SOA Principles Implementation Report

Course Name: Distributed Software Engineering

Date of Submission: 23.03.2022

—

Submitted by

Noshin Tahsin
MSSE 0909

Chapter 1: Scenario of the project

The Architectural Design Studio (ARC Studio) Business Process

ARC Studio is an architectural design studio. To automate the whole process of the design studio, I will design a preliminary service-oriented architecture and specify a list of web service-centric technology components required to implement the design. The principles of SOA (Service Contract Standardization, Service Coupling, Service Abstraction and Service Reusability) will be applied in this project.

Below are brief descriptions of the **service candidates:**

- The construction of architectural **structures** in the ARC Studio requires the use of specific **materials** that are applied according to predefined designs
- **Concrete, wood, stone, glass, brick** are **materials** ARC purchases from different manufacturers to build their structural models
- **Materials** are added to **inventory**
- **Notifications** are generated when any urgent conditions occur (for example, stock levels fall below certain levels)
- A periodic **patent sweep** is conducted to search if any recently issued patent is similar to ARC's planned architectural designs
- A **report** is generated after each time a **simulation** is successfully conducted and also after each time the simulation fails.

The next chapter presents the project description, planned services and their capabilities.

Chapter 2: Project Description

2.1 Introduction

In ARC Design Studio, architectural structures are simulated. Before the final simulation, the assembly of materials and both previously created and newly created architectural designs undergoes a series of verification checks. Using a customized user-interface, an architect assembles a combination of materials (purchased and/or developed) and retrieves either one or more existing base designs or creates one or more new base designs. A base design is essentially a pre-existing combination of materials.

After the verification checks, the design studio project is executed. If all of the needed materials are available, the simulation takes place, that is, the design solution interacts with a simulator program to graphically display the results of the combination. If any materials are missing or certain design combinations are not possible, the solution will reject the experiment configuration and terminate the project.

2.2 Description

Here are descriptions for the primary process steps, which are further displayed in the **workflow diagram (Figure 2.1).**

1. Issue a **stock level check** on all required materials. If any stock levels are lower than the requested quantities, terminate the process.
2. Retrieve information about required purchased materials. This can include **construction equipment, construction and design tools**, etc. in addition to materials used as ingredients for experiments.
3. Retrieve a list of the requested **base designs**, filtered **as per the need** of the current design project.

4. If a **new base design** is added, generate the base design record and add the design to the **base designs list** for this project.
5. Perform a **validation check** to ensure that all purchased and developed ingredients are **available** in order for the defined designs to be applied.
6. After validation check, submit all the necessary data to the **simulator**.
7. **Output** the results according to a **predefined report format**. If the simulation attempt fails, the returned report contains **error information**, and a **notification** is generated.

After identifying the nouns from the scenario, the service inventory blueprint is designed first (before building the actual services). **(Figure 2.2)**

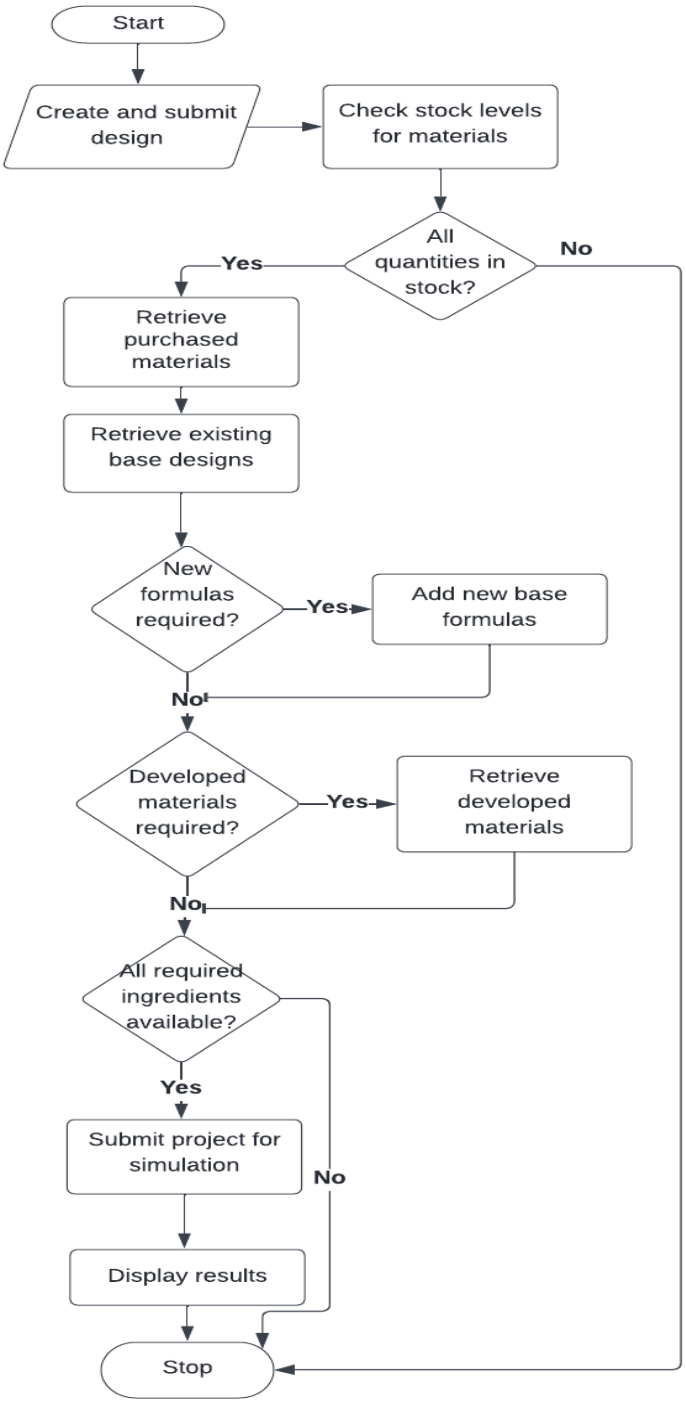

Figure 2.1: The workflow logic for the Design Studio Project

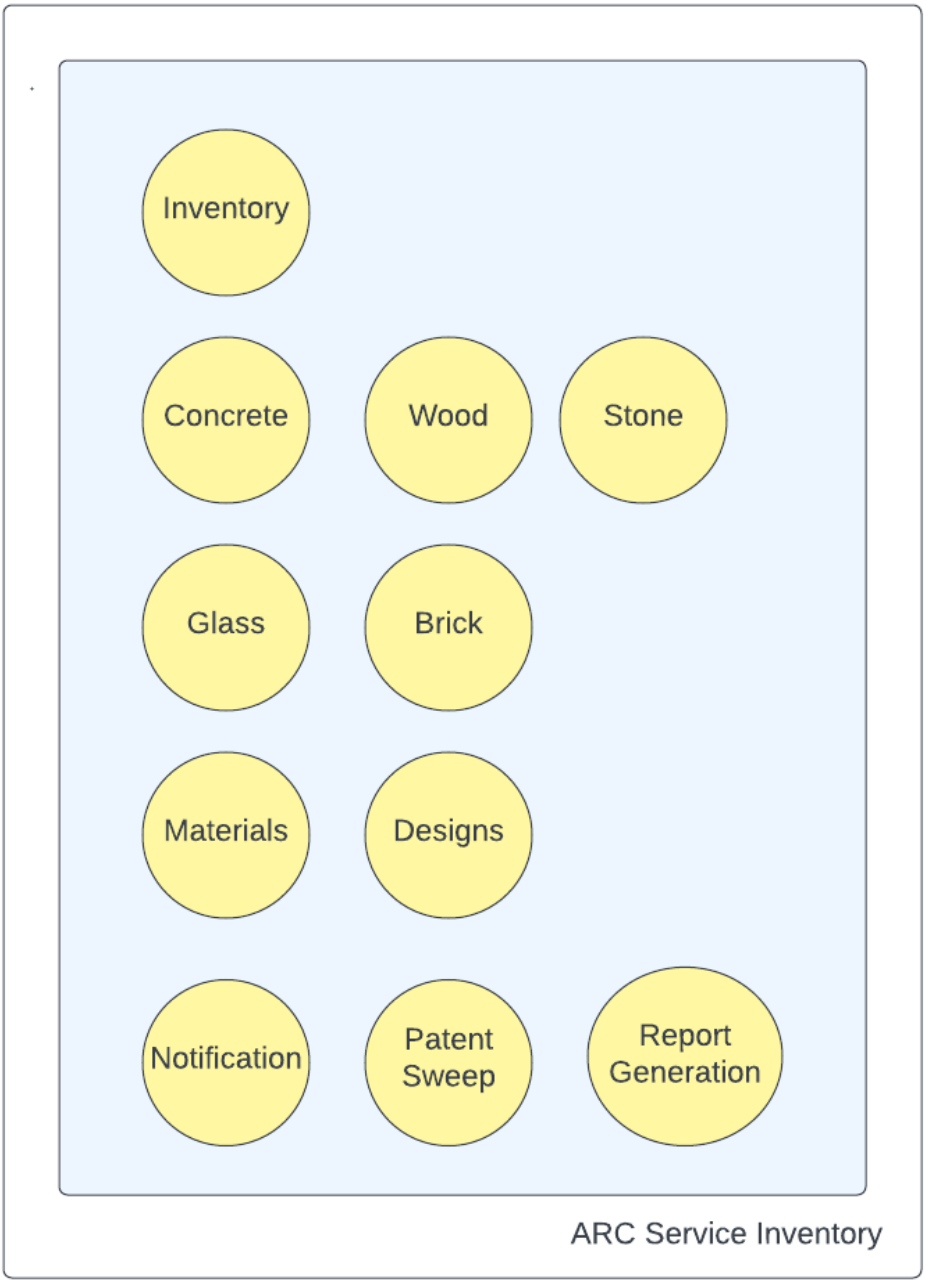

Figure 2.2: The initial set of services planned to support the following types of processes: tracking required construction materials, inventory management of constructed and purchased materials, keeping track of the designs, sending notification, generating reports and conducting periodic patent sweep operations.

All of the displayed services are based on the entity service model, except for the Notification, Patent Sweep and Report Generation services, which are basically utility services.

2.3 Planned Services, Their Capabilities and Models

The planned services, their capabilities and models are specified in this section.

2.3.1 Services

The entity services, utility services and task services are specified below:

Entity Services:

1. Materials
2. Designs
3. Inventory

Utility Services:

1. Notification
2. Patent Sweep
3. Report Generation

Task Service:

1. Run Design Studio

2.3.2 Capabilities

Capabilities of the Materials service:

1. GetDeveloped Operation
2. GetPurchased Operation
3. ReportStockLevels Operation

Capabilities of the Designs service:

1. AddBase Operation
2. Simulate Operation
3. Get Operation

Capabilities of the Inventory service:

1. AddItems operation
2. GetItem Operation
3. GetItemCount Operation
4. RemoveItems Operation

Capabilities of the Notification service:

1. GenerateNotification Operation

Capabilities of the Patent Sweep service:

1. PerformPatentSweep

Capabilities of the Report Generation service:

1. GenerateReport

Capabilities of the Run Design Studio service:

1. Start Operation

The next chapters present the application the SOA principles and explanation of the design decisions taken.

Chapter 3: Application of Service Contracts and Standardization

This principle states that: ***"Services share standardized contracts. Services within the same service inventory are in compliance with the same contract design standards."***

To address both the **functional expression** and **data representation** aspects for the initial set of service contracts, design standards can be put in place as follows:

In the service inventory established in the previous section for ARC, the following 3 Web services are identified to represent business entities involved in designing the structures:

- Materials
- Designs
- Inventory

These **agnostic services** are used to support various design projects in which different **materials are combined** as per **new and existing designs**. The **Inventory service** stores and keeps track of the materials. The **Run Design Studio** service is a task-centric service that encapsulates the automation requirements of the entire business process and fulfills these requirements by **composing the Materials and Designs services**. **(Figure 3.1)**

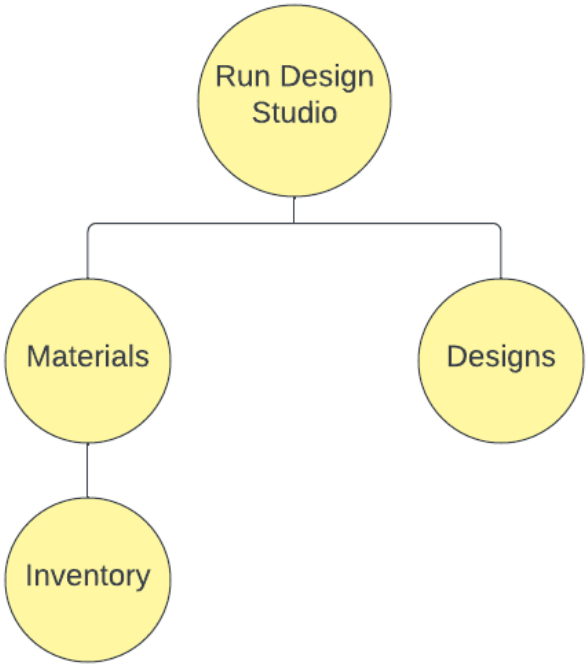

Figure 3.1: High-level look at the simple composition represented by the ARC services responsible for the design studio processes.

3.1 Design Standards Followed

The functional expression and data representation aspects for the initial set of service contracts specified here is followed throughout the design process:

3.1.1 Functional Expression Standards

- Entity services will be named in accordance with the corresponding business from which they are derived
- The names of task services will be based on the process the service is responsible for automating, further prefixed with an appropriate verb
- Operations for all services will be based on the following naming format: verb + noun
- The operation name cannot repeat the name of the service.

3.1.2 Data Representation Standards

- Whenever complex types representing data constructs already established by entity schemas are required, the existing complex types must be used. Therefore, all-encompassing, service-specific schema definitions are prohibited.
- Only when services need new complex types that fulfill processing requirements unique to the service are service-specific schema definitions allowed.
- All XML schema definitions must exist in separate files that are linked to the WSDL definition

3.2 Standardized WSDL Definition Profiles

ARC uses the functional expression standards to define preliminary service contract profiles. The following tables establish the operations related to the automation of the Studio Project Process for each of the services.

Materials Service	
GetDeveloped Operation	Input: unique material identifier and architect identifier Output: developed material document
GetPurchased Operation	Input: unique material identifier Output: purchased material document
ReportStockLevels Operation	Input: unique material identifier Output: stock level value

Table 3.1: Contract profile of the Materials Service (The three operations of the Materials Web service and their respective input and output requirements. Separate operations are provided for the two types of materials records.)

Designs Service	
AddBase Operation	Input: design document and architect identifier Output: acknowledgement code
Simulate Operation	Input: materials and designs identifiers required for this project plus an architect identifier Output: simulation report document
Get Operation	Input: unique design identifier and architect identifier Output: design document

Table 3.2: Contract profile of the Designs Service (In addition to the retrieval of design documents, the operations of the Designs Web service allow for the creation of base design records as well as the combining of base designs into compound designs.)

Inventory Service	
AddItem Operation	Input: standard inventory item document Output: acknowledgement code
GetItem Operation	Input: unique inventory identifier Output: standard inventory item document
GetItemCount Operation	Input: unique inventory identifier

	Output: stock level value
RemoveItems Operation	Input: unique inventory identifier for each item to be removed from inventory Output: acknowledgment code

Table 3.3: Contract profile of the Inventory Service (The inventory service is responsible for adjusting item quantity in the inventory, editing item records, getting items and item counts and reporting stock levels.)

Notification Service	
GenerateNotification Operation	Input: unique event identifier Output: a notification specific to the event associated with the identifier received as input

Table 3.4: Contract profile of the Notification Service (The Notification Service generates notification for specific events (like when stock level is low))

Patent Sweep Service	
PerformPatentSweep Operation	Input: Output: A report after each patent sweep, specifying the latest patent sweep status

Table 3.5: Contract profile of the Patent Sweep Service (After each periodic patent sweep to search for recently issued patents with similarities to ARC's planned architectural designs, the Patent Sweep service generates a report specifying the latest patent sweep status)

Report Generation Service	
GenerateReport Operation	Input: unique event identifier Output: a report specific to the event associated with the identifier received as input

Table 3.6: Contract profile of the Report Generation Service (This service generates reports for specific events, when required and invoked)

Run Design Studio Service	
Start Operation	Input: architect identifier and date value Output: acknowledgement code

Table 3.7: Contract profile of the Run Design Studio Service (The Run Design Studio task service has a simple operation that kicks off the Design Studio Process)

3.3 Standardized XML Schema Definitions

As part of this service delivery project, XML schema definitions are also defined. These schemas are required to provide the necessary **complex types for the input and output message definitions** listed in the preceding service description tables.

- DevelopedMaterial.xsd (entity)
- PurchasedMaterial.xsd (entity)
- Materials.xsd
- Designs.xsd (entity)
- BaseDesign.xsd

- Inventory.xsd
- Architect.xsd (entity)
- Notification.xsd
- Patent.xsd
- Report.xsd
- RunDesignStudio.xsd

As indicated in the preceding list, four of the schema definitions are derived from data models based on existing information sets or business entities. The other schemas provide complex types that support service-specific data exchange requirements.

3.4 Service Descriptions

Before specifying the underlying contract details, how the WSDL definitions relate to the schema definitions, is described first.

3.4.1 Materials Service:

- **Materials** in the ARC inventory are grouped into **two categories**: developed and purchased. Each represents a type of material document with different attributes and characteristics.
- The **Materials.wsdl** definition needs to expose the ability for consumer programs to process data associated with these two types of materials. Separate **GetDeveloped** and **GetPurchased** operations are therefore provided, each of which represents logic that accesses different databases that return **different document structures.**
- Almost all of the complex types required to define the input and output messages for the **GetDeveloped** and **GetPurchased** operations are defined in the respective **DevelopedMaterial.xsd** and **PurchasedMaterial.xsd** schema definitions.
- The **GetDeveloped** operation retrieves documents representing internally created materials considered intellectual property, so, it also requires the

architect ID of the person issuing the request. The complex type for this identifier is defined in the **Architect.xsd** schema.

- The **GetStockLevels** operation returns the **current in-stock value** of a given material. The input and output messages for this operation are defined separately in the **Materials.xsd** definition.

3.4.2 Designs Service

- Designs exist as separate records within the ARC Studio. There are **base designs** that are created and defined individually but which can also be combined.
- The **Designs.wsdl** definition provides operations for the **creation** of base designs and the **simulated** application of designs via the **AddBase** and **Simulate** operations. A simple **Get** operation is defined that **retrieves** one or more design documents.
- The input message for the **AddBase** operation and the output message for the **Get** operation are defined in the **BaseDesigns.xsd** schema which represents the official document structure for design records.
- The **acknowledgement code values** required by the output messages for the **AddBase** operation and the **simulation report structure** that is output by the Simulation operations are defined in the service-specific **Designs.xsd** definition.
- Design records are also considered private and secured information. The **identifiers of architects** working with designs therefore always need to be supplied. The architect identifier required as an input value by all three operations is defined in the **Architect.xsd** schema.

3.4.3 Inventory Service

- Both the **GetPurchased** and **ReportStockLevels** operations of the Materials service are required to interact with the **inventory** service.
- **GetItem** and **GetItemCount** provide the functionality required to carry out the Materials service's **GetPurchased** and **ReportStockLevels** operations respectively.

3.4.4 Notification Service

- The **GenerateNotification** operation generates notification for a specific event, when a unique event identifier is provided. The notification structure is defined in the **Notification.xsd.**

3.4.5 Patent Sweep Service

- The **PerformPatentSweep** operation performs a patent sweep periodically. The **latest patent status report structure** is defined in the **Patent.xsd.**

3.4.6 Report Generation Service

- The **GenerateReport** operation generates reports when invoked. The **report structure** is defined in the **Report.xsd.**

3.4.7 Run Design Studio Service

- The **RunDesignStudio.wsdl** definition exposes a Start operation that requests a **date value** as well as an **architect identifier** for the process to begin.
- The **input date value** and the **acknowledgement code** that is output when the process has been successfully initiated are defined in the **RunDesignStudio.xsd** definition.
- The **architect identifier** needed as input for the Start operation is defined in the **Architect.xsd** schema.

Chapter 4: Application of Service Coupling

This principle states that: ***"Services are loosely coupled. Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment."***

All the 7 services designed for the ARC studio exhibit a high level of **logic-to-contract coupling**, because they are custom services for which **standardized service contracts** are delivered.

The Materials, Designs and Inventory services are based on the entity service model, which deliberately **decreases potential functional coupling** to external or parent business process logic. **Run Design Studio**, which is a task-centric service, is bound to the Design Studio business process, which is a very specific procedure within the ARC Studio. As a result, this service shows **targeted functional coupling** which is intentionally done during its design.

Chapter 5: Application of Service Abstraction

This principle states that: ***"Non-essential service information is abstracted. Service contracts only contain essential information and information about services is limited to what is published in service contracts."***

Each of the services documented in the previous sections for the ARC Design Studio process has its own set of abstraction levels specified in the following.

5.1 Service Abstraction Levels

The following tables summarize the technology, functional, programmatic, and quality of service abstraction for these 7 services. Toward the end of this example, we'll discover how, upon review of the current abstraction levels, ARC makes some changes to refine and better support this principle.

Materials Service	
Functional Abstraction (Content Abstraction)	Concise (the service contract provides targeted functionality with limited constraints)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly documented and published as part of architecture specifications)
Programmatic Abstraction (Access Control)	Open Access (source code and design specifications are openly available on the local LAN)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table 5.1. Abstraction levels for the Materials service.

Designs Service	
Functional Abstraction (Content Abstraction)	Detailed (due to complex rules associated with the exchange of design data this service's contract has a low level of functional abstraction)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly docu- mented and published as part of architecture specifications)
Programmatic Abstraction (Access Control)	Open-to-Controlled Access (source code and design specifications for the Web service are openly available on the local LAN, but information about the Designs database is tightly guarded by a group of DBAs)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table 5.2. Abstraction levels for the Designs service.

Inventory Service	
Functional Abstraction (Content Abstraction)	Concise (a limited amount of available functionality is exposed via the service contract)
Technology Information Abstraction (Access Control)	Controlled Access (access to documentation of the older, more proprietary technology behind the legacy system encapsulated by this service is not openly available and requires permission)
Programmatic Abstraction (Access Control)	Controlled Access (source code and the original system design specifications are kept on a separate server with limited access)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table 5.3. Abstraction levels for the Inventory service

Notification Service	
Functional Abstraction (Content Abstraction)	Concise (a limited amount of available functionality is exposed via the service contract)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly documented and published as part of architecture specifications)
Programmatic Abstraction (Access Control)	Open Access (source code and design specifications are openly available on the local LAN)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table 5.4. Abstraction levels for the Notification service

Patent Sweep Service	
Functional Abstraction (Content Abstraction)	Concise (a limited amount of available functionality is exposed via the service contract)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are

	openly documented and published as part of architecture specifications)
Programmatic Abstraction (Access Control)	Open Access (source code and design specifications are openly available on the local LAN)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table 5.5. Abstraction levels for the Patent Sweep service

Report Generation Service	
Functional Abstraction (Content Abstraction)	Concise (a limited amount of available functionality is exposed via the service contract)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly documented and published as part of architecture specifications)
Programmatic Abstraction (Access Control)	Open Access (source code and design specifications are openly available on the local LAN)
Quality of Service	Open Access (SLA is published alongside

(Access Control)	service contract)

Table 5.6. Abstraction levels for the Report Generation service

Run Design Studio Service	
Functional Abstraction (Content Abstraction)	Optimized (the sole operation provided by this Web service has few constraints and could likely not be more efficiently designed)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly documented and published as part of architecture specifications)
Programmatic Abstraction (Access Control)	Open Access (source code and design specifications are openly available on the local LAN)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table 5.7. Abstraction levels for the Run Design Studio service

We can see from the above tables, there is an alarmingly **high amount of openly accessible meta information available** for most of these services. External consultants (that will need to be hired for upcoming service delivery projects) will be able to read up on any existing service details unless an access control process is put in place. But it is **also necessary** as it would **provide them with valuable background details** required.

So, ARC needs to **assign a service custodian** responsible for the **ownership** of all entity services. This will **effectively protect and limit** the technology and programmatic meta information to the future project teams.

Chapter 6: Application of Service Reusability

This principle states that: ***"Services are reusable. Services contain and express agnostic logic and can be positioned as reusable enterprise resources."***

We will now apply this principle in the case of the ARC Design Studio Project.

The contract profile for the Inventory Service looks like this so far.

Inventory Service	
AddItem Operation	Input: standard inventory item document Output: acknowledgement code
GetItem Operation	Input: unique inventory identifier Output: standard inventory item document
GetItemCount Operation	Input: unique inventory identifier Output: stock level value
RemoveItems Operation	Input: unique inventory identifier for each item to be removed from inventory Output: acknowledgment code

Table 6.1: Contract profile for the Inventory Service

6.1 Assessing Current Capabilities

The service contract for the Inventory service is **closely reviewed** with an **emphasis on facilitating service consumers beyond the Run Design Studio service.** To enhance this service and ensure reusability, I changed it as follows.

The Run Design Studio service needs to compose the following Inventory service operations:

- **AddItem:** Adds items in the inventory.
- **GetItem:** Used to retrieve an inventory item record that can then be examined by a studio team member/architect prior to deciding on whether to use it.
- **GetItemCount**: Used to retrieve the in-stock quantity of a particular item. Sometimes a certain quantity is required, and if the required amount is not available, an order needs to be immediately placed (or alternative items can be considered).
- **RemoveItems:** When one or more items are chosen, this operation decreases the stock level.

6.2 Modeling for a Targeted Measure of Reusability

Assessing the most immediate and critical inventory item-related functions that upcoming business processes will require, we identified the following **additional capabilities for the Inventory service:**

- Revising the cost of an existing inventory item record (usually in response to changing vendor prices)
- Generating a particular type of stock levels report that, based on specific criteria, summarizes all items nearing dangerously low stock quantities. This report would be used as the basis for a regular re-ordering (or advance ordering) process..

After **re-considering the above capabilities**, and to ensure **reusability**, the following changes are made to the Inventory service:

6.2.1 The New EditItemRecord Operation

An **EditItemRecord** operation is added, because it is determined that **AddItemRecord** and **DeleteItemRecord** operations would also be needed soon. These are **different from the existing AddItem and RemoveItem operations** in a sense that, the former actually **inserts and deletes inventory item records** in the system database, whereas the **latter two only increase and decrease inventory item quantity values**. Additionally, even though the immediate requirement was for the ability to change inventory item costs, this **new operation allows any editable part of an inventory item record to be updated**, including its cost.

6.2.2 The New ReportStockLevels Operation

A new operation called **ReportStockLevels** is added to enable a **generic operation capable of generating stock quantity-related reports**, including the specific "low stock quantity" report identified as an upcoming requirement.

6.2.3 The New AdjustItemsQuantity Operation

The original **AddItem** operation accepted a quantity value for a particular item record and then **increased the recorded stock level** of the item accordingly. The accompanying **RemoveItems** operation allowed multiple item ID and quantity values as input, **subsequently reducing stock levels**. The **new EditItemRecord operation** could now **replace these older operations** by allowing service consumers to submit new item record documents with updated quantity values.

Also, during the service modeling process, it was discussed **how this particular function is required repeatedly**, not just within the Design Studio business process, but in several others as well. It therefore would make sense to **define a more targeted operation**, even if its functionality is somewhat redundant. Essentially, this operation

would not accept an entire inventory item document as input, but only the inventory item ID and the revised quantity value.

We **combined the original AddItem and RemoveItems operations** into a single **AdjustItemQuantity** operation that would be able to increase or decrease the quantity of any given inventory item.

To enable the operation in **receiving and processing a range of items at the same time**, we get the new **AdjustItemsQuantity** operation capable of **accepting a range of item IDs and quantity values.**

6.3 Revised Inventory Service Profile

Table 16 shows how the **Inventory service's contract changed** after a **reusability centric remodeling and redesign effort**. By working toward a targeted measure of enhancement for the Inventory service, **several new processing requirements were accommodated.** These establish capabilities that are known to be useful for multiple business processes and are also capable of **carrying out their functions in an efficient manner**.

Inventory Service	
AdjustItemsQuantity Operation	Input: unique inventory identifier for each item to be removed from inventory Output: acknowledgement code
EditItemRecord Operation	Input: standard inventory item document Output: acknowledgment code
GetItem Operation	Input: unique inventory identifier Output: standard inventory item document

GetItemCount Operation	Input: unique inventory identifier Output: stock level value
ReportStockLevels Operation	Input: query criteria Output: summary in report forrmat

Table 6.2. The revised contract profile for the Inventory service

The **revised inventory service** now **ensures reusability**. It is responsible for adjusting item quantity in the inventory, editing item records, getting items and item counts and reporting stock levels. It can now revise the cost of an existing inventory item record (usually in response to changing vendor prices) and generate a particular type of stock levels report that, based on specific criteria, summarizes all items nearing dangerously low stock quantities.