EASE Application Architecture

Documentation

See "M:\7 DCCS Program Office\Solution Architecture\Del2B" for the current version of Technical Architecture for EASE.

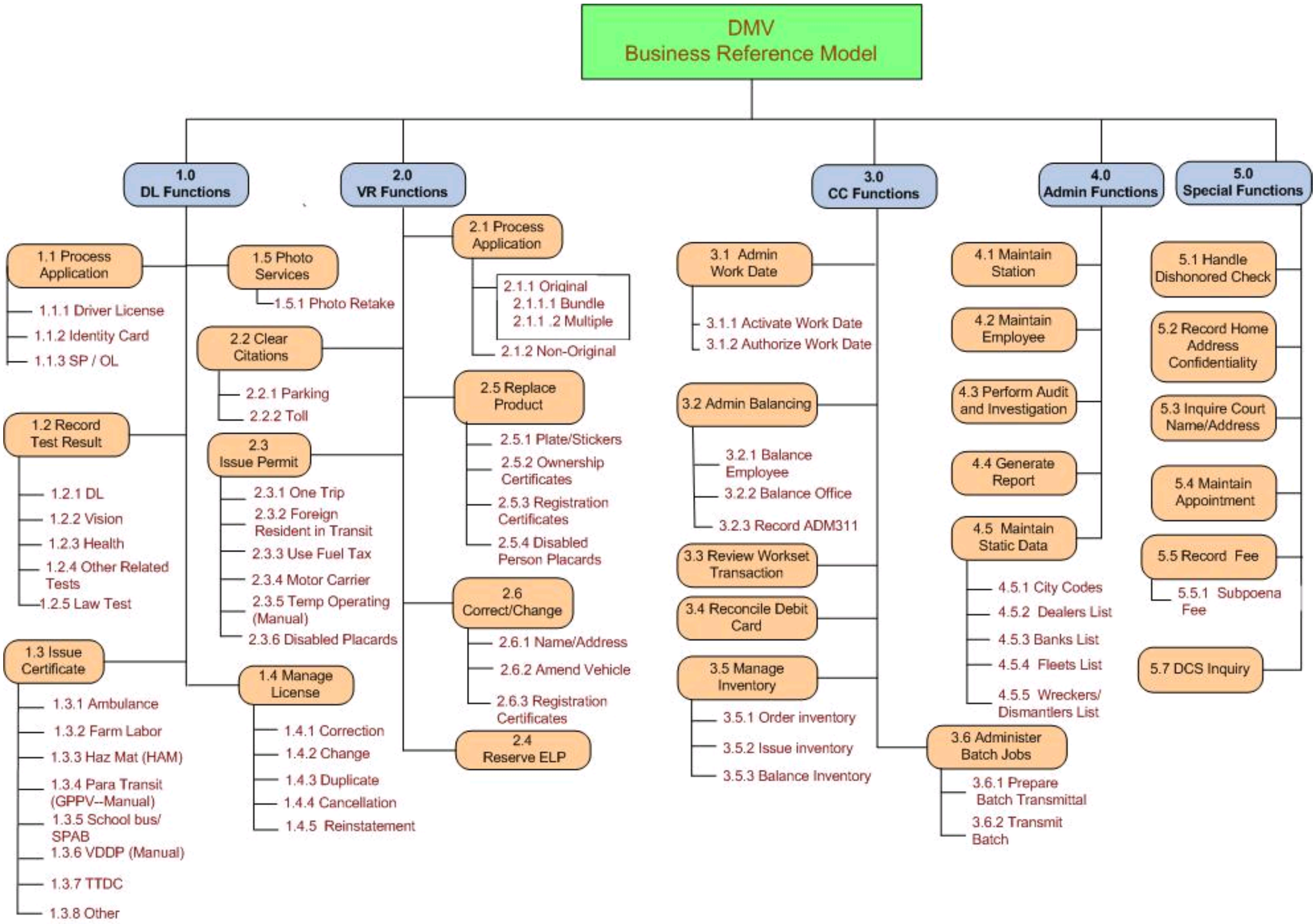
Background

What is Application Architecture?

"Application Architecture" denotes the internal organization of an application or system, including architectural style, highest-level decomposition into parts, and the patterns of collaboration among the parts, with a view to carry out given business functionality.

Application Architecture and the Business Function Reference Model

The business reference model (BRM) provides an organized, hierarchical construct for describing the day-to-day business operations of DMV. The diagram below shows the core business functions supported by the systems within the scope of the EASE project:

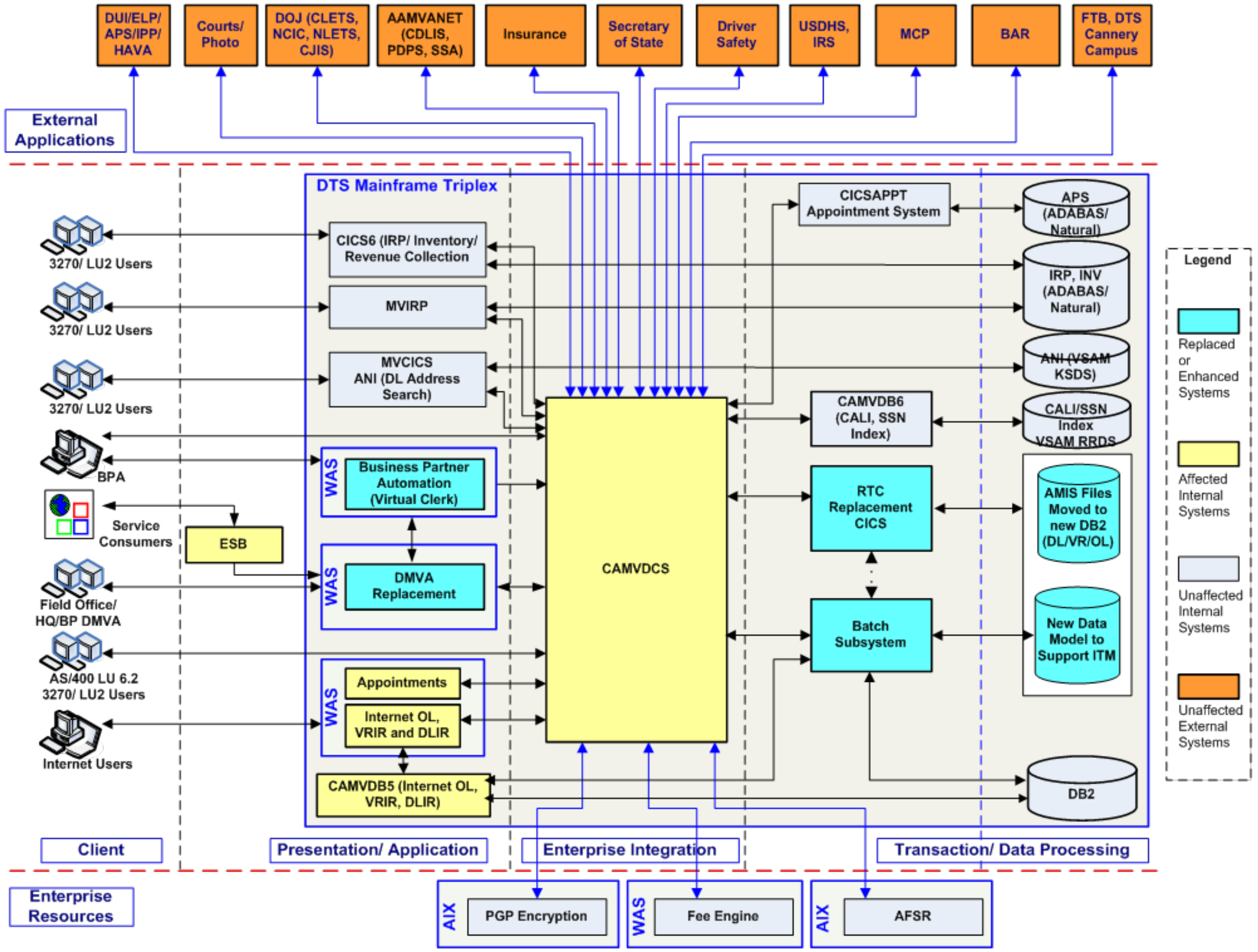


See Also

- EASE Component Architecture
- EASE Integration Architecture
- EASE Service Model

Overview of EASE Application Architecture

EASE Architecture Context


DUI/ELP/ APS/IPP/ HAVA
Courts/ Photo
DOJ (CLETS, NCIC, NLETS, CJIS)
AAMVANET (CDLIS, PDPS, SSA)
Insurance
Secretary of State
Driver Safety
USDHS, IRS
MCP
BAR
FTB, DTS Cannery Campus
External Applications
DTS Mainframe Triplex
3270/ LU2 Users
3270/ LU2 Users
3270/ LU2 Users
BPA
Service Consumers
ESB
Field Office/ HQ/BP DMVA
AS/400 LU 6.2 3270/ LU2 Users
Internet Users
CICS6 (IRP/ Inventory/ Revenue Collection
MVIRP
MVCICS ANI (DL Address Search)
WAS
Business Partner Automation (Virtual Clerk)
WAS
DMVA Replacement
WAS
Appointments
Internet OL, VRIR and DLIR
CAMVDB5 (Internet OL, VRIR, DLIR)
CAMVDCS
CICSAPPT Appointment System
CAMVDB6 (CALI, SSN Index)
RTC Replacement CICS
Batch Subsystem
APS (ADABAS/ Natural)
IRP, INV (ADABAS/ Natural)
ANI (VSAM KSDS)
CALI/SSN Index VSAM RRDS
AMIS Files Moved to new DB2 (DL/VR/OL)
New Data Model to Support ITM
DB2
Legend
Replaced or Enhanced Systems
Affected Internal Systems
Unaffected Internal Systems
Unaffected External Systems
Client
Presentation/ Application
Enterprise Integration
Transaction/ Data Processing
Enterprise Resources
AIX
PGP Encryption
WAS
Fee Engine
AIX
AFSR

EASE Architectural Style

The architectural style adopted for EASE is based on the follow ing styles:

- N-Tier, Layered: focuses on the grouping of related functionality w ithin an application into distinct layers that are stacked vertically on top of each other. Functionality w ithin each layer is related by a common role or responsibility. Communication betw een layers is explicit and loosely coupled.
- Service-Oriented: focuses on providing application functionality as a result of interaction of discrete services (functional and technical). See EASE Service Model
- Object-Oriented: focuses on encapsulating together state (data) and behavior (functions that operate on the data) as objects (rather than separately data and programs operating on that data), using abstraction, polymorphism, inheritance, and encapsulation.
- Component-Based - focuses on the decomposition of the design into individual functional or logical components that expose w ell-defined communication interfaces containing methods, events, and properties. See EASE Component Architecture

Architectural Principles in EASE

- Abstraction. Layered architecture abstracts the view of the system as w hole w hile providing enough detail to understand the roles and responsibilities of individual layers and the relationship betw een them.
- Encapsulation. No assumptions need to be made about data types, methods and properties, or implementation during design, as these features are not exposed at layer boundaries.
- Clearly defined functional layers. The separation betw een functionality in each layer is clear. Upper layers such as the presentation layer send commands to low er layers, such as the business and data layers, and may react to events in these layers, allow ing data to flow both up and dow n betw een the layers.
- High cohesion. Well-defined responsibility boundaries for each layer, and ensuring that each layer contains functionality directly related to the tasks of that layer, w ill help to maximize cohesion w ithin the layer.
- Reusable. Low er layers have no dependencies on higher layers, potentially allow ing them to be reusable in other scenarios.

- Loose coupling. Communication betw een layers is based on abstraction and events to provide loose coupling betw een layers.
- Separation of concerns

Benefits of the Adopted Style

- Domain alignment. Reuse of common services w ith standard interfaces increases business and technology opportunities and reduces cost.
- Rationalization. Services can be granular in order to provide specific functionality, rather than duplicating the functionality in number of applications, w hich removes

duplication.

- Abstraction. Layers allow changes to be made at the abstract level. You can increase or decrease the level of abstraction you use in each layer of the hierarchical stack.
- Isolation. Allow s you to isolate technology upgrades to individual layers in order to reduce risk and minimize impact on the overall system.
- Manageability. Separation of core concerns helps to identify dependencies, and organizes the code into more manageable sections.
- Performance. Distributing the layers over multiple physical tiers can improve scalability, fault tolerance, and performance.
- Reusability
- Testability

EASE Architectural Patterns

Clients

- Thin Web Clients (for human users)
- Web Service Clients (for non-human users)

Presentation Layer Patterns

- Model-View -Controller as present in JSF

Application Layer Patterns

- Business Process Definition and Execution
- Support for nested processes and Business Process Execution Contexts
- Supported types of Activities: Interaction, Business, Decision

Services Layer Patterns

- Taxonomy: Task, Entity, Utility (Business or Technical) Services
- Request/Response-based interaction w ith Services
- Factories for creation of Requests

Business Objects Layer Patterns

- Interface-expressed contracts
- Dependency Injection using Spring

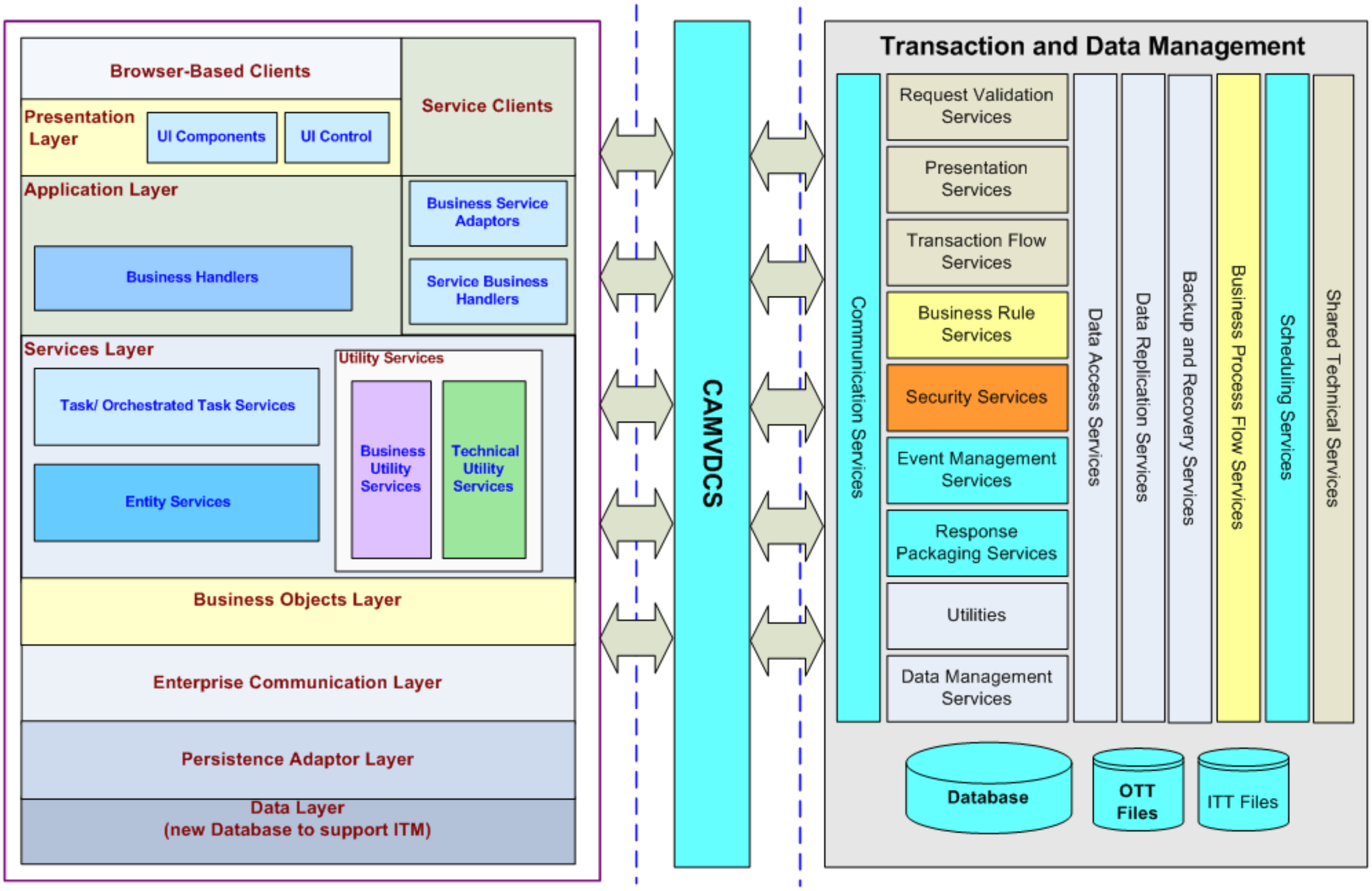
Persistence Layer Patterns

- Object-Relational Broker as present in Hibernate
- Coarse-grained persistence of object graphs (rather than single objects)

Integration Patterns

- Message-Oriented Interactions based on JMS and encapsulated in a service - see ECS
- Batch Processing

EASE Application Diagram


Browser-Based Clients
Service Clients
Presentation Layer
UI Components
UI Control
Application Layer
Business Service Adaptors
Business Handlers
Service Business Handlers
Services Layer
Utility Services
Task/ Orchestrated Task Services
Business Utility Services
Technical Utility Services
Entity Services
Business Objects Layer
Enterprise Communication Layer
Persistence Adaptor Layer
Data Layer
(new Database to support ITM)
CAMVDCS
Transaction and Data Management
Communication Services
Request Validation Services
Presentation Services
Transaction Flow Services
Business Rule Services
Security Services
Event Management Services
Response Packaging Services
Utilities
Data Management Services
Data Access Services
Data Replication Services
Backup and Recovery Services
Business Process Flow Services
Scheduling Services
Shared Technical Services
Database
OTT Files
ITT Files