Service Oriented Software Framework for Rapid Application Development

Rubasinghe N.

158773F

Faculty of Information Technology

University of Moratuwa

June 2018

Service Oriented Software Framework for Rapid Application Development

Rubasinghe N

158750H

Dissertation submitted to the Faculty of Information Technology, University of Moratuwa, Sri Lanka for the fulfilment of the requirements of Degree of Master of Science in Information Technology

June 2018

Declaration

We declare that this thesis is our own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Nishantha Rubasinghe ……………………………

Date: 2018-06-27 Signature of Student

Supervised by

Mr Chaman Wijesiriwardana
Senior Lecturer
Faculty of Information Technology
University of Moratuwa

…………………………………..

Date: 2018-06-27 Signature of Supervisor

Acknowledgements

I would like to express my gratitude to my supervisor Mr Chaman Wijesiriwardana, Senior Lecturer at the University of Moratuwa, Sri Lanka whose expertise. understanding. And patience added considerably to my research experience. I appreciate his vast knowledge and skill in many areas and his assistance in writing reports.

I would like to thank the other lecturers of University of Moratuwa, Sri Lanka. especially, Prof. Asoka Karunananda for the knowledge and assistance they provided at all levels of the research project.

I would also like to thank all the batch mates of the M.Sc. in IT degree program who gave their valuable feedback to improve the results of the research and my family for the support they provided me through my entire life and. I must acknowledge my wife Shashikala, without whose love, encouragement and editing assistance I would not have finished this thesis.

Abstract

This paper proposes a service-oriented rapid application development framework for application developments. It provides nine commonly use business patterns, the application under development with representations for the data model, business logic, API layer, user interfaces. Which are presented together with a diagram showing the precedence for checking the convenience of their utilization? The paper describes the layers, techniques, code generation, API testing tool and explores possible future extensions.

Many aspects of software development projects can be improved with an information architecture that provides a framework to automate the design, development, testing processes using pre-defined patterns and automate processes to ensure that the process is completed on time, within budget, and reliably with quality assurance. In this paper, we address a critical aspect of the time-consuming gaps that exists between the process of software design, implementation, and testing. The proposed pattern-based software rapid development framework is intended to fill these gaps and provide a more integrated approach in the development of a software system. We introduce the capability to orchestrate new integration technology that leverages recent advances in Web Services, including service-oriented architecture. This new approach can lead to significant improvement in software productivity and quality by providing a platform that bridges the gap between stakeholders and software engineers, as well as providing a reliable integration mechanism using pattern-based service-oriented approach to manage rapid changes in domain knowledge.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 **Prolegomena**

Today, business moves faster, and clients tend to change their minds more frequently over the course of a project’s development life cycle. Of course, they expect the development team to adapt to their needs and modify the structure of an application quickly. Software development process which minimizes the pre-planning phase, and results in more rapid software development lifecycle. RAD is a methodology for compressing the analysis, design, build, and test phases into a series of short, iterative development cycles. The idea behind this methodology is to start developing as early as possible so that clients can review a working prototype and offer additional direction.

we proposed pattern-based service-oriented framework for rapid development. which cover not only the development, design, code generation and API testing. such a rapid application development framework can improve developer productivity and improve the quality, reliability, and robustness of new software. Developer productivity is improved by allowing developers to focus on the unique requirements of their application instead of spending time on application infrastructure

1.2 Background and motivation

Every single CEO of any IT company wants to build software in best quality and faster. Time is the most expensive and valuable resource and quality is the most important factor of success. It is obvious that skills improve development speed. More skilled developers solve problems faster and create less complex solutions. Some say there can be 10x productivity difference between extremely skilled and less skilled developers. The next trivial question is what can be done to increase developers’ skills? First Option is, you can hire only skilled developers. That might work, but this model is not easily scalable. Skilled people tend to work on hard problems that demand their skills.

The software process model such as RAD consists of a set of activities undertaken to design, develop, testing and maintain software systems. A variety of software process models have been designed to structure, describe and prescribe the software development process. The software process models play a very important role in software development, so it forms the core of the software product. Software project failure is often devastating to an organization. Schedule slips, buggy releases, and missing features can mean the end of the project or even financial ruin for a company. Oddly, there is disagreement over what it means for a project to fail. In this paper, the discussion is done on current process models and analysis on a failure of software development, which shows the need for new research.

1.3 Problem Definition

There is a strong need for a new system development approach for application development with accompanying framework that reduces design, development testing time and software model reuse.

We understand that in traditional software development cycle most of the developers using previously used or common business patterns. in the current industrial scene, there is no service-oriented rapid application development framework with integrated patterns between data models, business models, service layer, UI templates and test automation

1.4 Aim & Objectives

1.4.1 Aim

The aim of this project is to developing service-oriented application development framework which can increase productivity in the major phases software development lifecycle. Especially in design, implementation, and testing

1.4.2 Objectives

- **Identify common business patterns and templates** for each category of development (includes five key elements based on software system's business patterns: database layer, business logic layer, API layer, UI templates)
- **Build a development framework with latest technologies** and higher development standards including the capability of customizing functionalities, Code maintainability, add new features, modify existing features, improve performance and readymade component collection such as role management and permission management

- **Build Integrated Code generator** for above 2 steps. this is the most important part of the research. even after identifying patterns & building templates for each pattern, developers may have to spend some more time to implement and integrate combined patterns, so our aim is to reduce development time as much as possible. Additionally, we can minimize testing/bug fixing by automated code generation and reducing human involvements for well-known and identified patterns
- **Build an automated test tool** for APIs Testing. this also a one of the important part of the research. the idea is reduced time for each level of development life cycle mainly in Design, Development, Testing

1.5 Summary

This chapter gave an overview of the proposed solution for rapid development. A brief introduction was given about rapid development methodology and this chapter provides a discussion of some of the background and motivation for this study. We defined the problem definition and aim and objectives. Next chapter shows the current developments and approaches of rapid development frameworks.

Chapter 2

Rapid Development Approaches

2.1 Introduction

Over the last 30 years, the information technology industry has experienced constant and rapid advances in computing devices, software products and technologies, and changing requirements as consumers find new ways to use this technology. industry's transition to distributed service-oriented applications that run on multiple computing platforms, such as laptops, handheld PDAs, and smartphones. Subsequently, advances in language paradigms, software design practices, and programming environments have been driven by the need to support reuse, adaptability, and the management of complexity within these software applications. These software characteristics are critical to support higher levels of agility, productivity, quality, maintenance, and evolution.

Business demand for new applications is increasing, but IT's ability to deliver them is not according to latest researches. turnaround expected by the business for custom development projects is shrinking. Rapid application development is quickly becoming a necessity.

Rapid application development frameworks prioritize speed and agility so that IT teams can increase their productivity and improve project outcomes. Instead of the typical turnaround of months or years for new applications, rapid application development enables, IT teams to deliver in a matter of days or weeks. improving the portability of the user interface across multiple platforms and improve scalability should be one of the main key features of feature RAD [1] frameworks

2.2 Rapid application development

Rapid Application Development (RAD) [1] is a condensed development process that produces a high-quality system with low investment costs. RAD process allows software developers to quickly adjust to shifting requirements in a fast-paced and constantly changing market. The ability to quickly adjust is what allows such a low investment cost. The Rapid Application Development method is divided into four phases: Requirements Planning, User Design, Construction, and Cutover. The user design and construction phases are repeated until the user approves that all the requirements are met.

RAD is most effective for projects with a well-defined business objective and a clearly defined user group, but which are not computationally complex. It is especially useful if the project is of small to medium size and time sensitive. However, it requires a stable team composition with highly skilled developers and users who are deeply knowledgeable about the application area. Deep knowledge is essential when working on a condensed development timeline that requires approval after each construction phase. If you don't have these requirements, RAD may not work well for your organization.

RAD takes advantage of automated tools and techniques to restructure the process of building information systems. This new process, extrapolated to the entire IS organization, results in a profound transformation of information systems development. RAD replaces hand-design and coding processes, which are dependent upon the skills of isolated individuals, with automated design and coding, which is an inherently more stable process. RAD may thus give an IS organization its first real basis for continuous improvement. In addition to being more stable, Rapid Application Development is a more capable process, as it is much faster and less error-prone than hand coding.

The success of Rapid Application Development is contingent upon the involvement of people with the right skills and talents. Excellent tools are essential to fast application development, but they do not, by themselves, guarantee success. Fast development relies equally heavily on the people involved. These people must thus be carefully selected, highly trained, and highly motivated. They must be able to use the tools and work together in close-knit teams. Rapid development usually allows each person involved to play several different roles, so a RAD project mandates a great degree of a cooperative effort among a relatively small group of people

2.3 Software Development Challenges

To have a successful software project, it is essential to identify what constitutes success. Projects succeed when enough factors go well to allow a project's objectives to be satisfied. Project success and failure can rarely be described in absolute terms. If failure is no accident, then success is no accident. Failure and success provide different perspectives on improvement: failure tells what not to do in future, whereas success shows what should be done again.

Using a structured systems development methodology is one of the critical success factors in a systems development project. You must set a realistic timetable for the completion of a systems project and expect that some deadlines will slip as unexpected setbacks inevitably arise. The data model is the core of the system. Without a carefully constructed data model, any system is doomed to failure.

2.4 Software Patterns

Some years ago Alexander Christopher noted that a pattern “describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice” [10] On the other hand, Gamma et all defined design patterns as "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context" [11] From these definitions it is understandable that patterns have two important parts: a problem and a solution, Where the solution can be reused several times in different problem domains

The book Design Patterns: Elements of Reusable Object-Oriented Software [11] launched an avalanche of best practices and share knowledge in the world of software engineering and put the design patterns in the centre of software design and development in a very practical way It is one of the most important pattern languages in the software industry. A design pattern describes a commonly-recurring structure of communicating components that solve a general design problem in a particular context.

Architectural patterns [12] express fundamental structural organization schemas for software systems. They provide a set of predefined subsystems, specify their responsibilities, and include rules and guidelines for organizing the relationships between them.

Cruz A, M [13], identified the most common use case patterns found on use case models of data-oriented applications. These patterns are well - recognized from the different types of relationships (e.g., master, master-detail or master-reference relationships) established from the entities defined in a domain model level and, also from the common CRUD operations found on these entities.

2.5 Latest RAD Frameworks & Approaches

Rapid application development focuses on four major components: tools, people, methodology, and management. Current, powerful computing technology is essential to support such tools as application generators, UI(screen/form) frameworks & generators, report generators, pattern languages, relational or object-oriented database tools, automated testing tools and CASE tools. People include users and the development team. The methodology stresses prototyping and joint application design. In this section, we are discussing few frameworks and approaches

Bootstrap [2] is a good example of UI framework, rather than coding from scratch, Bootstrap enables you to utilize readymade blocks of code to help you get started. Combine that with cross-browser compatibility and CSS-Less functionality, many hours of coding can be saved. Pre-styled Components: Some basic components are needed by all the web pages and we need not write it for all the web applications. For example Sidebars, Navigation bars, Tabs etc. All these styled and readymade for design noobs like us who barely use CSS. Bootstrap is the most popular framework for UI, but its only for UI development such as (HTML, CSS, and Client-Side Components such as jQuery plugins and Charts est.)

COTS Approach

Commercial off-the-shelf (COTS) [14] is a very common name for most of the organizations nowadays. There are different types of COTS products, but here COTS software is referred to as COTS. It is pre-built software. Here different components are bought from different vendors and then integrating them we get the final product. Its reusability property facilitates us with the variety of features with shrinking budget and less time duration. The only successful way for a commercial off-the-shelf (COTS) implementation to be successful is to decide at the outset that you are going to re-engineer your business to fit the limitations of the COTS package. These packages usually imply a specific method of doing business with a corresponding set of capabilities. A managerial decision must be made that whatever the package does is good enough and that customization is not an option.

2.6 Code Generations

Code generation is an important part of today’s software development. Using code generation can increase code quality, ease maintenance and shorten development time. It can be used for development of different parts of software systems like database access layers, business logic, API layer, user interface and many others. code generators may be ready to use products or developed in-house for project’s specific requirements. There are different tools and environments for the development of code generators.

Eclipse modelling framework (EMF) is a Java framework and code generation facility for building applications and tools, based on structured models. EMF started as an Implementation of MOF specification, but now it can be thought of as a highly efficient Java implementation of a core subset of the MOF API. The meta-model definition or meta meta-model in EMF is called Ecore

2.6 API Testing Tools

Several approaches to simplify testing RESTful APIs exist. commonly used tools like JUnit1, NUnit and other xUnit frameworks aim at unit-testing. Their tight-coupling with the implementation language of the subject under test makes it difficult to use them for testing web-services. Therefore, many teams are searching for techniques to improve testing such services. One approach is SoapUI, a tool which can configure test cases for web-services by using a Service-Oriented-Architecture. The fact that it can mainly be configured via its graphical interface makes it difficult to enable an automated test case generation. Thus, it did not fulfil our requirements. Apache JMeter [15] is an “open-source testing tool” developed by “Apache Software Foundation (ASF)”. JMeter’s main function is to load test client/server. Moreover, JMeter is used in regression testing by generating test scripts.

Even though there are several good REST API Testing Frameworks available in the market today, they may not always suit your application, may need more configurations for each API, sequential testing and model testing as I explained in approach chapter

[3.5.4]. since we are proposing code generator for our framework we can automate most of the configuration with our own testing tool

2.7 Summary

This chapter gave an overview of the rapid application development and A brief introduction was given about software development challenges, software patterns, latest RAD frameworks & approaches, code generation tools and API testing tools. Next chapter shows our approach for proposed rapid development framework of rapid development frameworks.

Chapter 3

Approach

3.1 Introduction

Chapter 2 presented the rapid development approaches for software developments. This chapter presents our approach to developing pattern-based service-oriented rapid application development framework with industry’s most popular standards and technologies

3.2 Hypothesis

Increasing productivity, reducing cost and improving quality, the major challenges facing software development organizations can be summarized as more, better, and faster by using pattern-based service oriented rapid application development framework.

3.5 Major Phases of The Proposed Framework Development

From the framework development perspective, 6 different phases can be highlighted

1) Identifying Common Business Patterns
2) Develop Data Model
3) Develop API Model
4) Develop UI Model
5) Code Generation Module
6) Automate API testing Module

3.5.1 Identifying Common Business Patterns

Our main objective of this project was identifying set of integrated patterns that apply to data models, business models, API models and UI models. Based on those patterns we proposed a service-oriented rapid application development framework to improve productivity and quality in all design, development and testing phases. In chapter 4 [4.4.1] we explain all patterns with pattern details

3.5.2 Code Generation Module

this section we will describe our approach for code generation of the proposed framework. Source code generation is one of the main assets of the proposed framework if you wish to increase developer productivity.

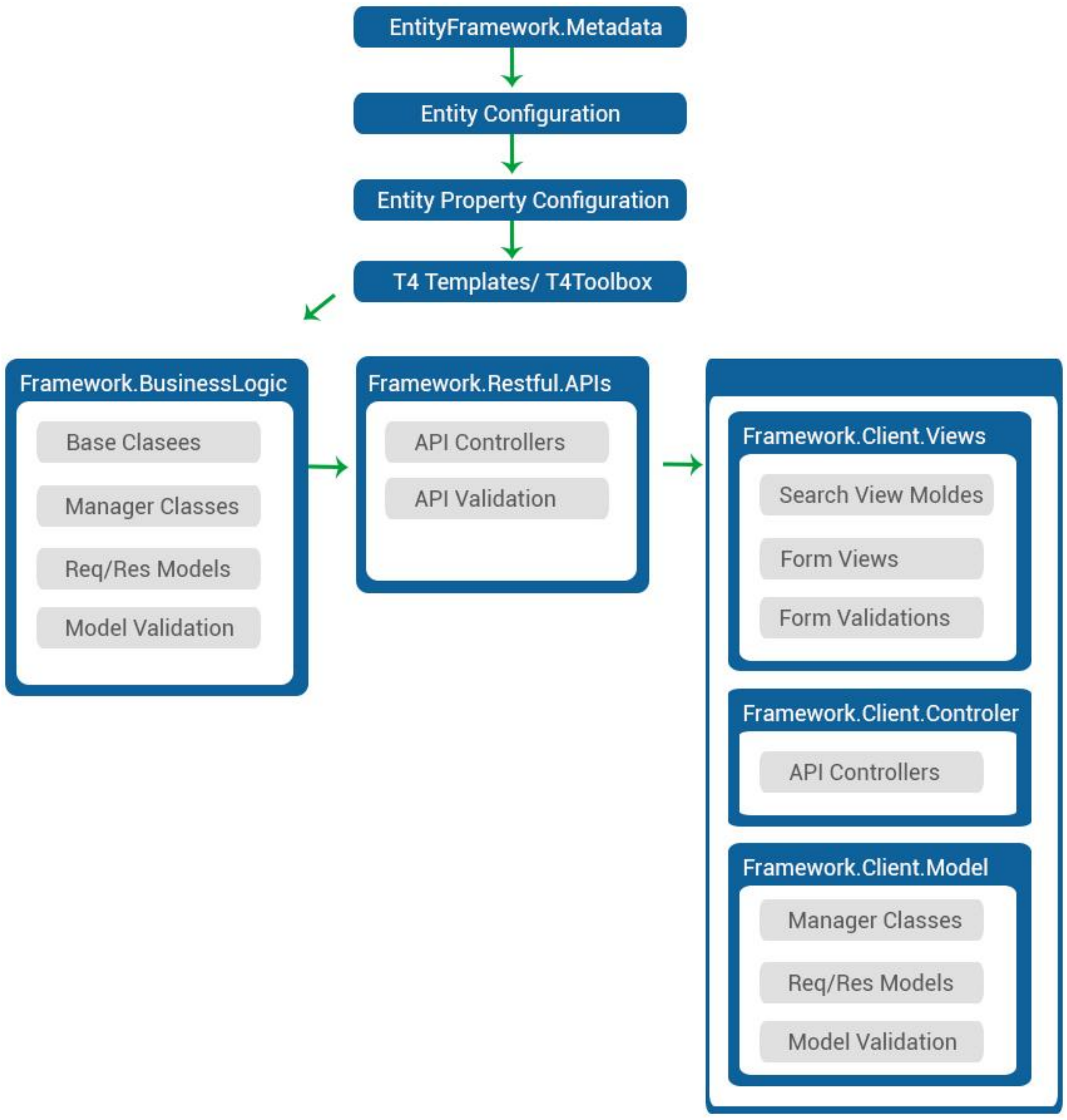


Figure 4.5.1: Workflow of the code generation

this code generator populating code based on characteristics and behaviors of pre-identified 9 patterns explained in [chapter 4.4]. purpose of this code generator is deliver code which is consistent with the pre-decided set of functionalities. Additionally, these generators can generate complex API and interface components including forms, filter and several other advanced navigation features. The generated code is comparable to the one written by the senior developers. since identified base code patterns and functionalities written by senior developers. such a code generator create code and then have nothing to do with the project. in other hand developers can focus on the unique requirements of their application instead of spending time on application infrastructure

Our code generators are capable of:

1. Proper standard code base with infrastructure
2. Providing code generation any specific data model using identified patterns
3. Generating source code native to the target environment or platform (Exposing Business Logic as REST API)
4. Providing easily modifiable and well-organized source code
5. Preserving customizations and modifications during subsequent application regenerations

3.5.3 Automate API testing Module

From the test implementation perspective, different design patterns can be understood in automation module

1. Single API Tests

   Filtering Options will be available to select APIs from Existing Restful API list. once you selected an API, input section will be automatically populated. Ex: request JSON, API input parameters

2. API Sequence Tests

   Apart from testing of each API, the sequence of APIs to be called also a challenge and may end up with inadequate coverage. For example, consider the sample cashflow APIs as below:

1) Create User
2) Create Account
3) Create Sample Transaction
4) List Transactions by account

Apart from testing of each API, there can various sequences of flow can be made from it. The output of an API can be used as an input to another API. All the sequence flow should be covered to deliver a bug-free application.

3. Model Tests

   This is also some kind of sequence testing, but the purpose of this type automatically tests the model after generating source code

   For example, consider the item model APIs as below:

   1) Create Item
   2) Create Update Item
   3) Create Delete Item
   4) Read Item
   5) Search Items
   6) Advanced Search Items

   The idea is automating testing process for all the generated APIs

4.6 Advantages & Limitations

This section describes advantages and limitations of proposed framework

4.6.1 Advantages

Business Logic as Restful API

the REST protocol totally separates the user interface from the server and the data storage. This has some advantages when making developments. For example, it improves the portability of the interface to other types of platforms, it increases the scalability of the projects and allows the different components of the developments to be evolved independently.

The REST API always adapts to the type of syntax or platforms being used, which gives considerable freedom when changing or testing new environments within the development. With a REST API, you can have PHP, Java, Python or Node.js servers. The only thing is that it is indispensable that the responses to the requests should always take place in the language used for the information exchange, normally XML or JSON.

Pre-Build Components

Another major advantage of the proposed framework is providing a comprehensive set of pre-built components that are commonly used in enterprise web applications. It offers a solid foundation with components such as:

- Notification components
- Security Components
- User Management & Permission Components

Technical Skills

Another significant advantage is that the developers don't need to become technical architecture experts: They can leverage the expertise of the enterprise's technical architects. Similarly, it can help consultants come up to speed more quickly. And it can also be used as a requirement in consulting contracts to ensure compliance with enterprise standards.

Design Time

Designing data model, UI model, test model is much easier with our proposed identified 9 patterns. Your application consists of one or more entity models and you have to one or more of patterns to generate an entity model.

Code Generator

Using Code generation module to write code is a major advantage of the proposed framework, otherwise, must be written by a person. This offers many benefits including saving developers' time, ensuring consistency and guaranteeing accuracy. By removing the repetitive basic code creation chores, your developers are free to work on more customization and logical tasks that require their individual skills. I should have mentioned that Generating code that much closer to hand-written quality with good industry standard was my major intention. in that way developers can easily customize generated source code

Testing Time

Test automation saves testing time and resources. Test tools can execute tests faster than a person can, and in most cases, they can do so in an unattended mode. So, test automation should reduce test cycle time or the number of testers needed.

4.6.1 Limitations

1. Designed for data-oriented application development
2. Designed for **relational database** data model, this concept cannot be applied to No SQL or any other data structure
3. UI code generation is only focused on backend development, admin, and applications (not for front-end websites and apps)

4.7 Summary

This chapter gave an overview of our approach proposed solution and provides a discussion of each module of proposed solution such as identifying common business patterns, develop data model, develop API model, develop UI model, code generation module and automate API testing module. We defined the problem definition and the hypothesis for this thesis.

Chapter 4

Design & Implementation

4.1 Introduction

Chapter 3 presented the approaches for major components of the proposed rapid development framework. This chapter presents details of designing & implementation of each component, our approach is to complete development framework with industry’s most popular standards and technologies

Technologies are constantly evolving and as developers, we need to cope up with what's the latest or at least popular nowadays. As developers, we might find having a hard time catching up with latest technologies because it will give you more confusion as to what sets of technologies to use and where to start. We know that there are tons of resources out there that you can use as a reference to learn but you still find it hard to connect the dots in the picture. Sometimes you might think of losing the interest to learn and give up. When we are thinking about framework development, we should mainly focus on selecting wildly using technologies

4.2 Over roll framework design

In this paper, we propose a development framework, code generation, and API test automation tool for rapid application development. The architecture of the development framework is illustrated in Figure 5.3.1. As shown, the system builds the service layer (Restful APIs) [3] for the presentation layer. After that, the code is generated from the models with the real devices and the new service is registered. Users can then pick up a registered service and deploy the generated code in an execution engine. The engine provides an execution environment where a service can be loaded and scheduled to run.

4.3 Top Level Architecture

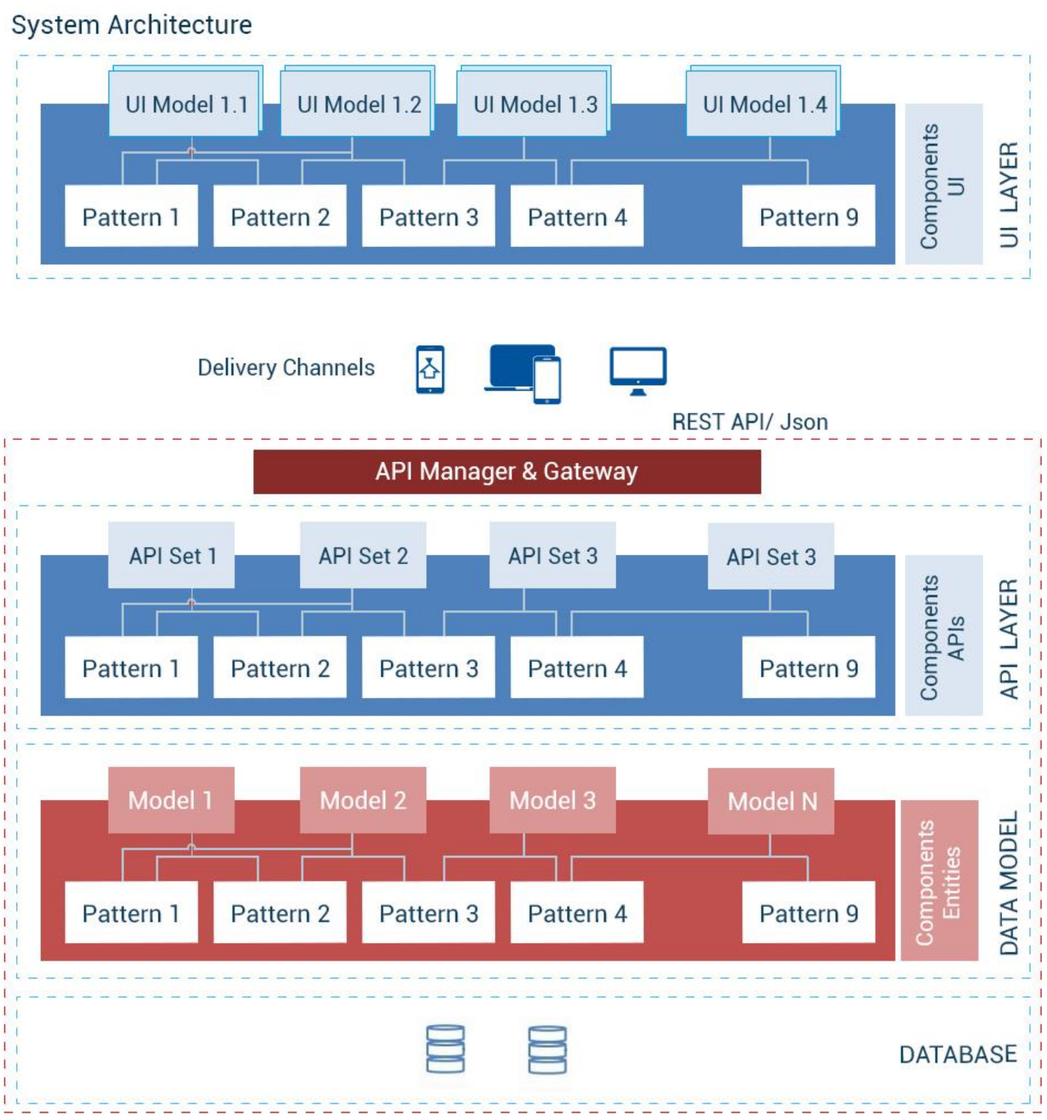


Figure 4.3.1: System Architecture

4.4 Analyzing & identifying patterns

From this section, we will provide a set of predefined subsystems, specify their characteristics, responsibilities, and include rules and guidelines for recognizing each pattern.

4.4.1 Pattern 1 – Simple Master Pattern

Context: Brand names in phone shop inventory, country list in e-commerce portal are samples for this pattern. generally, this pattern is using for categories or types of resources and settings.

Forces: independent entity**,** simple entity type without foreign keys

Problem: How can we manage categories of products or parties in a simplest and reusable way?

Example: Brand Names (Toyota, Nissan, Audi) in Auto Classified site

Solution:

Data Model:

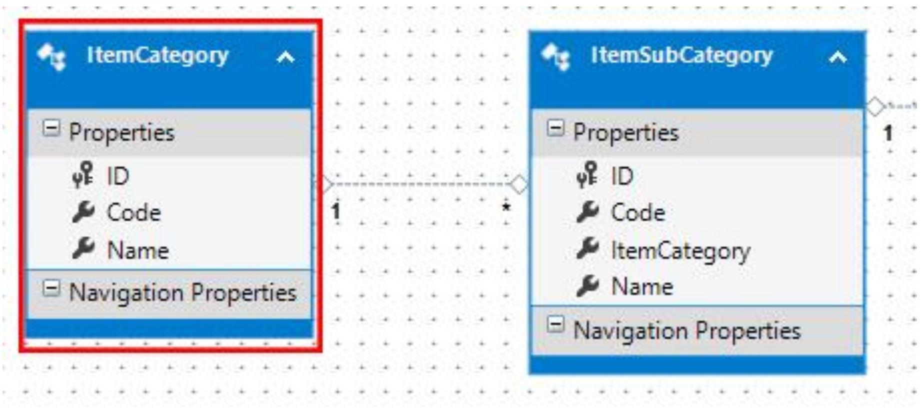


Figure 4.4.1.1: Data Model of Pattern 1

API Structure:

```
GET: list, search, advanced search, and info
- API Template
[
  {
    "ID": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
    "Code": 2085,
    "Name": "Name 1"
  },
  {
    "ID": "23c61fd3-f2f5-4e94-87a6-0071304fd832",
    "Code": 1053,
    "Name": "Name 2"
  }, ...
]
```

```
POST: new
- API Template
{
  "Name": "Item Category 1",
  ...
}
```

```
POST: update
- API Template
{
   "ID":"13c61fd3-f2f5-4e94-87a6-0071304fd832",
   "Name": "Item Category 1",
   ...
}
```

UI:

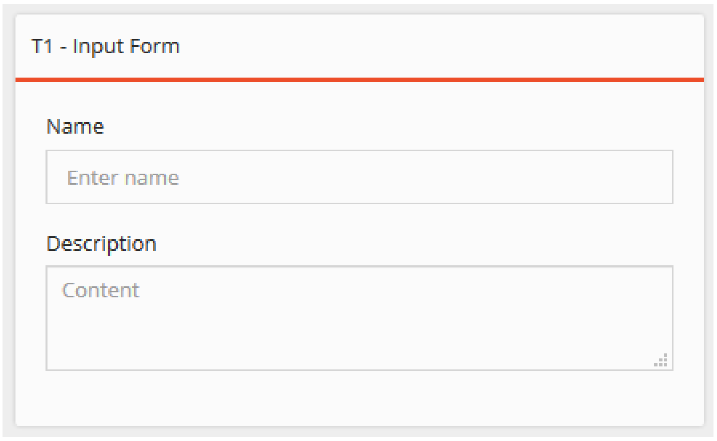


Figure 4.4.1.2: Basic UI Input form of Pattern 1

4.4.2 Pattern 2 – Association Pattern

Context: models in phone shop inventory, city list in e-commerce portal are samples for this pattern. Generally, this pattern is using for referencing with another entity.

Forces: Dependent entity, at least one attribute which is a foreign key

Problem: How can we manage a many-to-one relationship entity data in a simplest and reusable way? in this pattern, primary entity is a child entity

Example: Modes (Prius, Allion, Premio) and BrandNames (Toyota, Nissan, Audi) in Auto Classified site

Solution:

Data Model:

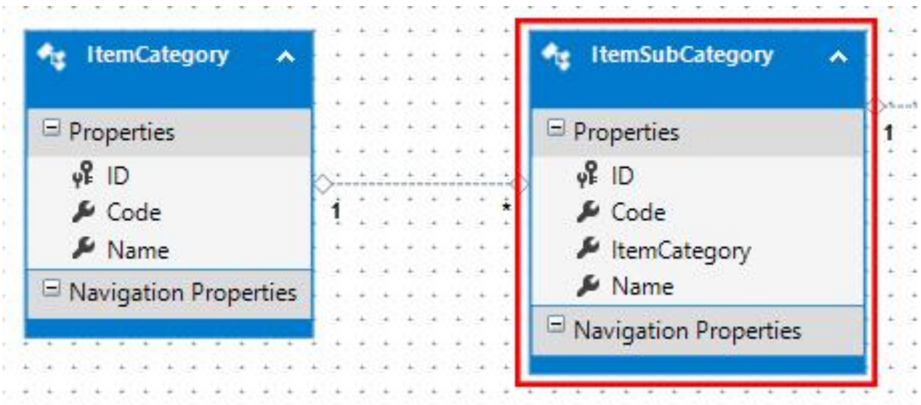


Figure 4.4.2.1: Data Model of Pattern 2

API Structure:

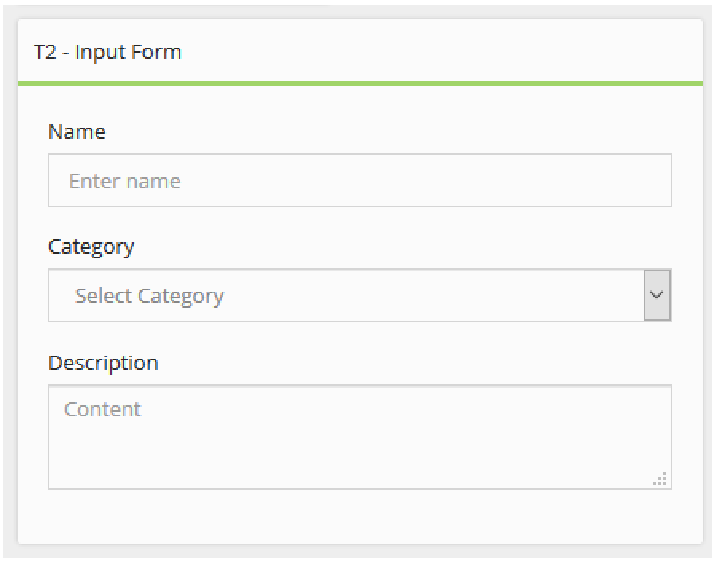
```
GET: list, search, advanced search, and info
- API Template
[
  {
    "ID": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
    "Code": 2085,
    "Name": "Name 1",
    "ItemCategory": {
      "ID": "d189a497-e7a8-4c13-ac88-149690bde176",
      "Code": 1,
      "Name": "ItemCategory 1"
    }
  }, ...
]
```

```
POST: new
- API Template
{
  "Name": "Item Sub Category 1",
  "ItemCategory": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
  ...
}


POST: update
- API Template
{
   "ID":"abc61fd3-f2f5-4e94-87a6-0071304fd832",
   "Name":"Item Sub Category 1",
   "ItemCategory": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
   ...
}
```

Pattern-specific APIs:

```
GET: advanced search – new filtering option for category


GET: get list by category (GetListByCaregory)
- API Template
same as list json sample
```

UI:


T2 - Input Form

Name

Enter name

Category

Select Category

Description

Content

Figure 4.4.2.2: Basic UI Input form of Pattern 2

5.4.3 Pattern 3 - One to One Pattern

Context: Employee – employee contact info in a payroll, customer – customer membership info in an e-commerce site. Generally, this pattern is using keep partial details of an entity.

Forces: a) secondary entity primary key same as the primary entity

b) may need to update secondary data separately

c) secondary entity data retrieval may be optional for some cases

Problem: How can we manage a one-to-one relationship entity data in a simplest and reusable way?

Example: Customer - customer financial information in an account management system

Solution:

Data Model:

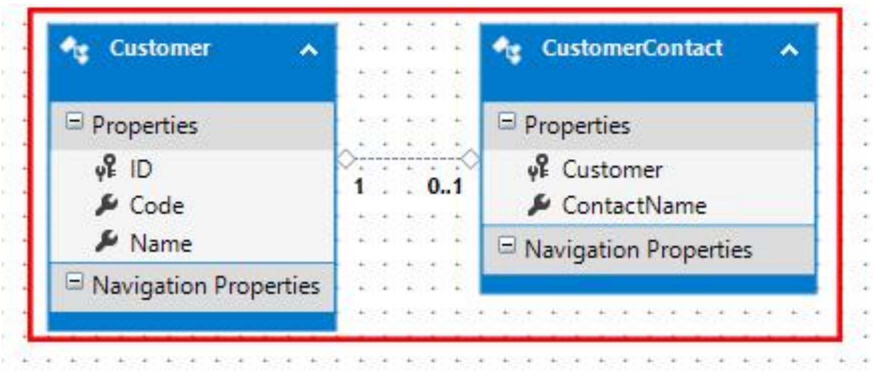


Figure 4.4.3.1: Data Model of Pattern 3

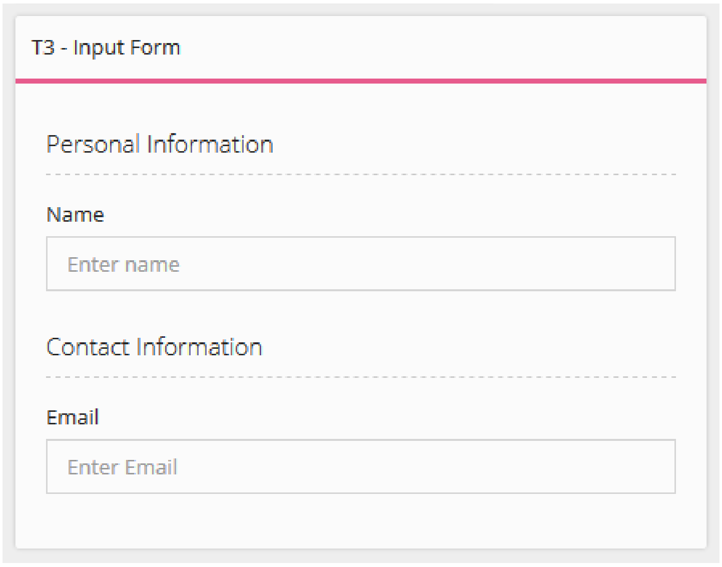
API Structure:

```
GET: list, search, advanced-search & info
- API Template
[
  {
    "ID": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
    "Code": 2085,
    "Name": "Customer 1 ",
    "ContactInfo": {
      "Email": "test@test.com",
      ...
    }
  }, ...
]

POST: new
- API Template
{
  "Name": "Customer 1",
  "ContactInfo": {
    "Email": "test@test.com",
    ...
  }, ...
}

POST: update
- API Template
{
  "ID": "abc61fd3-f2f5-4e94-87a6-0071304fd832",
  "Name": "Customer 1",
  "ContactInfo": {
    "Email": "test@test.com",
    ...
  }, ...
}
```

Pattern specific APIs:

```
POST: update secondary data (UpdateContactInfo)
- API Template
{
  "ID": "abc61fd3-f2f5-4e94-87a6-0071304fd832",
  "ContactInfo": {
    "Email": "test@test.com",
    ...
  }
}

POST: read secondary data (GetContactInfo)
- API Template
{
    "ID": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
    "ContactInfo": {
      "Email": "test@test.com",
          ...
        }
}
```

UI:


T3 - Input Form

Personal Information

Name

Enter name

Contact Information

Email

Enter Email

Figure 4.4.3.2: Basic UI Input form of Pattern 3

4.4.4 Pattern 4 – Model Divide Pattern

Context: Employee – employee contact info in a payroll, customer – customer membership info in an e-commerce site. Generally, this pattern is using keep partial details in the same primary entity.

Forces: a) no secondary entity, keep partial data in the same primary entity

b) may need to update secondary data separately

c) secondary entity data retrieval may be optional for some cases

c) data-model changed but others same as pattern 3

Problem: How can we manage a simple entity data with separation in a simplest and reusable way?

Solution:

Data Model:

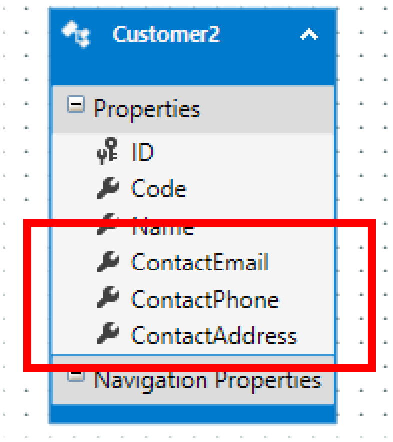


Figure 5.4.4.1: Sample Data Model of Pattern 4

API Structure:

```
GET: list, search, advanced-search & info
- API Template
[
  {
    "ID": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
    "Code": 2085,
    "Name": "Customer 1 ",
    "ContactInfo": {
      "Email": "test@test.com",
      ...
    }
  }, ...
]
```

```
POST: new
```

```
- API Template
{
  "Name": "Customer 1",
  "ContactInfo": {
    "Email": "test@test.com",
    ...
  }, ...
}
```

```
POST: update
```

```
- API Template
{
  "ID": "abc61fd3-f2f5-4e94-87a6-0071304fd832",
  "Name": "Customer 1",
  "ContactInfo": {
    "Email": "test@test.com",
    ...
  }, ...
}
```

Pattern specific APIs:

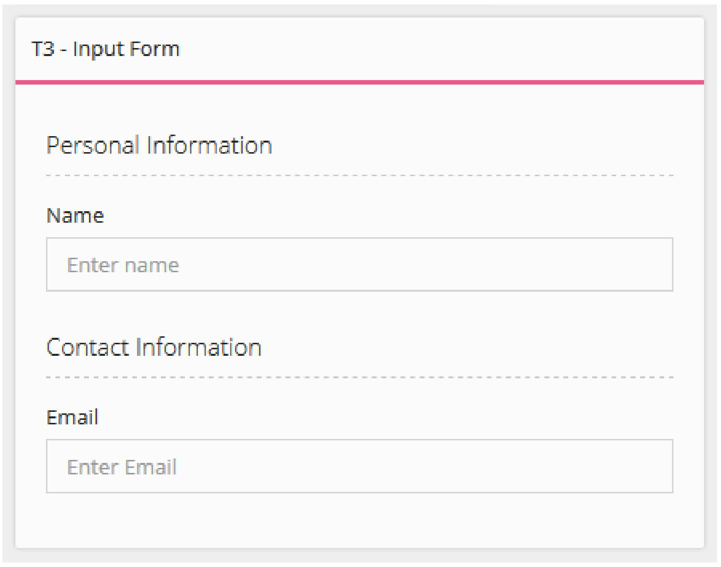
```
POST: update secondary data (UpdateContactInfo)
```

```
- API Template
{
  "ID": "abc61fd3-f2f5-4e94-87a6-0071304fd832",
  "ContactInfo": {
    "Email": "test@test.com",
    ...
  }
}
```

```
POST: read secondary data (GetContactInfo)
```

```
- API Template
{
    "ID": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
    "ContactInfo": {
        "Email": "test@test.com",
        ...
    }
}
```

UI:


T3 - Input Form

Personal Information

Name

Enter name

Contact Information

Email

Enter Email

Figure 4.4.4.2: Basic UI Input form of Pattern 4

4.4.5 Pattern 5 - Inheritance Pattern

Context: Ad – Auto Ad, Property Ad in a classified site, Employee – Manager in an HRM system. Generally, this pattern is using inheritance entity types.

Forces:

a) a one-to-one relationship between primary entity & secondary entities
b) both base entity and derived entity (secondary entities) functionalities should be implemented

Problem: How can we manage inheritance entity data in a simplest and reusable way?

Solution:

Data Model:

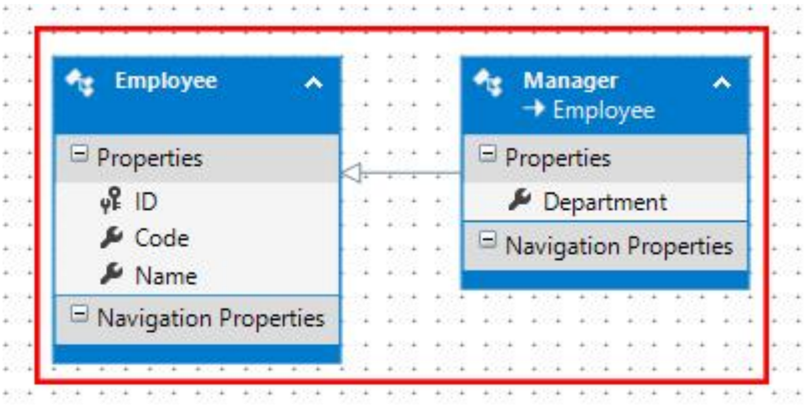


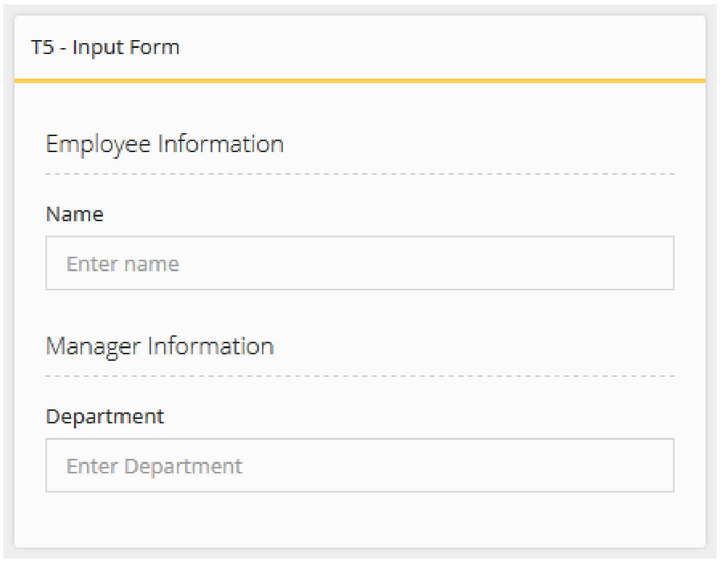
Figure 4.4.5.1: Data Model of Pattern 5

API Structure:

```
GET: list, search, advanced search, and info

- API Template
[
  {
    "ID": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
    "Code": 2085,
    "Name": "Manager 1 ",
    "Department": "HR",
    ...
  }, ...
]
```

```
POST: new
- API Template
{
  "Name": "Manager 1",
  "Department": "HR",
  ...
}

POST: update
- API Template
{
  "ID": "abc61fd3-f2f5-4e94-87a6-0071304fd832",
  "Name": "Manager 1",
  "Department": "HR",
  ...
}
```

UI:


T5 - Input Form

Employee Information

Name

Enter name

Manager Information

Department

Enter Department

Figure 4.4.5.2: Basic UI Input form of Pattern 5

5.4.6 Pattern 6 - Associated Collection Pattern

Context: Ad- Image list in a classified site, this pattern is using secondary entity for keeping a collection of items with reference to the primary entity.

Forces: a) secondary entity is a week entity

b) small number of items attached to primary item

c) secondary has only one dependent (primary entity)

d) most of the time, lazy loading enabled

Problem: How can we manage entity which has a collection of independent items in a simplest and reusable way?

Example:

Advertisement and Image list of the advertisement in a classified site

Solution:

Data Model:

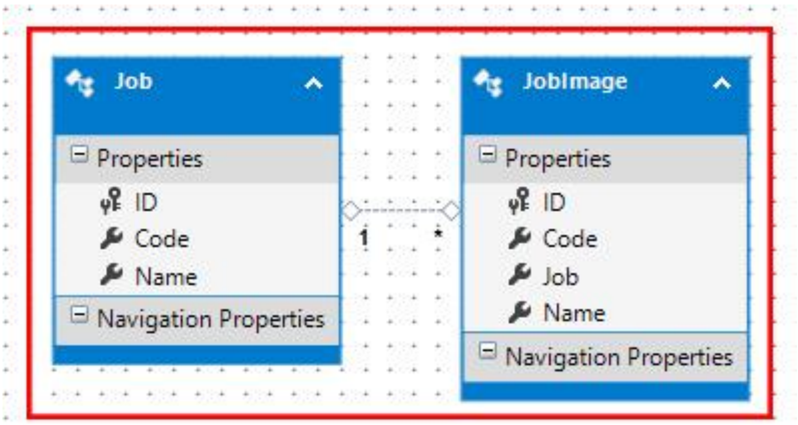


Figure 4.4.6.1: Data Model of Pattern 6

API Structure:

```
GET: list, search, advanced search, and info
- API Template
[
  {
    "ID": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
    "Code": 2085,
    "Name": "Job 1 ",
    "JobImages": [
      {
        "ID": "a6c1cbf3-8bc2-4011-9491-10f9702a9221",
        "Name": "Image 1"
      }, ...
    ]
  }, ...
]
```

```
POST: new
- API Template
{
  "Name": "Job 1",
  "JobImages": [
    {
      "Name": "Image 1"
    }, ...
  ]
}
```

```
POST: update
- API Template
{
  "ID": "abc61fd3-f2f5-4e94-87a6-0071304fd832",
  "Name": "Job 1",
  "JobImages": [
    {
      "ID": "ergb1fd3-f2f5-4e94-87a6-0071304fd832",
      "Name": "Image 1"
    }, ...
  ]
}
```

Pattern specific APIs:

```
POST: update secondary data (UpdateImageList)
- API Template
{
  "ID": "abc61fd3-f2f5-4e94-87a6-0071304fd832",
  "JobImages": [
    {
      "ID": "ergb1fd3-f2f5-4e94-87a6-0071304fd832",
      "Name": "Image 1", ...
    }, ...
  ]
}
```

```
POST: read secondary data for selected record (GetImageList)
{
    "ID": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
    "JobImages": [
      {
        "ID": "a6c1cbf3-8bc2-4011-9491-10f9702a9221",
        "Name": "Image 1"
      }, ...
    ]
 }
```

UI:

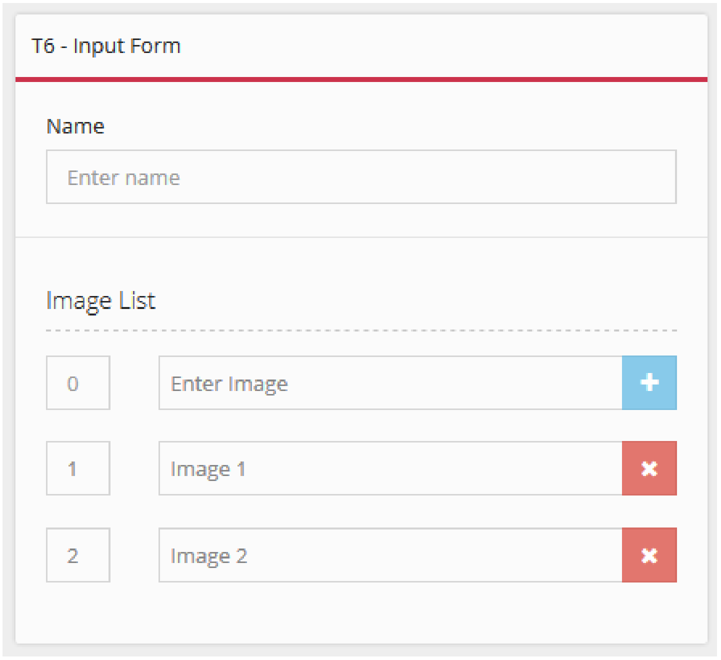
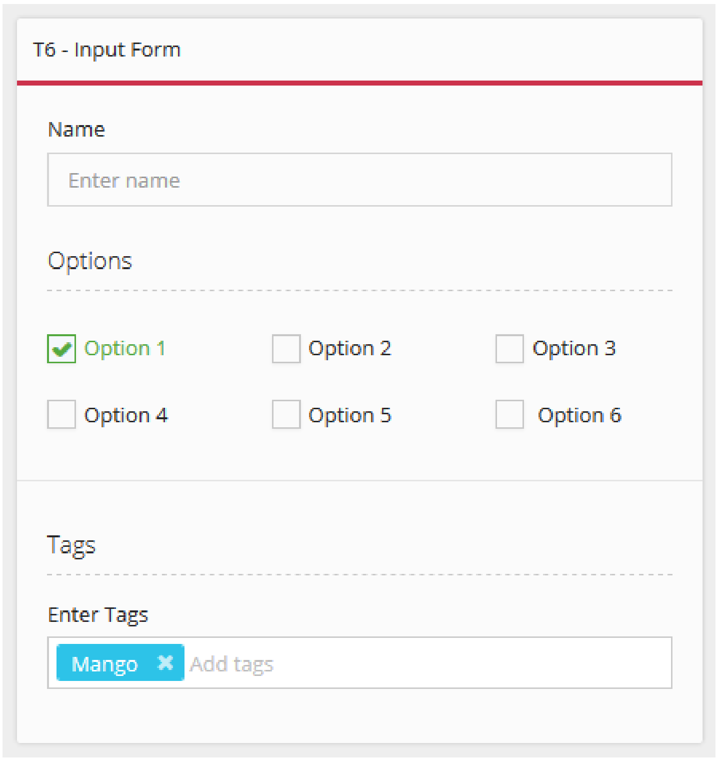


Figure 4.4.6.2: Basic UI Input form of Pattern 6


T6 - Input Form

Name

Enter name

Options

[✓ Option 1] Option 2 Option 3

Option 4 Option 5 Option 6

Tags

Enter Tags

Mango ✖ Add tags

Figure 4.4.6.3: simplest UI Input form of Pattern 6

4.4.7 Pattern 7 - Simple Composite Pattern

Context: Student- Subjects in a school management system, select admins from existing user list in CMS. this pattern is using composite entity for keeping a collection of items with reference to the primary entity.

Forces: a) using a composite relationship to keep secondary item collection

b) small number of items attached with primary item

c) most of the time, lazy loading enabled

Problem: How can we manage a many-to-many relationship entity data in a simplest and reusable way?

Example:

Student- Subjects in a school management system

Solution:

Data Model:

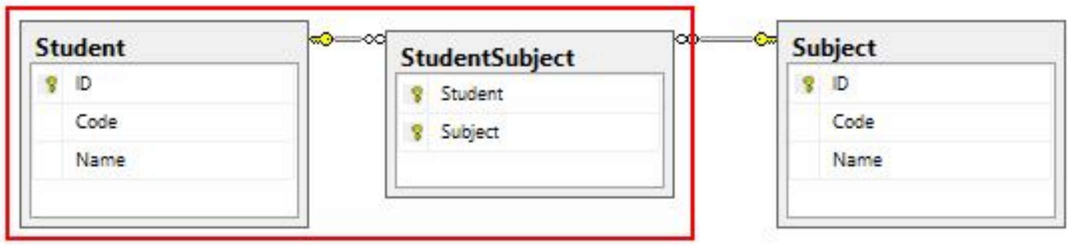
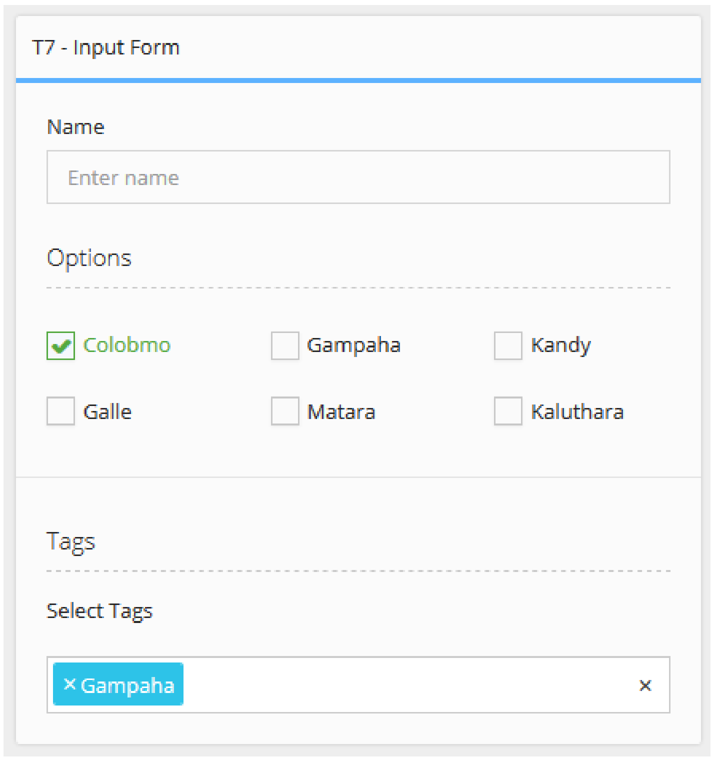


Figure 4.4.7.1: Data Model of Pattern 7

API Structure:

```
GET: list, search, advanced search, and info
- API Template
[
  {
    "ID": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
    "Code": 2085,
    "Name": "Student 1 ",
    "Subjects": [
      {
        "ID": "a6c1cbf3-8bc2-4011-9491-10f9702a9221",
        "Name": "Subject 1"
        ...
      }, ...
    ]
  }, ...
]
POST: new
- API Template
{
  "Name": "Student 1",
  "Subjects": [
    {
      "Subject 1",
      "Subject 12", ...
  ], ...
}

POST: update
- API Template
{
  "ID": "abc61fd3-f2f5-4e94-87a6-0071304fd832",
  "Name": "Student 1",
  "Subjects": [
    {
      "Subject 1",
      "Subject 12", ...
    }, ...
  ]
}
```

UI:


T7 - Input Form

Name

Enter name

Options

[✓ Colobmo] Gampaha Kandy

Galle Matara Kaluthara

Tags

Select Tags

× Gampaha ×

Figure 4.4.7.2: Basic UI Input form of Pattern 7

4.4.8 Pattern 8 - Composite Pattern

Context: Set Branches including branch's additional info like (phone), based on City entity in a financial application. this pattern is using composite entity for keeping a collection of items with reference to the primary entity.

Forces: a) using a composite relationship to keep secondary item collection

b) small number of items attached with primary item

c) most of the time, lazy loading enabled

Problem: How can we manage a many-to-many relationship entity with additional data in a simplest and reusable way?

Example:

Student- Subjects in a school management system

Solution:

Data Model:

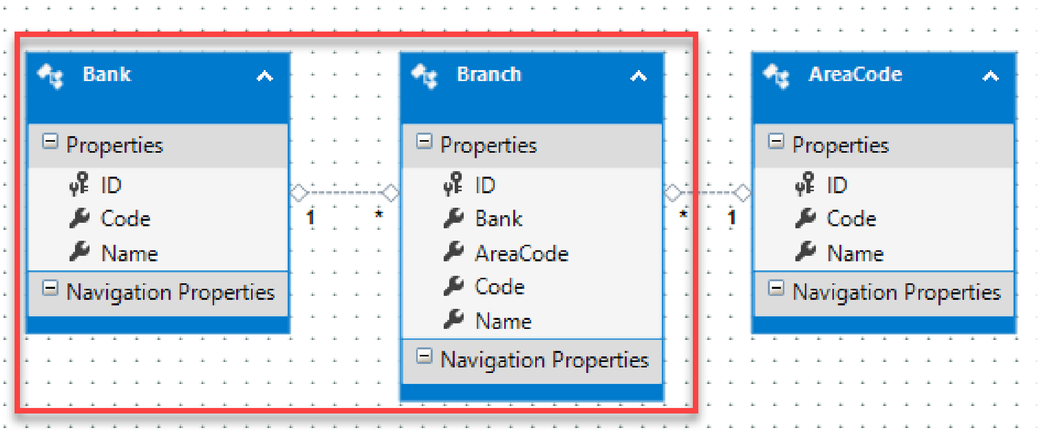


Figure 4.4.8.1: Data Model of Pattern 8

API Structure:

```
GET: list, search, advanced search, and info

- API Template
[
  {
    "ID": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
    "Code": 2085,
    "Name": "Sampath Bank",
    "Branches": [
      {
        "ID": "a6c1cbf3-8bc2-4011-9491-10f9702a9221",
        "Name": "Kandy Branch"
      }, ...
    ]
  }, ...
]
```

```
POST: new
- API Template
{
  "Name": "Sampath bank",
  "Branches": [
    {
        "Name": "Kandy Branch"
    }, ...
  ]
}
```

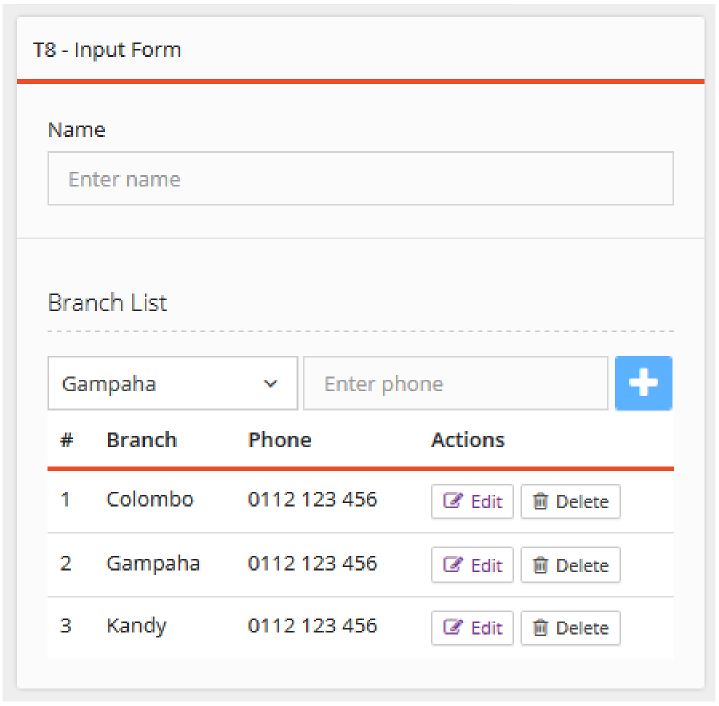
```
POST: update
- API Template
{
  "ID": "abc61fd3-f2f5-4e94-87a6-0071304fd832",
  "Name": " Sampath bank ",
  "Branches": [
    {
      "ID": "ergb1fd3-f2f5-4e94-87a6-0071304fd831",
      "Name": "Kandy Branch"
    }, ...
  ]
}
```

Pattern specific APIs:

```
POST: update secondary data (UpdateBranchList)
- API Template
{
  "ID": "abc61fd3-f2f5-4e94-87a6-0071304fd832",
  "Branches": [
    {
      "ID": "ergb1fd3-f2f5-4e94-87a6-0071304fd831",
      "Name": "Kandy Branch"
    }, ...
  ]
}
```

```
POST: read secondary data for the selected record (GetBranchList)
{
    "ID": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
    "Branches": [
      {
        "ID": "a6c1cbf3-8bc2-4011-9491-10f9702a9221",
        "Name": "Kandy Branch"
      }, ...
    ]
}
```

UI:



T8 - Input Form

Name

Enter name

Branch List

Gampaha | Enter phone

#	Branch	Phone	Actions
1	Colombo	0112 123 456	Edit Delete
2	Gampaha	0112 123 456	Edit Delete
3	Kandy	0112 123 456	Edit Delete

Figure 4.4.8.2: Basic UI Input form of Pattern 8

4.4.9 Pattern 9 - Independent Association Pattern

Context: Article- Comment list in a CMS, this pattern is using secondary entity for keeping a collection of items with reference to the primary entity.

Forces: a) secondary entity is a week entity

b) can manage secondary entity data independently

b) a large number of items attached with primary item

c) most of the time, lazy loading disabled

Problem: How can we manage entity which has a huge collection of independent items in a simplest and reusable way?

Example:

and Image list of the advertisement in a classified site

Solution:

Data Model:

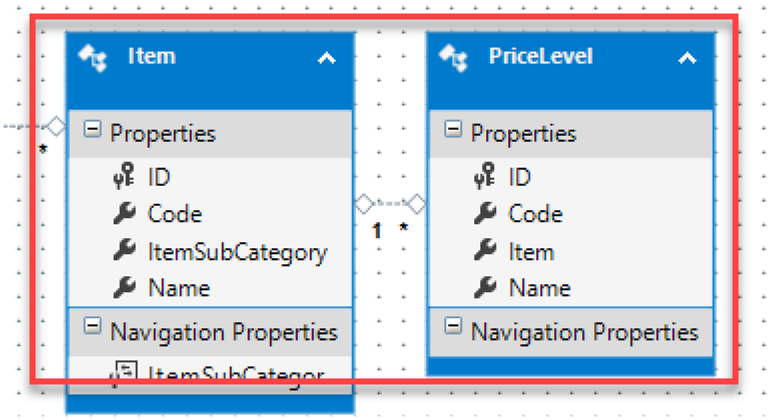


Figure 4.4.9.1: Data Model of Pattern 9

API Structure:

```
GET: list, search, advanced search, and info
- API Template
[
  {
    "ID": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
    "Code": 2085,
    "Name": "Item 1",
    "PriceLevels": [
      {
        "ID": "a6c1cbf3-8bc2-4011-9491-10f9702a9221",
        "Name": "Price 1"
      }, ...
    ]
  }, ...
]
```

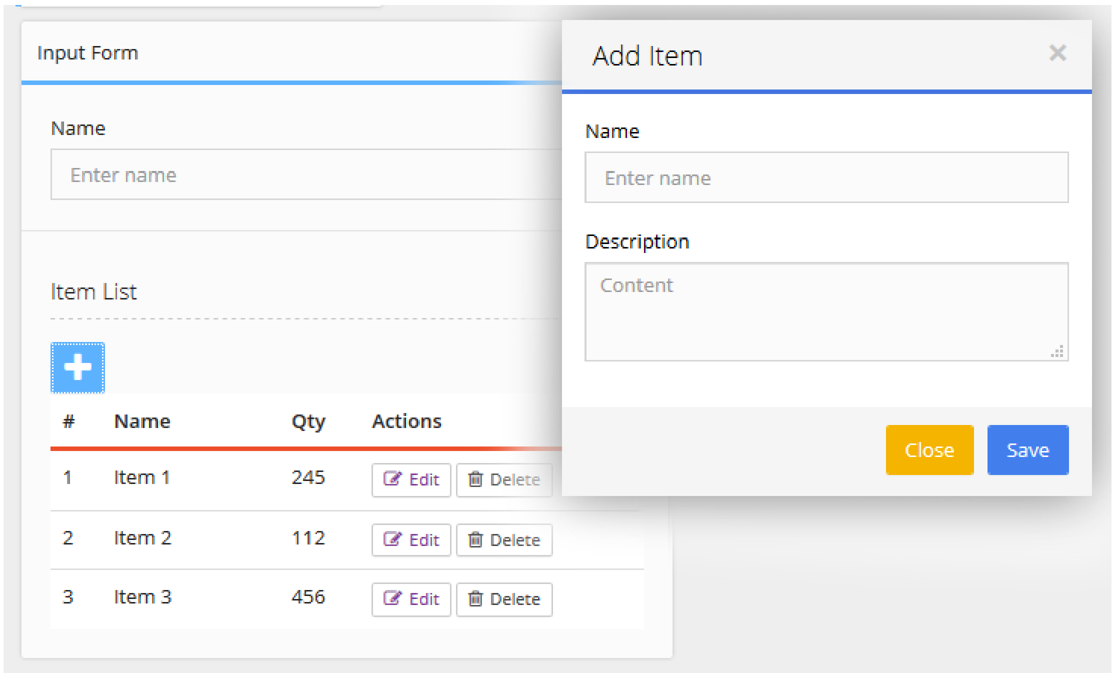
```
POST: new
- API Template
{
  "Name": "Price Level 1",
  "Item": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
  ...
}
```

```
POST: update
- API Template
{
   "ID":"abc61fd3-f2f5-4e94-87a6-0071304fd832",
   "Name":"Price Level 1",
   "Item": "13c61fd3-f2f5-4e94-87a6-0071304fd832",
   ...
}
```

Pattern-specific APIs:

```
POST: read secondary data for the selected record (GetPriceLevels)
- API Template
[
    {
      "ID": "a6c1cbf3-8bc2-4011-9491-10f9702a9221",
      "Name": "Price 1"
    }, ...
]
```

UI:



	#	Name	Qty	Actions
	1	Item 1	245	Edit Delete
	2	Item 2	112	Edit Delete
	3	Item 3	456	Edit Delete

Figure 4.4.9.2: Basic UI Input form of Pattern 9

4.5 Common Functions of an Entity Model

. Entity model can be generated from a combination of patterns (one or more). below list is the common finicalities of an entity model, in addition to that there may be pattern specific new APIs and UI components, we have mentioned some of them pattern detail sections. Ex: Pattern 2 (Association) adding new filter option for advanced search API and UI (presentation layer). in next section, we will explain how we can use unique characteristics for generating source code.

Action	Rest API	UI	Description
Search	Yes	Yes	General text search
AdvancedSearch	Yes	Yes	Advanced search for each pattern
Create	Yes	Yes	Create new record
Info	Yes	Yes	Read existing record
Update	Yes	Yes	Update existing record
Delete	Yes	Yes	Delete existing record
Backup	Yes	Yes	Backup a record
Restore	Yes	Yes	Restore backup record
GetBackupsByID	Yes	Yes	List backups by record id
GetBackupByID	Yes	Yes	Get backup by backup id

Table 4.5.1: Common API/UI function list for a pattern model

4.5 Code Generation

Source code generation is one of the main assets of the proposed framework if you wish to increase developer productivity. T4 template handles generating source code for each layer from an entity framework metadata and entity configuration & entity property configuration as explained below.

4.5.1 Entity Configuration

Entity model level preparties for code generation, all properties are optional, default properties will be generated from data table properties

Sample JSON setting

```
{
  "ReLoad": "true",
  "MenuGroup": "Sales",
  "GenarateAPI": "True",
  "GenarateUI": "True",
  "DisplayColumn": "True",
  "BasePattern": "SimpleEntity",
}
```

JSON Property	**Description**
ReLoad	Reload entity model
GenarateAPI	Generate APIs for this entity model
GenarateUI	Generate UI for this entity model
BasePattern	For recognize base pattern of the entity
MenuGroup	Menu group is generated
Model Title	Title for UI
Description	Model description

Table 4.5.2: entity configuration settings

4.5.2 Entity Property Configuration

Entity model property level preparties for code generation, all properties are optional default properties will be generated from data table column properties.

Sample JSON setting

```
{
  "InputType": "dropdownlist",
  "DisplayName": "Item Category",
  "Searchable": "True",
  "Orderable": "True",
  "DisplayColumn": "True",
  "Validations": [
    {
      "Type": "Required",
      "Message": "Select Item Category"
    }
  ],
  "DataItems": [
    {
      "Name": "English",
      "Value": "English"
    }
  ]
}
```

JSON Property	Description
InputType	Input type Ex: drop-down list
DisplayName	Label for input (optional)
Searchable	Search Option enable for API and UI
Orderable	Order by Option enable for API and UI
DisplayColumn	Show this column in list view/ search result
Validations	Collection of validations for validate API and UI
DataItems	Values for a drop-down list

Table 4.5.3: entity property configuration settings

Developer manages all the process of code re-generation, and he doesn't need to do anything special to generate code for the model. the code is generated for a model every time when the changed model is saved so that you always have all updates are available in the code in each layer. if developing a model one by one or module by module, you may disable code re-generation by using the “reload” property of an entity configuration

T4 template handles generating source code for each layer from an entity framework metadata and entity configuration & entity property configuration as I explain above. T4 is aimed at developers who would like to either create modules based on identified patterns to minimize handwritten code. With T4 we can generate code which simply wraps calls to business logic, create Request/Response classes, and interface files such as MVC components (model, view controller)

Since REST services should simply be a wrapper around our business logic, T4 we can automate creation of REST APIs or any other service technology based on our interface and model. This leaves the developer more time to focus on the business logic written in C# and customization of the application

4.6 API Testing Tool

Even though there are many API Testing Frameworks available in the market today, they may not always suit your application, may need more configurations for each API, sequential testing and model testing steps as I explained in approach chapter. This may call for the creation of your own rest testing framework.

Especially when you have code generation tools,

The framework should be able to execute the basic REST operations (GET, POST, PUT, PATCH, DELETE) and perform the validations on the code, message, headers and body of the response.

Single API Tests

Implement Filtering Options will be available to select APIs from Generated Existing Restful API list. once you selected an API, input section will be automatically populated. Ex: request JSON, API input parameters [Figure 6.5.3].

API Sequence Tests

Apart from testing of each API, there can various sequences of flow can be made from it. The output of an API can be used as an input to another API. All the sequence flow should be covered by testing of any use cases.

For example, consider the sample cashflow APIs as below:

1. Create User
2. Create Account
3. Create Sample Transaction
4. List Transactions by account

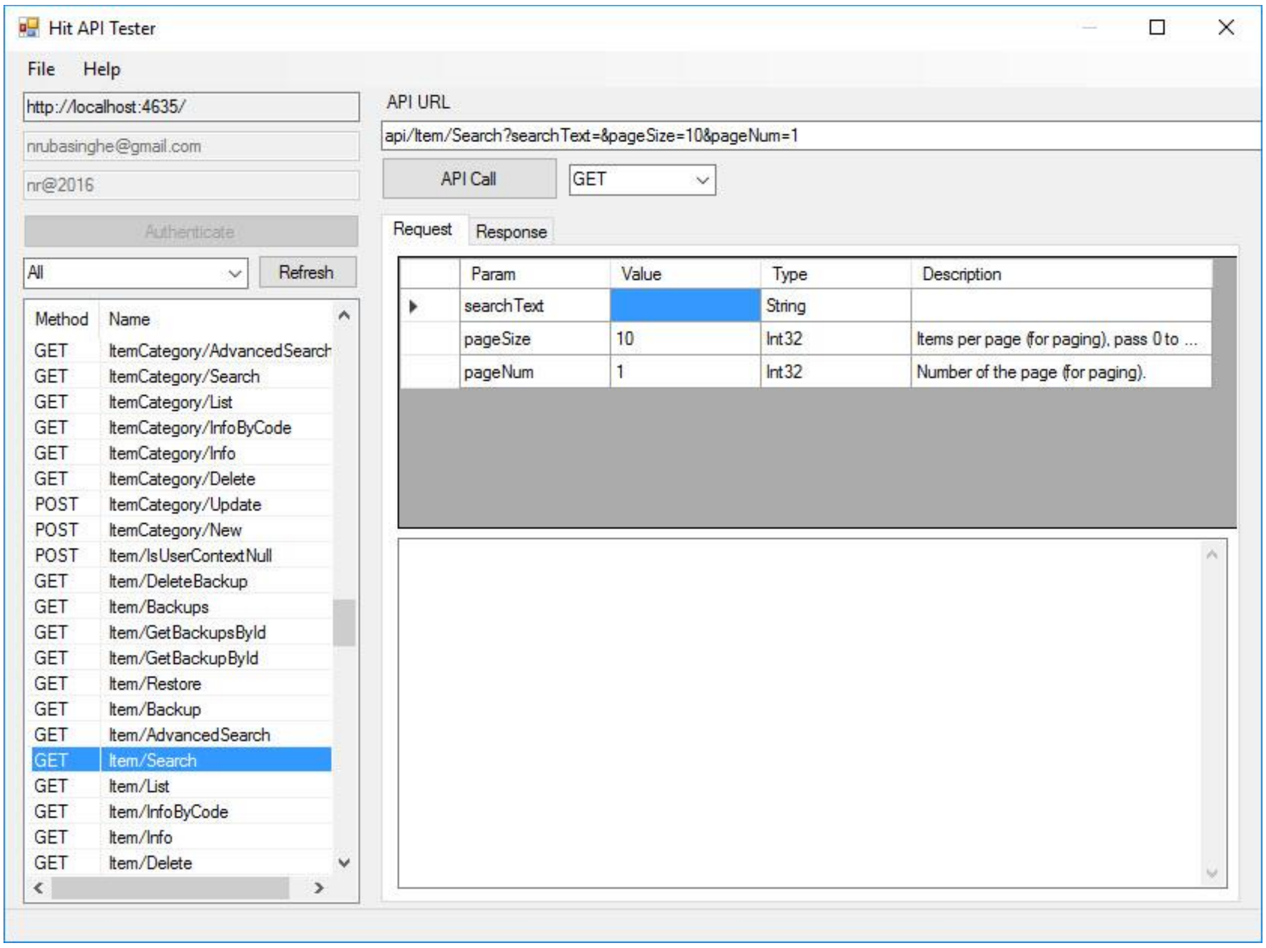
Model Tests

This is also some kind of sequence testing, but the purpose of this testing mode is automatically tested all the APIs of a model after generating source code

For example, consider the item model APIs as below:

1. Create Item
2. Create Update Item
3. Create Delete Item
4. Read Item Sad
5. Search Items
6. Advanced Search Items

The idea is automating testing process for all the generated APIs,



http://localhost:4635/

nrubasinghe@gmail.com

nr@2016

All

API URL: api/Item/Search?searchText=&pageSize=10&pageNum=1

GET

Method	Name
GET	ItemCategory/AdvancedSearch
GET	ItemCategory/Search
GET	ItemCategory/List
GET	ItemCategory/InfoByCode
GET	ItemCategory/Info
GET	ItemCategory/Delete
POST	ItemCategory/Update
POST	ItemCategory/New
POST	Item/IsUserContextNull
GET	Item/DeleteBackup
GET	Item/Backups
GET	Item/GetBackupsById
GET	Item/GetBackupById
GET	Item/Restore
GET	Item/Backup
GET	Item/AdvancedSearch
GET	Item/Search
GET	Item/List
GET	Item/InfoByCode
GET	Item/Info
GET	Item/Delete

Param	Value	Type	Description
searchText		String	
pageSize	10	Int32	Items per page (for paging), pass 0 to ...
pageNum	1	Int32	Number of the page (for paging).

Figure 6.5.3: a screenshot of a single API test mode

4.7 Summary

This chapter fully described the implementation of all the phrases of the proposed rapid application development framework. Next chapter explains the evaluation of the framework that we have already developed.

Chapter 5

Evaluation

5.1 Introduction

The previous chapter discussed the details on the implementation of all the modules of the
proposed solution. This chapter justifies and evaluates the identified patterns and proposed framework.

5.2 Case Study Evaluation

We use two case studies to evaluate your development framework with two different software development teams. design of the case study covers major phases of our framework such as design, the code generating & implementation and API testing

1) A classified website like ikman.lk
2) Cashflow application

5.2.1 Case Study 1:

We have selected a classified website like ikman.lk for evaluate our identified patterns and framework. Figure 6.1.1 shows entity relationship and how its matched with our identified patterns. Entity mapping fully fitted with our identified patterns. consider mapping details as below

* - secondary entity

Entity	Used Patterns	Description
BrandName	P1	Auto brands for vehicle ad type, Ex: Toyota,
Model	P2	Vehicle models. Ex: Premio
User	P3	User details with contact details
UserMembershipInfo	*P3	User membership details
Ad	P4, P5	Ad details with contact details
AutoSaleAd	*P5	Auto Sale Ad details (Base type is Ad)
AdImage	*P4	Collection of ads for Ad
Comment	*P9	Collection of comments for Ad
Shop	P1, P7, P8	Shop Details Ex: Car sale
ShopAdmin	*P7	Collection of users for Shop
Branch	*P8	Member financial overview
City	P2	Location cities
District	P1	Location districts

Table 5.2.1.1: Case Study 1 - Entity mapping with patterns

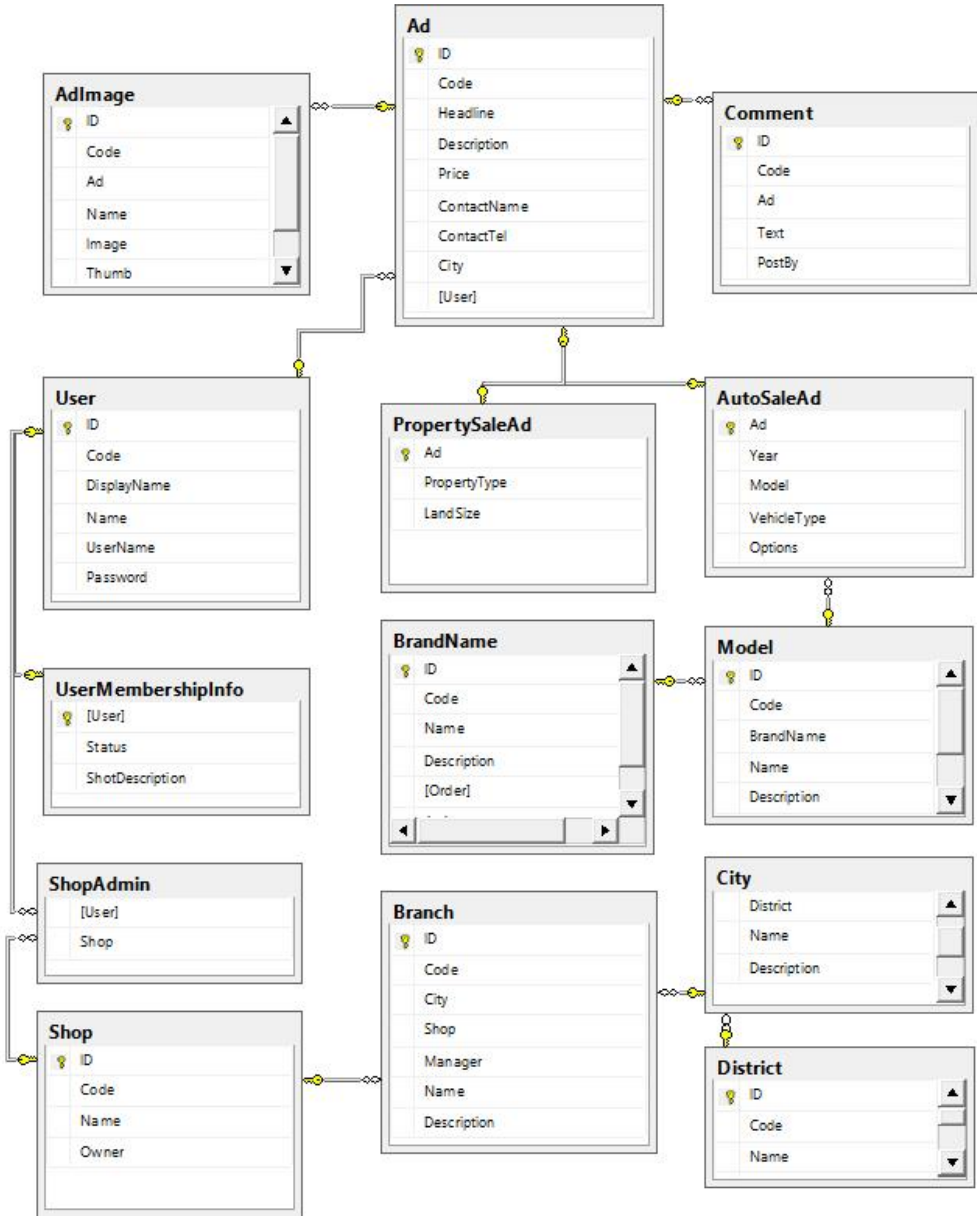


Figure 5.2.1.2: Data Model of case study 1

5.2.2 Case Study 2:

The other selected case study is a Cashflow application, including bank accounts, transactions, categories and member information.

Figure 6.1.2 shows entity relationship and how its matched with our identified patterns

Entity mapping fully fitted with our identified patterns. consider mapping details as below

* - secondary entity

Entity	Used Patterns	Description
Account	P2, P5, P9	Base Type of Accounts
CreditCardAccount	*P5	Credit account details
TermDepositAccount	*P5	Term Deposit account details
MortgageAccount	*P5	Mortgage loan account details
Transaction	*P9	Accounts transaction details
Member	P1, P3, P4	Member personal & contact info
MemberEmploymentInfo	*P3	Member employment info
MemberFinancialOverview	*P3	Member financial overview
Category	P1	Transaction category, Ex: Travel
SubCategory	P2	Transaction Subcategory, Online booking

Table 5.2.2.1: Case Study 2 - Entity mapping with patterns

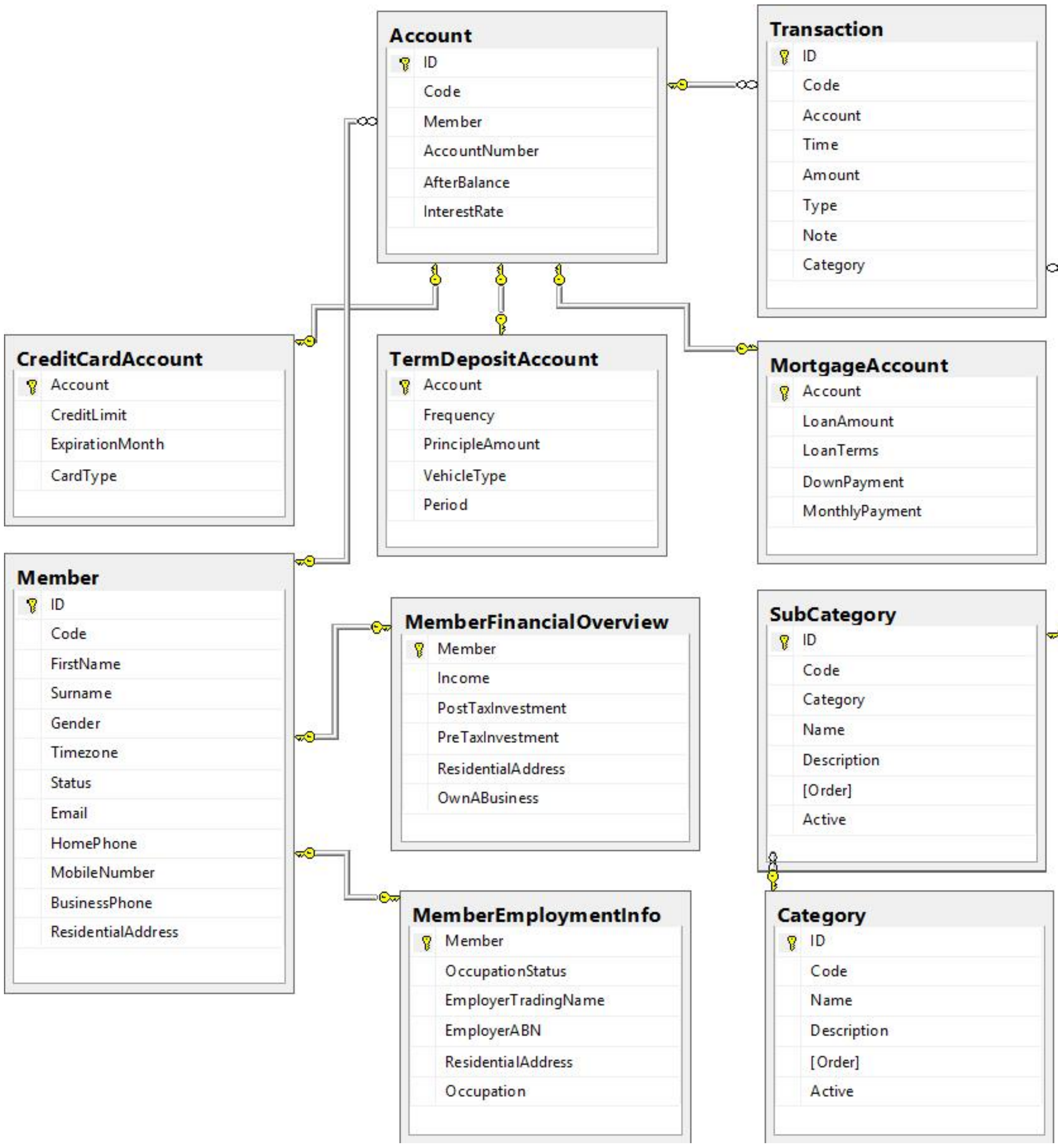


Figure 5.2.2.2: Data Model of cash study 2

5.3 Case Study Outcomes

- Data model & Entity mapping 100% fitted with our identified patterns
- Design decisions ware very easy and fast with pre-defined patterns
- Its took acceptable time to set up entity & entity property configuration [chapter 4.5.1], [chapter 4.5.2]
- Code generation was successful & finally, we got working application module within the very small period. Generated project is not 100% suitable for production. but it's a more than 60-65% completed application with high-quality industry standard code base.
- The generated source code is customizable, but it may take some developer to familiarize with coding standard and unfamiliar technologies
- API testing tool is productive, Filtering options available for generated API list. API testing configuration time can be reduced by automated API testing modes [chapter 4.6]

5.4 Summary

This chapter fully described the evaluation of the proposed framework by using case study evaluation method. we successfully used two case studies for evaluation for our main objective of proposed framework. Next chapter explains the conclusion & further works.

Chapter 6

Conclusion & Further work

6.1 Introduction

One of the biggest challenges that app developers faced before was ... The platform supports rapid application development. One rapid application development tool for building web, mobile, desktop, and server-based applications

6.2 Conclusion

Nowadays, business moves faster, and clients tend to change their minds more frequently over the course of a project's development life cycle. Of course, they expect the development team to adapt to their needs and modify the structure of an application quickly. Software development process which minimizes the pre-planning phase, and results in more rapid software development lifecycle. RAD is a methodology for compressing the analysis, design, build, and test phases into a series of short, iterative development cycles. The idea behind this methodology is to start developing as early as possible so that clients can review a working prototype and offer additional direction.

we proposed pattern-based service-oriented framework for data-oriented (information systems) rapid application developments based on the most commonly used integrated patterns. which cover not only the development, design, code generation and API testing. such a rapid application development framework can improve developer productivity and improve the quality, reliability, and robustness of new software. Developer productivity is improved by allowing developers to focus on the unique requirements of their application instead of spending time on application infrastructure

6.4 Further work

Our main objective of this project was identifying set of integrated patterns that apply to data models, business models, API model and UI templates. Based on those patterns we proposed a service-oriented rapid application development framework to improve productivity and quality in all design, development and testing phases. there are some other areas in software development lifecycle which can be applied. especially in use case specification in requirement engineering domain.

6.5 Summary

This chapter concludes the thesis by implementing a pattern-based service-oriented rapid application framework and how it can be enhancing further to improve the software development productivity and quality

Chapter 7

Reference

[1] M. James, *Rapid application development (RAD)*. Macmillan Publishing Co., Inc. Indianapolis, IN, USA ©1991, 1991.

[2] Twitter, "Bootstrap HTML, CSS, and JS framework by Twitter." [Online]. Available: http://getbootstrap.com/.

[3] R. T. Fielding and R. N. Taylor, *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000.

[4] J. Cowan, J. Paoli, F. Yergeau, E. Maler, C. M. Sperberg-McQueen, and T. Bray, "*Extensible Markup Language (XML) 1.1,*" *W3C CR CR-Xml11-20021015*, 2002.

[5] C. Douglas, "JSON (JavaScript Object Notation)," 1999. [Online]. Available: http://www.json.org/.

[6] B. Don *et al.*, "Simple Object Access Protocol (SOAP) 1.1." 2002.

[7] Boyd, Lorraine. *Business Patterns of Association Objects*. In "Martin, Robert C. (ed.); Riehle, Dirk (ed.) and Buschmann, Frank (ed.) Pattern Languages of Program Design 3", Addison-Wesley, pp. 395-408, 1998.

[8] Coad, P.; North, D.; Mayfield, M. *Object Models: Strategies, Patterns and Applications*, Yourdon Press, 2nd edition, 1997.

[9] Johnson, Ralph and Woolf, Bobby. *Type Object*. In "Martin, Robert C. (ed.); Riehle, Dirk (ed.) and Buschmann, Frank (ed.) Pattern Languages of Program Design 3", Addison-Wesley, pp. 47-65, 1998.

[10] Christopher, A., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-king, I., Angel, S., *A Pattern Language*. Oxford University Press, New York, 1977.

[11] Gamma, E., Helm, R., Johnson, R., Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison Wesley Professional, 1994.

[12] L. Bass, P. Clements, and R. Kazman,*Software Architecture in Practice*,3rd ed., P. Gordon, Ed. Boston, Massachusetts, USA: Addison-Wesley Professional, 2012.

[13] CRUZ, A.M., *A Pattern Language for Use Case Modeling*, Proceedings of MODELSWARD 2014, INSTICC Press,2014.

[14] M. Moristo, C. B. Seaman, V.R. Basili, A. T. Parra, S.E. Kraft, S. E.Condo. *COTS-based software development: Process and open issues*, The Journal of Systems and Software, September 2002.

[15] Halili, E.(2008). *Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites*, Birmingham: Packt Publishing Limited.

[16] Coad, Peter. *Object-Oriented Patterns. Communications* of the ACM, V. 35, nº9, pp. 152-159, September 1992.

[17] Johnson, Ralph and Woolf, Bobby. *Type Object*. In "Martin, Robert C (ed.); Riehle, Dirk (ed.) and Buschmann, Frank (ed.) Pattern Languages of program design 3", Addison-Wesley, pp. 47-65, 1998.

[18] Coad, P.; North, D.; Mayfield, M. *Object Models: Strategies, Patterns and Applications*, Yourdon Press, 2nd edition, 1997.

[19] Boyd, Lorraine. *Business Patterns of Association Objects.* In “Martin, Robert C. (ed.); Riehle, Dirk (ed.) and Buschmann, Frank (ed.) Pattern Languages of Program Design 3”, Addison-Wesley, pp. 395-408, 1998.

Fowler, Martin. *Analysis Patterns*. Addison-Wesley, 1997.