HEXAGON | Xalt

DATA SHEET

THE DUAL MODEL ARCHITECTURE

1. INTRODUCTION

Xalt is a revolutionary technology platform for key business operations across all industries. It enables the convergence of the physical world with the digital world, operational technologies with information technologies, and the seamless integration of emerging innovation with legacy systems and data.

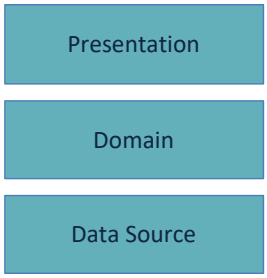
A scalable "platform of platforms," Xalt encompasses five critical functions that accelerate the digital transformation of business models, operational and job-specific workflows, and entire industry ecosystems. Its core functional capabilities are cloud enablement, intelligent edge connectivity, enterprise integration, built-in mobility, and artificial intelligence (AI) across all functions.

The dual model architecture for cloud enablement and built-in mobility is a new way of organising enterprise software, designed to run next-generation business software using cloud technologies. In addition to leveraging cloud technologies, the architecture was designed to support a wide variety of user interfaces including, but not limited to, mobile applications running natively on devices like iPhones, iPads, and Androids. This paper explains the breakthrough technology behind this approach to enterprise software architecture.

2. SINGLE MODEL ARCHITECTURE

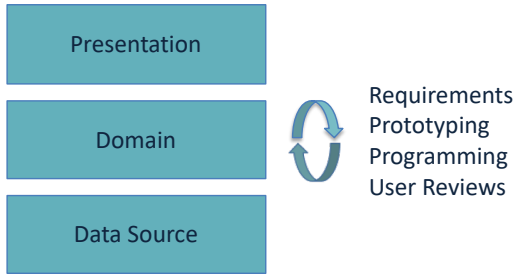
2.1 CLASSIC 3-TIER ARCHITECTURE

The classic 3-tier architecture is a client-server architecture that separates presentation, domain logic, and data sources into logically separate processes. The diagram below illustrates a standard view of a 3-tier architecture, where in a strict implementation, a layer only depends on the layer immediately below. This separation of concern minimises dependencies, promotes reuse, and allows a layer to be upgraded or replaced with minimal impact on the other layers.



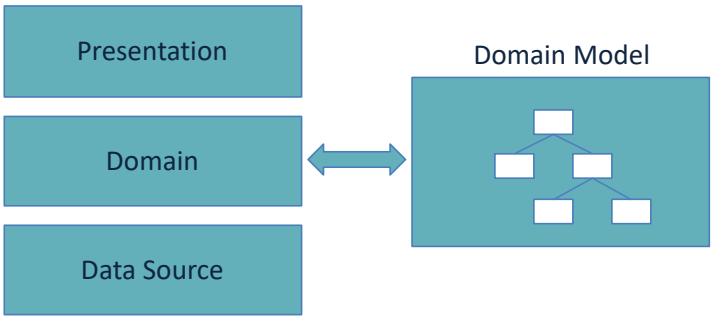
2.2 THE DOMAIN LAYER

Most enterprise software systems deploy some variation of the 3-tier architecture. Of the three layers, the domain layer is of most interest and is considered the heart of software development. The domain layer is developed through an iterative process, which begins with business concepts surrounding the problem domain and deploys as software components adapted to the target platform.



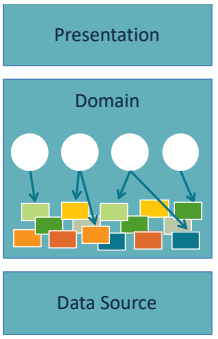
2.3 THE DOMAIN MODEL

Corresponding to the domain layer is a collection of diagrams known as the domain model. As the name suggests, the domain model is a representation of the software contained in the domain layer. The domain model is a tool for reasoning about software design and is used to document and drive the development of the software layer. The domain model and domain layer evolve through iterative development in an incremental and lockstep fashion. Consequently, effects on the domain model affect the domain layer, and vice versa.



2.4 THE CONTROLLER LAYER

A common approach to handling domain requests is to insert a controller layer into the domain layer, which serves as an API. The presentation layer acts on the domain objects purely through the controller layer. In a well-factored system, the controller layer forwards real requests to the underlying domain objects. In this case, the controllers are easier to use by the presentation layer because the controllers are organised around use cases. Application controllers also serve as a convenient point for adding transactional wrappers and security checks.



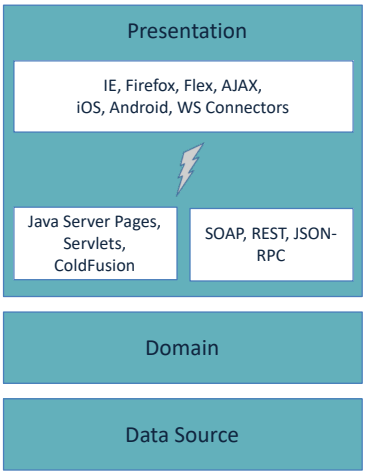
The presentation layer acts on the domain objects purely through application controllers.

Application controllers are organized around use cases and forward requests to domain objects.

Each domain object contains business logic relevant to only it.

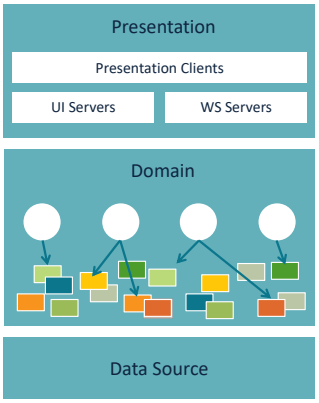
2.5 USER INTERFACES AND WEB SERVICES

User interfaces and web services are typically supported with subsystems located in the presentation layer. These subsystems act as presentation servers, and their job is to map client requests to the domain layer. Web-based user interfaces are commonly implemented using an MVC architecture where the model is mapped to the domain layer. Some popular frameworks for implementing web-based MVC are Java Server Pages, Servlets, and ColdFusion. Likewise, web service programs are implemented with frameworks depending on the web service protocols desired. The common web service protocols used today are SOAP, REST, and JSON-RPC.



2.6 TYPICAL SINGLE MODEL ARCHITECTURE

The following diagram summarises the single model architecture into a logical architecture view:



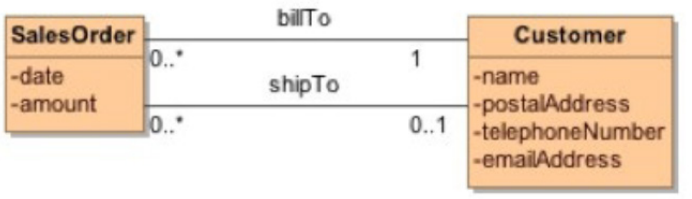
1. User Interface Servers and Web Service Servers map client requests onto Application Controllers.
2. Application Controllers present use case scenarios as APIs and map scenarios into requests on domain objects.
3. Domain Objects contain the fine grained business data and business calculations.

3. DUAL MODEL ARCHITECTURE

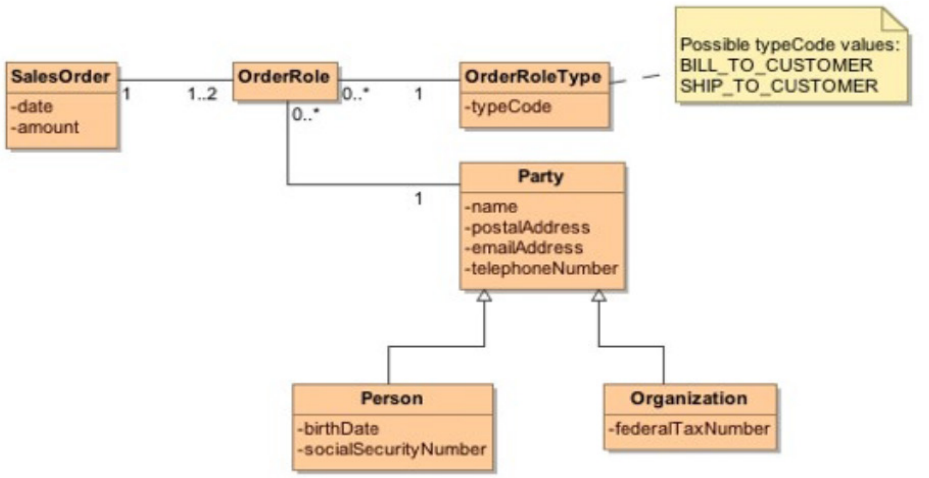
3.1 SPLITTING THE DOMAIN MODEL

The fundamental goal of the dual model architecture is to gain advantages from a higher separation of concern in the domain layer. To achieve this goal, the domain model is split into two models: the conceptual model and the implementation model. To better understand this approach, an example is presented of a conceptual model followed by its implementation model.

The following conceptual model specifies that a Sales Order must have one Bill-To Customer and optionally one Ship-To Customer. It's reasonable to assume that end users would think of Customers and Sales Orders in this way.



The following implementation model supports the same data as the conceptual model, but contains a different entity structure to avoid update anomalies and to provide a flexible structure for enhancements. This implementation example is taken from The Data Model Resource Book, a collection of widely used data model patterns.



The implementation model in the diagram above includes a few details that were abstracted over by the conceptual model:

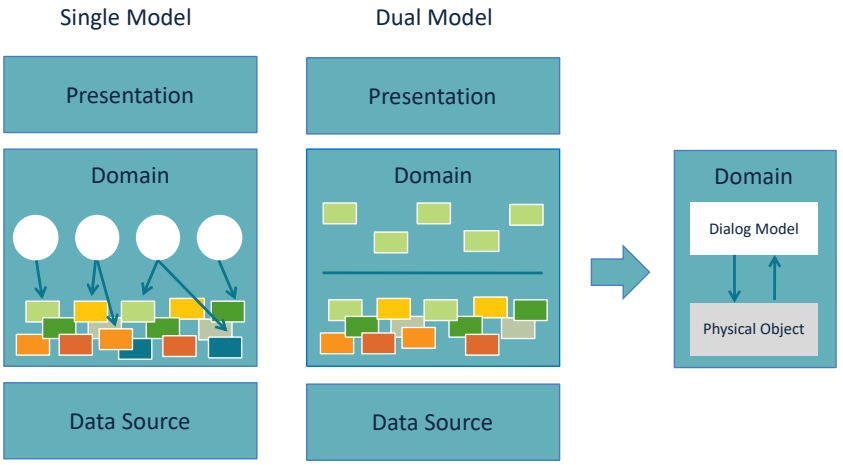
1. The Customer class is generalised into a Party class. This is a step towards a more normalised data model and allows for greater reuse in an enterprise database. For example, this approach allows a Party to participate as a Customer and an Employee without data duplication.
2. The Bill-To and Ship-To relationships are soft-coded as "type codes" in the Order Role Type entity. This makes it easy to add new role types, like SALES_PERSON, without changing the database schema.
3. The implementation model supports the subclasses of Person and Organisation, which are orthogonal to the Customer class found in the conceptual model.

In the dual model architecture, the presentation layer can only access entities found in the conceptual model. Entities found in the implementation model are completely hidden, as illustrated in the diagram on the next page.

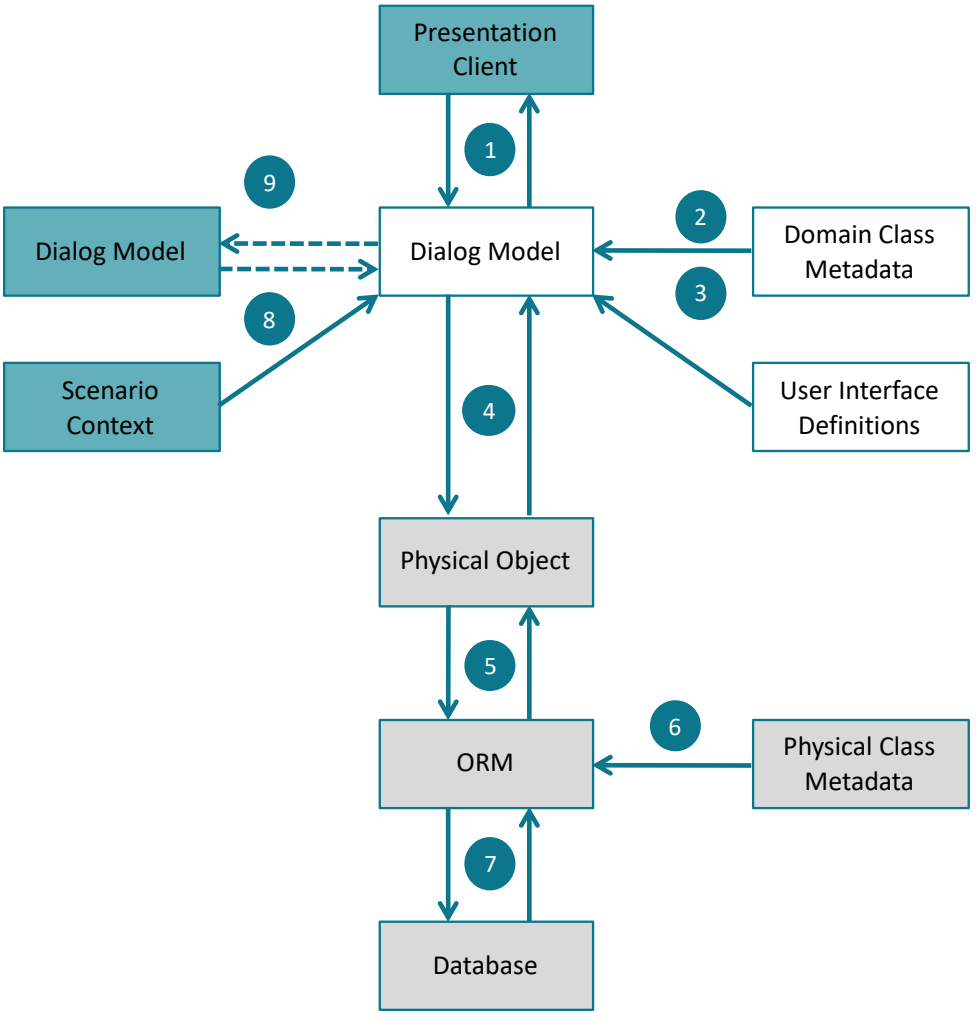
3.2 DIALOG MODELS

There are countless ways to create conceptual and implementation layers within the domain. The dual model architecture uses a specific US Patent number 8423561 approach called a Dialog Model. The dialog model, from a presentation view, looks like a conceptual entity. The dialog model handles all requests by the presentation layer and is a direct replacement for the controller used in a single model approach. The dialog model accomplishes the essence of the dual model architecture.

The diagram below shows the placement of a dialog model in a side-by-side comparison of logical architectures:



To the outside world, a dialog model simply looks like a conceptual entity. However, the capabilities and responsibilities are quite sophisticated. The diagram below details the essential inputs and outputs of a dialog model.



1. All requests from the presentation layer are object-oriented and delivered in terms of the conceptual domain model. Although the requests are object-oriented, they are delivered and returned in relatively large-grained messages to achieve good performance over wide area networks.

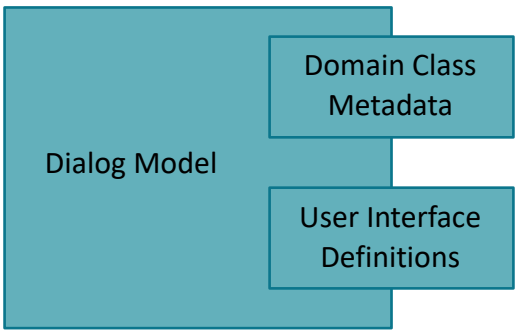
2. Dialog models are protocol handlers for domain classes. Each domain class is fully described with metadata. All requests (read, write, query, navigate, and perform) received by a dialog model are validated and processed against the domain class's metadata, which contains information like schema structure and security permission.

3. Dialog models can have an associated set of user definitions. Dialog models being used in forms-based interfaces have the notion of a selected view and can deliver user definitions on demand. This ability forms the basis of a dynamic and metadata driven user interface.

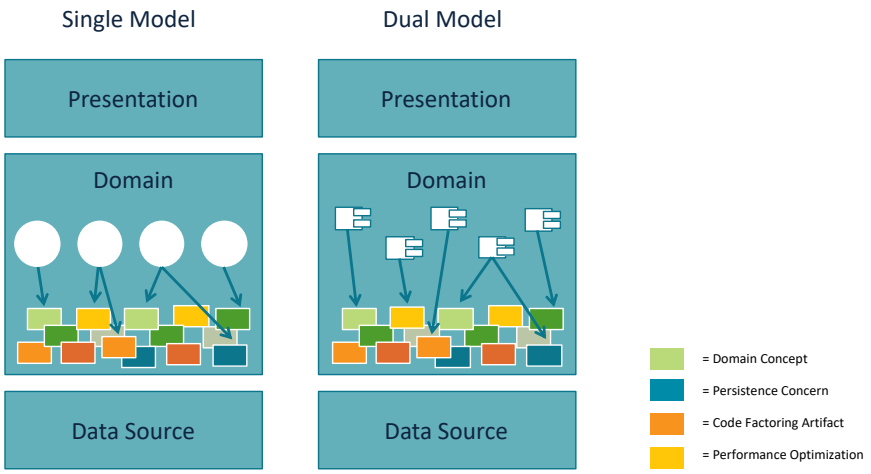
4. Dialog models map logical requests into physical requests. Each dialog request is checked against security and bracketed by a transaction. A single large-grained dialog request is mapped to many small-grained method calls to physical objects. This is an important performance feature when crossing network boundaries. In this example, the physical objects are mapped to a database using an object-relational mapping (ORM) framework. It's important to note that in a dual model approach you can have two sets of independent metadata — domain class metadata and physical metadata — to achieve a better separation of concern.

5. Dialog models are not just strict proxies for their domain class. They can take into consideration details of the current scenario. Scenario attributes like user role and GPS location can be used to condition dialog behaviour.

6. Client programs can directly navigate the domain model using navigate messages. But more importantly, every request to a dialog model has the potential to redirect to a different dialog model. This ability is a core feature, and it is the reason that native user interfaces can be developed to run on remote devices but be controlled by the server. Fundamentally, standard object-oriented messaging that returns either a value or generates an exception has been extended to return a value, generate an exception, or generate a redirection. The previous dialog diagram can be reduced to a single icon representing its public view to the presentation layer. There are two sets of information requests handled by a dialog model: Domain Class and User Interface.

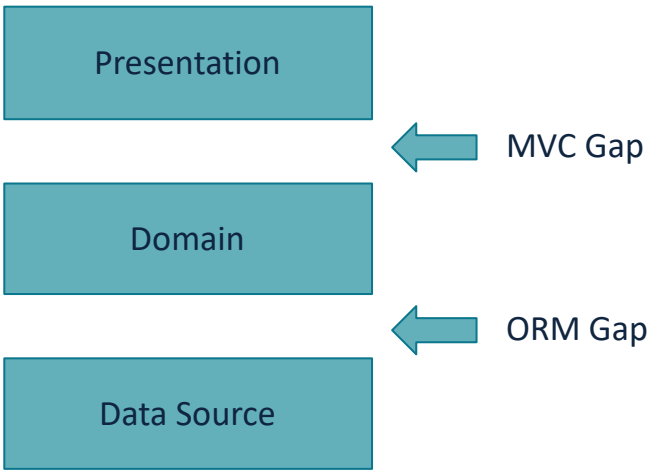


The following diagram presents a side-by-side comparison of the logical architectures, while considering the enhanced view of the conceptual domain model and the new dialog model icon.



3.3 ADVANTAGES OF A DUAL MODEL ARCHITECTURE

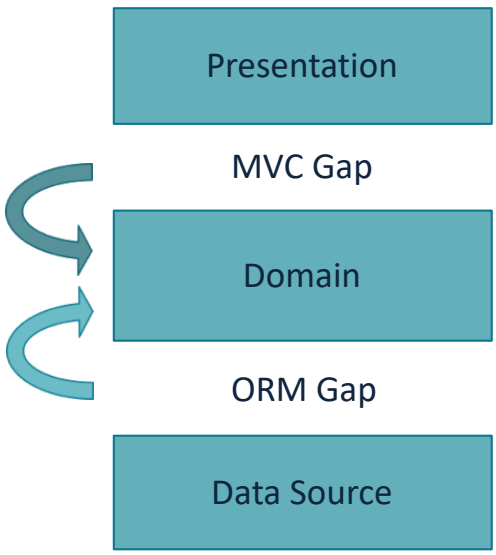
The fundamental goal of the dual model architecture is to gain advantages from a higher separation of concern. The advantages can be seen at the highest level as a decrease in representational gaps. Representational gaps exist above and below the domain layer. The Presentation/Domain gap is usually bridged with Model View-Controller technology, and the Domain/Data Source gap is bridged with Object-Relational Mapping technology when using a database.



In the single model architecture, the domain layer is an amalgamation heavy with implementation concerns, so application controllers are used to transform data for the presentation layer. In the dual model architecture, however, the presentation layer is accessing conceptual entities much closer to the form expected by the end users, which requires less transformation. In other words, the conceptual model creates a smaller MVC gap, which requires less code to fill it.

A similar advantage is found below the domain model. In the single model architecture, designers try to preserve domain concepts to maintain continuity with domain requirements. This is especially true in the Domain Driven Design methodology. Mapping conceptual objects to normalised data sources is a complex problem and has challenged ORM technologies for decades. For example, efficiently mapping object structures based on diagram 3.1 to a data source schema based on diagram 3.2 is a non-trivial task. In the dual model architecture, however, the presentation layer does not access implementation entities. Designers are free to optimise the implementation entities to the underlying data source technology. This optimisation creates a smaller ORM gap, which requires less code to fill it.

Moving the domain closer to the presentation while at the same time moving the domain closer to the data source was an exercise in architectural code factoring. As illustrated in diagram 3.9, programming concerns that are normally dealt with using MVC and ORM technology were factored (and encapsulated) into the domain layer, between dialog models and physical objects.



3.3.1 PRESENTATION ADVANTAGES

There are several features of the architecture that work together to bring advantages to the presentation layer.

1. Conceptual views of the domain model
2. External facing object-oriented interface
3. Large grained object-oriented protocol
4. Metadata driven domain classes
5. Metadata driven user interfaces
6. Server directed user interfaces

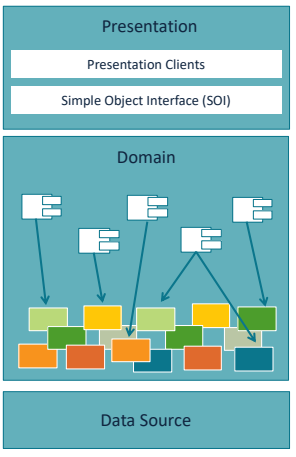
These features make it possible to build user interface programs that are rich interface “engines,” but small in program size. A rich interface engine is a native program that takes its direction from the server and adapts user interface definitions to its device using native controls and look-and-feel.

3.3.2 DATA SOURCE ADVANTAGES

The advantages realised at the data source layer are due to physical objects being completely independent from application controllers and presentation programs. Working with databases is easier. There is a natural affinity between physical objects and databases, so freeing the physical objects from higher layers allow ORM implementations to better encapsulate database rows with objects. Working with web services is easier. Because the dialog model receives large-grained requests, it’s easier to relay requests to remote data sources like web services.

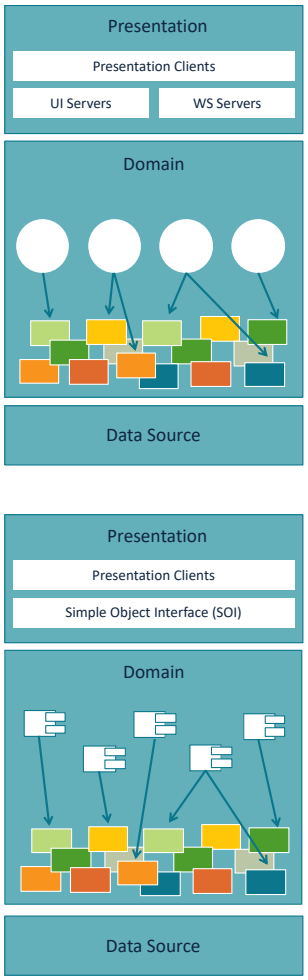
3.4 SIMPLE OBJECT INTERFACE (SOI)

The dialog model mechanism provides an object-oriented interface to the domain, but it’s not appropriate for direct access by the network. Instead, the presentation layer accesses the system through a Simple Object Interface (SOI). This interface accomplishes a few things. First, it narrows the dialog model protocol to only what’s needed by external actors. The remaining protocol is private to intra-dialog communication on the server. Second, it converts data structures to and from a form that’s appropriate for the network. The SOI is a universal interface to the domain and is suitable for user interfaces and web services alike.



3.5 SINGLE MODEL VS. DUAL MODEL

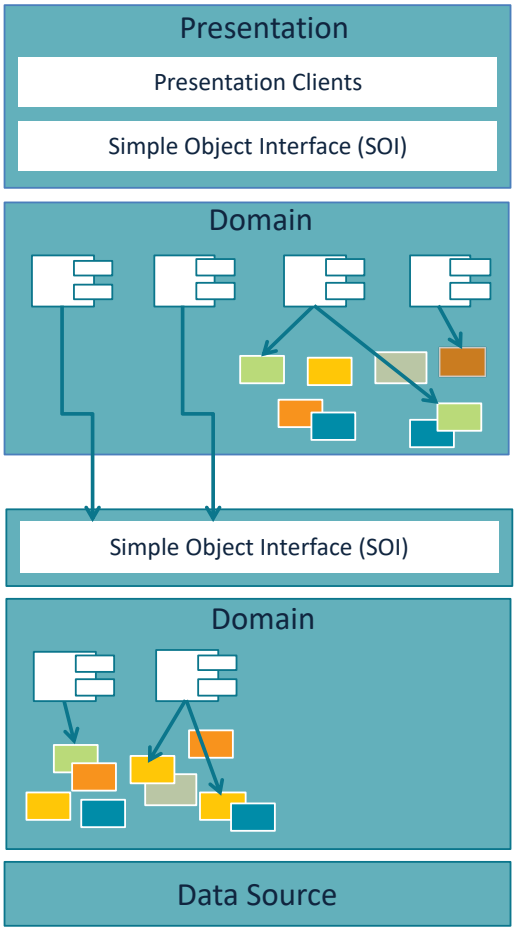
The following diagram presents a side-by-side comparison of the single model and dual model architectures.



4. CLOUD COMPUTING

4.1 API GATEWAY

The API Gateway introduces the notion of distributed computing. In the diagram below, the system at the top of the diagram has two of its dialog models mapped to a remote system, which is called the API Gateway. Notice that data sources located in multiple locations can be mashed up into a single view. This example pulls from local and remote date sources.



4.2 CONNECTOR PROVIDER INTERFACE AND ARCHITECTURE

Physical objects are completely independent of dialog models, but connecting dialog models to physical objects and diverse data sources is a non-trivial task. The Connector Provider Interface was devised to make this task simpler and to establish a well-defined contract for pluggable data sources.

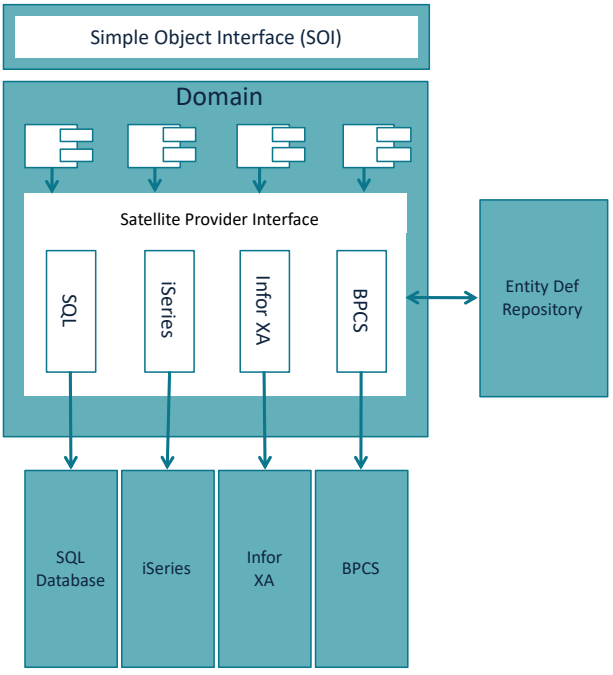
The diagram below illustrates a number of features within the satellite provider architecture.

1. The Connector Provider Interface hides connector provider details. In other words, dialog models communicate with satellite providers using a generic interface (the SPI).

2. Multiple providers can coexist within the same runtime. This means that mash-ups can be created from disparate ERP systems.

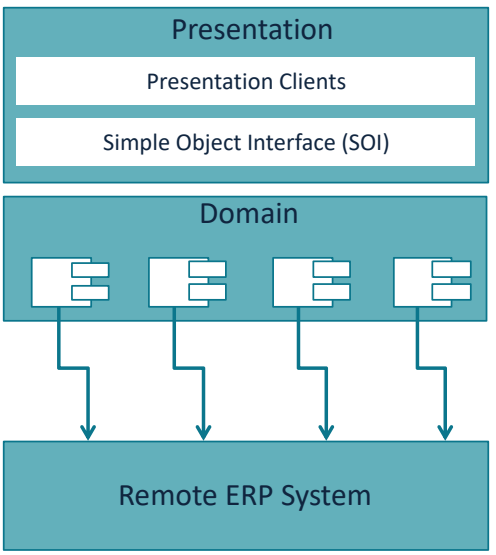
3. There are many aspects of Entity definitions that are common across databases and ERP systems. A common entity repository service exists for all providers. Entity definitions are extensible so that providers can attach vendor specific information.

4. Connector providers can coexist with a direct implementation of physical objects, like an ORM implementation using Hibernate. (This aspect is not illustrated.)



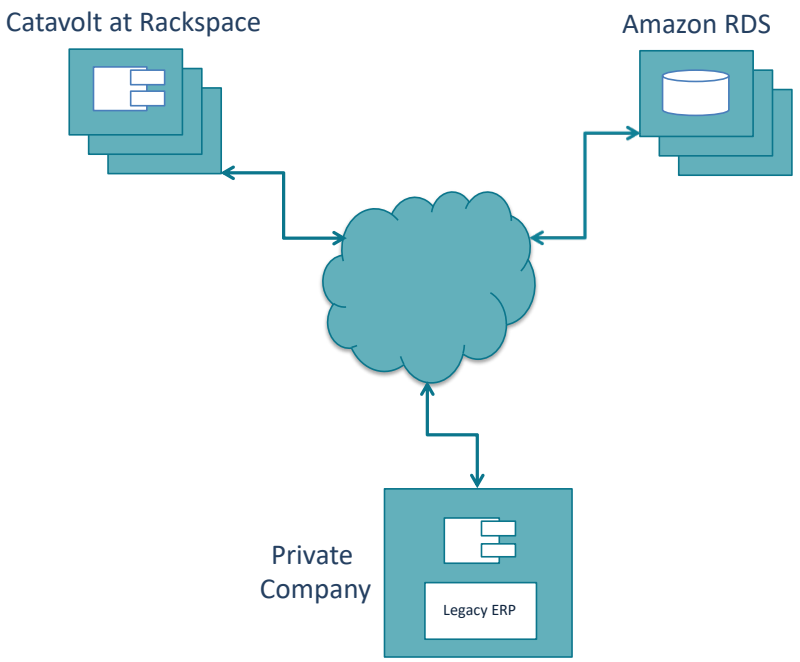
4.3 USER INTERFACE AS A SERVICE

Modernising legacy systems with new user interfaces is a novel application of the dual model architecture. The diagram below shows how a thin layer of dialog models can be used to reface a legacy ERP system, which would open the system to access by iPhones, Galaxy Tablets, Flex Browsers, and more.



4.4 HYBRID CLOUD COMPUTING

Most enterprises have a significant investment in information technology and the thought of moving that technology to the cloud is daunting. The modularity of the dual model architecture allows moving to the cloud in incremental steps. The diagram below depicts a private company who has multiple backend systems networked through the cloud. The Xalt system at Rackspace is the user interface server as well as the primary application server. It has two data sources: one data source mapped to an Amazon RDS (Relational Database Service) and another data source mapped to the legacy ERP system, by way of a satellite server



5. BIBLIOGRAPHY

[Cat11] Osborne, Glenn; Mashini, George. System and Method for Mapping Requests on a Logical Model to Requests on a Physical Model. Patent-Pending Publication No. US-2011-0004603-A1.

[Eva03] Evans, Eric. Domain Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 2003.

[Fow02] Fowler, Martin, et al. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.

[Lar04] Larman, Craig. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition. Prentice Hall, 2004.

[Sil01] Silverston, Len. The Data Model Resource Book. Revised Edition. Wiley, 2001.

About Hexagon

Hexagon is a global leader in digital solutions that create Autonomous Connected Ecosystems (ACE), a state where data is connected seamlessly through the convergence of the physical world with the digital, and intelligence is built-in to all processes.

Hexagon's industry-specific solutions leverage domain expertise in sensor technologies, software, and data orchestration to create Smart Digital Realities™ that improve productivity and quality across manufacturing, infrastructure, safety and mobility applications.

Learn more about Hexagon (Nasdaq Stockholm: HEXA B) at hexagon.com and follow us @HexagonAB.