Architecture Description of The Layered architecture for

*Optimum Hair Finder*

**Introduction**

* 1. **The Layered Architecture**

The Layered or Tiered software architecture has four standard layers in the Optimum Hair Finder software system. And, each layer in the architecture deals with its’ own tasks that need to be done to achieve a particular request. Each request moves from one layer to the next layer concurrently, however, all components for each layer are still independent and changes made to a certain layer will only affect components from other layers that are associated with that change.

* + 1. **Presentation layer**

This is the user interface layer that deals with the main function in the system, to search hair salons.

* + 1. **Business layer**

This layer, also called the domain logic layer, is responsible for executing specific user requests that are sent into the system.

* + 1. **Persistence layer**

This layer converts requests and puts them in proper form in order to send these requests to the system.

* + 1. **Database layer**

This layer deals with all stored data on database and accessible for user requests through architecture layers system.

**1.2** **Supplementary information**

Authors: Raesetje Bonjo Sefala, Laura Bokgoshi, Siphamandla Mzobe, Tshepo Molefe, Keku Mashego

Reviewers: Raesetje Bonjo Sefala, Laura Bokgoshi, Siphamandla Mzobe, Tshepo Molefe, Keku Mashego

Approving authority: Raesetje Bonjo Sefala

Issuing organization: Opi Team

Date of issue: First version 15 August, 2016; Final version: October, 2016

Status: Work in progress

Change history: No changes to the design have been documented yet.

Version control information: Revision number 1

Configuration management information: No updates yet.

Scope: Allows users to search salon of choice on system and receive results of prices, hairstyles, location and distance to salons searched for. System should be able to search salons by specific field and give details about salons.

Context: This document contains the software architecture description of the Optimum Hair Finder system. The descriptions are detailed using architecture design information and UML interaction diagrams. This document is prepared using ISO/IEC/IEEE 42010, Systems and software engineering—The Architecture Description Template

Summary: This document is for use to read architectural design information on the Optimum Hair Finder system.

Glossary:

Chiskop: hair cut, cut all hair off.

Fade: hair cut, ct short around and leave longer in the middle

Weave: hair extensions saw in tracks

Braids: hair extensions, individual hair extension, braid by braid

References:

* ISO/IEC/IEEE 42010, Systems and software engineering—The Architecture Description Template, Template prepared by: Rich Hilliard
* Software Architecture Patterns, Understanding Common Architecture Patterns and When to Use Them, Mark Richards
* Architectural Blueprints—The “4+1” View Model of Software Architecture, Philippe Kruchten
* Software Engineering, Ivan Marsic

**1.3** **Other information**

1. **Architecture evaluations**

Overall agility

Rating: Low

Making changes to system takes time even with the layered system we have since system design is implementations is a big system.

Ease of deployment

Rating: Low

Changes made to the system will not be an issue because all information or data used in the system will be uploaded from the database dynamically. One change will not cause entire system to be updated only the section that deals with that change.

Testability

Rating: High

Each layer can be tested individually, so the overall rating is high.

Performance

Rating: Low

The use of layers will cause time problems as some tasks do not need passing through certain layers but still have to and so wasting valueable time.

Scalability

Rating: Low

The system will be hard to scale since it consists of multiple layers which are all linked somehow and the only way to scale the system would be to split the layers up which is not efficient towards the system.

Ease of development

Rating: High

The use of layers makes development of the software manageable because the different layers will be familiar and so will models within each layer.

1. **Rationale for key decisions**

System architecture consists of four layers because they work best with the system.

**Stakeholders and concerns**

**2.1** **Stakeholders**

The stakeholder for the architecture is the Opi Team.

**Opi Team**

Stakeholder 1: Raesetje Bonjo Sefala

Stakeholder 2: Laura Bokgoshi

Stakeholder 3: Siphamandla Mzobe

Stakeholder 4: Tshepo Molefe

Stakeholder 5: Keku Mashego

**2.2** **Concerns**

**What are the purpose(s) of the system-of-interest?**

To search hair salons in a certain area.

**What is the suitability of the architecture for achieving the system-of-interest’s purpose(s)?**

The layered architecture works best for a search-and-find type system since all hair salon data will be in a database. There is a finite amount of salons in a certain areas, use of a database allows quick access to the salon information.

**How feasible is it to construct and deploy the system-of-interest?**

The deployment of the system is very feasible since all salon data requested from a search is stored in a database and so will allow easy access to that data.

**How is the system-of-interest to be maintained and evolved?**

The system needs to be maintained by continuous updating of salon information, e.g. salon location, new hairstyles or prices at the salons, new salon promotions and concepts. The system will be evolved to spread among more than one city and grow to include more salon interaction with clients through system.

**2.3** **Concern–Stakeholder Traceability**

Table 2.1: Example showing association of stakeholders to concerns in an

AD

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Stakeholder 1 | Stakeholder 2 | Stakeholder 3 | Stakeholder 4 | Stakeholder 5 |
| Concern 1 | X | X | X | X | X |
| Concern 2 | - | - | X | - | X |
| Concern 3 | X | X | - | X | - |
| Concern 4 | X | X | X | X | X |

**Viewpoints**

**3.1 Logical view**

**3.2** **Overview**

The logical view outlines the functional requirements, what the system should provide to the user. This view also outlines the architecture of the system; what is the type of architecture used for this system? how many subsystems exist, how are they related, etc. The outline of the system is not only to show functionality that went into the system but also to show common mechanisms and design elements across the various parts of the system.

**3.3** **Concerns and stakeholders**

1. **Concerns**

What is thepurpose of the system-of-interest?

Does the system-of-interest do what it is supposed to be doing?

How does the system manage fault?

**3.3.2** **Typical stakeholders**

Raesetje Bonjo Sefala

Laura Bokgoshi

Siphamandla Mzobe

Tshepo Molefe

Keku Mashego

**3.1 Process view**

**3.2** **Overview**

This view outlines non-functional requirements, the extra capabilities of the system; performance, testability, usability, scalability, reliability, availability, etc. This view will deal with the order of events, distribution and efficiency of the system including the logical view concepts.

**3.3** **Concerns and stakeholders**

1. **Concerns**

What is the suitability of the architecture for achieving the system-of-interest’s purpose(s)?

**3.3.2** **Typical stakeholders**

Siphamandla Mzobe

Keku Mashego

**3.1 Development view**

**3.2** **Overview**

The development view deals with the software module organization on the software development environment. The software is packaged in a way that allows groups of developers of work on certain parts of the system. This view is represented by module and subsystem diagrams showing all important relationships. This view takes into account internal requirements that allows for easy development, software management and reuse.

**3.3** **Concerns and stakeholders**

1. **Concerns**

How is the system-of-interest to be maintained and evolved?

**3.3.2** **Typical stakeholders**

Raesetje Bonjo Sefala

Laura Bokgoshi

Siphamandla Mzobe

Tshepo Molefe

Keku Mashego

**3.1 Physical view**

**3.2** **Overview**

The physical view takes into account the non-functional requirements of the system. The software executes on a network of physical computers that need to be mapped onto the various processing nodes. Usually we use physical parts for development and testing, and we try for efficient physical components so that they do not interfere with the actual code.

**3.3** **Concerns and stakeholders**

1. **Concerns**

How feasible is it to construct and deploy the system-of-interest?

**3.3.2** **Typical stakeholders**

Raesetje Bonjo Sefala

Laura Bokgoshi

Tshepo Molefe

**3.4** **Model kinds**

? Identify each model kind used in the viewpoint per ISO/IEC/IEEE 42010, 7c.

In the Standard, each architecture view consists of multiple architecture models. Each model is governed by a model kind which establishes the notations, conven-tions and rules for models of that type. See: ISO/IEC/IEEE 42010, 4.2.5, 5.5 and 5.6.

Repeat the next section for each model kind listed here the viewpoint being speci-fied.

**3.5** <**Model Kind Name**>

? Identify the model kind.

1. <**Model Kind Name**> **conventions**

? Describe the conventions for models of this kind.

Conventions include languages, notations, modeling techniques, analytical methods and other operations. These are key modeling resources that the model kind makes available to architects and determine the vocabularies for constructing models of the kind and therefore, how those models are interpreted and used.

It can be useful to separate these conventions into a language part: in terms of a metamodel or specification of notation to be used and a process part: to describe modeling techniques used to create the models and methods which can be used on the models that result. These include operations on models of the model kind.

The remainder of this section focuses on the language part. The next section focuses on the process part.

The Standard does not prescribe how modeling conventions are to be documented. The conventions could be defined:

10

**I)** by reference to an existing notation or language (such as SADT, UML or an archi-tecture description language such as ArchiMate or SysML) or to an existing technique (such as M=M=***4*** queues);

**II)** by presenting a metamodel defining its core constructs;

**III)** via a template for users to fill in;

**IV)** by some combination of these methods or in some other manner.

Further guidance on methods I) through III) is provided below.

Sometimes conventions are applicable across more than one model kind – it is not necessary to provide a separate set of conventions, a metamodel, notations, or oper-ations for each, when a single specification is adequate.

**I) Model kind languages or notations** (**optional)**

Identify or define the notation used in models of the kind.

Identify an existing notation or model language or define one that can be used for models of this model kind. Describe its syntax, semantics, tool support, as needed.

**II) Model kind metamodel** (**optional)**

A metamodel presents the AD elements that constitute the vocabulary of a model kind, and their rules of combination. There are different ways of representing meta-models (such as UML class diagrams, OWL, eCore). The metamodel should present:

**entities** What are the major sorts of conceptual elements that are present in mod-els of this kind?

**attributes** What properties do entities possess in models of this kind?

**relationships** What relations are defined among entities in models of this kind?

**constraints** What constraints are there on entities, attributes and/or relationshipsand their combinations in models of this kind?

NOTE: Metamodel constraints should not be confused with architecture constraints that apply to the subject being modeled, not the notations used.

In the terms of the Standard, entities, attributes, relationships are AD elements per ISO/IEC/IEEE 42010, 3.4, 4.2.5 and 5.7.

In the Views-and-Beyond approach [[1](#page23)], each viewtype (which is similar to a view-point) is specified by a set of elements, properties, and relations (which correspond to entities, attributes and relationships here, respectively).

When a viewpoint specifies multiple model kinds it can be useful to specify a single viewpoint metamodel unifying the definition of the model kinds and the expression

11

of correspondence rules. When defining an architecture framework, it may be help-ful to use a single metamodel to express multiple, related viewpoints and model kinds.

**III) Model kind templates** (**optional)**

Provide a template or form specifying the format and/or content of models of this model kind.

1. <**Model Kind Name**> **operations** (**optional)**

Specify operations defined on models of this kind.

See x[3.6](#page15) for further guidance.

1. <**Model Kind Name**> **correspondence rules**

? Document any correspondence rules associated with the model kind.

See x[3.7](#page16) for further guidance.

**3.6** **Operations on views**

Operations define the methods to be applied to views and their models. Types of operations include:

**construction methods** are the means by which views are constructed under thisviewpoint. These operations could be in the form of process guidance (how to start, what to do next); or work product guidance (templates for views of this type). Construction techniques may also be heuristic: identifying styles, patterns, or other idioms to apply in the synthesis of the view.

**interpretation methods** which guide readers to understanding and interpretingarchitecture views and their models.

**analysis methods** are used to check, reason about, transform, predict, and eval-uate architectural results from this view, including operations which refer to model correspondence rules.

**implementation methods** are the means by which to design and build systemsusing this view.

Another approach to categorizing operations is from Finkelstein et al. [[2](#page23)]. The work plan for a viewpoint defines 4 kinds of actions (on the view representations): assembly actions which contains the actions available to the developer to build a

12

specification; check actions which contains the actions available to the developer to check the consistency of the specification; viewpoint actions which create new view-points as development proceeds; guide actions which provide the developer with guidance on what to do and when.

**3.7** **Correspondence rules**

? Document any correspondence rules defined by this viewpoint or its model kinds.

Usually, these rules will be across models or across views since, constraints within a model kind will have been specified as part of the conventions of that model kind.

See: ISO/IEC/IEEE 42010, 4.2.6 and 5.7

**3.8** **Examples** (**optional)**

Provide helpful examples of use of the viewpoint for the reader (architects and other stakeholders).

**3.9** **Notes** (**optional)**

Provide any additional information that users of the viewpoint may need or find helpful.

1. **Sources**

? Identify sources for this architecture viewpoint, if any, including author, history, bibliographic references, prior art, per ISO/IEC/IEEE 42010, 7e.

13

**Chapter 4**

**Views+**

Much of the material in an AD is presented through its architecture views. Each view follows the conventions of its governing viewpoint. A view is made up of archi-tecture models.

? Include an architecture view for each viewpoint selected in x[3.](#page10)

Repeat and complete the following section for each architecture view in the AD.

**4.1** **View:** <**View Name**>

* Give the architecture view a <View Name>.
* Provide any identifying and supplementary information about <View Name>.

The details of this information will be as specified by the organization and/or project. See x[1](#page5) for examples of identifying and supplementary information.

Views have their own identifying and supplementary information distinct from ADs because they may be developed and evolve separately over the lifetime of a project.

? Identify the viewpoint governing this view from among those identified in x[3.](#page10)

See also: ISO/IEC/IEEE 42010, 5.5

1. **Models+**

An architecture view is composed of one or more architecture models.

? Provide one or more architecture models adhering to the governing viewpoint.

14

? The models must address all of the concerns framed by the view’s governing view-point and cover the whole system from that viewpoint.

Repeat the section below for each model.

1. <**Model Name**>

* Each architecture model shall include version identification as specified by the organization and/or project.
* Each architecture model shall identify its governing model kind and adhere to the conventions of that model kind from x[3.5.](#page13)

See ISO/IEC/IEEE 42010, 5.4.

An architecture model may be a part of more than one architecture view. This enables sharing of details and addressing distinct but related concerns without redundancy. Other uses of multiple models: aspect-oriented style of architecture description: architecture models shared across architecture views can be used to express architectural perspectives [[7](#page23)] and architecture textures [[6](#page23)]. Architecture models can be used as containers for applying architecture patterns or architec-ture styles to express fundamental schemes (such as layers, three-tier, peer-to-peer, model-view-controller) within architecture views.

1. **Known Issues with View**

? Document any discrepancies between the view and its viewpoint conventions. Each architecture view must adhere to the conventions of its governing architec-ture viewpoint.

Known issues could include: inconsistencies, items to be completed, open or unre-solved issues, exceptions and deviations from the conventions established by the viewpoint. Open issues can lead to decisions to be made. Exceptions and deviations can be documented as decision outcomes and rationale.

15

**Chapter 5**

**Consistency and correspondences**

This chapter describes consistency requirements, recording of known inconsisten-cies in an AD, and the use and documentation of correspondences and correspon-dence rules.

**5.1** **Known inconsistencies**

? Record any known inconsistencies in the AD.

Although consistent ADs obviously are to be preferred, it is sometimes infeasible or impractical to resolve all inconsistencies for reasons of time, effort, or insufficient information.

2 An architecture description should include an analysis of consistency of its archi-tecture models and its views.

**5.2** **Correspondences in the AD**

? Identify each correspondence in the AD and its participating AD elements. Iden-tify any correspondence rules governing

Correspondences are used to express, record, enforce and analyze consistency be-tween models, views and other AD elements within an architecture description, between ADs, or between an AD and other forms of documentation.

AD elements include instances of stakeholders, concerns, viewpoints and views,

16

model kinds and models, decisions and rationales. Constructs introduced by view-points and model kinds are also AD elements.

Correspondences are n-ary mathematical relations. Correspondences can be de-picted via tables, via links, or via other forms of association (such as in UML).

**5.3** **Correspondence rules**

? Identify each correspondence rule applying to the AD.

Correspondence rules can be introduced by the AD, by one of its viewpoints, or from an architecture framework or architecture description language being used.

? For each identified correspondence rule, record whether the rule holds (is satisfied) or otherwise record all known violations.

17

**Appendix A**

**Architecture decisions and rationale**

It is not required by the Standard to capture architecture decisions. This section describes recommendations (“shoulds”) for their recording.

**A.1** **Decisions**

2 Provide evidence of consideration of alternatives and the rationale for the choices made.

2 Record architecture decisions considered to be key to the architecture of <System of Interest>.

Areas to consider to selecting key decisions include those:

affecting key stakeholders or many stakeholders essential to project planning and management expensive to enforce or implement

highly sensitive to changes or costly to change involving intricate or non-obvious reasoning

pertaining to architecturally significant requirements requiring major expenditures of time or effort to make resulting in capital expenditures or indirect costs

2 When recording decisions, the following information items should be considered:

unique identifier for the decision

18

statement of the decision

correspondences or linkages concerns to which it pertains owner of the decision

correspondences or linkages to affected AD elements rationale linked to the decision

forces and constraints on the decision assumptions influencing the decision

considered alternatives and their potential consequences

See [[3](#page23)] and references there for various approaches to documenting decisions com-patible with the Standard.

The template ends here!

19

**Bibliography**

Clements, Paul C. et al. Documenting Software Architectures: views and beyond. 2nd. Addison Wesley, 2010.

Finkelstein, A. et al. “Viewpoints: a framework for integrating multiple per-spectives in system development”. In: International Journal of Software Engineering and Knowledge Engineering 2.1 (Mar. 1992), pp. 31–57.

Heesch, Uwe van, Paris Avgeriou, and Rich Hilliard. “A Documentation Framework for Architecture Decisions”. In: The Journal of Systems & Software 85.4 (Apr. 2012), pp. 795–820. DOI: [10.1016/j.jss.2011.](http://dx.doi.org/10.1016/j.jss.2011.10.017) [10.017](http://dx.doi.org/10.1016/j.jss.2011.10.017).

IEEE Std 1471, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. Oct. 2000.

ISO/IEC/IEEE 42010, Systems and software engineering — Architecture description. Dec. 2011, pp. 1–46.

Ran, Alexander. “ARES Conceptual Framework for Software Architecture”. In: Software Architecture for Product Families Principles and Practice. Ed. by M. Jazayeri, A. Ran, and F. van der Linden. Addison-Wesley, 2000, pp. 1–29.

Rozanski, Nick and Eoin´ Woods. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. 2nd. Addison Wes-ley, 2011.

20

**Contents**

|  |  |
| --- | --- |
| [**Using the template**](#page2) | **i** |
| [License](#page2) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | i |
| [Version History](#page4) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 1 |
| [Template editions](#page4) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 1 |
| [Comments](#page4) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| [**1 Introduction**](#page5) | | | **2** |
| [1.1](#page5) | [Identifying information](#page5) . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 2 |
| [1.2](#page5) | [Supplementary information](#page5) . . . . . . . . . . . . . . . . . . . . . . . . | | 2 |
| [1.3](#page6) | [Other information](#page6) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 3 |
|  | [1.3.1](#page6) | [Overview (optional)](#page6) . . . . . . . . . . . . . . . . . . . . . . . . . | 3 |
|  | [1.3.2](#page7) | [Architecture evaluations](#page7) . . . . . . . . . . . . . . . . . . . . . . | 4 |
|  | [1.3.3 Rationale for key decisions](#page7) . . . . . . . . . . . . . . . . . . . . . | | 4 |

|  |  |  |
| --- | --- | --- |
| [**2 Stakeholders and concerns**](#page8) | | **5** |
| [2.1](#page8) | [Stakeholders](#page8) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [2.2](#page9) | [Concerns](#page9) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 6 |
| [2.3](#page9) | [Concern–Stakeholder Traceability](#page9) . . . . . . . . . . . . . . . . . . . . . | 6 |

|  |  |  |
| --- | --- | --- |
| [**3 Viewpoints+**](#page10) | | **7** |
| [3.1](#page11) | [<Viewpoint Name>](#page11) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 8 |
| [3.2](#page11) | [Overview](#page11) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 8 |

21

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | [3.3](#page11) | [Concerns and stakeholders](#page11) . . . . . . . . . . . . . . . . . . . . . . . . . | | 8 |
|  |  | [3.3.1](#page11) | [Concerns](#page11) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 8 |
|  |  | [3.3.2](#page12) | [Typical stakeholders](#page12) . . . . . . . . . . . . . . . . . . . . . . . . . | 9 |
|  |  | [3.3.3 “Anti-concerns” (optional)](#page13) . . . . . . . . . . . . . . . . . . . . . . | | 10 |
|  | [3.4](#page13) | [Model kinds+](#page13) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 10 |
|  | [3.5](#page13) | [<Model Kind Name>](#page13) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 10 |
|  |  | [3.5.1 <Model Kind Name> conventions](#page13) . . . . . . . . . . . . . . . . . | | 10 |
|  |  |  | [I) Model kind languages or notations (optional)](#page14) . . . . . . . . . | 11 |
|  |  |  | [II) Model kind metamodel (optional)](#page14) . . . . . . . . . . . . . . . | 11 |
|  |  |  | [III) Model kind templates (optional)](#page15) . . . . . . . . . . . . . . . . | 12 |
|  |  | [3.5.2 <Model Kind Name> operations (optional)](#page15) . . . . . . . . . . . . | | 12 |
|  |  | [3.5.3 <Model Kind Name> correspondence rules](#page15) . . . . . . . . . . . | | 12 |
|  | [3.6](#page15) | [Operations on views](#page15) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 12 |
|  | [3.7](#page16) | [Correspondence rules](#page16) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 13 |
|  | [3.8](#page16) | [Examples (optional)](#page16) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 13 |
|  | [3.9](#page16) | [Notes (optional)](#page16) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 13 |
|  | [3.10](#page16) | [Sources](#page16) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 13 |
| [**4**](#page17) | [**Views+**](#page17) | |  | **14** |
|  | [4.1](#page17) | [View: <View Name>](#page17) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 14 |
|  |  | [4.1.1](#page17) | [Models+](#page17) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 14 |
|  |  | [4.1.2 <Model-Name>](#page18) . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 15 |
|  |  | [4.1.3 Known Issues with View](#page18) . . . . . . . . . . . . . . . . . . . . . . | | 15 |
| [**5**](#page19) | [**Consistency and correspondences**](#page19) | | | **16** |
|  | [5.1](#page19) | [Known inconsistencies](#page19) . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 16 |
|  | [5.2](#page19) | [Correspondences in the AD](#page19) . . . . . . . . . . . . . . . . . . . . . . . . . | | 16 |
|  | [5.3](#page20) | [Correspondence rules](#page20) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 17 |
| [**A**](#page21) | [**Architecture decisions and rationale**](#page21) | | | **18** |

22

[A.1 Decisions](#page21) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

23