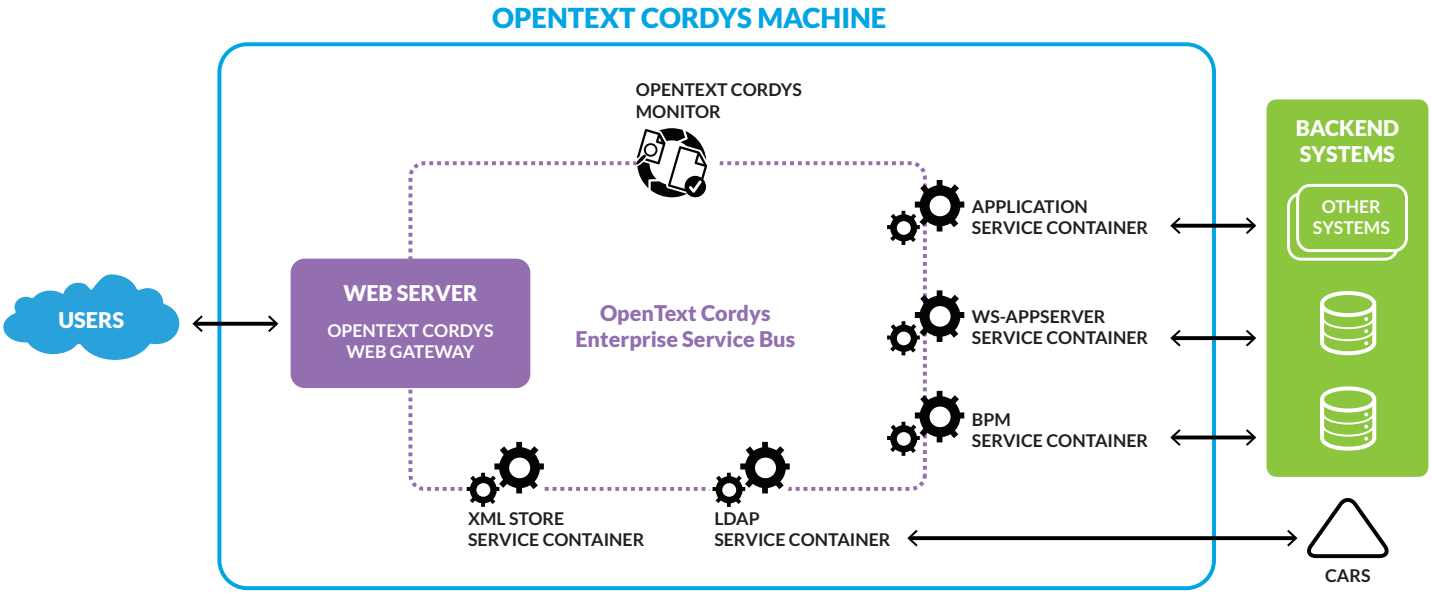
- Artifact type or ID
- User who performed the action
- Date range
- Organization context
- Action performed on the audited artifact, e.g. deploy or undeploy

Gateway

The OpenText Cordys web gateway[28] is the HTTP interface of OpenText Cordys SOA Grid. It is meant to be a light-weight component, hence its functionality is very minimal. In brief, it is the HTTP end-point of all web services hosted by OpenText Cordys. Besides that, it also supports some security features[29].

The primary functions of the web gateway are:

- Authentication (optional)
  - Integrated authentication, e.g. Microsoft Active Directory®
  - OpenText Cordys 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 above picture gives a short representation of web gateway role in OpenText Cordys.

User Interface (UI) tasks

UI applications in OpenText Cordys are referred 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 any web service invocations, or just 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 '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 backend 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 OpenText Cordys system in a secure and trusted way.

OpenText Cordys has multiple options for user authentication:

- Web server authentication
- OpenText Cordys 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 will handle user authentication and pass-on the user identity to OpenText Cordys.

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 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 OpenText Cordys.

OpenText Cordys authentication

With OpenText Cordys authentication, the user authentication is done by OpenText Cordys. The web server will not do any authentication of the user; this is left to OpenText Cordys Single Sign-On service (SSO).

User authentication is done 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 SAL artifact which is 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 will look up the corresponding SAML Assertion for the given SAML artifact and use this internally in OpenText Cordys.

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

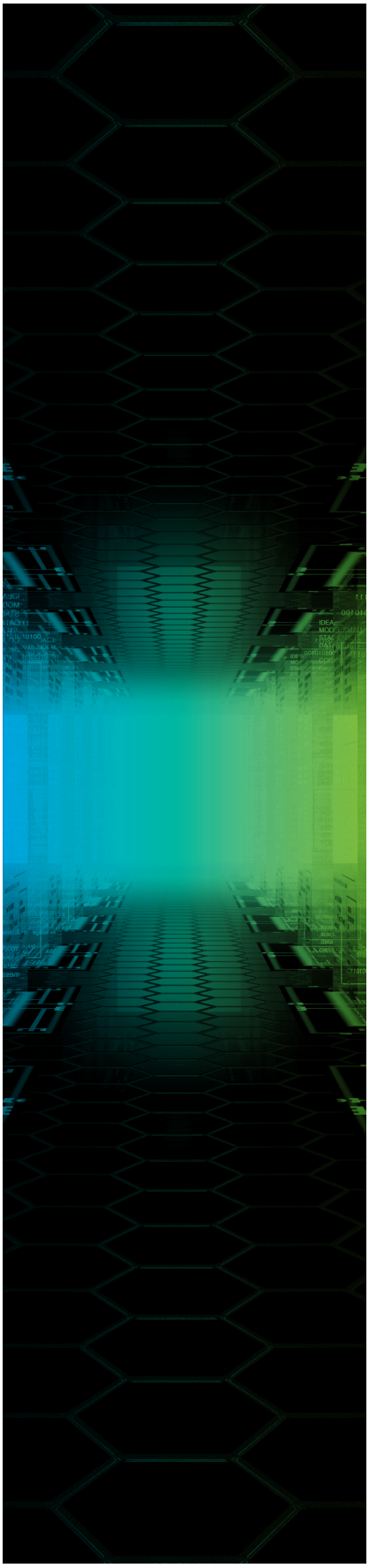
OpenText Cordys Authentication is extensible in a way that custom login forms can be used in the front-end. The OpenText Cordys SSO service also has an extension mechanism, so that one can also authenticate against other sources (e.g. Microsoft Active Directory or a database).

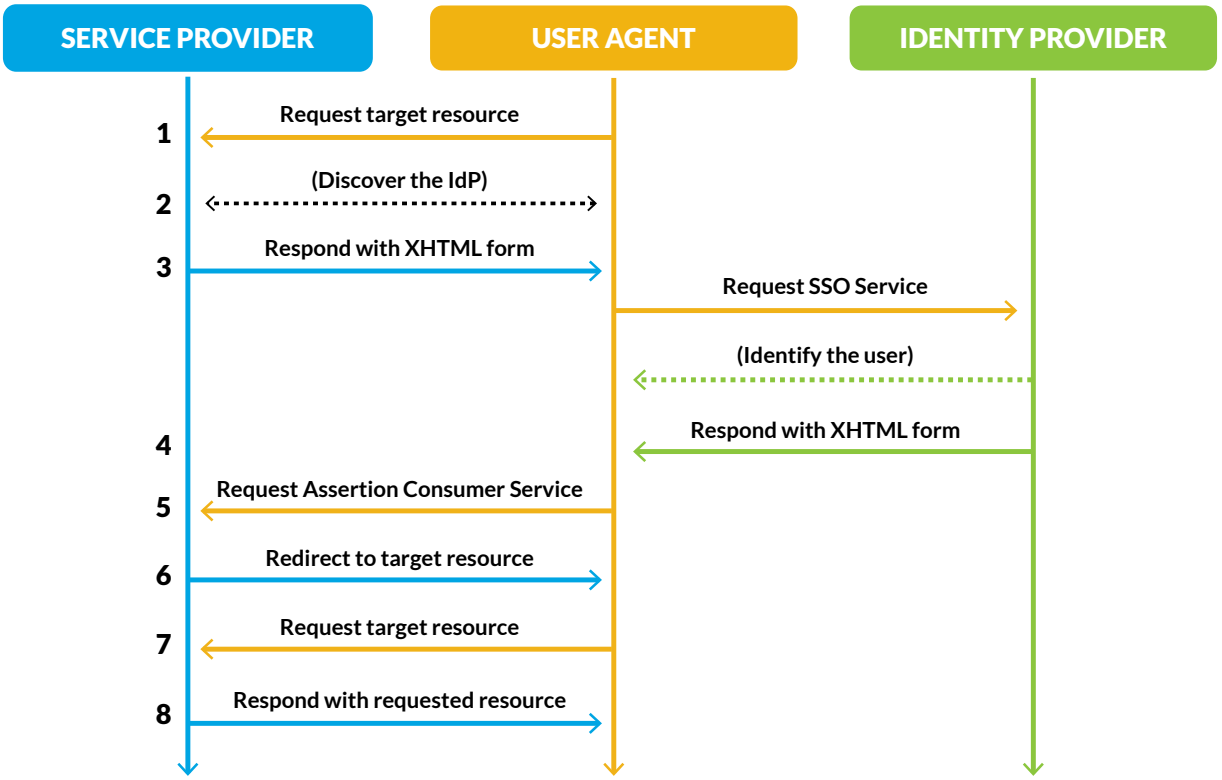
As complete user authentication is done by OpenText Cordys, authentication in the web server needs to be disabled.

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)**. OpenText Cordys implements the SAML 2 Web browser SSO profile[30], which means the external IDP must support the SAML 2.0 standard.

When OpenText Cordys needs to authenticate the user, the browser will be redirected in a separate window to the configured external IDP which handles the authentication. After the user is authenticated, the external IDP Posts a 'SAMLResponse' back to the OpenText Cordys SSO service. The SAMLResponse contains a signed SAML 2.0 Assertion which states the identity of the user. This external SAMLResponse will be validated and matched with the available users in OpenText Cordys. 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.

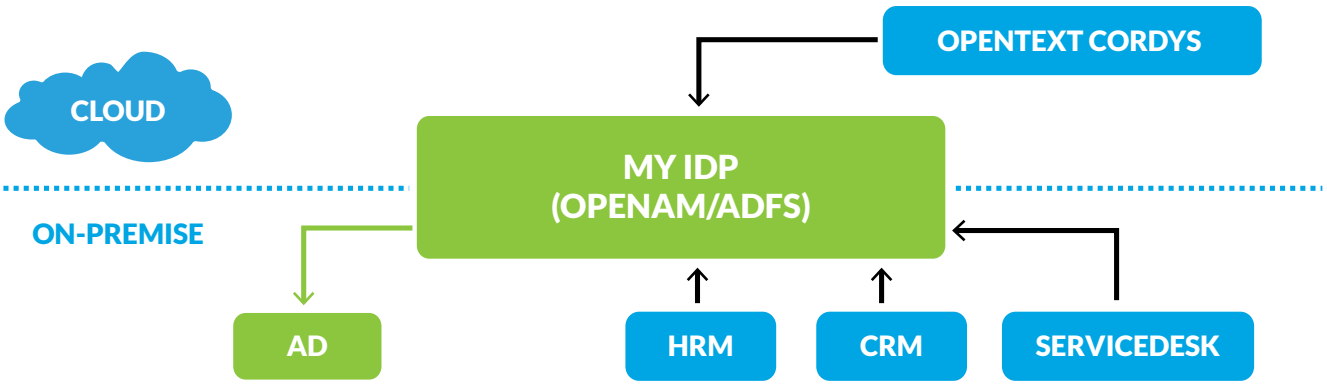




The flow between the browser, Service Provider and Identity Provider is given above. In this flow, OpenText Cordys SSO is the Service provider and the User Agent is the user's browser.

The external authentication is based on a two-way trust configuration between OpenText Cords and the external IDP. The Security Admin Console in OpenText Cordys is used to configure one part of this trust. See the Security Management space[31] for documentation and examples.

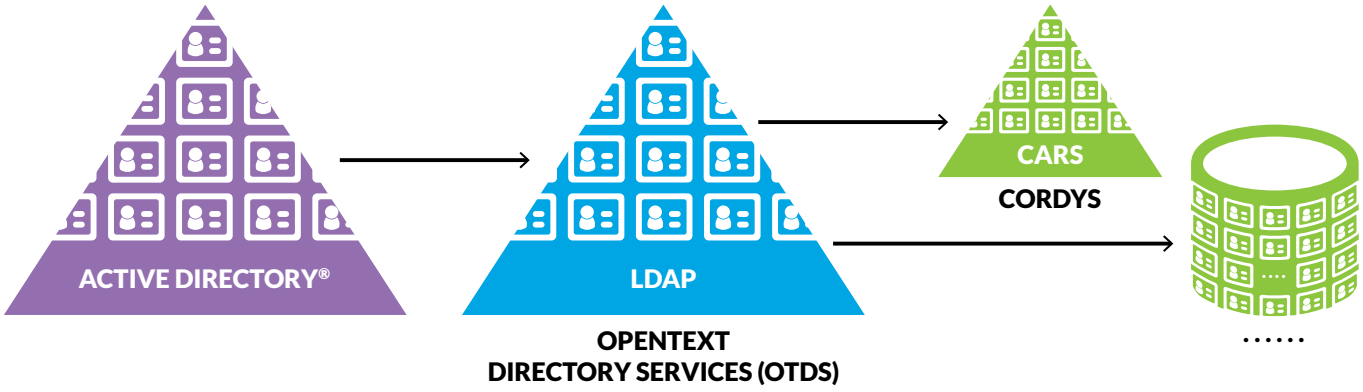
The external IDP might be cloud-based or on-premise. Direct communication between the external IDP and OpenText Cordys is not required. The information exchange is done through browser redirects and automatic form submits only.



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, user and group management can now be done centrally, in OTDS or its identity provider (for instance Microsoft Active Directory), across all Open-Text products.



Advanced authentication features

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

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

Authorization

OpenText Cordys 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

- Package role: Role defined during application development, defining 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.
- Organizational role: Role defined during run time, normally aggregating package roles. This is created on an organization level. 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 OpenText Cordys system. It is implemented as part of the ISAPI extension and the Apache module running in the web server.

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

Conclusion

OpenText Cordys offers a strong foundation to build business applications. This article describes the design goals of the platform and offers an insight into the design-time architecture as well as the run-time architecture of the platform.

We could only cover the highlights of the architecture here. To learn more, join the OpenText Cordys 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.

Applicable standards

OpenText Cordys plays in the domains of business process management, cloud computing, and web services. These domains are being standardized by standards bodies like W3C, Oasis, and OMG, each producing numerous standards. Below you will find an overview of the most important standards supported by OpenText Cordys.

Note: A * in a version column indicates partial support.

WEB SERVICES, XML AND INTERNET STANDARDS		
CSS	Style sheet language to describe the 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* 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. BOP doesn't support Kerberos and REL.	1.0*
BPMN	Standardized graphical notation for drawing business processes in a workflow	1.1*
XPDL	Standardized XML based language for import/export of BPM	2.0
SECURITY STANDARDS		
SSL / TLS	Commonly-used protocol for managing the security of a message transmission on a network	3.0/1.0
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 to manage and monitor applications, system objects, devices etc.	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://www.w3.org/MarkUp/Forms/
9	http://en.wikipedia.org/wiki/Right-to-left
10	http://www.bpmn.org/
11	http://www.w3.org/TR/scxml/
12	http://en.wikipedia.org/wiki/Create,_read,_update_and_delete
13	https://wiki.cordys.com/display/PI/2009/06/23/Business+Objects+as+XML
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	http://en.wikipedia.org/wiki/Gossip_protocol
22	https://wiki.cordys.com/display/PI/2009/11/10/Distributed+cache+invalidation
23	https://wiki.cordys.com/display/PI/2009/12/01/State+registry+in+Cordys
24	https://wiki.cordys.com/display/PI/2010/02/11/SSU+Framework+highlights+-+1
25	https://wiki.cordys.com/display/PI/2010/03/19/SSU+-+Framework+highlights+-+2
26	https://wiki.cordys.com/display/dsc/Connector+Directory
27	http://en.wikipedia.org/wiki/Tag_(metadata)
28	https://wiki.cordys.com/display/PI/2009/08/20/The+how+and+why+of+Cordys+HTTP+Gateway
29	https://wiki.cordys.com/display/PI/2009/09/10/Cordys+Web+Gateway+Security+-+Part+1
30	http://en.wikipedia.org/wiki/SAML_2.0#Web_Browser_SSO_Profile
31	https://wiki.cordys.com/display/SecMan

www.opentext.com

NORTH AMERICA +800 499 6544 • UNITED STATES +1 847 267 9330 • GERMANY +49 89 4629-0
UNITED KINGDOM +44 (0) 1189 848 000 • AUSTRALIA +61 2 9026 3400

Copyright ©2013-2014 Open Text Corporation OpenText is a trademark or registered trademark of Open Text SA and/or Open Text ULC. The list of trademarks is not exhaustive of other trademarks, registered trademarks, product names, company names, brands and service names mentioned herein are property of Open Text SA or other respective owners. All rights reserved. For more information, visit:http://www.opentext.com/2/global/site-copyright.html (01/2014)1680EN