

Software Architecture Illustration

Wang Hao (Hawk)

Wanghao_buaa@yahoo.com

<http://beyondtest.objectis.net/hawkwang>

Revision History

Date	Version	Description	Author
10/April/2005	0.1	This document is used to illustrate how to analyze and design software architecture.	Hawk Wang (汪浩)

Table of Contents

Introduction.....	4
Purpose.....	4
The Audience for This Document	4
What's not in this topic	4
Requirement.....	5
Calculator Requirement	5
High Level Use Case Diagram.....	6
Architecture Requirement	6
Architectural Solutions	8
Map Use Cases onto Architecture Requirements	8
Solutions	9
Solutions for improving software development, maintenance, and configuration.....	10
Solutions for improving the usability of user interface.....	12
Solutions for improving computation	13
Solutions for improving data access, storage, exchange, and deliverables	14
Packaging the component Solutions	14
Architecture Views.....	17
Conclusion	19

Introduction

Purpose

Although software architecture are mentioned and used frequently, still a lot of software engineer, even so-called architect does not know the essential about it. So, this document is presented to make it clear.

In order to make it easy to be understood, a simple application is used as the example. Cf. Requirement section.

The Audience for This Document

This document is written for software engineers as an introduction of how to generate software architecture.

What's not in this topic

1. This is not a formal learning lecture for software development.
2. In order to increase the effectiveness, seldom formal templates or guidelines are used here.
3. Only idea or high level thinking is listed, no detailed information provided.

Requirement

Calculator Requirement

1. The application should support basic computation algorithm and consistent UI.
 - ✓ Addition
 - ✓ Subtraction
2. The computation result can be viewed using multiple views, which should be defined by concrete calculator itself
3. User can display the history data and event and generate deliverables, for example, xml files
4. The application should support extendable algorithm except for addition and subtraction.
That means that the application should support:
 - * API or plug-in mechanism for other vendor :-)
5. Etc.

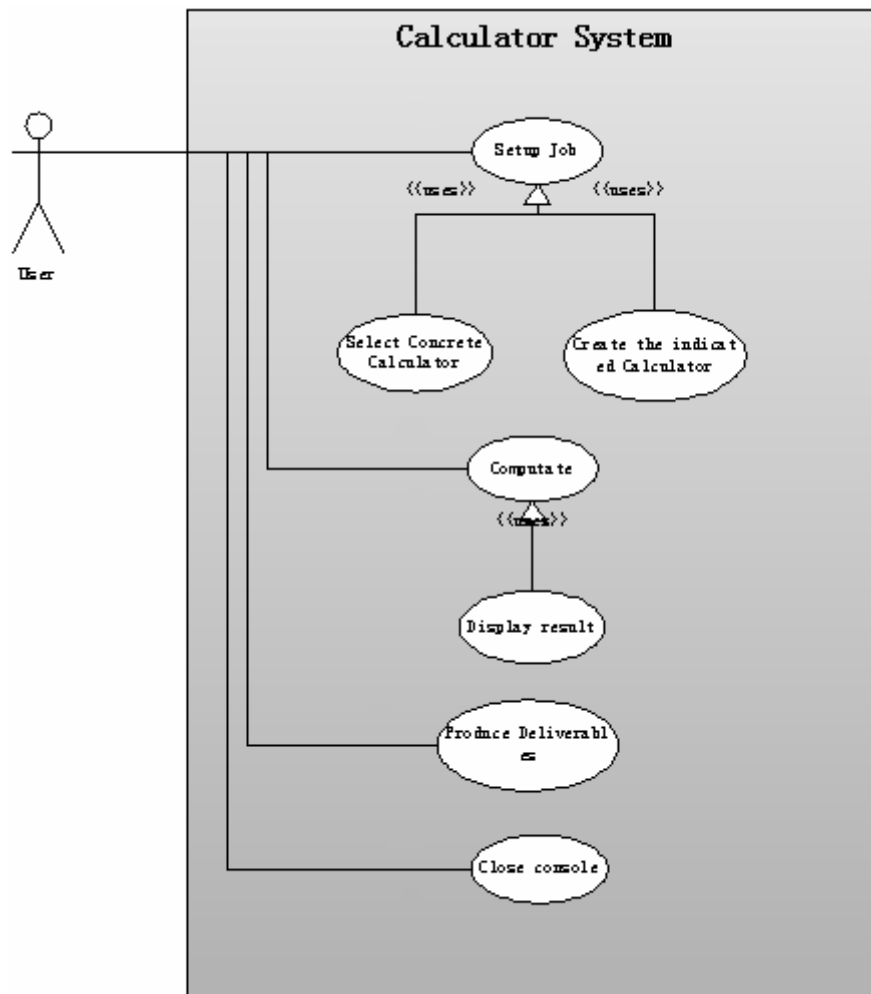
Important announcement:

The calculator application itself is a simple and small application, but we will regard its implementation as a long term product line projects, which will include product platform (framework) development and product development.

Other notes for this document:

1. Normally the terms, such as "etc", "TBD", should not appear in the requirement related docs.
2. This file is alive for bringing new ideas to illustrate software development issues.

High Level Use Case Diagram



Notes:

1. This high level use cases are used to capture main functionality of calculator application, it's an abstraction from concrete calculator, for example, Addition calculator.
2. Need detailed description about each use cases.

Architecture Requirement

Here presents the software requirements for the calculator architecture. The requirements are categorized into one of the following four groups:

1. Requirements for improving software development, maintenance, and configuration

Public

-
- a) The architecture must support rapid, independent software development across product teams, or even product centers
 - b) The architecture must support independent software release schedules across product teams, or even product centers
 - c) The architecture must support calculator configurations of software at runtime
- 2. Requirements for improving the usability and efficiency of user interfaces**
- a) The architecture must support the development of user interfaces that support computation workflows
 - b) The architecture must support the execution of computation
 - c) The architecture must support the displaying of result data from computation workflows
- 3. Requirements for improving calculator computation**
- a) The architecture must support calculator specific computation
- 4. Requirements for improving data access, storage, exchange, and deliverables**
- a) The architecture must support a traceable record of computation data and events

Architectural Solutions

Map Use Cases onto Architecture Requirements

The four top-level use cases, from which we derive computation workflows, are list below.

- **Setup Job** – the tasks users perform to select calculator and define deliverables
- **Computation** – the tasks users perform to acquire and compute input data
- **Display result** – the tasks users perform to display result with different views
- **Generate deliverables** – the tasks users perform to generate report of result of computation, and record of computation data, events.

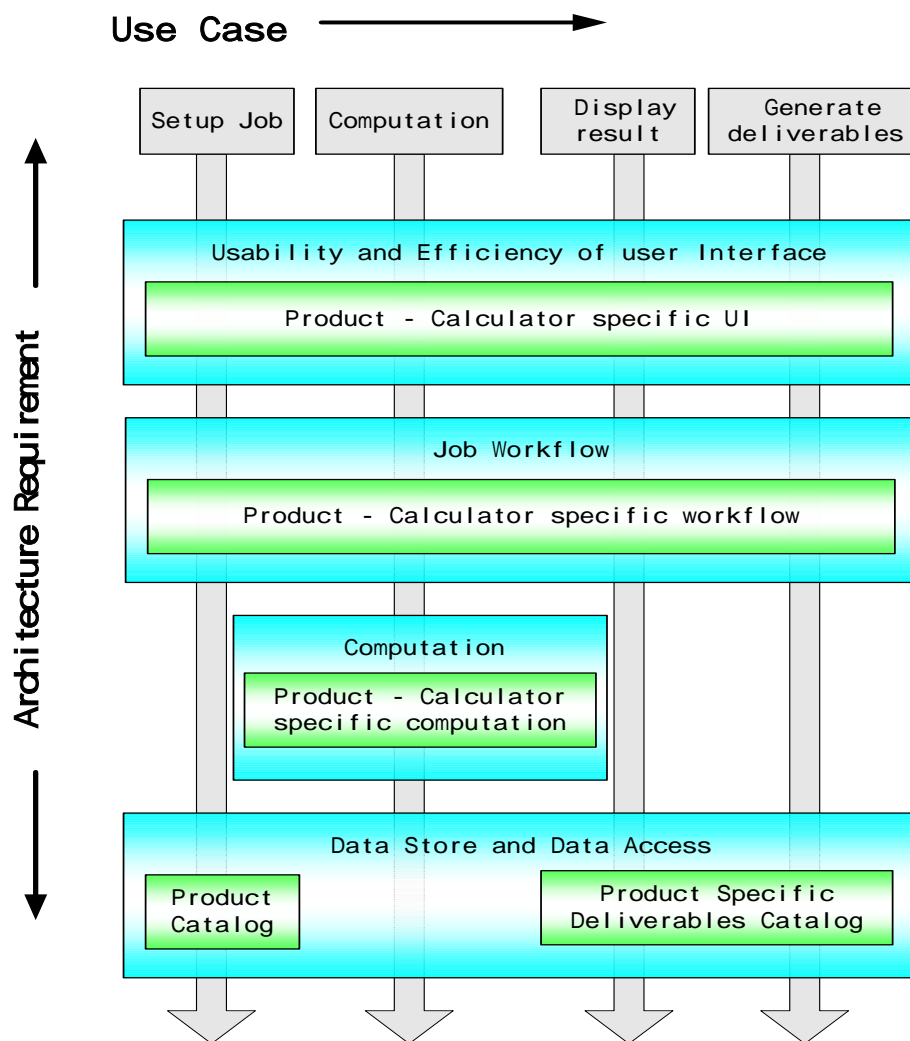


Diagram 1 Map Use Cases onto Architecture Requirements

Solutions

The diagram shows how the solutions are grouped to fulfill the architecture requirements and acquisition use cases.

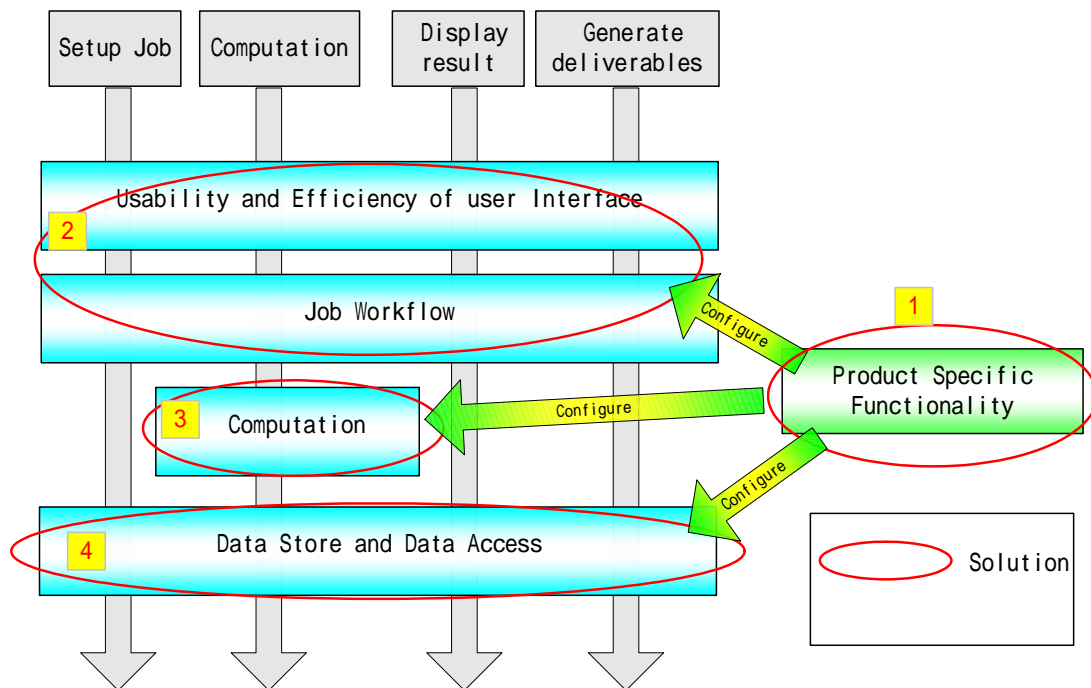


Diagram 2 Architectural Solutions

Solutions for improving software development, maintenance, and configuration

Here, Component Architecture, three-tier architecture and application programming interface (API) are the three main solutions.

The solutions can be understood by following two diagrams.

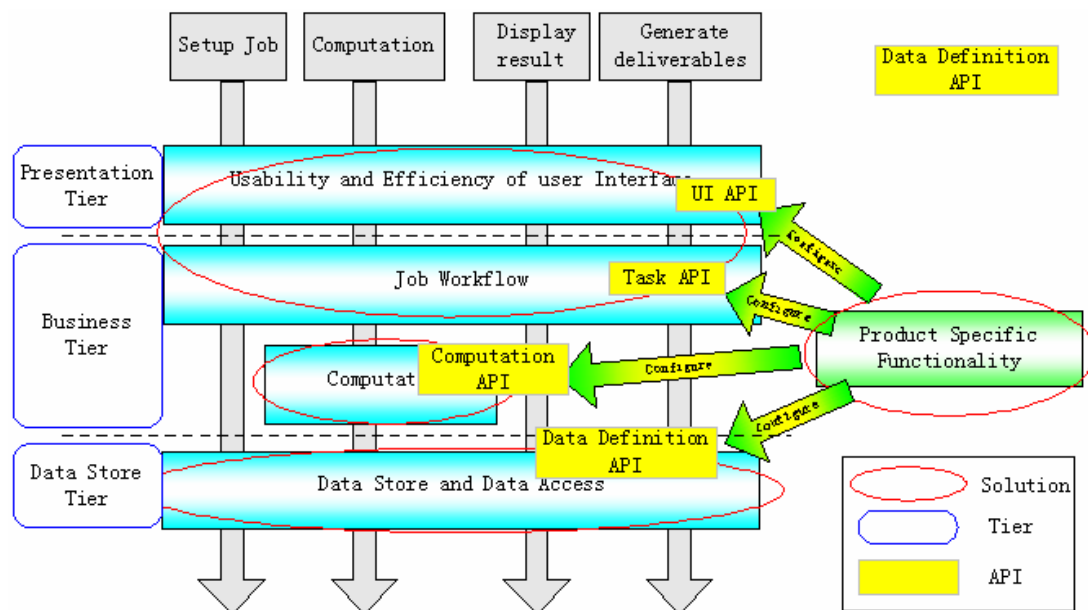


Diagram 3 Three-Tier Component Architecture

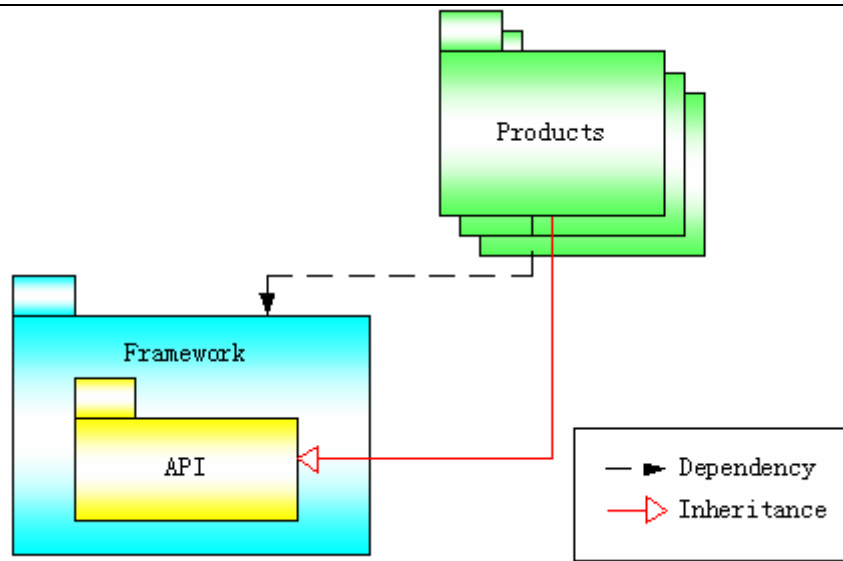


Diagram 4 Component Architecture & API Usage

Solutions for improving the usability of user interface

Diagram 5 presents the solutions for improving the usability of user interface:

1. using **calculator use cases** to develop user interfaces that support computation workflows
2. using **console** and **user interface component** to display data for calculator specific workflows
3. using **UI servers** and **task components** to execute calculator specific workflow
4. using **in-memory data** and **UI Utility** to display and edit computation data
5. using **UI server** to support internationalization and localization features
6. using **tasks** to manage concrete computation jobs

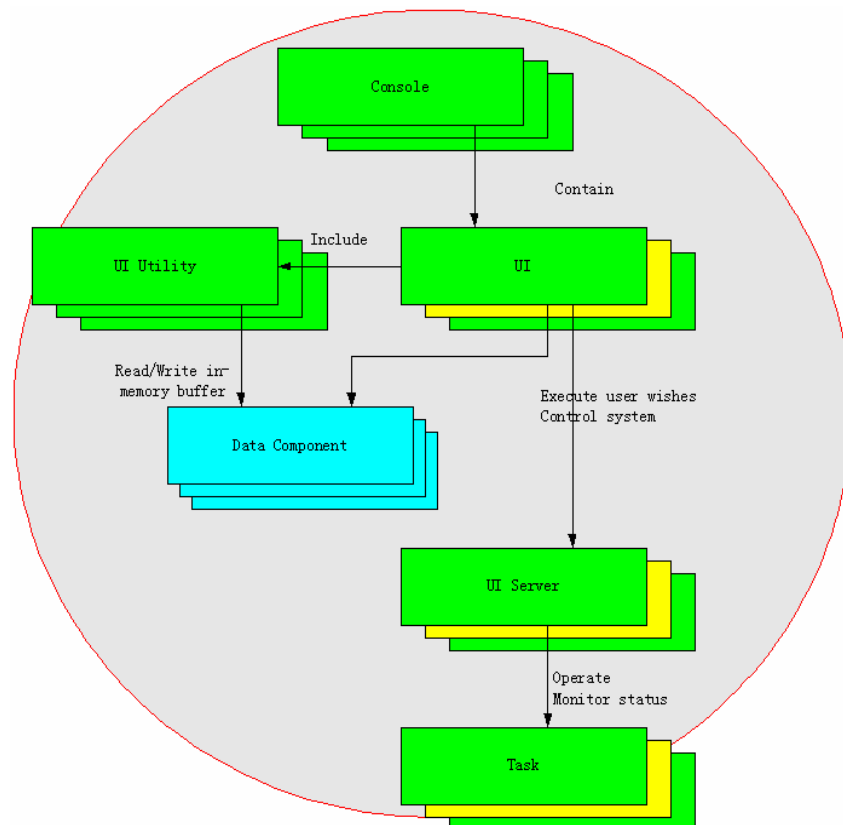


Diagram 5 Components that improve the usability of user interfaces

Solutions for improving computation

Diagram 6 presents the solutions for improving the computation capability:

1. using **network components** to manage computations and re-computations
2. using computation module components to support computation steps

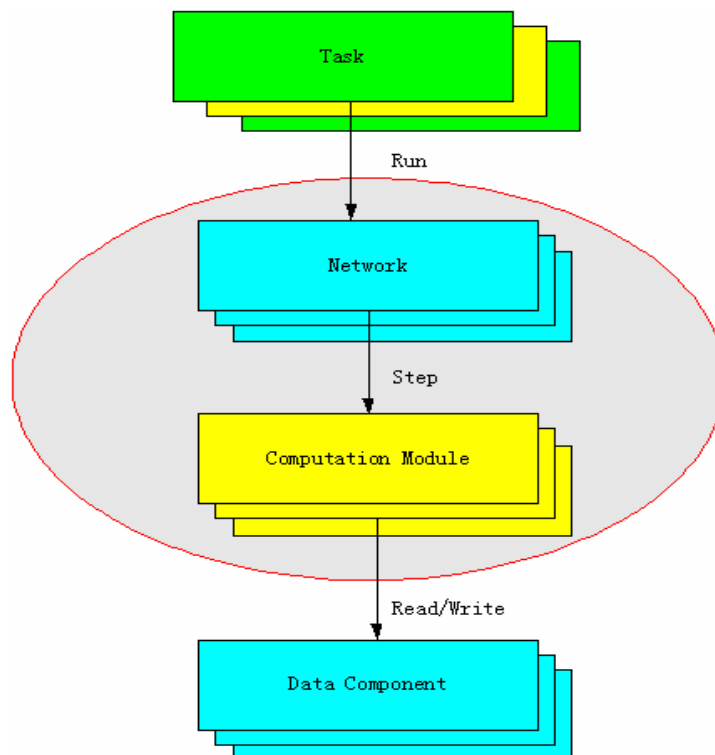


Diagram 6 Components that improve computation

Solutions for improving data access, storage, exchange, and deliverables

Diagram 7 presents the solutions for improving data access, storage, exchange, and deliverables:

1. using **MMFs (Memory Map Files), Data Component, Data Store Clients, and Data Store Server components** to provide access to current and history data
2. using **Ensemble** and **Information** components to implement the domain objects defined by calculator related data model
3. using Catalog components to configure product specific services, for example, deliverables

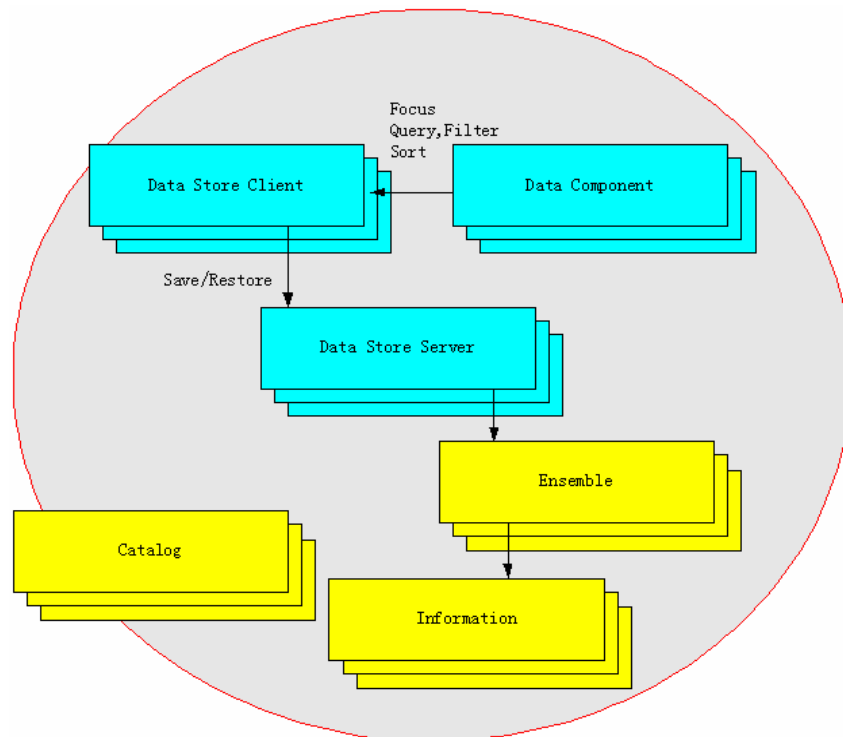


Diagram 7 Components that improve data access, storage, exchange, and deliverables

Packaging the component Solutions

Functionality components are packaged into framework packages, while definition components are packaged into API packages. Following diagrams are corresponding packaging solutions for Public

all the components mentioned above.

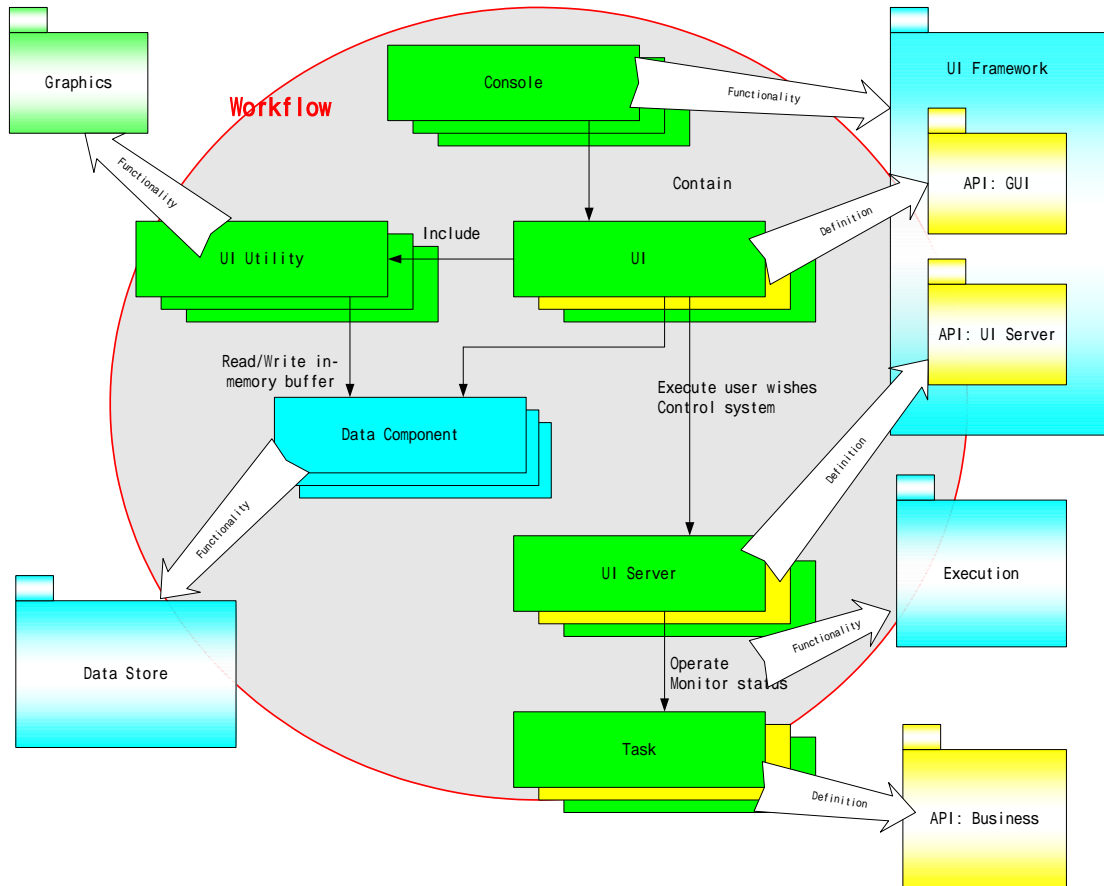


Diagram 8 Packaging components that improve the usability of user interfaces

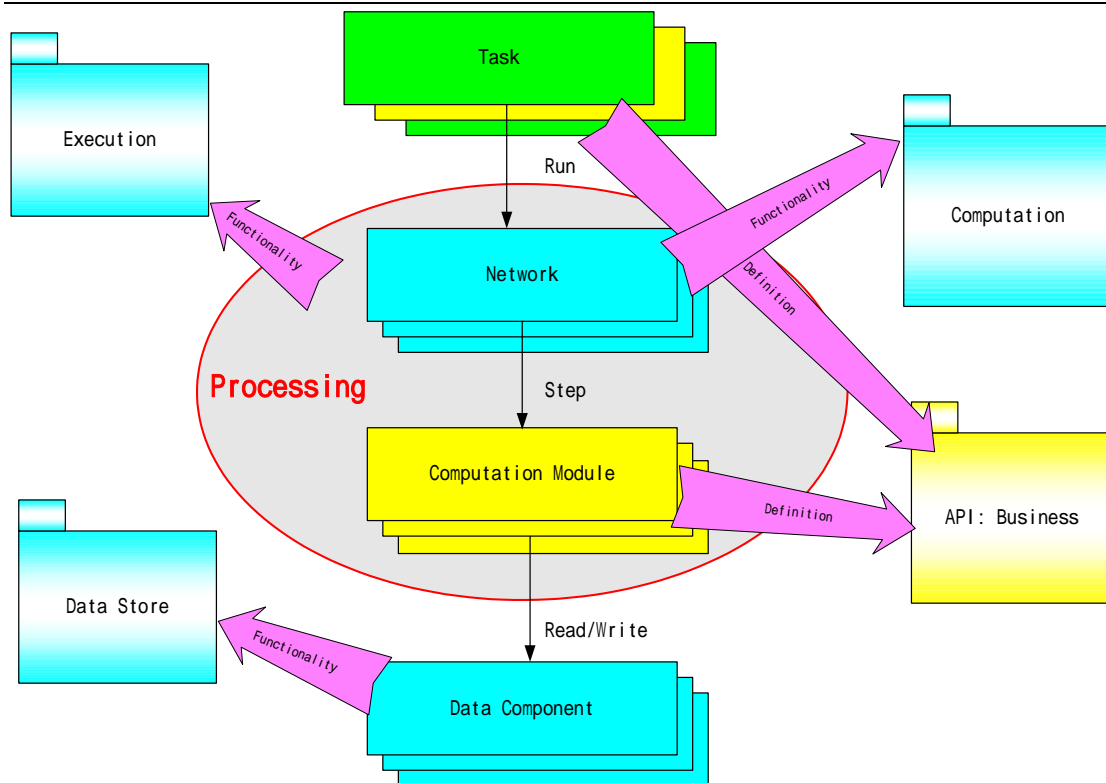


Diagram 9 Packaging components that improve computation capability

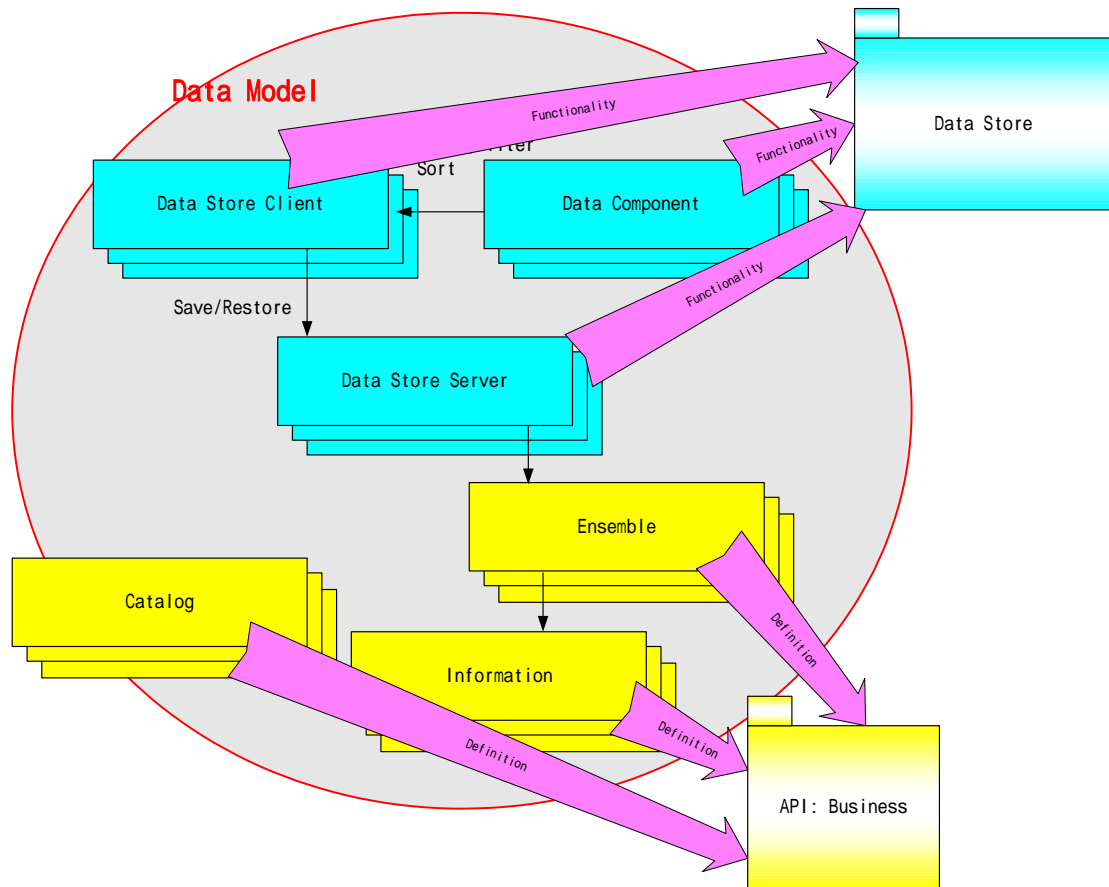
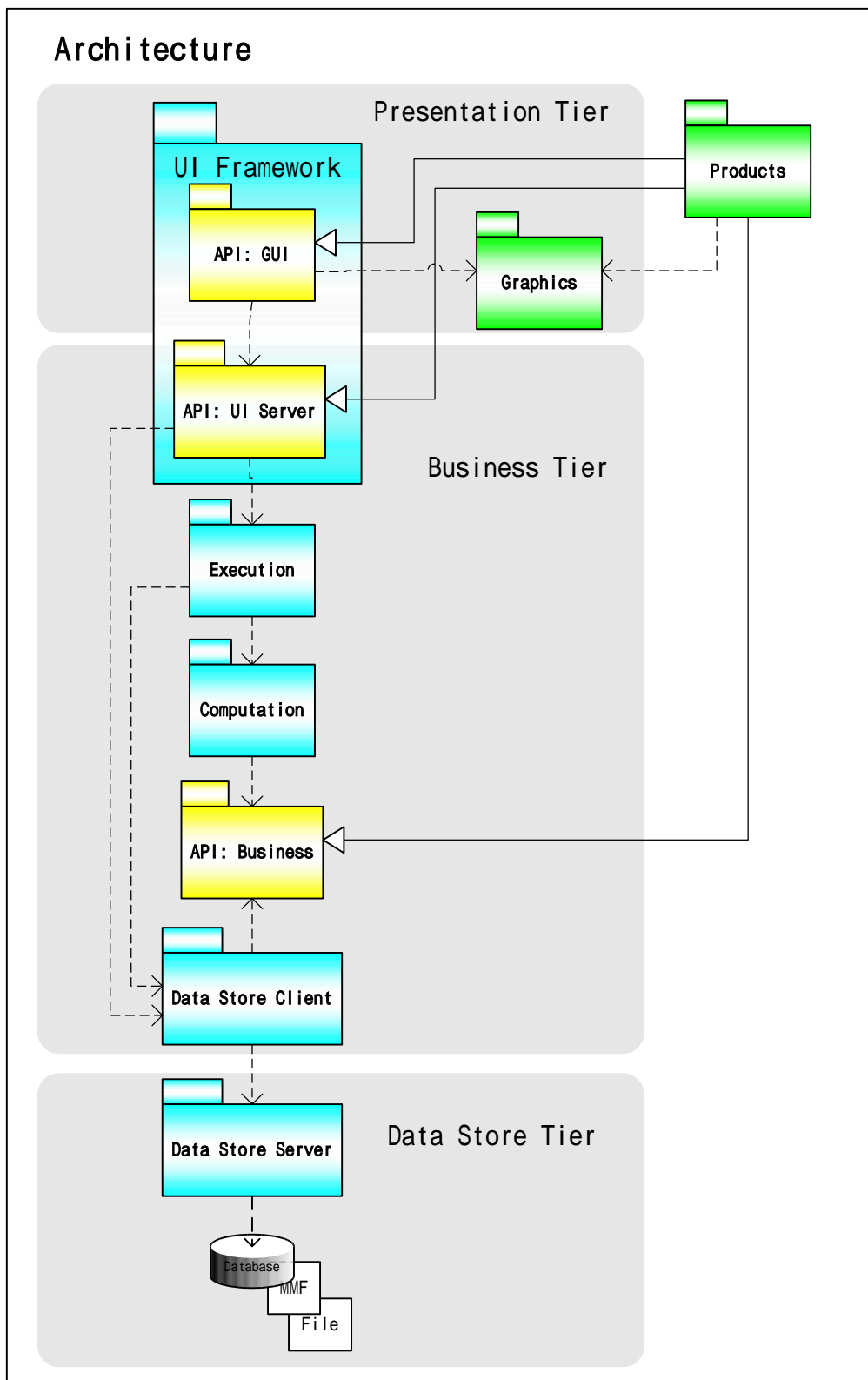


Diagram 10 Packaging components that improve data access, storage, exchange, and deliverables

Architecture Views

Normally, in order to describe architecture clearly, views including logical, workflow, event, process, data, development, etc are needed.

Here only logical view is presented to simplify this document.



The other views will come soon.

Conclusion

In this paper, the method of making architecture was presented. Three main steps were discussed:

1. Identify architecturally significant requirements and use cases
2. Design solutions
 - a) Map requirements onto Use Case
 - b) Build concepts
 - c) Define components
3. Package components

Due to time limited, only logical view was generated here. In order to grasp system architecture, other important views are also needed. The related topic will come soon.