A Demonstrator Framework for Consistency Management Approaches

by

Arjya Shankar Mishra

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

A Demonstrator Framework for Consistency Management Approaches

Master's Thesis

Submitted to the Fakultät für EIM, Institut für Informatik
in Partial Fulfillment of the Requirements for the
Degree of

Master of Science

by
Arjya Shankar Mishra

Supervisors:
Jun. Prof. Dr. Anthony Anjorin
Prof. Dr. Gregor Engels

Paderborn, April 23, 2017

Declaration

I hereby declare that I prepared this thesis entirely on my own and have not used outside sources without declaration in the text. Any concepts or quotations applicable to these sources are clearly attributed to them. This thesis has not been submitted in the same or substantially similar version, not even in part, to any other authority for grading and has not been published elsewhere.

City, Date

Signature

Contents

1 Introduction **1**
1.1 Problem Statement . . . 2
1.2 Solution Idea . . . 2
1.3 Approach Overview . . . 3
1.4 Structure . . . 5

2 Foundation **6**

3 Requirements **12**
3.1 Functional . . . 13
3.2 Non-Functional . . . 13
3.3 Technical . . . 14
3.4 Choices and Threats . . . 14

4 Related Work **14**
4.1 Existing Demonstrators . . . 15
4.2 Virtual Machines . . . 15
4.3 Handbooks & Tutorials . . . 15
4.4 Example Repositories . . . 16
4.5 Discussion . . . 16

5 Design **17**
5.1 Choosing an Example . . . 17
5.1.1 Construction . . . 17
5.1.2 Selection . . . 19
5.2 BX Tool Selection . . . 19
5.2.1 Theory . . . 19
5.2.2 Selection . . . 20
5.2.3 Benchmarx . . . 21
5.3 Architecture Design . . . 21
5.3.1 Design Decision . . . 21
5.3.2 Framework . . . 21
5.3.3 Controller . . . 23
5.3.4 Model . . . 23
5.3.5 View . . . 23
5.4 Challenges . . . 24
5.5 Walkthrough . . . 24

6 Implementation **24**
6.1 UML Diagrams 24
6.1.1 Component Diagram 24
6.1.2 Class Diagram 24
6.1.3 Sequence Diagram 24
6.2 Core Details 24
6.3 Challenges 24

7 Evaluation **26**
7.1 Discussion and Feedback 26
7.2 Evaluation and Results 26

8 Summary and Future Work **26**
8.1 Conclusion 26
8.2 Future Work 26

9 Appendix **27**

List of Figures **28**

List of Tables **29**

References **30**

1 Introduction

In the era of Information Technology, the usage of softwares and applications are growing day by day and have become an important part of our lives. This makes the software industry one of the largest industries on the world and many others companies are built around the development of softwares. With the growing usage of softwares, software development process has changed drastically and currently more solution oriented. Nowadays, the entire focus is making the software development process fast, less complex and more human-friendly.

One of the approach to reduce the complexity of software development is abstraction and separation of concerns [22]. In recent times, software modeling has become an effective way for implementing this principle. In traditional approach, developers manually writes programs and checks the specifications realted to software models, which is often costly, incomplete, informal and carries a major risk of failure. On the contrary, model-driven software development (referred as **MDSD** from now on) approach improves the way softwares are built by moving the focus from programming language code and representing the essential aspects of software in the form of software models. It increases the reduces the development costs and increases the reusability and maintainability of software. The objective of MDSD [22] is to increase productivity and reduce time-to-market by enabling development at a higher level of abstraction and by using concepts closer to the problem domain at hand, rather than the ones offered by programming languages.

The core idea of MDSD approach is based on models, modeling and model transformations. In this approach, developers depits the real world system under discussion in the form of models with certain level of abstraction. These models are used to form the problem domain and the relationships between them helps to form the solution domain. Models can be changed to represent different state/view of the system. But, it is necessary to change between different states of a system with equivalent level of abstarction by ensuring their overall consistency. This phenomenon is handled by model transformation. This certainly increases the developers productivity and the quality of the models.

Bidirectional transformation (denoted by "bx", referred as **bx** from now on) is a technique used to synchronize two (or more) instance of different meta-models. Both models are related, but don't necessarily contain the same information. Changes in one model thus lead to changes in the other model [1]. Bx makes sure that two models that can change over time have to be kept constantly consistent with each other.

Bidirectional transformation is used to deal with scenarios like:

- change propagation to the user interface as a result of underlying data changes
- synchronization of business/software models

- refreshable data-cache incase of database changes
- consistency management between two artifacts by avoiding data loss

and many more....

Bx community has been doing research and development work in many fields like software development, database, mathematics and much more to increase awareness and to reach more people [2][1]. As a result, many kinds of bx tools are being developed, e.g., eMoflon [9], Echo [12]. These bx tools are based on various approaches, such as graph transformations, bidirectionalization, update propagations [10] and can be used in different areas of application.

1.1 Problem Statement

Bidirectional transformation is an emerging concept. In the past, many efforts have been made by conducting international workshops, seminars and through experiments conducted by developers / bx community to identify its potential. Also, in addition to the development of bx tools and bx language, benchmarks are being created for bx tools for systematic comparison [4].

Although a significant amount of work has been done in this field, some basic problems still remain:

- **Reachability:** Reachability to relevant communities is not significant due to the absence of a common vocabulary for bx across research disciplines [3]. Seminars are still conducted for exchanging ideas in different communities to define a common vocabulary of terms and properties for bx [2].
- **Applicability:** Bx tools and their applicability is still not widely known even in the developers' communities. Many developers and researchers are still using non-bx transformation tools to achieve properties which can be easily supported by bx-tools [3]. This is because of the existing conceptual and practical challenges associated with configuring/trying a bx-tool for knowing its potential, building software systems using bx-tools, etc.
- **Demonstrator:** Absence of a simple yet interactive bx tool demonstrator to depict the potential of *bidirectional transformation* over preferred non-bx tool demonstrators among developers' and researchers' communities [3].

1.2 Solution Idea

To solve the problems as described in Section 1.1, in this thesis, my goals are as follows:

- Design and implement an interactive demonstrator.

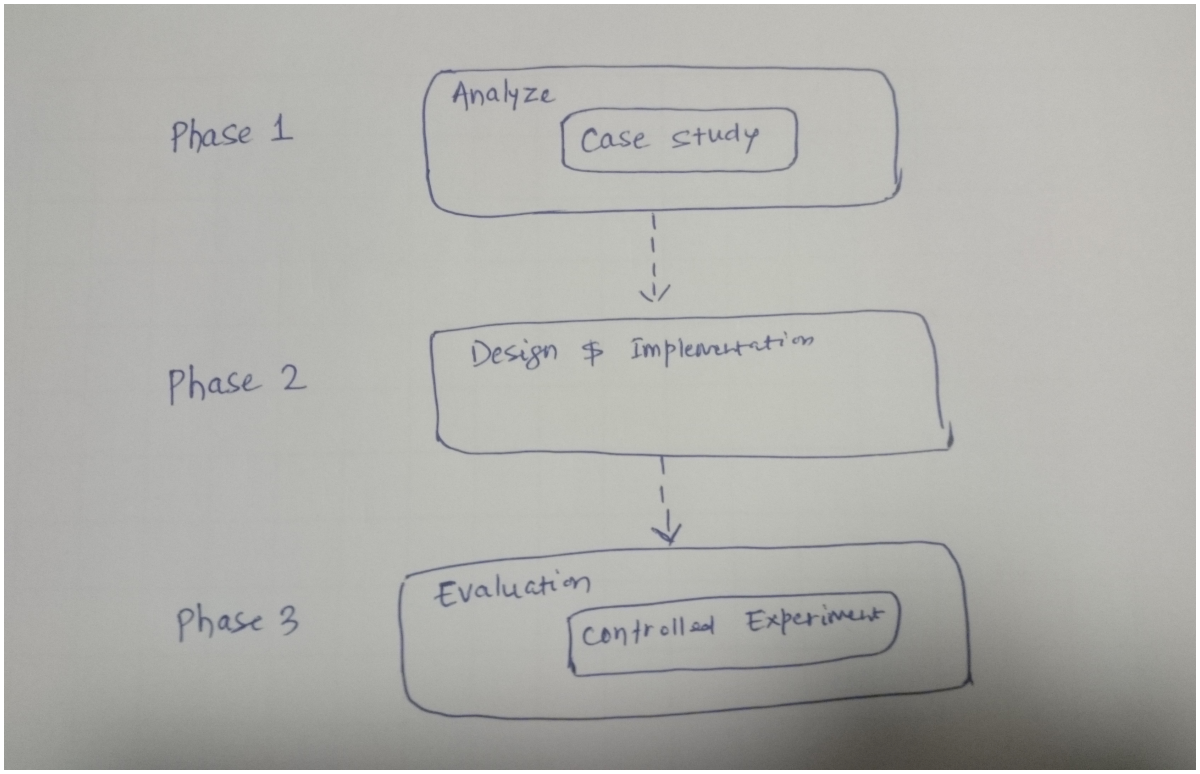

Figure 1: Approach Overview

- Spreading the basic concepts of bx to a wide audience and making them accessible and understandable.

An existing bx tool will be used as a part of the demonstrator to realize *bidirectional transformation*. The final prototype will be interactive and easily accessible to users to help them understand the potential, power and limitations of bx.

1.3 Approach Overview

My approach for providing a solution to the problems described in Section 1.1 consists of various stages which focuses on designing and implementing a successful bx tool demonstrator for mitigating the problems as shown in figure 1.

Analysis This stage is based on the research method called Case Study [18]. In any research, there could be many cases and each case could focus on a number of different research questions, each of which leads to a different direction in developing solution strategies [18]. Hence, this method aims at selecting cases that are most relevant to the research and research questions

are pinned down to the exact problems in hand.

Initially, the case study was based on the existing work/research done on bx, existing tools available in the market, flexibility in usage, time and technical expertise required to use these bx tools, and the implementation of these tools in different areas. With the initial study and knowledge gathered, I have divided the problems into following relevant cases and associated research questions (referred to as **RQ** from now on):

Case: Demonstrator

- ***RQ1*** – What are the core requirements for implementing a successful bx demonstrator ?
- ***RQ2*** – What kind of interactivity and to what extent is it required in the bx demonstrator ?
- ***RQ3*** – Which goals can be particularly well addressed in a bx demonstrator and why ?

Case: Applicability

- ***RQ4*** – To what extent is such a bx demonstrator reusable?
  RQ4 can be split into the following sub-questions:
  RQ4.1 – Is the implementation of the demonstrator bx tool-specific ?
  RQ4.2 – Is the implementation of the demonstrator example-specific?
  RQ4.3 – What part(s) of the demonstrator can be reused in implementing a different example ?

Case: Reachability

- ***RQ5*** – Is it possible to teach the concepts of bx through a demonstrator ?
- ***RQ6*** – Does an interactive GUI helps an user to increase his/her understanding of bx concepts ?

All of my work is directly or indirectly related to the above research questions.

Design and Implementation This stage is consist of all the steps related to designing and implementing the demonstrator by keeping a focus on the research questions.

First step was to choose a bx-tool for my demonstrator. Based on the gathered information and taking account implementation related issues, I have chosen a bx-tool to be used as a part of the demonstrator to realize bx. Section 5.2 explains the process in detail.

Secondly, I have constructed a few examples which can be implemented covering the requirements and showing the usability of bx tools through demonstrator. Then, taking account the availabilities of resources and usability factor, I finally chose the best suitable example to

implement and build the final prototype. Section **??** describes list of all the examples.

Next step was to set up the entire application framework for implementing the demonstrator. First, I did some research by going through materials on software design patterns and web application architecture. Then, I prepared a few proof of concepts(POC) for checking the feasibilty of the architecture designs before finalising my application framework. Section 5.3 explains the process in detail.

Evaluation This stage is based on the research method called Controlled Experiments [18]. This experiment is based on one or more hypothesis, which guide all steps of the experimental design, deciding which variables to include and how to measure them. So, it is an investigation of one or more hypothesis where one or more independent variables are manipulated to measure their effect on one or more dependent variables. Finally, data is collected and analyzed to measure the outcome. Section **??** explains the process in detail.

1.4 Structure

This document is structured as follows:

Chapter 1 (introduction) contains the introduction and motivation about the thesis with a solution strategy.

Chapter 2 discusses the related terminologies with respect to bidirectional transformation.

Chapter 3 describes the requirements for implementing a successful bx demonstrator.

Chapter 4 explains the related work that has been done on bx in last few years and the related problems.

Chapter 5 describes all the high-level concepts of my implementation work in brief with related diagrams.

Chapter 6 provides the in-depth details of each implementation layer along with UML diagrams.

Chapter 7 presents a walkthrough of the application from UI perspective.

Chapter 8 contains the feedback from user groups, evaluation results and learning goals(based on research questions).

Last chapter summarizes all the work which was done as part of this thesis and draws useful conclusions followed by future work.

2 Foundation

This chapter describes some commonly used terminologies with respect to bx and their definitions in the software industry. These definitions will help the reader in understanding the further chapters.

As already mentioned, **"bx is a technique to maintain consistency(synchronization) between two(or more) related models"**.

Model is an artefact or external representation of reality which reflects certain relevant aspects of a real system. It is denoted as "M".

- Artefact, because it is something artificial, not real.
- External representation, because it can be a drawing, a physical object or software generated on a computer.
- Certain relevant aspects, because it contains some properties related to the real system.

A model depicts the structure and/or behavior of the system under discussion from a certain viewpoint and at a certain level of abstraction which helps in managing and understanding the complexities of a system [19] [20].

Example: In my demonstrator, the example that I have implemented has two models e.g., Kitchen and Grid. Both the models represent the reality "Kitchen". A relation between reality and models is shown in Table 2.

Modeling is the process which allows a developer to specify, visualize and capture a variety of important characteristics of a system in corresponding models which helps in decision making in all the development stages [19]. Nowadays, Unified Modeling Language (UML) [19] is used to visualize models in an object-oriented fashion.

Example: In my demonstrator example, the process of identifying and constructing the "Grid Model" with the characteristics that is required for the task. A basic model of the "Grid Model" is shown in figure 2.

State denotes the condition of an artefact at any given point of time.

Delta is the change done to one or more properties of an artefact. It denotes the relationships between models from the same model space [6]. It is denoted as δ: M $\longrightarrow$ M' where M' is an updated version of M. Refer Figure 3.

Reality	Model	Aspects Covered
Kitchen	Kitchen Model	• Area of a kitchen as a white space • 4 Walls of a kitchen • contains the objeccts of a kichen • Creating, Moving, Deletion of kitchen objects
Kitchen	Grid Model	• Area of a kitchen as a block structure • 4 Walls of a kitchen • contains the objeccts of a kichen • Creating, Deletion of kitchen objects

Table 1: Model and Reality

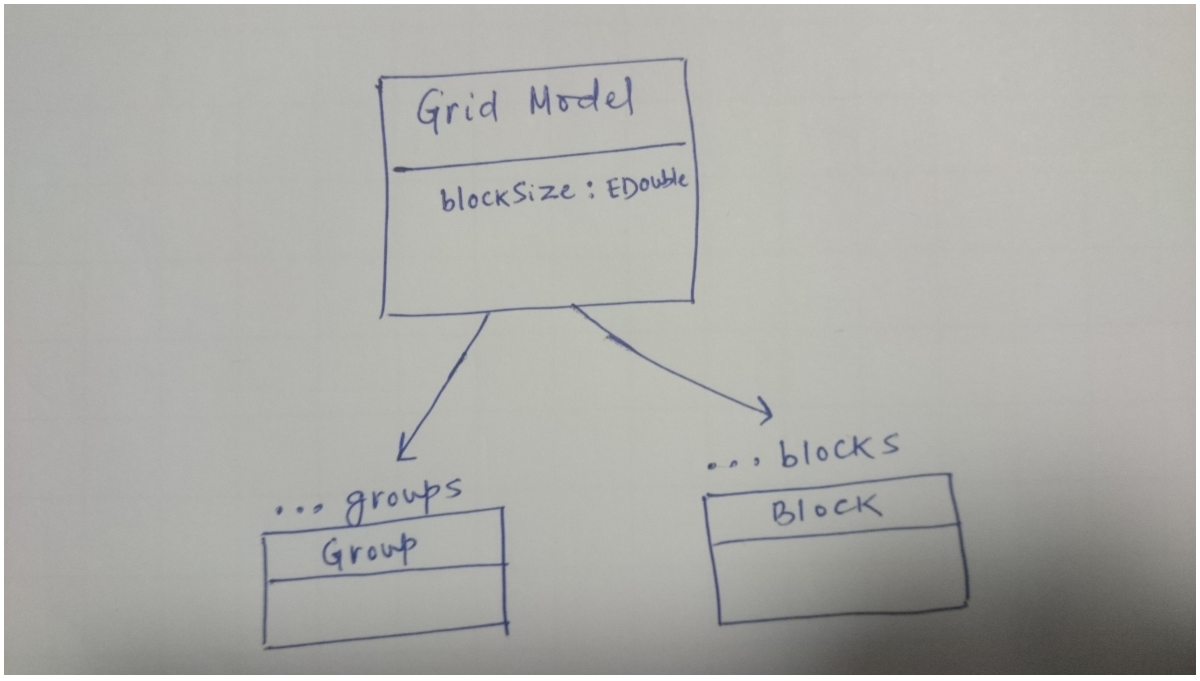

Figure 2: Modeling Example

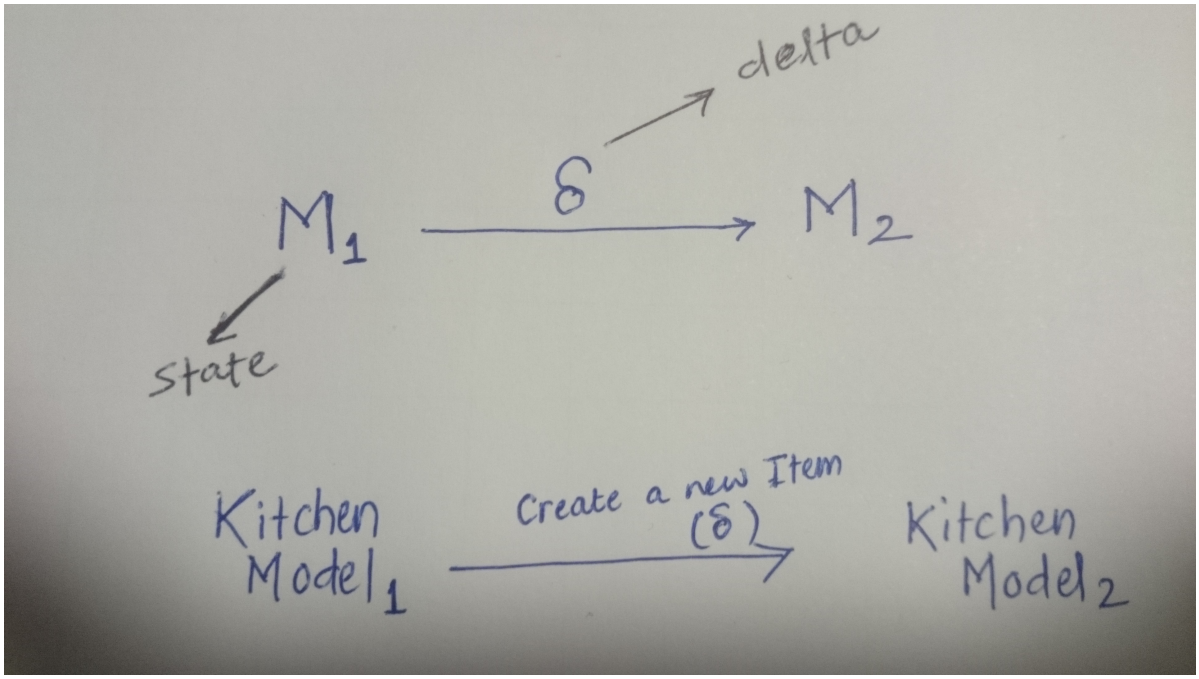

Figure 3: Delta Propagation

Example: In my demonstrator example, deltas related to the "Kitchen Model" are described in Table 2.

Structural Delta denoted as "s-delta" is the collection of changes done to a model to update it, which is compatible with structure [6]. The way the changes are applied to the original model to result in a updated model (models of the same model space) don't necessarily have to be in the order in which they are performed.

Operational Delta denoted as "o-delta" is the collection of operational specification changes done to a model to update it [6]. It is important that the changes are applied to the original model to result in a updated model (models of the same model space) have to be in the order in which they are performed.

Correspondence Links Relationships between models from different model spaces are called correspondence links, or just corrs [6]. A corr is a set of links r(a; b) between elements (a in A, b in B), which are compatible with the models' structure and denoted by double bidirectional arrows, e.g., R: A $\Longleftrightarrow$ B.

Example: In my demonstrator example, an example of a corr is r(group; itemSocket) where

Model	**Delta (δ)**
Kitchen Model	• Creating a new Item • Deleting an existing Item • Moving an Item

Table 2: Examples of Delta

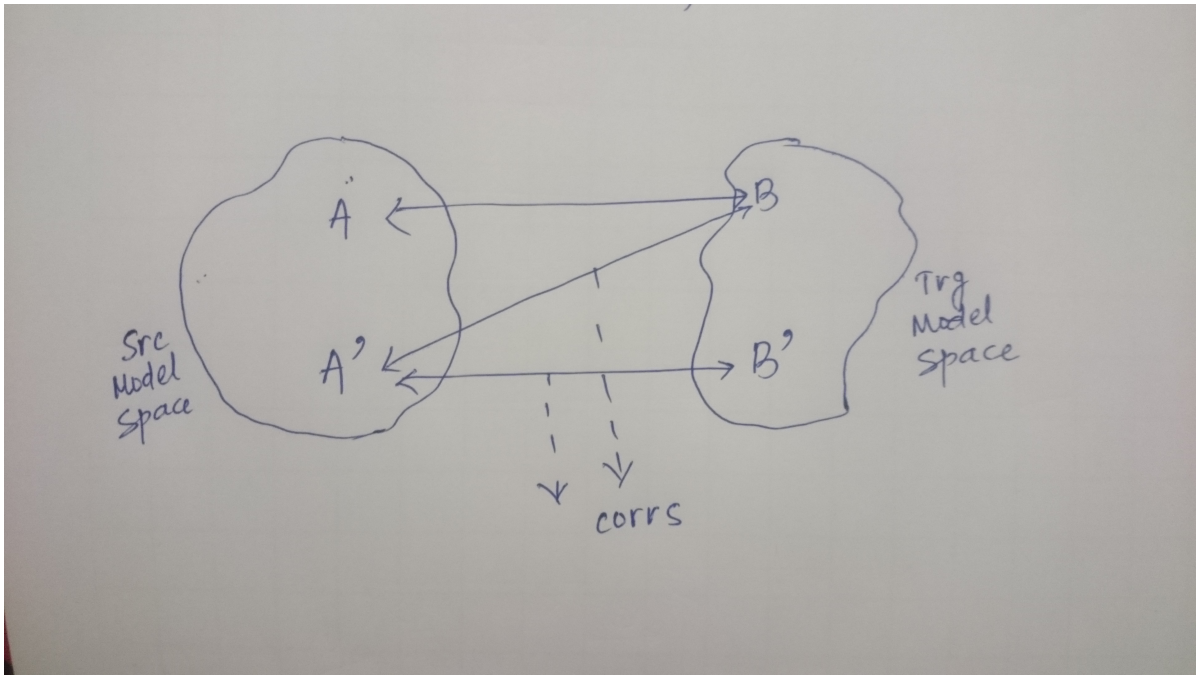

Figure 4: Correspondence Links

group, *itemSocket* are the elements of Grid and Kitchen Model respectively. Refer Figure 4.

Transformation is the process in which an artefact undergo a change from one model to another model (models from different model spaces) expressed in either same abstraction level or in different abstraction level. Refer Figure 5.

Forward Transformation is the transformation from *Source* model to *Target* model.

Backward Transformation is the transformation from *Target* model to *Source* model.

Example: In my demonstrator example, "Grid Model" is the *Source* model and "Kitchen Model" is the *Target* model. Forward transformation will cause "Grid Model" to change into

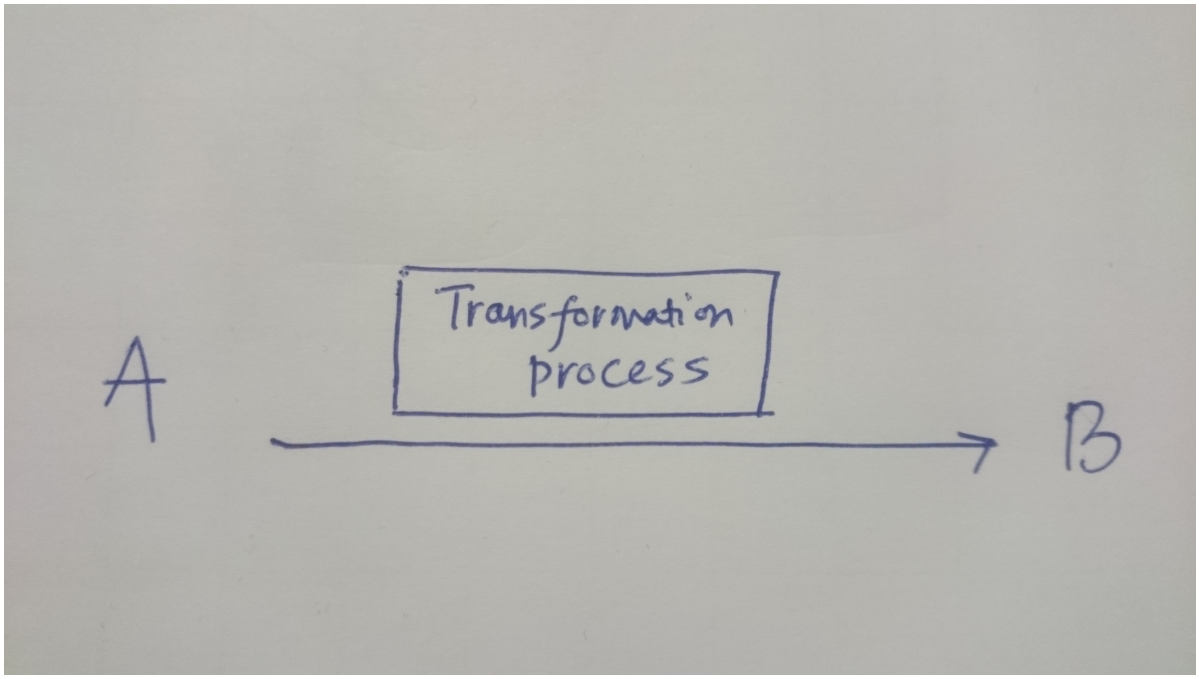

Figure 5: Transformation

"Kitchen Model" and Backward transformation will cause "Kitchen Model" to change into "Grid Model". Refer Figure 6.

State Space describes all the states of an artefact and all the deltas which lead from one state to another.

Example: In my demonstrator example, all the states of Kitchen Model along with its deltas are shown in Figure 7.

Consistency In layman's term, consistent is a state which always involves 2 or more objects/artefacts but doesn't involve ambiguity between them. This is the most important part of the model transformation and the entire focus revolves around it. To be consistent with each other the models have to be inline with respect to their states. Changes in one model may or may not cause any change in other model (models from different model spaces) but their states must not contain any contradiction. It is checked on the set of all corrs related to both the models, R: A $\Longleftrightarrow$ B, i.e., the corr R is consistent, or R is inconsistent [6].

Unidirectional vs.Bidirectional Transformation Unidirectional is the simplest one out of all kinds of transformation. It always takes one type of input and produces same type of output.

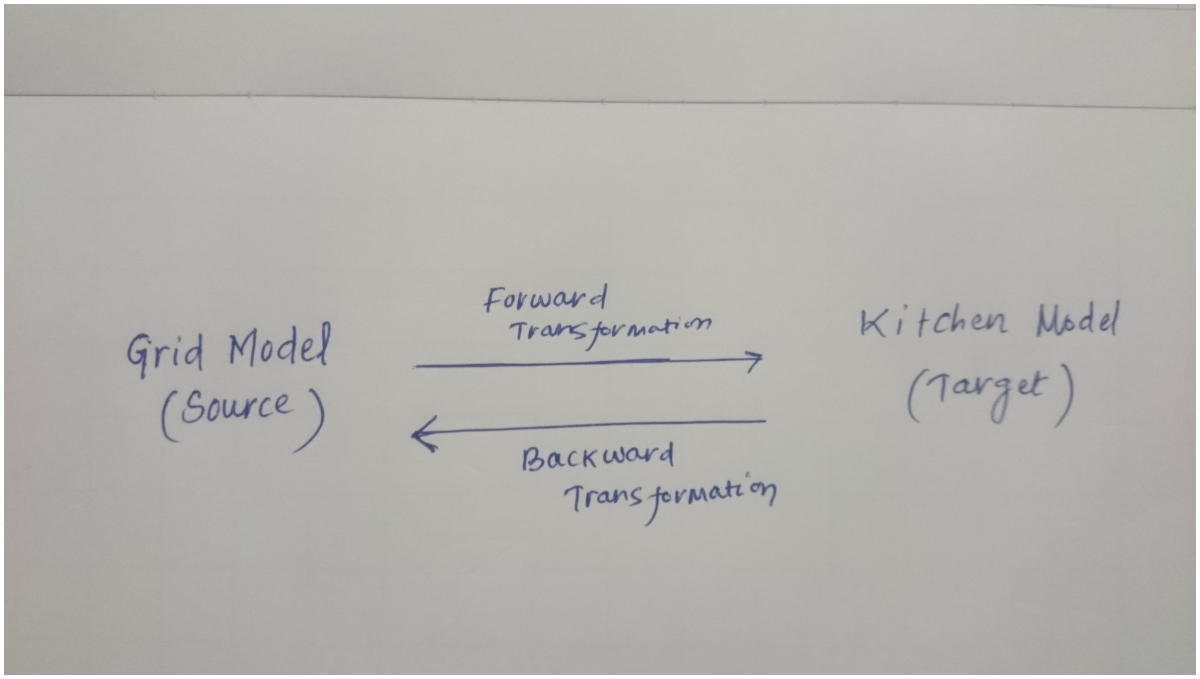

Figure 6: Forward and Backward Transformation

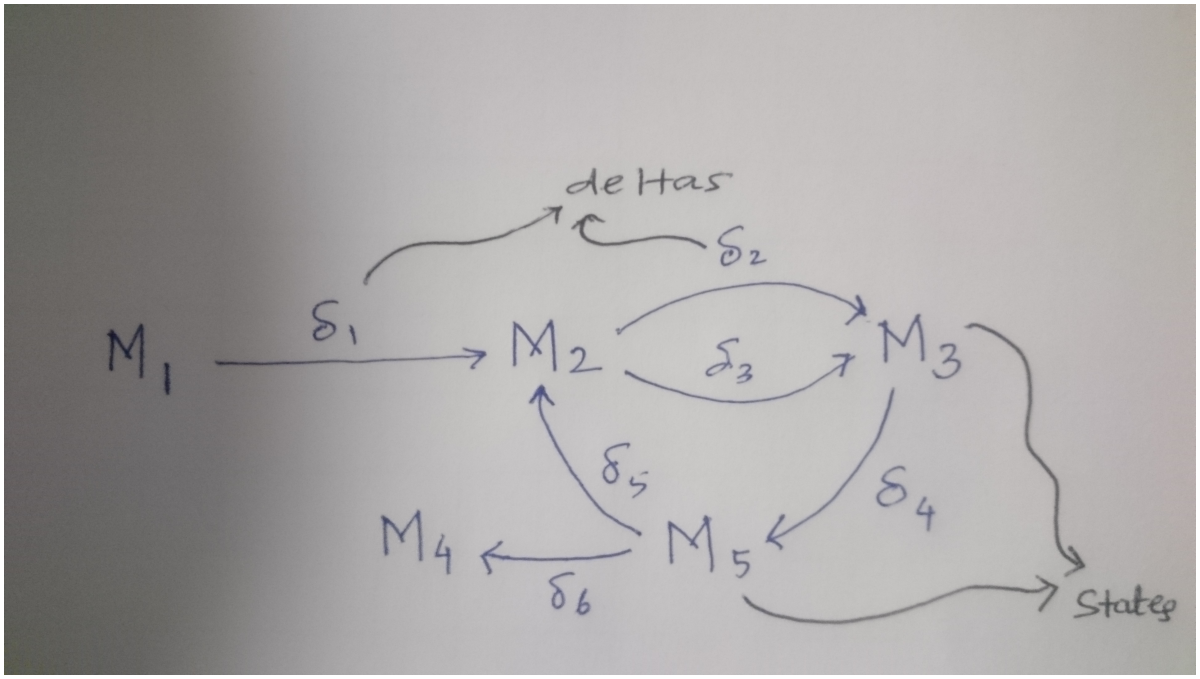

Figure 7: State Space

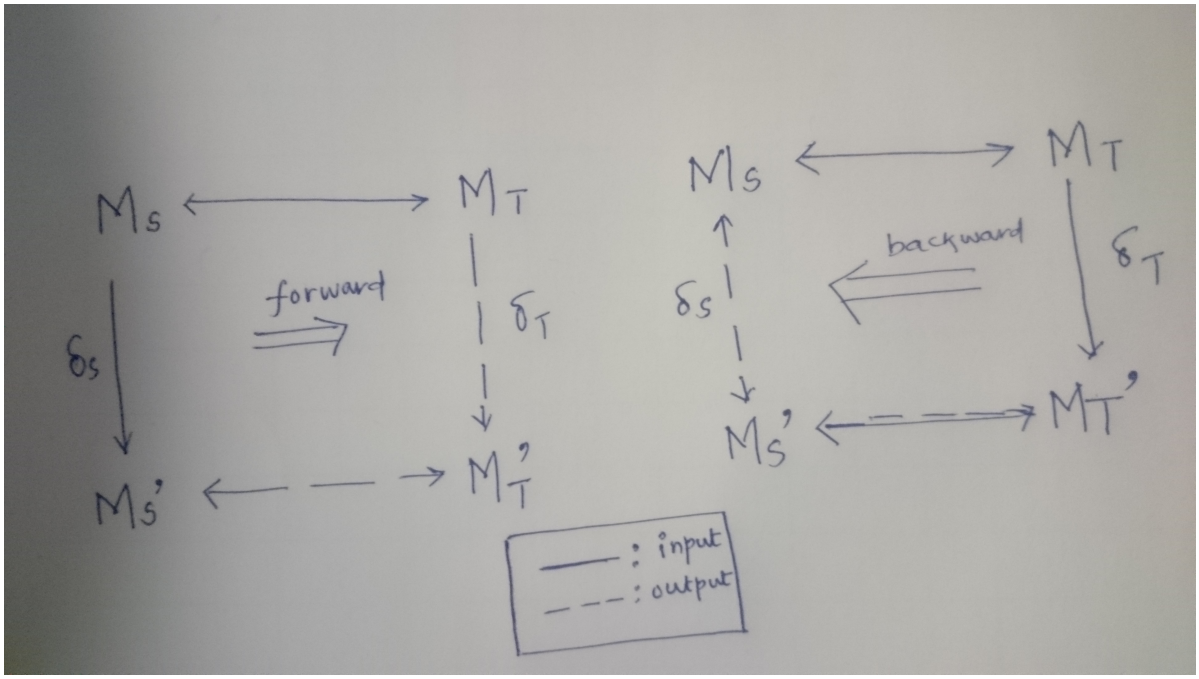

Figure 8: Bidirectional Transformation

The concept of consistency is very simple as the input model is consistent with the output model, only.
Whereas, bidirectional transformationis a pair of transformation which takes place in both forward and backward direction. One model can be input sometimes and output at some other time. The concept of consistency is relatively complex as more than one model and the models must be kept consistent. Source model (M_S) is transformed to target model (M_T) in forward transformation and vice-versa in backward transformation. It is denoted by "BX".
Figure 8 describes the concept in detail.

Bijective Transformation In bijective transformation, every element of a model is paired with exactly one element of another model (models from different model spaces). Refer Figure 9.

3 Requirements

This chapter explains all the requirements needed to implement a successful demonstrator. Section 3.1 describes all the functional requirements. Afterwards, Section 3.2 deals with all the non-functional requirements. Then, Section 3.4 describes some of the choices that I made during research and implementation work and its associated threats.

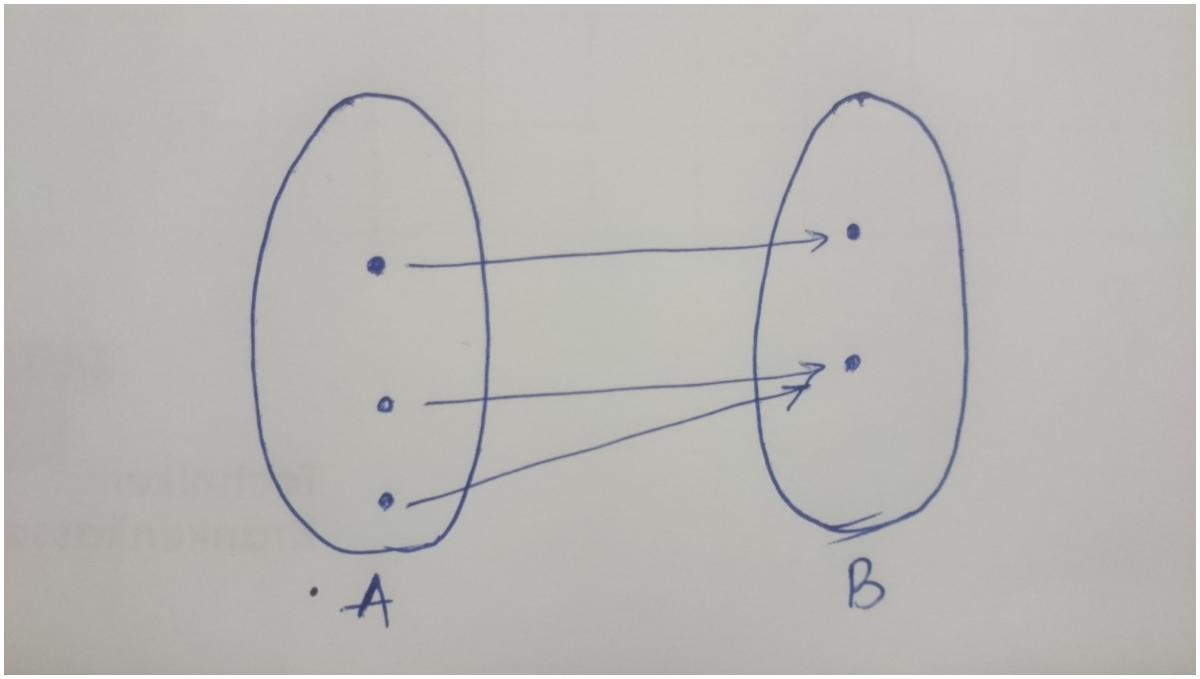

Figure 9: Bijective Transformation

3.1 Functional

Learning Goals Along with exploring the challenges as described in Section **??**, I am also focussing on teaching some basic concepts(functional requirements, referred to as **FREQ** from now on) to the user about bx in the process of trying/playing with the demonstrator.

What am I trying to teach?

- ***FREQ1*** – Bidirectional is not always bijective.
- ***FREQ2*** – Not all changes can be propagated. In this case, consistency needs to be preserved.
- ***FREQ3*** – In BX, Challenge is to avoid or minimise information loss.
- ***FREQ4*** – Current limitation - you can only change one side.
- ***FREQ5*** – Synchronisation is interactive (to handle non-determinism).

3.2 Non-Functional

As my thesis is more focussed on the implementation of a demonstrator for consistency management based on a bx tool, following are some of the non-functional end results (referred

to as **NREQ** from now on) I am trying to achieve:

- ***NREQ1*** – Reduce the installation time of the bx tool by making the demonstrator available online. Hence, the user is just a click away to try the bx tool and needs nothing to install on his/her machine.
- ***NREQ2*** – Demonstartor should be interactive and fun to play with.
- ***NREQ3*** – Demonstrator's example should not be too technical and rather be simple to attract more users. Also, user should be able to relate to the bx concepts through the example.
- ***NREQ4*** – Demonstrator should be able to guide the user in the whole playing/learning process.

3.3 Technical

3.4 Choices and Threats

The implementation process and the final prototype was driven by many choices and threats. For example,

- selection of the bx tool and the most suitable example to implement has impact on deciding the usefullness of the demonstrator.
- seletion of the bx tool and the example to implement is more or less influenced by the ideas given by my supervisor.
- my decisions on designing the application's framework for implementation are impacted by the existing availability and usability issues of the finalized bx-tool.
- During evaluation, I might not have got proper feedback from my friends & colleagues due to my acquaintance and time constraints.

4 Related Work

This chapter sums up all the related work that has been done on bx. Section 4.1, 4.2, 4.3, 4.4, and describe the various ways that bx community and developer's group have tried to make the work done on bx visible to the world. Then, Section 4.5 explains the related problems.

Model transformation is a central part of Model-Driven Software Development [1] [2]. Bx community has been constantly doing research and development work in many fields to help people understand and increase awareness about bx. Nowadays, researchers from different

areas are actively investigating the use of bx to solve a variety of problems. A lot of work has been done in terms of building usable tools and languages for bx. These tools can be used in various fields, for achieving *bidirectional transformation*. To understand these tools, several handbooks, tutorials and examples have been created so that users and developers can understand the core concepts. Following sections will describe these concepts in detail.

4.1 Existing Demonstrators

Also, we have analyzed an existing demonstrator available along with the test cases of a domain-specific language, BiYacc [15] which is based on BiGUL [13].

Being an online demonstrator, a user can try it out instantly and check how it works. It doesn't require any installation or technical expertise to get the example running and also, it makes the features of bx noticeable.

4.2 Virtual Machines

Also, we have analyzed a web-based virtual machine, e.g., SHARE [16]. Basically, this is a web portal used for creating and sharing executable research papers and acts as a demonstrator to provide access to tools, softwares, operating systems, etc., which are otherwise a headache to install [16].

This provides the environment that the user requires to execute his/her tool or program. Hence, it reduces the overhead of a user for maintaining and organizing all software framework related stuff and simplify access for end-users.

4.3 Handbooks & Tutorials

As a part of our research, I have analyzed some tutorials and tools and below are my findings.

Anjorin et al.[9] present the concept for *bidirectional transformation* using Triple Graph Grammars (denoted by "TGG") [7]. To demonstrate their core idea and the usage of the tool, they have described an example by transforming one model (source) into another (target) through TGG transformations [7][8]. The whole tutorial is about 42 pages long which guides the user to get the example running through a series of steps. These steps include installing Eclipse[1], getting their tool as an Eclipse plugin, setting up the workspace, creating TGG schema and specifying its rule and much more. If the user is able to execute each step correctly, then

[1] *Integrated Development Environment (IDE) for programming Java*

finally he/she can view the final output. It took me 4 days to get the tool up and running.

We have analyzed a tutorial[14] on a bidirectional programming language BiGUL[13]. The core idea with BiGUL is to write only one putback transformation, from which the unique corresponding forward transformation is derived for free. The whole tutorial is about 45 pages long which includes a lot of complex formulas, algorithms, and guides the user to get the example running through a series of steps. These steps include installing BiGUL, setting up the environment, achieving bx through BiGUL's bidirectional programming and much more. If the user is able to execute each step correctly, then finally he/she can view the final output.

4.4 Example Repositories

A rich set of bx examples repository [11] has been created based on many research papers. These examples cover a diverse set of areas such as business process management, software modeling, data structures, database, mathematics and much more.

A user can find relevant information about the examples on the respective web pages. Some of the examples are very well documented along with class diagrams, activity diagrams, object diagrams etc. and source code of a few examples are available as well.

4.5 Discussion

Some associated problems that I found with the above paragraphs are as follows:

- Installation requires technical expertise and time consuming as the user typically has to setup and install the tool.
- Required knowledge is too tool/area specific and it can be challenging if the user is not familiar with the technological space, e.g., User must possess knowledge about Java and Eclipse framework or a Java programmer might find the tool chain for Haskell unfamiliar.
- Steps to get the example running needed technical expertise, e.g., In some cases, domain specific knowledge includes mathematics and specific coding language. What is showcased and discussed is tied to a specific technological space and might not be easily transferable to other bx approaches.
- Not very helpful to understand what bx is before deciding which bx tool (and corresponding technological space) to use.
- The existing demonstrator's visual representation doesn't create interest in the target audience as it does not exploit the potential of using an interactive GUI and colors, etc. It

just makes use of two text fields and is comparable to a console-based interface that is accessible online.

- In the existing demonstrator, there is very limited guidance provided concerning what the user can do and try out with the demonstrator. Especially for non-experts, it is not clear what to do with the demonstrator after a few minutes.
- The existing demonstrator is based on a rather technical example that might not be relevant, interesting, or convincing for a large group of potential bx users.
- For security reasons, virtual machines are not like other web portals where a user need to simply sign-up and can host/create/access data, rather it includes a series of request-grant cycle for getting access to an environment and hosting/managing data. Also, some actions require special authorization and take time to complete the whole process.
- User needs at least 3 Gigabyte or more space on his/her local system for configuring the virtual machines depending on the requirement of the environment, which sometimes creates an overhead.

5 Design

In this chapter, I am going to describe all high-level technical details realized during the implementation of the demonstrator. Section **??** describes all the examples that I have conceptualized before choosing the final one for the implementation. This is then followed by the description of the steps taken for selecting the bx-tool in Section 5.2. Afterwards, Section 5.3 deals with the decisions taken for finalizing the apllication's architecture design.

5.1 Choosing an Example

To solve the problems as described in Section **??**, the main idea is to design and implement an interactive bx tool demonstrator based on an intuitive example.

5.1.1 Construction

I have constructed a few examples for implementation as follows:

Task Management This prototype can be used for allocating tasks in a team. It contains two views e.g., supervisor's view and employee's view. A Supervisor can allocate tasks to their subordinates. An employee can view the tasks assigned to him. Then the task will go through

a life cycle as the work progresses, i.e., Assigned, In Progress, Testing, Done. Supervisor's view shows aggregate information from multiple projects and multiple employees, but does not contain detailed information, e.g., tasks have fewer states than for assigned employees. Bx rules control how updates are handled and states are reflected in the different views of the project, e.g., the employee's view will be updated for each state change, whereas the supervisor's view is only updated when a task is completed and not for intermediate changes.

Quiz This prototype can be used for an online quiz game. It contains two views e.g., administrator's view and participant's view. There will be a large set of questions related to different areas, e.g, history, geography, politics, sports, etc. The administrator can select the areas from which the questions will be shown to the participant and initiate the game. The participant can override the selection of the areas and start the quiz. Randomly questions will be shown to the participant from the selected areas with 4 options. The administrator's view contains less information than the participant's view, e.g., only the result of each question will be shown to the administrator, whereas participant can see questions along with its options. As soon as the participant chooses the answer to any question, bx rules control how updates are handled and states are reflected in the different views of the project.

Playing with Shapes It contains two views e.g., low-level view (depicts UI[2] for low-level language, i.e., UI with less functionality) and high-level view (depicts UI for high-level language, i.e., UI with more functionality). User will draw a geometric shape, i.e., triangle / square / rectangle / circle with some notations similar to the shape on the low-level view and if the notations are correct, the high-level view tries to recognize the shape and draws it with default parameters and vice-versa. Basically the transformation will happen between a low-level language and a high-level language and bx rules control how updates are handled and states are reflected in the different views of the project. In high-level view, more functionalities will be present, i.e., moving one shape from one place to another, creating a clone of an existing shape, etc. which is not possible in low-level view.

Arranging a Kitchen It contains two views e.g., low-level view (depicts a grid structure containing blocks) and high-level view (empty space which depicts UI for kitchen). High-level view has more functionalities such as creating/ deleting/ moving an kitchen item, etc. out of which only a few will be available in low-level view. User will create/ delete/ move a kitchen item, i.e., sink / table on the high-level view and if changes done on the high-level view are according to the rules defined in the bx tool then items will be reflected on the low-level view with same colored blocks and vice-versa. Basically the transformation will happen between a low-level language and a high-level language and bx rules control how updates are handled and states are reflected in the different views of the project.

[2] *short for User Interface*

Person and Family It contains two views e.g., Family view and Person view. In Family view, it contains many families and each family consist of members. Whereas, Person view contains persons (the members of each family). We assume that the surnames are unique and allow us to differentiate between different families. Addition of a new person to the Person view will be reflected on the family view and vice-versa. Also, due to the uniqueness of the surnames, person created will be automatically assigned to the related family. Bx rules control how updates are handled and states are reflected in the different views.

5.1.2 Selection

Selecting an example to implement through the demonstrator was not random, rather I have taken many factors into considerations before choosing the final one. It was a very important decision, as selection of the example and its implemenation will directly effect the research questions ***RQ2***, ***RQ4.2***, ***RQ6*** described in Section 1.3 and requirements ***NREQ2***, ***NREQ3*** described in Section 3.1.

Hence by taking into account all these factors, I have finally chosen the **Arranging a Kitchen** example to be implemented by the demonstrator as a part of my research. Following were the driving factors for the selection of the example:

- Any user can relate to the example very well as everybody is familiar with a kitchen and its environment.
- Example is very simple and no technical details are involved.
- Interactivity and rules won't be a overhead for the user, rather intuitive.
- Associted scenarios are part of day to day life, so user will be able to relate to the learning concepts through the example.

5.2 BX Tool Selection

Next step in the design process was selection of a bx tool which takes care of the bx part of the demonstrator and upon which the entire framework of the demonstrator will be constructed. My gathered information during the case study phase led the foundation for the selection process.

5.2.1 Theory

I further investigated on the existing bx tools from the point of view of practical application and usage of these tools in terms of building softwares. Even after a significant amount of work has

been done by developers community and bx community, the main problems are still revolving around below points [3]:

- the conceptual or theoretical challenges, practical challenges associated with using bx, and tool/technology challenges involved with building software systems that supported or exploited bx.
- limited benchmarks for comparison of complete bx solutions.
- no common repository of bx scenarios or problems that can be used to test and evaluate bx solutions.
- there exists tools to support particular bx scenarios but with very limited interoperability and integration.

To focus particularly on the above issues, bx-community had conducted a series of technical workshops at relevant conferences and organised week-long intensive research seminars in the year 2013 before the *BX 2014* workshop [3]. Some of the main outcome of these seminars are listed below:

- focus on the need of benchmarks and further categorizing them into functional and non-functional ones.
- scenarios for bx were developed based on database, synchronization, model-view architecture etc.
- software tool support for bx was shown in terms of demos which includes tool like eMoflon, Echo etc.

5.2.2 Selection

Again, it was a very important decision, as selection of the bx tool and the implemenation on the top of it will directly effect the research questions ***RQ1***, ***RQ4.1*** described in Section 1.3 and requirements ***NREQ1*** described in Section 3.1.

The outcome of the seminars as discussed in previous section 5.2.1 led me to concentrate and analyze the bx tools like eMoflon, Echo, BIGUL. Also, I came across the benchmark [5] [6], the first non-trivial benchmark where Anjorin et al. has provided a practical framework to compare and evaluate three bx tools. After analyzing these tools, benchmarks and taking into consideration the reserach questions and requiements, I have finally chosen **eMoflon** as the bx tool to handle the bx part of my demonstrator. Following were the driving factors for the selection of the bx tool:

- sample implementation with framework to implement the eMoflon tool was already available.

- my supervisor/prof. is a core member of the eMoflon community which gave me added advantage of knowing the tool inside out.
- extra knowledge about the tool can be really handy and it actually helped me in solving the implemenation issues/challenges regarding the tool as described in Section6.3.

5.2.3 Benchmarx

A bx benchmark (referred to as **Benchmarx** from now on) is a bx example that has a precise and executable definition of a binary consistency relation on source and target artefacts; an explicit definition of, or a generator for, input artefact elements; a set of precisely defined update scenarios for certain input artefact elements; and a set of executable metric definitions [3].

Anjorin et al. [6] tried to solve the main problem with benchmarking bx tools by creating a common design space, in which different bx tools architecture can be accomodated irrespective of the fact that they can have different input data.

Design Space

5.3 Architecture Design

Last step in the design process was application architecture design which is the most important part of my thesis and also the starting phase of the implementation of the prototype.

5.3.1 Design Decision

5.3.2 Framework

Figure 10 shows a very high-level architecture diagram of my prototype. It consist of 3 layers e.g., View, Controller and Model. View is responsible for all the graphical user interface management and consist of technologies like HTML, CSS, JavaScript and Jquery. Controller is responsible for event handling and basically consist of the technology Servlet. Model is responsible for all the tasks related to business logic, business rules, data, meta-models, state of meta-models etc. and consist of java classes.

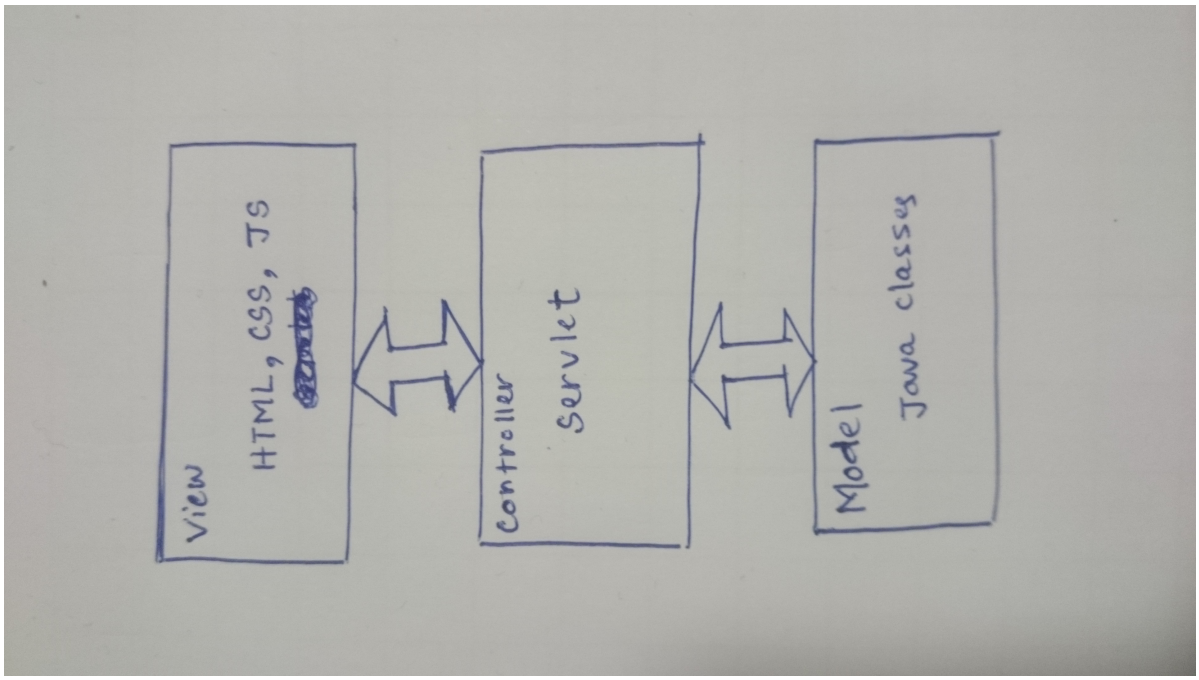

Figure 10: High Level Architecture Diagram

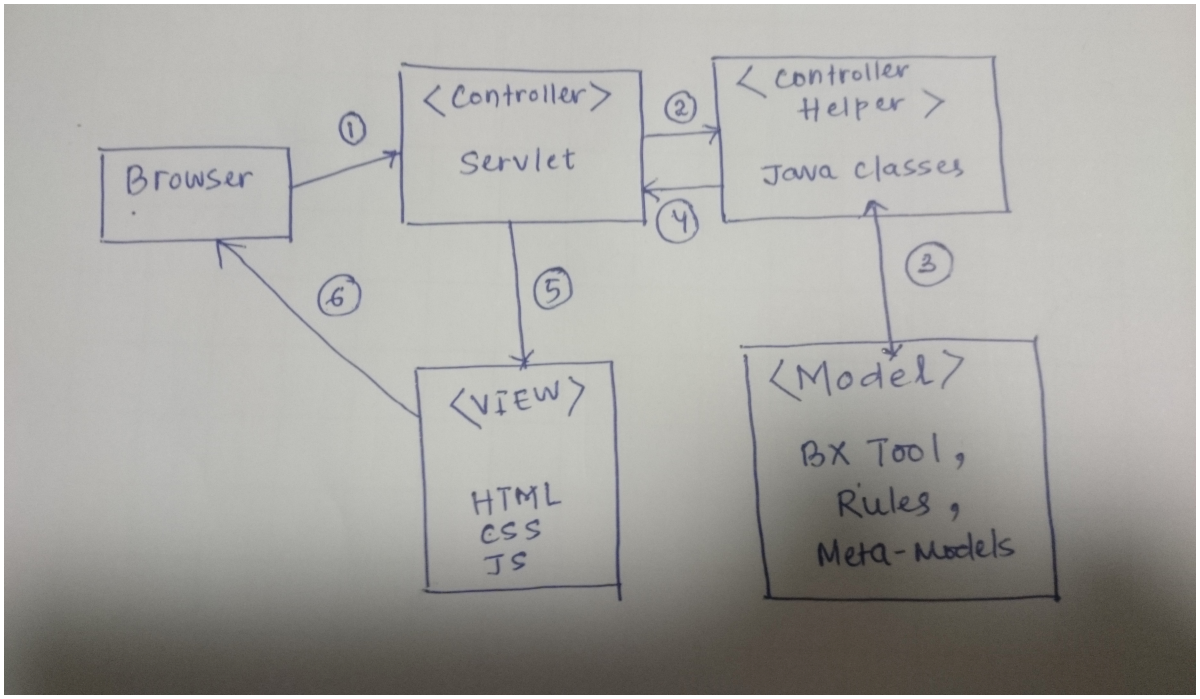

Figure 11: Detail Architecture Diagram

Request-Response Cycle Figure 10 shows a detailed architecture diagram with a complete request-response cycle.

5.3.3 Controller

5.3.4 Model

UI Models

Kitchen Layout

Grid Layout

5.3.5 View

External Design

Internal Design

5.4 Challenges

architecture design issues- dependencies of projects

UI design - high level/low level views, deltas, low level view user interaction(color), user interaction

5.5 Walkthrough

6 Implementation

6.1 UML Diagrams

6.1.1 Component Diagram

Component diagram of my prototype is given in Figure 12.

I am using Model-View-Controller (MVC) pattern for my application framework. On the top, `View` (Web Browser) component is present which contains a graphical user interface and functionalities that belong to the user. With the changes on `View` component, data are being provided through interface `IDoAnalysis` to the `Controller` component and after the calculations are done, the results are sent back to the `View` through interface `IProvideResults`. Both `View` and `Controller` resides on the same machine. `Model` (Bx Tool) component encapsulates and manages the state of all models by communication with the `Controller` through interface `IRules`.

6.1.2 Class Diagram

6.1.3 Sequence Diagram

6.2 Core Details

6.3 Challenges

Bx tool - implementing atomic deltas to track successful/failed changes translation, user interaction

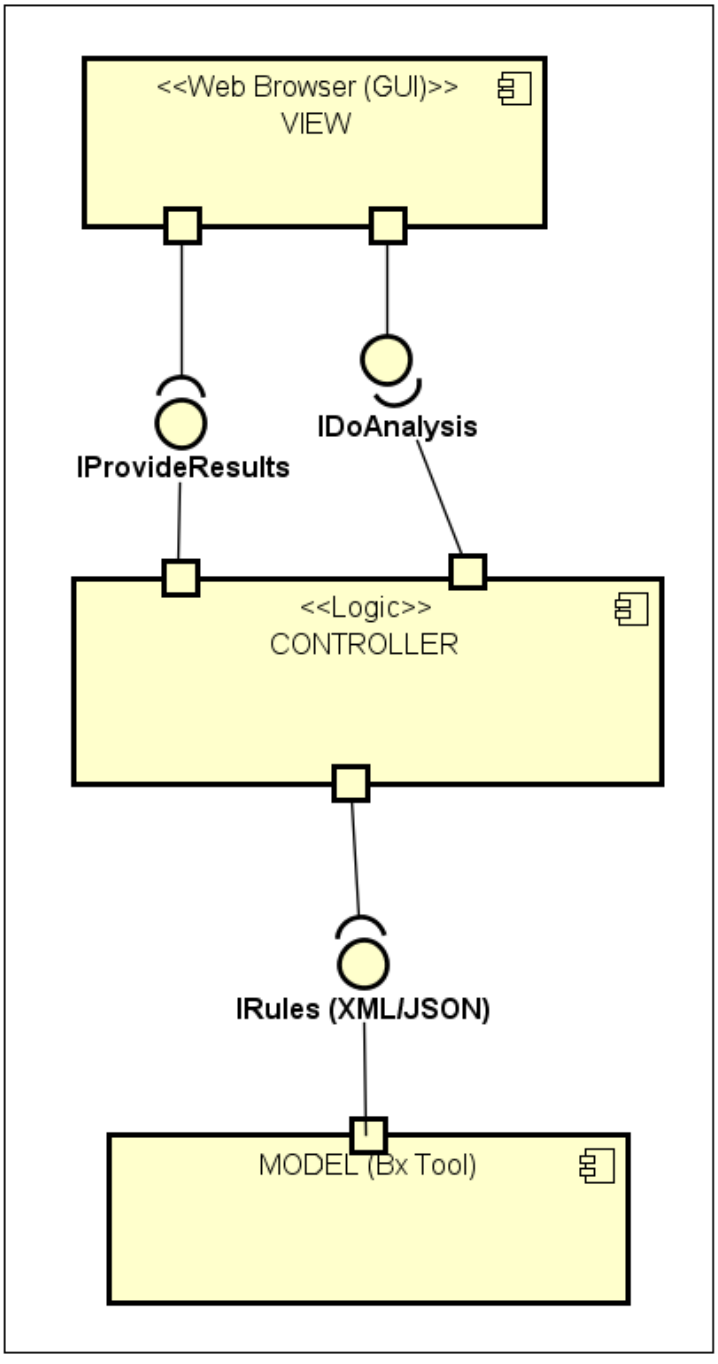

Figure 12: Component Diagram

7 Evaluation

7.1 Discussion and Feedback

7.2 Evaluation and Results

8 Summary and Future Work

8.1 Conclusion

Finding the limitations of the tool, benchmarx.

8.2 Future Work

1. Another tool 2. Another example 3. Tool as a webservice

9 Appendix

The Appendix can be used to provide additional information, e.g. tables, figures, etc.

List of Figures

1 Approach Overview 3
2 Modeling Example 7
3 Delta Propagation 8
4 Correspondence Links 9
5 Transformation 10
6 Forward and Backward Transformation 11
7 State Space 11
8 Bidirectional Transformation 12
9 Bijective Transformation 13
10 High Level Architecture Diagram 22
11 Detail Architecture Diagram 23
12 Component Diagram 25

List of Tables

1 Model and Reality 7
2 Examples of Delta 9

References

[1] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger, *Bidirectional Transformations: A Cross-Discipline Perspective*, GRACE International Meeting , Shonan, Japan, 2008. 1, 2, 14

[2] Z. Hu, A. Schürr, P. Stevens, and J. Terwilliger, *Dagstuhl Seminar* #11031 on Bidirectional Transformations "bx", Dagstuhl Reports, Vol. 1, Issue 1, pages 42-67, January 16-21 , 2011. 2, 14

[3] J. Gibbons, R. F. Paige, A. Schürr, J. F. Terwilliger and J. Weber, *Bi-directional transformations (bx) - Theory and Applications Across Disciplines*, [Online]. Available: https://www.birs.ca/workshops/2013/13w5115/report13w5115.pdf. 2, 20, 21

[4] R. Oppermann and P. Robrecht, *Benchmarks for Bidirectional Transformations.* Seminar Maintaining Consistency in Model-Driven Engineering: Challenges and Techniques, University of Paderborn: Summer Term 2016. 2

[5] A. Anjorin, A. Cunha, H. Giese, F. Hermann, A. Rensink, and A. Schürr, *Benchmarx*. In K. S. Candan, S. Amer-Yahia, N. Schweikardt, V. Christophides, and V. Leroy, editors, Proceedings of the Workshops of the EDBT/ICDT 2014 Joint Conference (EDBT/ICDT 2014), CEUR Workshop Proceedings, pages 82-86. CEUR-WS.org, 2014. 20

[6] A. Anjorin, Z. Diskin, F. Jouault, Hsiang-Shang Ko, E. Leblebici, and B. Westfechtel, *Benchmarx Reloaded: A Practical Benchmark Framework for Bidirectional Transformations*. In R. Eramo, M. Johnson (eds.): Proceedings of the Sixth International Workshop on Bidirectional Transformations (Bx 2017), Uppsala, Sweden, April 29, 2017, published at http://ceur-ws.org. 6, 8, 10, 20, 21

[7] A. Schürr. *Specification of graph translators with triple graph grammars.* In E. W. Mayr, G. Schmidt, and G. Tinhofer, editors, *Graph-Theoretic Concepts in Computer Science, 20th International Workshop, WG 94*, volume 903 of LNCS, pages 151-163, Herrsching, Germany, June 1994. 15

[8] A. Bucaioni and R. Eramo, *Understanding bidirectional transformations with TGGs and JTL*, in Proc. of the Second Workshop on Bidirectional Transformations (BX 2013), no. 57, 2013. 15

[9] A. Anjorin, E. Burdon, F. Deckwerth, R. Kluge, L. Kliegel, M. Lauder, E. Leblebici, D.Tögel, D. Marx, L. Patzina, S. Patzina, A. Schleich, S. E. Zander, J. Reinländer, and M. Wieber, *An Introduction to Metamodelling and Graph Transformations with eMoflon. Part IV: Triple Graph Grammars.* [Online]. Available: https://emoflon.github.io/eclipse-plugin/release/handbook/part4.pdf 2, 15

[10] BX Community, [Online]. Available: http://bx-community.wikidot.com/ 2

[11] BX Community, [Online]. Available: http://bx-community.wikidot.com/examples:home 16

[12] N. Macedo, T. Guimaräes and A. Cunha, *Model repair and transformation with Echo*. In Proc. ASE 2013, ACM Press, 2013. 2

[13] Hsiang-Shang Ko, Tao Zan, and Zhenjiang Hu, *BiGUL: A formally verified core language for putback-based bidirectional programming.* In Partial Evaluation and Program Manipulation, PEPM'16, pages 61-72, ACM, 2016. 15, 16

[14] Zhenjiang Hu and Hsiang-Shang Ko, *Principle and Practice of Bidirectional Programming in BiGUL*, Tutorial [Online]. Available: http://www.prg.nii.ac.jp/project/bigul/tutorial.pdf 16

[15] *BiYacc, tool designed to ease the work of writing parsers and printers*, [Online]. Available: http://biyacc.yozora.moe/ 15

[16] *SHARE - Sharing Hosted Autonomous Research Environments*, [Online]. Available: http://is.ieis.tue.nl/staff/pvgorp/share/ 15

[17] *Wikipedia. "Model transformation."* Retrieved March, 2017, from https://en.wikipedia.org/wiki/Model_transformation.

[18] S. Easterbrook, J. Singer, M.-A. Storey, & D. Damian. *Selecting Empirical Methods for Software Engineering Research. Guide to Advanced Empirical Software Engineering*, 285-311, 2008. Retrieved from http://doi.org/10.1007/978-1-84800-044-5_11 3, 5

[19] J. Rumbaugh, I. Jacobson and G. Booch, *Unified Modeling Language Reference Manual, The (2nd Edition)*, Pearson Higher Education, pages 1-21, 2004. 6

[20] S. Beydeda, M. Book and V. Gruhn, *Model-Driven Software Development*, ACM Computing Classification, pages 1-8, 1998. 6

[21] S. Sendall and W. Kozaczynski, *Model transformation: the heart and soul of model-driven software development*, IEEE, Vol. 20, Issue: 5, pages 42-45, Sept.-Oct. 2003. 1

[22] S. Sendall and W. Kozaczynski, *Model transformation: the heart and soul of model-driven software development*, IEEE, Vol. 20, Issue: 5, pages 42-45, Sept.-Oct. 2003. 1