Architecture description template for
Borrow My Books

Architecture Description of *Express JS, Client-server and asynchronous calls* For *Borrow My Books*

"Bare bones" edition
version: 2.3

Template prepared by:
Rich Hilliard
r.hilliard@computer.org

License

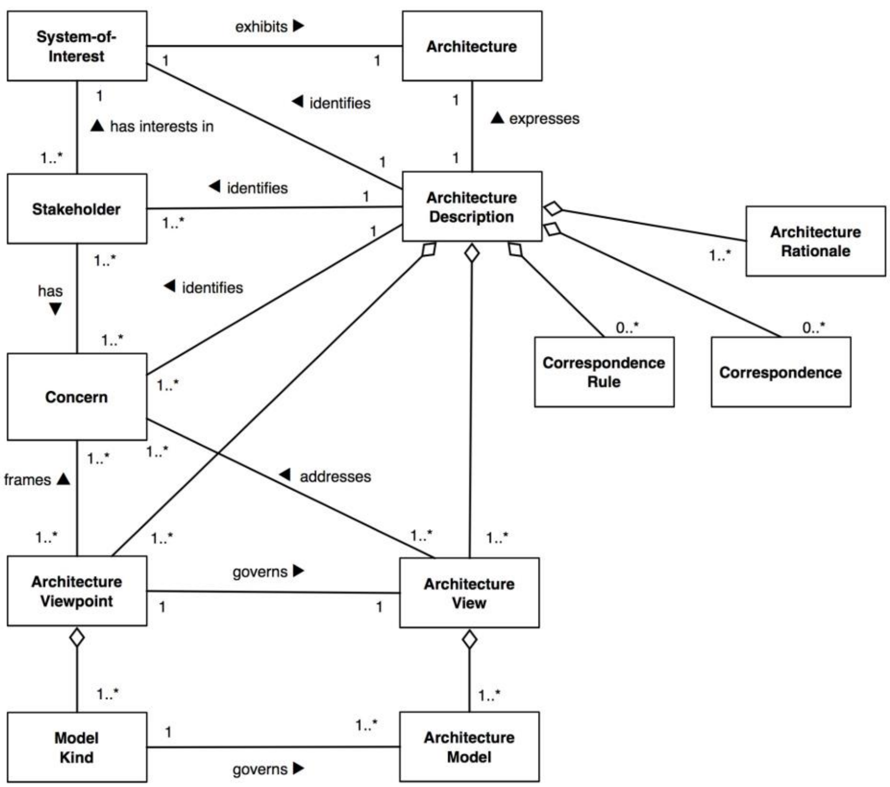

Figure 1: Content model of an architecture description

The latest version is always available at:
http://www.iso-architecture.org/42010/templates/.

Chapter 1

Introduction

This chapter describes introductory information items of the AD, including identifying and supplementary information.

1.1 Identifying information

? Identify the architecture being expressed, such as via an <Architecture Name>, or as appropriate.

? Identify the <System of Interest> for which this is an architecture description.

The Software Architecture chosen for the development of the BorrowMyBooks system is a hybrid one, consisting of a Model View Controller (MVC) type of system called ExpressJS for the implementation of relationships between user interfaces and underlying data models, and a Client Server in order to manage the network aspects of the system.

1.2 Supplementary information

Massive Dynamic (MD) proposes a system which will help students trade, borrow and sell second hand books – mainly textbooks. The system's main objective is to allow users to easily search for, locate and purchase the books through it. It will have gamification aspects like Uber where users can rate other users. Additionally, it will include a reporting function in order to allow reporting of abusive users. There will be an administrative profile/s in the system, to aid for the maintenance of the system. The technology we have chosen to implement this on is a web based technology, namely the MEAN stack. The system will not cover using online payment methods and will only facilitate the locating and transferal of physical goods.

1.3 Other information

This system makes use of web 2.0 technologies. We are using JavaScript processed by the V8 engine through NodeJS. We make heavy use of the callback code pattern. The system is based on a single process model which is event driven. These event occur mainly with user input and/or asynchronous calls. Other code patterns we have used when they are necessary are both the yield/await directives and also the async.waterfall model. (Found here: http://caolan.github.io/async/) These other patterns have been used to reduce the amount of nesting and callbacks used in the code. They are not anti-patterns but they are not the standard that comes with our use of the database connection driver (mongoose) and ExpressJS.

The live version of the site is hosted on Heroku (https://www.**heroku**.com/). The URL is https://borrowmybooks.herokuapp.com/. This site provides hosting for the server side code and public assets. The MongoDb database is stored on another site who is affiliated with Heroku called mLab (https://www.mlab.com/).

1.3.1 Architecture evaluations

? Include results from any evaluations of the <Architecture Name> being documented.

1.3.2 Rationale for key decisions

? An architecture description shall include rationale for each decision considered to be a key architecture decision (per ISO/IEC/IEEE 42010, 5.8.2).

See §A for further guidance about decisions and rationale.

Chapter 2

Stakeholders and concerns

2.1 Stakeholders

There are 2 main stakeholders which take part in the functioning of the system. These are namely, the Administrator and the user.

However, the user role has 2 perspectives based on the purpose of the system.

One user is the Advertiser, who uploads books for sale or for auction. The other perspective of the user role is that of searching for books to purchase or bid on.

2.2 Concerns

The purpose of the BorrowMyBooks system is to create an easy to use platform where people can trade their textbooks. The architecture we have chosen to apply to this system is suitable due to the fact that the system will be developed as a Web application, where the separation of concerns (as supported by the MVC architecture) and the isolation of application logic from the user interface is necessary.

Furthermore, the Client-server network architecture will be suitable for the functioning of the BorrowMyBooks system as the system is mainly based on a large database of data relating to users, administrators and books listed for trading. This architecture will aid in the handling of the database and the communications of each user with it.

The feasibility of the system is dependent on the demand which students have for second hand textbooks. At tertiary-education level, the demand is high due to the fact that brand new textbooks are expensive, and in some cases, textbooks are only used for 6 months, after which they become useless unless they are sold. This system will therefore make it easier for students to trade their textbook with others which is currently a cumbersome task. The deployment of the system is feasible.

There are few risk factors involved for the stakeholders of the system. The system will include password protection for the users, namely password salting, which will minimize the risk of user accounts being hacked into.

Furthermore, users will not be requested to submit financial data, such as credit card numbers and bank details. No transaction handling will be done through the system, it is solely developed for the location and transferal of textbooks between pupils.

The maintenance of the system will be carried out by the main developers, who will make sure that the system runs fluidly, any bugs are eliminated and that database communications are fast and reliable.

Lastly, the evolution of the system will depend on the future requirements of its users.

2.3 Concern–Stakeholder Traceability

? Associate each identified concern from §2.2 with the identified stakeholders from §2.1 having that concern.

This association can be recorded via a simple table or other depiction.

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

	Stakeholder 1	Stakeholder 2	Stakeholder 3	. . .
Concern 1	–	x	x	. . .
Concern 2	x	–	x	. . .
Concern 3	–	x	x	. . .
. . .				

Chapter 3

Viewpoints+

An AD contains multiple architecture views; each view adheres to the conventions of an *architecture viewpoint*. This chapter describes the requirements on documenting viewpoints for an AD.

? Include a specification for each architecture viewpoint used in this AD.

? Viewpoints must be chosen for the AD such that each identified concern from §2.2 is framed by at least one viewpoint.

? Provide a rationale for each viewpoint used.

Rationale could include discussion in terms of its stakeholders, the concerns framed by the viewpoint, relevance of its model kinds and modeling conventions.

? Each architecture viewpoint used in the AD must be specified in accordance with the provisions of ISO/IEC/IEEE 42010, 7.

A detailed template for specifying viewpoints in accordance with the Standard is included in §3.1 below.

NOTE: *The latest version of the viewpoint template can be found at http://www.iso-architecture.org/42010/templates/.*

Repeat and fill-in viewpoint template as needed for each viewpoint used in the AD.

An AD contains one or more architecture views and an architecture viewpoint definition for each view. There is no required ordering of the views or viewpoints within an AD. Readers of the AD will need to refer to the viewpoint specifications to understand the subject of a view, its notations, models and the modeling conventions used. Given a set of views (v_i) and their viewpoints (VP_i), the architect might consider the following possible arrangements:

- Viewpoints, first: VP_i, followed by the views: v_i
- Interleaved views with their viewpoints: v_i, VP_i, v_j, VP_j, . . .
- Views up front: v_i with the viewpoints deferred to appendices, VP_i

3.1 <Viewpoint Name>

? Provide the name for the viewpoint.

If there are any synonyms or other common names by which this viewpoint is known or used, record them here.

3.2 Overview

Provide an abstract or brief overview of the viewpoint.

Describe the viewpoint's key features.

3.3 Concerns and stakeholders

Architects looking for an architecture viewpoint suitable for their purposes often use the identified concerns and typical stakeholders to guide them in their search. Therefore it is important (and required by the Standard) to document the concerns and stakeholders for which a viewpoint is intended.

3.3.1 Concerns

? Provide a listing of architecture-relevant concerns to be framed by this architecture viewpoint per ISO/IEC/IEEE 42010, 7a.

Describe each concern.

Concerns name "areas of interest" in a system.

NOTE: *Following ISO/IEC/IEEE 42010,* ***system*** *is a shorthand for any number of things including man-made systems, software products and services, and softwareintensive systems such as "individual applications, systems in the* *traditional sense, subsystems, systems of systems, product lines, product families, whole enterprises, and other aggregations of interest".*

Concerns may be very general (e.g., *Reliability*) or quite specific (*e.g., How does the system handle network latency?*).

Concerns identified in this section are critical information for an architect because they help her decide when this viewpoint will be useful.

When used in an architecture description, the viewpoint becomes a "contract" between the architect and stakeholders that these concerns will be addressed in the view resulting from this viewpoint.

It can be helpful to express concerns *in the form of questions* that views resulting from that viewpoint will be able to answer. E.g.,

- *How does the system manage faults?*
- *What services does the system provide?*

NOTE: *"In the form of a question" is inspired by the television quiz show, Jeopardy!*

ISO/IEC/IEEE 42010, 5.3 contains a candidate list of concerns that must be considered when producing an architecture description. These can be considered here for their relevance to the viewpoint being specified:

- What are the purpose(s) of the system-of-interest?
- What is the suitability of the architecture for achieving the system-of-interest's purpose(s)?
- How feasible is it to construct and deploy the system-of-interest?
- What are the potential risks and impacts of the system-of-interest to its stakeholders throughout its life cycle?
- How is the system-of-interest to be maintained and evolved?

See also: ISO/IEC/IEEE 42010, 4.2.3.

3.3.2 Typical stakeholders

? Provide a listing of the typical stakeholders of a system who are in the potential audience for views of this kind, per ISO/IEC/IEEE 42010, 7b.

Typical stakeholders would include those likely to read such views and/or those who need to use the results of this view for another task.

Stakeholders to consider include:

- users of a system;

- operators of a system;
- acquirers of a system;
- owners of a system;
- suppliers of a system;
- developers of a system;
- builders of a system;
- maintainers of a system.

3.3.3 "Anti-concerns" (optional)

It may be helpful to architects and stakeholders to document the kinds of issues for which this viewpoint is *not appropriate or not particularly useful*.

Identifying the "anti-concerns" of a given notation or approach may be a good antidote for certain overly used models and notations.

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, conventions 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 specified.

3.5 <Model Kind Name>

? Identify the model kind.

3.5.1 <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:

I) by reference to an existing notation or language (such as SADT, UML or an architecture 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 operations 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 metamodels (such as UML class diagrams, OWL, eCore). The metamodel should present:

entities What are the major sorts of conceptual elements that are present in models 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 relationships and 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], each viewtype (which is similar to a viewpoint) 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 of correspondence rules. When defining an architecture framework, it may be helpful 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.

3.5.2 <Model Kind Name> operations (optional)

Specify operations defined on models of this kind.

See §3.6 for further guidance.

3.5.3 <Model Kind Name> correspondence rules

? Document any correspondence rules associated with the model kind.

See §3.7 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 this viewpoint. 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 interpreting architecture views and their models.

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

implementation methods are the means by which to design and build systems using this view.

Another approach to categorizing operations is from Finkelstein et al. [2]. 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 specification; *check actions* which contains the actions available to the developer to check the consistency of the specification; *viewpoint actions* which create new viewpoints 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.

3.10 Sources

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

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 architecture models.

? Include an architecture view for each viewpoint selected in §3.

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 §1 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 §3.

See also: ISO/IEC/IEEE 42010, 5.5

4.1.1 Models+

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

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

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

Repeat the section below for each model.

4.1.2 <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 §3.5.

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] and architecture textures [6]. Architecture models can be used as containers for applying architecture patterns or architecture styles to express fundamental schemes (such as layers, three-tier, peer-to-peer, model-view-controller) within architecture views.

4.1.3 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 architecture viewpoint.

Known issues could include: inconsistencies, items to be completed, open or unresolved 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.

Chapter 5

Consistency and correspondences

This chapter describes consistency requirements, recording of known inconsistencies in an AD, and the use and documentation of correspondences and correspondence 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 architecture models and its views.

5.2 Correspondences in the AD

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

Correspondences are used to express, record, enforce and analyze consistency between 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, model kinds and models, decisions and rationales. Constructs introduced by viewpoints and model kinds are also AD elements.

Correspondences are n-ary mathematical relations. Correspondences can be depicted 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.

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
- 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] and references there for various approaches to documenting decisions compatible with the Standard.

The template ends here!

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 perspectives 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.2011.10.017.

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 Wesley, 2011.

Contents