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, June 17, 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 and Motivation

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

One of the approaches for reducing the complexity of software development is abstraction and separation of concerns [23]. In recent times, (software) modeling has become an effective way of implementing this principle. In a traditional approach, developers manually write programs and check the specifications, which is often costly, incomplete, informal, and carries a major risk of failure. In contrast, model-driven software development (referred to as **MDSD** from now on) improves the way software is built by moving the focus from code to representing the essential aspects of software in the form of software models [23]. It reduces development costs and increases the reusability and maintainability of software. The objective of MDSD [23] 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, rather than what is directly offered by programming languages.

The core idea of the MDSD approach is based on models, modeling and model transformations. In this approach, developers represent real world systems as models at a suitable level of abstraction. Different models can be used to represent different views of a system. Although these views are separate and result in models that can be independently manipulated by different developers, there are still numerous relations between models that must be taken into account to ensure that the entire system, described by the state of all models, is consistent. This can be handled by model transformation to increase the developers' productivity and quality of the models [22].

Bidirectional transformation (referred to as **bx** from now on) is a technique used to synchronize two (or more) models. Such models are related but do not necessarily contain the same information. Changes in one model can thus lead to changes in other models [1].

Bidirectional transformation is used to deal with scenarios like [3]:

- change propagation to the user interface as a result of underlying data changes
- synchronization of business/software models
- refreshable data-cache in case of database changes
- consistency management between two artifacts by avoiding data loss

The bx community (`http://bx-community.wikidot.com/`) has been doing research in many fields including software development, databases, mathematics and much more, to increase awareness for bx [1][2]. As a result, many kinds of bx tools are being developed. These bx tools are based on various approaches, such as graph transformations e.g., eMoflon [9], bidirectionalization e.g., BIGUL [13], constraint solving e.g., Echo [12] and can be used in different areas of application [10].

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 on bx, a general awareness and understanding for basic concepts, the involved challenges, and reasonable expectations are not really given. Hence, there exist conceptual and practical challenges with building software systems using bx-tools and as a result, bx tools and their applicability is still not widely known and used [3].

From experience with working with master students (future software developers) and bx researchers, we have identified that imparting knowledge about certain core bx concepts can improve the situation. However, the problem is that it is relatively hard and challenging to do this as current possibilities (virtual machines, handbooks, etc) are either tool specific or ineffective because installation process to get the tool running is a time-consuming process and sometimes requires technical expertise in a specific area/tool/programming language. Even after you get the tool running, it doesn't necessarily help in understanding bx concepts because of lack of proper explanations. Just examples are not interactive enough, anything tool-specific gets into too much detail of the respective tool, etc. So a platform on which one can easily prepare high-quality, easily understandable, and engaging learning material for spreading the bx concepts is missing.

To keep a focus on the above described problem statements and to relate to the work done directly or indirectly during my entire thesis, I have formulated the following associated research questions (referred to as **RQ** from now on):

RQ 1: What are the core requirements for implementing a successful bx demonstrator?

RQ 2: Which goals can be particularly well addressed in a bx demonstrator and why?

RQ 3: Is it possible to teach the concepts of bx through a demonstrator?

RQ 4: Does an interactive GUI helps a user to increase his/her understanding related to bx concepts?

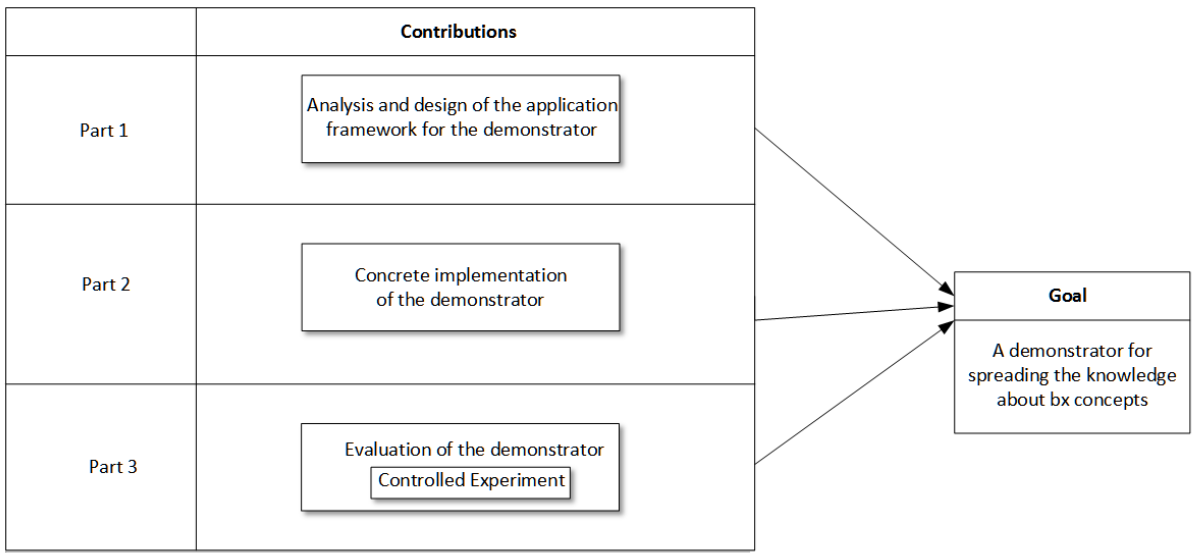

Contributions
Part 1
Analysis and design of the application framework for the demonstrator
Part 2
Concrete implementation of the demonstrator
Part 3
Evaluation of the demonstrator
Controlled Experiment
Goal
A demonstrator for spreading the knowledge about bx concepts

Figure 1: Contributions

1.2 Contribution

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

- Design and implement an interactive demonstrator.
- Spread 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.

Besides giving an overview of previous work and comparing their advantages and disadvantages with my demonstrator, Figure 1 shows my contributions towards providing a solution to the problems described in Section 1.1. It can be categorized into three parts:

1. Analysis and design of the application framework for the demonstrator
2. Concrete implementation of the demonstrator
3. Evaluation of the demonstrator

After analyzing the existing work on bx and their related problems (detail explanation described in Section 4), to solve the problems in hand, designing an application framework for the implementation of an interactive demonstrator is needed. I have designed a working, fully functional application framework based on MVC pattern for implementing a bx tool demonstrator along with an interactive user interface for the accomplishment of point (1). This framework can be used to implement a demonstrator encapsulating a bx tool with a concrete example.

Along with this achievement, a concrete implementation of a demonstrator can be done to check the validity of the application framework. I have implemented a fully functional online demonstrator based on the application framework designed earlier for the accomplishment of point (2). This demonstrator is based on a bx tool i.e., eMoflon and a concrete example leveraging the functionalities of the bx tool and explaining some basic bx concepts.

After the implementation, evaluation of the demonstrator to check the effectiveness and its impact on the mass audience is needed. I have evaluated the demonstrator based on the research method called controlled experiments [20] for the accomplishment of point (3). This experiment is based on one or more hypothesis, which guides all steps of the experimental design, deciding which variables i.e., dependent and independent variables to include and how to measure them. A well-designed experiment is performed with different groups of participants, data is collected and analyzed to measure the outcome.

All of my contributions help to achieve the goal of this thesis i.e., a demonstrator for spreading the knowledge about bx concepts.

1.3 Structure of Thesis

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 their related problems. Chapter 5 describes all the high-level design details and decisions done for each layer along with UML diagrams during the implementation work. Chapter 6 contains the learning goals (based on research questions), evaluation methodology, and evaluation results. The last chapter summarizes all the work which was done as part of this thesis and draws conclusions followed by future work.

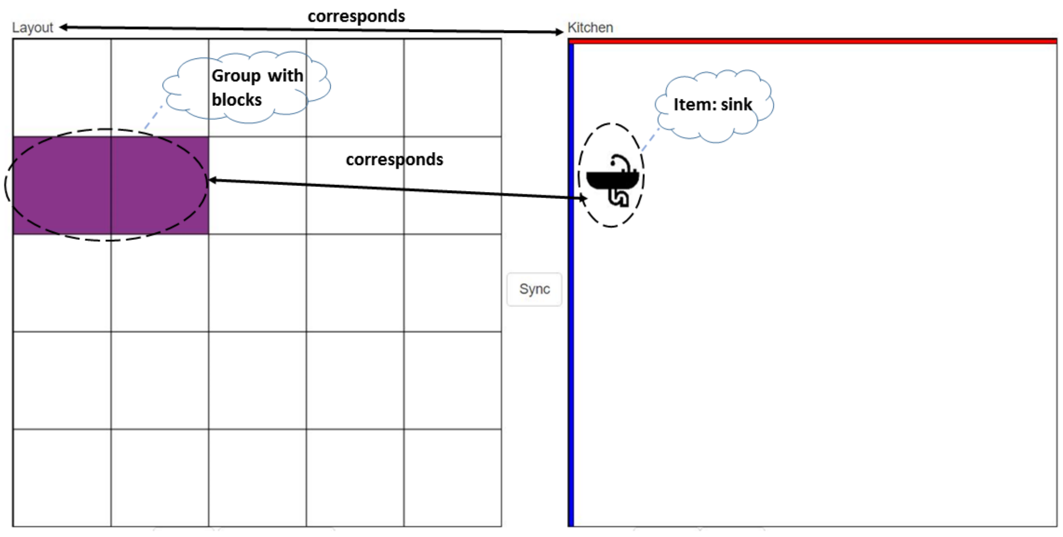


Figure 2: Running Example: Layout and Kitchen

2 Foundation

This chapter provides an overview of my running example in Section 2.1 followed by definitions of some commonly used terminologies with respect to bx in Section 2.2. This chapter will lay the foundation for understanding the basics for the reader and will help him/her in apprehending the concepts explained in further chapters.

In this thesis, bidirectional transformation will be discussed by referring to a *Kitchen Model* and a *Grid Model* of a software system. Both models describe the structure and behavior of the real system "kitchen" but from different perspectives.

2.1 Running Example

This example is a simplified kitchen planner. Figure 2 describes the relation between the *Kitchen Model* denoted by *Kitchen* (right-hand side canvas) and the *Grid Model* denoted by *Layout* (left-hand side canvas).

Kitchen contains items e.g., sink, fridge, table etc. You can create, delete or move these items in the kitchen, and press "Sync" to propagate your changes to the layout. The *Layout* consists of groups of a certain number of blocks. This shows how much space the objects occupy as

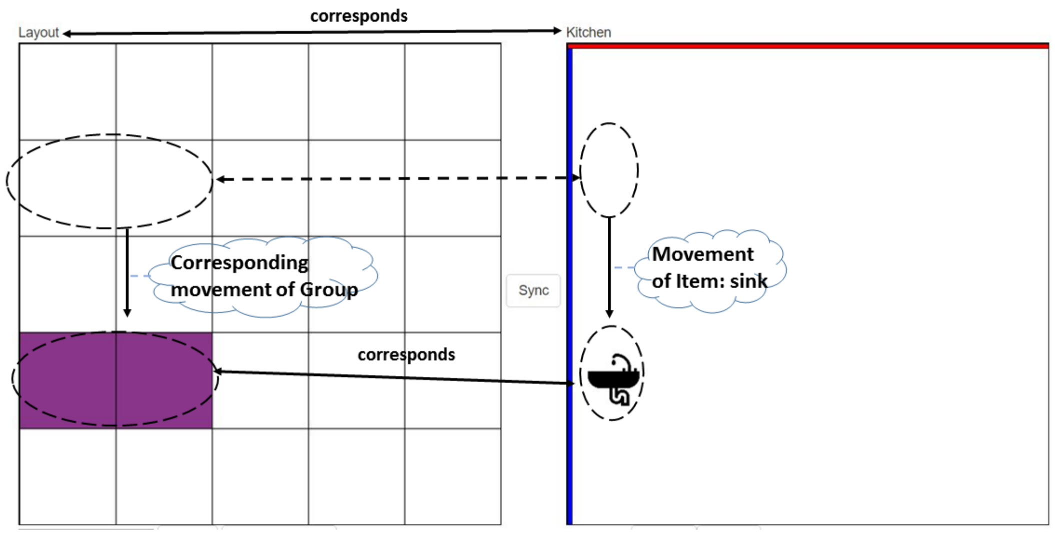


Figure 3: Running Example: Consistency Preservation

colored groups of blocks organized in a grid. Here, the *Kitchen* corresponds to the *Layout* and a single *Item* corresponds to a *Group*. This is described in Figure 2 where an item e.g., *sink* created in the *Kitchen* corresponds to a group in the *Layout*.

Both artifacts are created and evolve together during the lifecycle of the application representing a kitchen workspace. Thus, changes in one domain should be propagated to the other domain in order to ensure consistency between these related artifacts. This is shown in Figure 3.

Model transformation and synchronization is discussed with this scenario throughout this thesis.

2.2 BX Basics

Definition 1 *(Model and Meta-Model)*
A model depicts the structure and/or behavior of a real system under discussion from a certain point of view and at a certain level of abstraction which helps in managing and understanding the complexities of a system [21] [22]. Model creation helps in keeping a clear focus on selected concepts and rules relevant for a particular concern and omitting irrelevant details.

A metamodel describes a set of models, i.e., defines a modeling language.

Example: In my demonstrator, the example that I have two meta-models i.e., *Kitchen* and *Grid*.

Reality	Model	Aspects Covered
Kitchen	Kitchen Model	• Area of a kitchen as a white space • 4 walls of a kitchen • Contains the objects of a kitchen • Creation, movement, deletion of kitchen objects
Kitchen	Grid Model	• Area of a kitchen as a block structure • 4 walls of a kitchen • Contains the objects of a kitchen • Creation, deletion of kitchen objects

Table 1: Model and Reality

Both the models represent the reality "Kitchen". A relation between reality and models is shown in Table 1.

Figure 4 and Figure 5 depict the *Grid* and *Kitchen* meta-model respectively as a class diagram from an object oriented point of view showing the related classes and the relationship between them.

Figure 6 illustrates an abstract and concrete view of the *Kitchen* model. The left-side figure shows an abstract syntax i.e., an instance of the kitchen. Whereas, right-side figure describes a concrete syntax of the kitchen as visualized in the user interface.

Definition 2 *(Delta)*
A delta is a change applied to one or more properties of an artefact. It denotes the relationships between models from the same model space [6]. It is denoted δ: M $\longrightarrow$ M' where M' is an updated version of M.

Example: In my demonstrator example, deltas related to the "Kitchen Model" are described in Table 2. Whereas, Figure 7 shows a concrete example of delta propagation, where the creation of a new item e.g., sink causes the change from Kitchen to Kitchen'.

Definition 3 *(Model Space)*
A Model Space describes all the states of an artefact (models of the same type) and all the deltas which lead from one model to another.

Example: In my demonstrator example, a concrete example of a subset of a model space is shown in Figure 8. GS contains the states i.e., Grid1, Grid2, Grid3 along with deltas i.e., $\delta 1$,

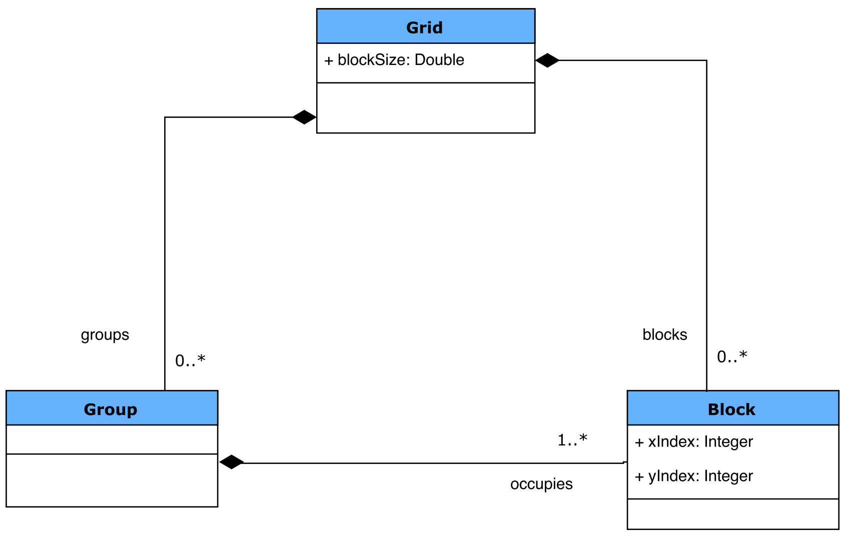


Figure 4: Grid MetaModel

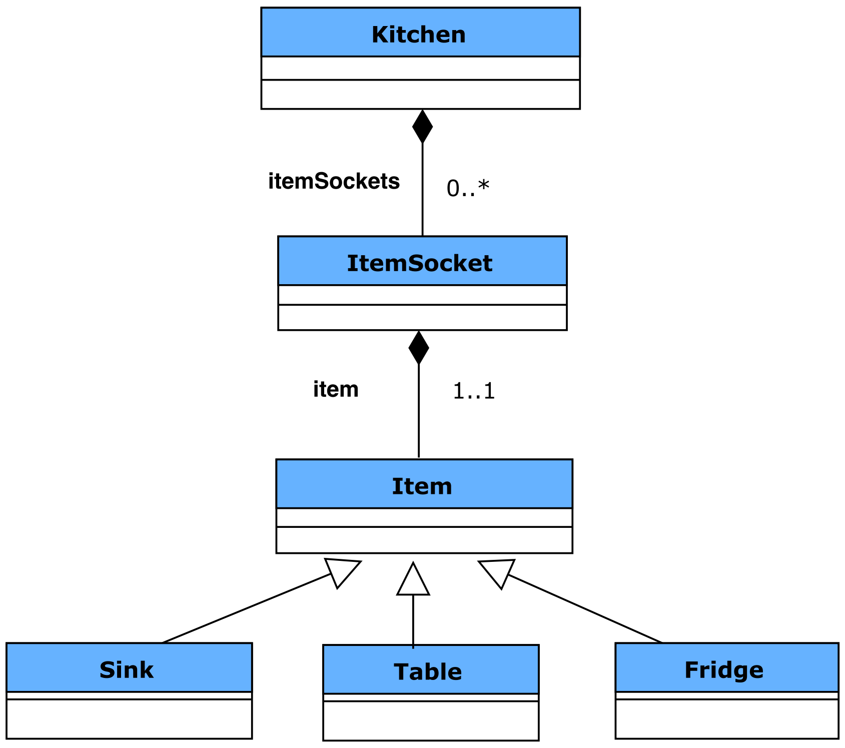


Figure 5: Kitchen Meta-Model

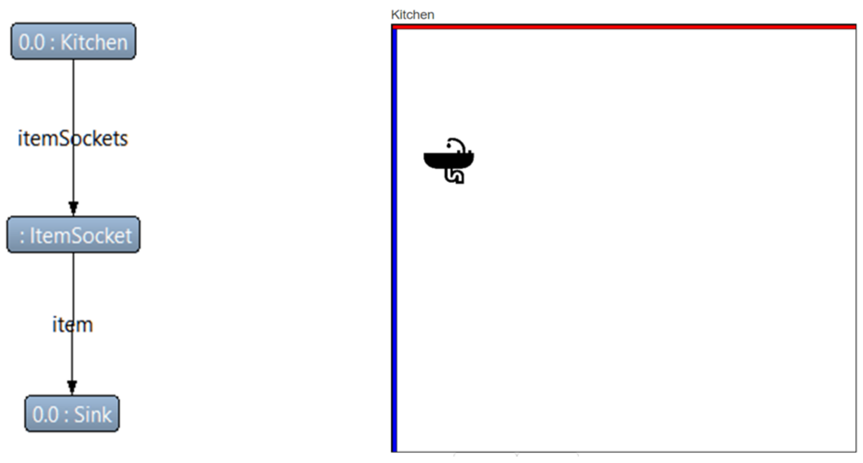


Figure 6: Kitchen Model (Abstract & Concrete)

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

Table 2: Examples of Delta in Kitchen Model

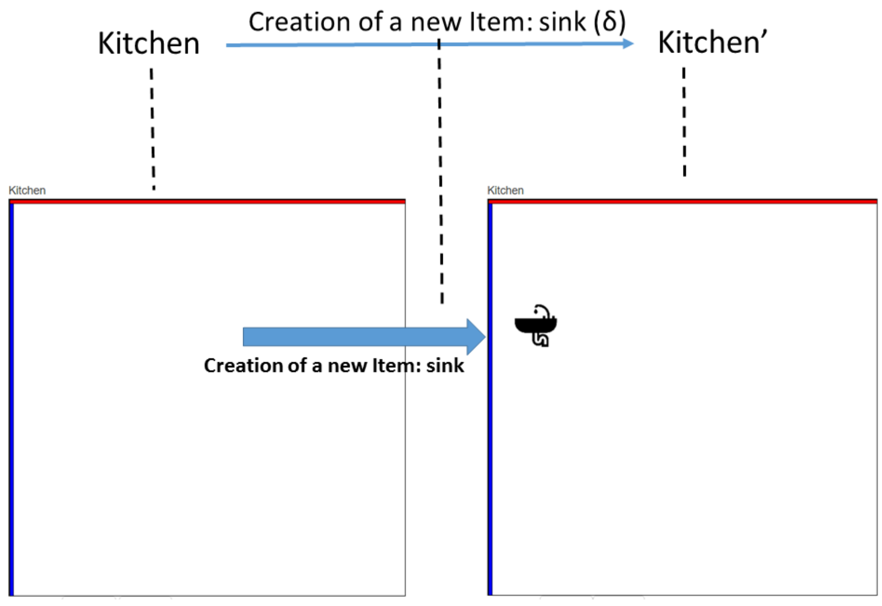

Creation of a new Item: sink (δ)
Kitchen
Kitchen'
Kitchen
Kitchen
Creation of a new Item: sink

Figure 7: Delta Propagation

$\delta 2$ of the *Grid* Model. Whereas, KS contains the states i.e., Kitchen1, Kitchen2, Kitchen3 along with deltas i.e., $\delta 3$, $\delta 4$ of the *Kitchen* Model.

Definition 4 *(Correspondence Links)*
Relationships between models from different model spaces are called correspondence links, or just corrs [6]. Corrs are used to relate two model spaces. A corr is a set of links c(a; b) between models (a in A, b in B), where a, b are models in model spaces A and B respectively.

Example: In my demonstrator example, an example of a corr c(Grid2; Kitchen2) is shown in Figure 8. A concrete example of the corr is given in Figure 9.

Definition 5 *(Consistency)*
Changes in one model may or may not cause any change in another correspondence model (model from different model space) but their states must not contain any contradiction. Consistency in model transformation is measured on the correctness and completeness of the operations performed [19]. For example, consistency between two models is ensured when a change in one model causes correct restoration of all the association/corrs between the models (correctness) and all valid inputs are propagated (completeness).

Example: In my demonstrator example, a *Kitchen* model is consistent with a *Grid* model when a mapping between *itemSocket* and *group* can be established such that itemSocket containing an item is paired with a group containing block(s). Figure 2 and Figure 3 describe a concrete

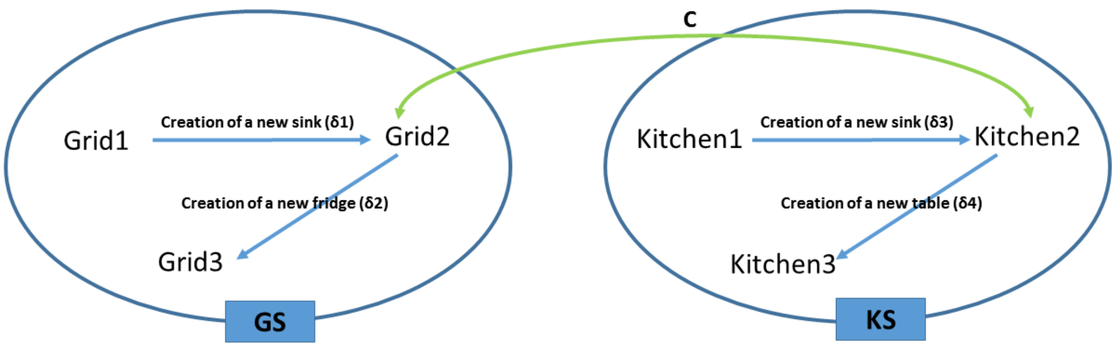

c
Grid1
Creation of a new sink (δ1)
Grid2
Creation of a new fridge (δ2)
Grid3
GS
Kitchen1
Creation of a new sink (δ3)
Kitchen2
Creation of a new table (δ4)
Kitchen3
KS

Figure 8: Model Space

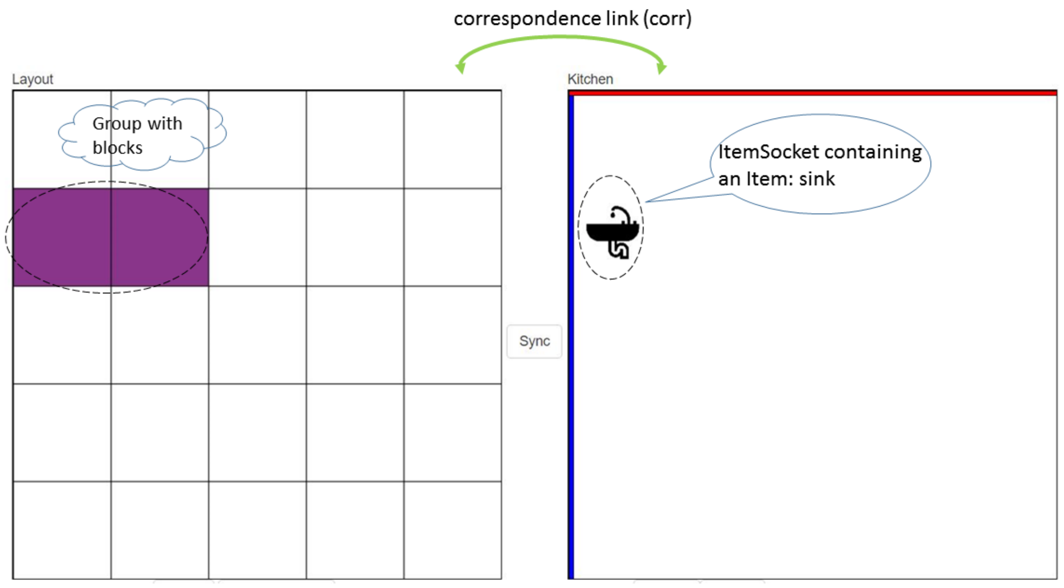

correspondence link (corr)
Layout
Group with blocks
Sync
Kitchen
ItemSocket containing an Item: sink

Figure 9: Correspondence Links

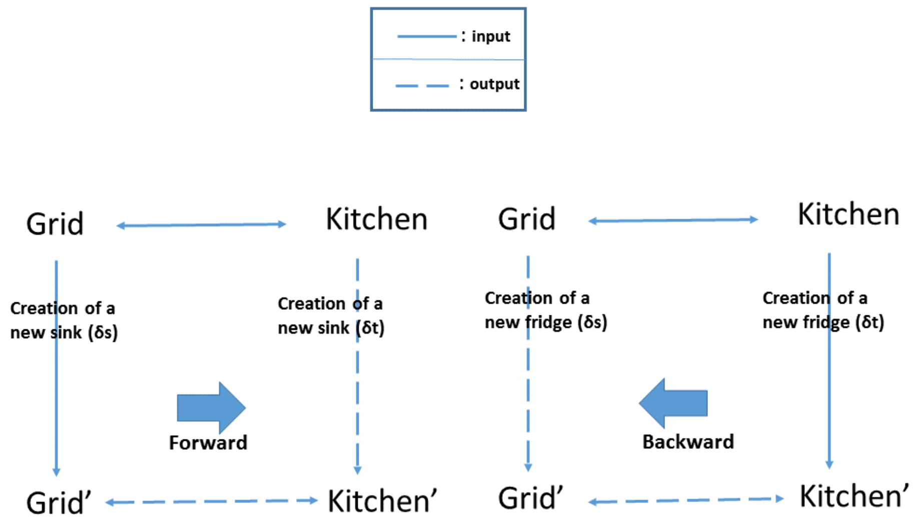

: input
: output
Grid
Kitchen
Grid
Kitchen
Creation of a new sink (δs)
Creation of a new sink (δt)
Creation of a new fridge (δs)
Creation of a new fridge (δt)
Forward
Backward
Grid'
Kitchen'
Grid'
Kitchen'

Figure 10: Bidirectional Transformation

example of consistency preservation where, changes in *Kitchen* (Kitchen model) i.e., movement of the sink should ensure the movement of the corresponding group with two horizontal blocks in *Layout* (Grid model) in order to ensure consistency.

Definition 6 *(Forward and Backward Transformation)*
Model transformation is defined as a process for propagating the changes (deltas) from one model of one domain to a corresponding model in another domain using forward and/or backward transformation operations. After a transformation operation, consistency of source and target model is always ensured as model transformation ensures that for each consistent source model there exist a consistent target model [19].

In forward transformation, only the changes that occur in the source model is propagated to the target model ensuring the consistency between source and target model.

In backward transformation, only the changes that occur in the target model is propagated to the source model ensuring the consistency between source and target model.

Given two models, the bidirectional transformation is a pair of transformation which takes place in both forward and backward direction maintaining the consistency relation between them [18].

Example: In my demonstrator example, "Grid" is the *Source* model and "Kitchen" is the *Target* model. Forward transformation will cause "Grid Model" to transform into "Kitchen Model" and Backward transformation will cause "Kitchen Model" to transform into "Grid Model". Figure 10 shows an abstract example of the "BX" describing both forward and backward

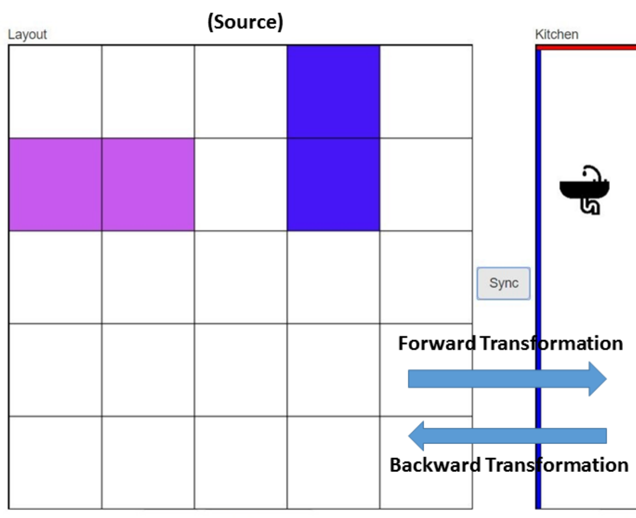


Figure 11: Transformation (Concrete Diagram)

transformation. Whereas, Figure 11 depicts a concrete example of the "BX" process where *Layout* and *Kitchen* are described as source and target respectively in the process of forward and backward transformation.

3 Requirements

This chapter explains all the requirements needed to implement a successful demonstrator. Section 3.1 describes all the functional requirements. Afterward, Section 3.2 specifies all the non-functional requirements.

3.1 Functional

Functional requirements define the behavioral attributes of a system [37] or in simple terms what a system should do.

As my thesis is more focused on the implementation of a demonstrator for consistency management based on a bx tool, following are some of the functional end results (referred to as **FR** from now on) I have implemented in the demonstrator:

FR1: User should be able to manipulate models' instances

Being a demonstrator for bx tool, a user must go through the entire process by himself/herself in order to try and learn bx concepts out of the demonstrator. During the process, the user should be able to play with the models' data i.e., Kitchen and Grid and manipulate them with the set of actions e.g., creation, deletion etc. defined separately in both the views i.e., Kitchen and Layout.

FR2: User should be able to trigger the synchronization process

Rather being automatic, the synchronization process should be triggered by the user during the process of trying and learning bx concepts out of the demonstrator. After manipulating the models' data, the user should decide when to initiate the synchonization process and visualize the views with updated models.

FR3: User can make changes only in one view before synchronization

During the process of manipulation of models' data before the initiating the synchronization process, a user can make changes only in one view i.e., Kitchen or Layout but not in both. The synchronization process will be initiated only if one of the views contains changes otherwise, changes will be discarded and the user has to start the process once again from the previous consistent state.

FR4: User can reset the models' states at any time

A user can reset the models' states and the views at any time during the process and start from the beginning.

3.2 Non-Functional

Non-functional requirements cover all the remaining requirements which are not covered by the functional requirements. Hence it describes all the quality attributes of a system [37].

Following are some of the non-functional end results (referred to as **NR** from now on) I have implemented in the demonstrator:

NR1: Reduce the installation time of the bx tool

The demonstrator should not take much time to install and should be up and running very fast. By making the demonstrator available online, the user needs nothing to install on his/her machine. The user has to enter the url in the browser and hence just a click away from trying the demonstrator based on a bx tool.

NR2: Demonstrator should be fun to play with.

The demonstrator should be able to attract more users by exploiting the features of the GUI and colors. It should be interactive and fun to play with during the entire process so that the user sticks to it.

NR3: Demonstrator's example should be simple

The user is going to try/use the demonstrator based on an example. This example should rather be less complex and less technical to understand to create interest and convince more target audience. Also, the user should be able to relate to the bx concepts through the example.

NR4: Demonstrator should be able to guide the user

During the entire time, while the user is using the demonstrator, it should be able to provide clear instructions/guidelines on what the user can do and try with it. The user should not feel like what to do with the demonstrator after a few minutes. Demonstrator should guide the user through all the aspects that are intended to be taught/learned with it.

NR5: Demonstrator should be informative

Rather being just an online playing tool, the user should learn certain concepts of bx by using the demonstrator. The user should be able to relate to the concepts taught by the demonstrator with its scenarios.

NR6: Public access

The demonstrator should be accessible worldwide to all. Being a demonstrator for a bx tool, it's accessibility shouldn't be restricted to a few groups. rather, it should be accessible to all whoever wants to use the tool.

NR7: Robust & Platform independent

Application architecture should be easily maintainable and extendable. With changing requirements and needs, application architecture should be able to accommodate new changes and future enhancements. Also, the application should be platform independent i.e., it should run on any operating system and all kinds of browser. Different environment on user's machine should not affect the application.

NR8: Non-Occupancy of Users' machine space

The application should occupy very less or no space on users' machine during installation or while running the example.

NR9: Easily deployable project

Application's deployment process to get the example running should not be complex. Rather, it should involve simple steps so that it can be carried out by anyone in no time.

NR10: Products, Technologies, and tools

Updated version of the products, technologies, and tools should be used as per the requirements during the implementation work so that future enhancements can be carried out easily.

4 Related Work

This chapter sums up all the related work that has been done on bx. Section 4.1, 4.2, 4.3, and 4.4 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 and their comparison with the Demon-BX tool.

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.

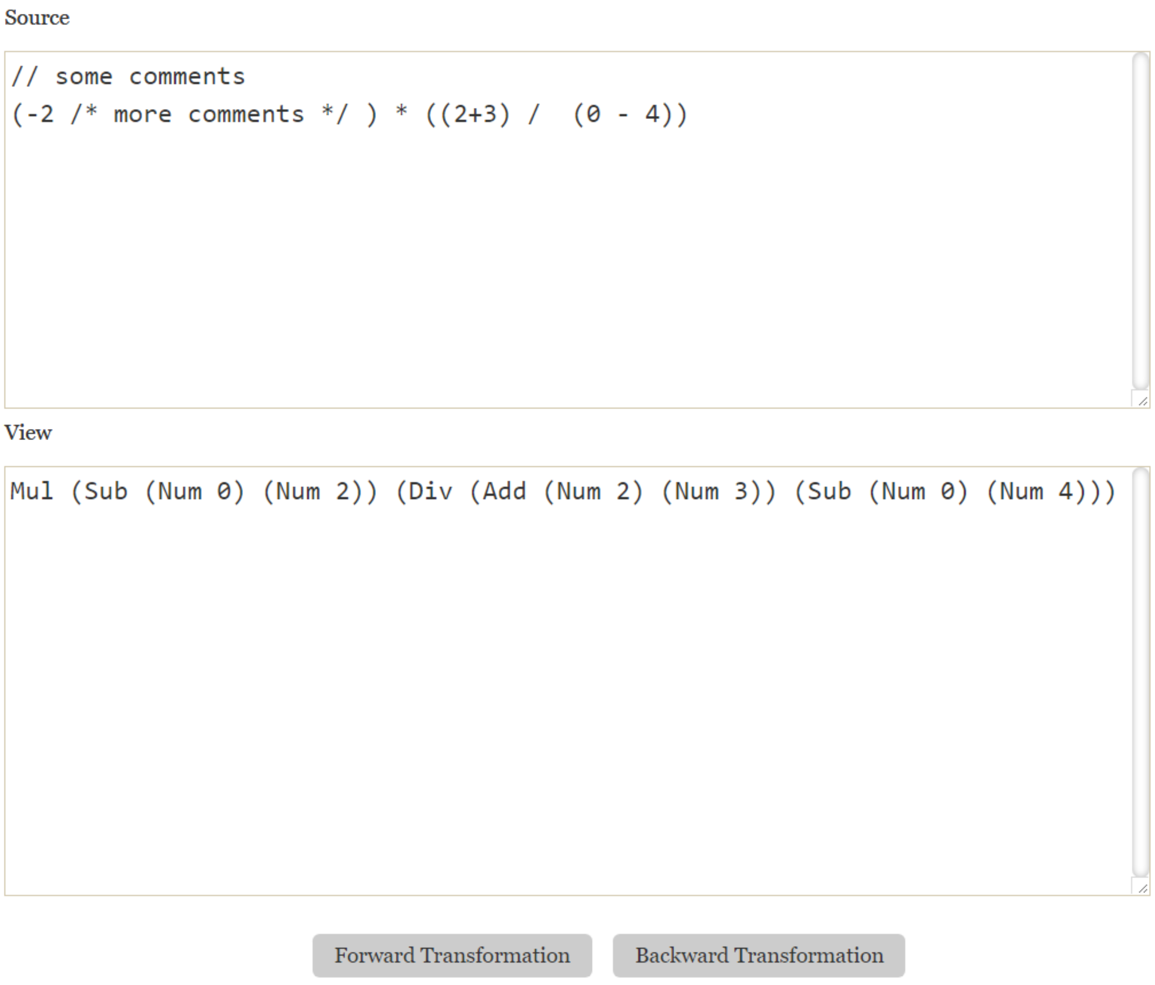
4.1 Existing Demonstrator

First, I analyzed an existing demonstrator available along with the test cases of a domain-specific language, BiYacc [15] which is based on BiGUL [13]. BiYacc designed to keep the parsers and printers needed by the language designers unified, in a single program, and consistent throughout the evolution of a language. Based on bidirectional transformations theory, BiYacc constructs the pairs of parsers and reflective printers and guarantees that they are consistent.

Figure 12 shows the web interface of the demonstrator. It contains two views i.e, source and view. Depending on the user selection, "source" contains an arithmetic expression or a program with some comments. After the forward transformation is done, "view" is loaded with the parsed version (machine readable format) of the sources' text.
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.

4.2 Virtual Machines

Secondly, I 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, software, operating systems, etc., which are otherwise a headache to install [16]. Also, I have analyzed VM Virtual Box [17], an open source virtualization product used for creating and sharing enterprise, academic, and personal work.



```
// some comments
(-2 /* more comments */ ) * ((2+3) /  (0 - 4))
```

```
Mul (Sub (Num 0) (Num 2)) (Div (Add (Num 2) (Num 3)) (Sub (Num 0) (Num 4)))
```

Figure 12: Web GUI for Demonstrator: Biyacc

These virtual machines provide 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. To access the shared material, virtual machine either have to accessed online or deployed on users' machine.

4.3 Handbooks & Tutorials

As a part of my research, I also 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 finally he/she can view the final output. It took me 4 days to get the tool up and running.

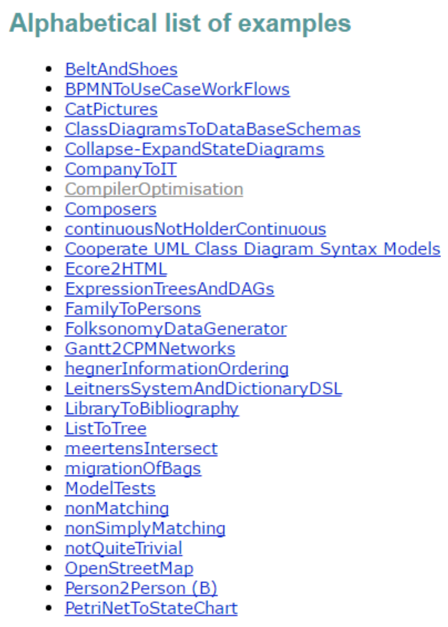
We have analyzed a tutorial[14] on a bidirectional programming language BiGUL[13]. The core idea with BiGUL is to write only one putback transformation, from which the unique corresponding forward transformation is derived for free. The whole tutorial is about 45 pages long which includes a lot of complex 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.

Figure 13 shows a screenshot of the example repositories listed alphabetically on the bx community web-site (`http://bx-community.wikidot.com/`). 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.

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


Alphabetical list of examples

- BeltAndShoes
- BPMNToUseCaseWorkFlows
- CatPictures
- ClassDiagramsToDataBaseSchemas
- Collapse-ExpandStateDiagrams
- CompanyToIT
- CompilerOptimisation
- Composers
- continuousNotHolderContinuous
- Cooperate UML Class Diagram Syntax Models
- Ecore2HTML
- ExpressionTreesAndDAGs
- FamilyToPersons
- FolksonomyDataGenerator
- Gantt2CPMNetworks
- hegnerInformationOrdering
- LeitnersSystemAndDictionaryDSL
- LibraryToBibliography
- ListToTree
- meertensIntersect
- migrationOfBags
- ModelTests
- nonMatching
- nonSimplyMatching
- notQuiteTrivial
- OpenStreetMap
- Person2Person (B)
- PetriNetToStateChart

Figure 13: BX Example Repositories

4.5 Discussion

In this section, I will discuss the related problems associated with each of the related work discussed in the previous sections and a comparison of these related work with the Demon-BX tool.

4.5.1 Related Problems

Associated problems that I found with the above discussed related work are described in the following paragraphs:

Existing Demonstrator 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.

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.

Virtual Machines For security reasons, web-based virtual machine, e.g., SHARE is 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.

The user needs at least 3 Gigabyte or more space on his/her local system for configuring the virtual machine i.e., VM Virtual Box depending on the requirement of the environment, which sometimes creates an overhead.

Handbooks & Tutorials As described in handbooks and tutorials, 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., the 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 and also, not very helpful to understand what bx is before deciding which bx tool to use.

Example Repositories In today's fast paced and visually enriched world, "what you see is what you believe" and that is exactly what these examples are lacking in. None of the examples have a demonstrator showing how it works. Hence, it is very difficult for a user to realize the examples just by going through the documentation and UML diagrams. Even if the source code is present, it takes a lot of time and requires technical expertise to set-up the framework and get the example running.

4.5.2 Comparison

Associated problems as discussed in Section 4.5.1 gave me the foundation for forming the requirements of the Demon-BX tool. I took the shortcomings from each related work and designed the requirements i.e., functional and non-functional.

Table 3 illustrates a comparison between all the related work discussed above with Demon-BX tool based on the requirements described in Section 3. "✓" denotes the fulfillment of the requirement, "✗" denotes the non-fulfillment of the requirement, and "?" denotes insufficient information related to the requirement. This comparison table clearly shows that Demon-BX tool overcomes all the problems associated with these related work.

	Requirements													
Related Work	**FR1**	**FR2**	**FR3**	**FR4**	**NR1**	**NR2**	**NR3**	**NR4**	**NR5**	**NR6**	**NR7**	**NR8**	**NR9**	**NR10**
Existing Demon.	✗	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	?	?
Virtual Machines	✓	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
Handbooks & Tutorials	✓	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
Example Repositories	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Demon-BX	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 3: Comparison of related work and Demon-BX based on requirements

5 Design and Implementation

In this chapter, I am going to describe the design and implementation steps realized for the demonstrator. Section 5.1 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. Afterward, Section 5.3 deals with the decisions taken for finalizing the application's architecture design. Then, Section 5.4 describes the application's framework in detail along with its components. Section 5.5 describes the challenges that I faced during the design and implementation of the entire framework. At last, Section 5.6 describes some of the choices that I made during the design phase and its associated threats.

5.1 Choosing an Example

To solve the problems as described in Section 1.1, 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., a low-level view (depicts UI[2] for low-level language, i.e., UI with less functionality) and a high-level view (depicts UI for high-level language, i.e., UI with more functionality). The 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., layout view (depicts a grid structure containing blocks) and symbolic view (empty space which depicts UI for a kitchen). The symbolic view has more functionalities such as creating/deleting/moving a kitchen item, etc. out of which only a few will be available in layout view. The user will create/delete/move a kitchen item, i.e., sink/table on the symbolic view and if changes are according to the rules defined in the bx tool then items will be reflected on the layout 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.

Person and Family The concept of this example is taken from the example *FamilyToPersons* located in the bx examples repository [11]. 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. The addition of a new person to the Person view will be reflected in the family view and vice-versa. Also, due to the uniqueness of the surnames, a 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 for 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 the

[2]*short for User Interface*

Examples	Evaluation Criterias EC1	EC2	EC3	EC4
Task Management	✗	✗	✓	✓
Quiz	✓	✗	✓	✓
Playing with Shapes	✗	✗	✓	✓
Person and Family	✓	✓	✗	✓
Arranging a Kitchen	✓	✓	✓	✓

Table 4: Comparison of examples based on evaluation criteria

selection of the example and its implementation will directly affect the research question *RQ 4* specified in Section 1.1 and requirements *NR 2*, *NR 3* described in Section 3.2.

Hence by taking into account all these factors, I have constructed a few evaluation criteria (referred to as **EC**) for the selection process of the most suitable example as listed below:

EC 1: User should be familiar with the example.

EC 2: Example should be simple without the involvement of any technical details.

EC 3: Interactivity should not create an overhead for the user but rather be intuitive.

EC 4: User should be able to relate to the learning concepts through the example.

Table 4 shows the comparison of the examples based on the evaluation criteria. "✓" denotes the fulfillment of the evaluation criteria whereas, "✗" denotes the non-fulfillment of the evaluation criteria. This comparison table clearly shows that *Arranging a Kitchen* is the most suitable example. It fulfills all the evaluation criteria because 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 an overhead for the user as the environment is a part of day to day life and hence user will be able to relate to the learning concepts through the example. So, I have finally chosen this example for the demonstrator and also as part of my research.

5.2 BX Tool Selection

Next step in the design process was the 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. Again, it was a very important decision, as the selection of the bx tool and the implementation on the top of it will directly affect the research questions *RQ 1*, *RQ 2* specified in Section 1.1.

My gathered information during the case study phase led the foundation for the selection process. 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 software. Even after a significant amount of work has been done by developers community and bx community, the main problems were revolving around the conceptual, practical challenges associated with using bx, and tool/technology in building software systems and absence of benchmarks for comparison of complete bx solutions [3]. To focus particularly on these issues, bx-community had conducted a series of technical workshops at relevant conferences and organized week-long intensive research seminars [3]. Some of the main outcomes of these seminars were (i) focus on the need of benchmarks and further categorizing them into functional and non-functional ones, (ii) software tool support for bx was shown in terms of demos which includes tool like eMoflon, Echo etc.

The outcome of the seminars 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 research questions and requirements, I have finally chosen *eMoflon* as the bx tool to handle the bx part of the demonstrator. Following were the driving factors for the selection of the bx tool:

- 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 accommodated irrespective of the fact that they can have different input data. Hence, sample implementation with a framework to implement the eMoflon tool was already available.
- my supervisor/prof. is a core member of the eMoflon developer team which gave me the added advantage of knowing the tool inside out. This extra knowledge actually helped me in solving the implementation issues/challenges related to the tool.
- eMoflon is Java based and was thus easier for me to integrate and work with, compared to BiGUL (Haskell-based).

5.3 Architecture Design

The 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.

I decided to build a web application to address the requirements *NR1*, *NR6*, *NR7*, and *NR8* described in Section 3.2.

Web application development has evolved drastically from single page design to very complex layering structures since the beginning of World Wide Web. Many design patterns [24] [25] consisting of different technologies and programming languages were proposed, adopted and implemented in different time to address the demands of customers and users on the web. With the rapid changes occurring on World Wide Web, technologies are becoming obsolete and losing its demand day by day. Nowadays, the main focus is on improving the user interaction and allow the developers to build powerful web applications rapidly.

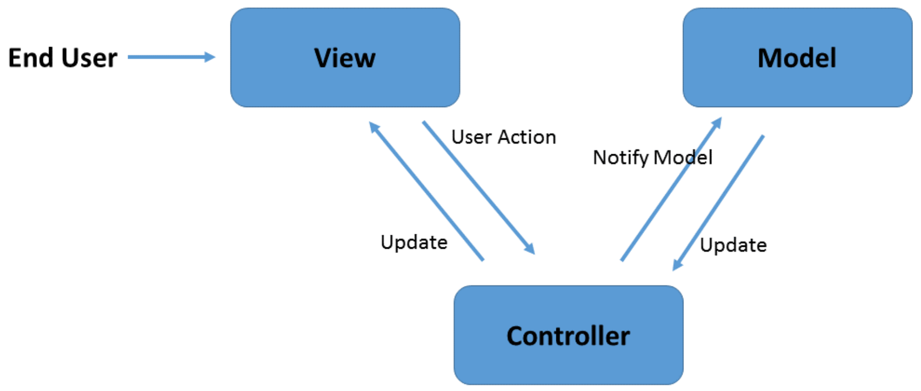


Figure 14: Simple MVC Architecture

I have analyzed a few design patterns and commonly used architecture designs in today's world by working on a few Proof of Concepts (PoC). The main goal of my thesis is to design and implement a demonstrator based on a bx-tool as described in Section 1.2 and prior to this stage, I have already selected *eMoflon* as my bx-tool in Section 5.2. So, The main idea was to check the feasibility of the architecture and the data flow within its components on the top of the interface provided by the *Benchmarx* (proposed by Anjorin et al. [6]) to access the bx-tool. While working on the PoCs, I came across a few problems such as maintainability and reusability of code, dependencies of components etc. To avoid these problems and to address the requirement *NREQ3* described in Section 3.1, I finally chose Model-View-Controller (referred as **MVC** from now on) architecture for my application framework. Figure 14 shows a simple MVC architecture with its layers. Following were the driving factors for the selection of the MVC architecture [25] [28]:

- it differentiates the layers of a project in Model, View, and Controller for the re-usability and easy maintenance of code.
- it splits responsibilities into three main roles which allow the developers to work independently without interfering in each other's work and for more efficient collaboration.
- due to the separation of concern, the same Model can have any no.of Views. Enhancements in Views and other support of new technologies for building the View can be implemented easily.

- a person who is working on View does not need to know about the underlying Model code base and its architecture.

5.4 Architecture Layers

This section provides an overview of the entire architecture followed by the description of each component e.g., Model, View, and Controller in detail.

5.4.1 Overview

The architecture has three components e.g., Model, View and Controller to signify the MVC pattern. Figure 15 describes the high-level architecture of the Demon-BX tool in the form of a component diagram. `Model` component is present at the below which contains the bx tool and all transformation logic.

`View` component is present on the top left, which contains a graphical user interface along with web technologies inside a web browser and a few java classes. As a component, it is shown on a web browser as a part of a web application, provides an interface for a user to interact, and responsible for capturing all the user-related actions.

The `controller` component is present on the top right, which acts a bridge between view and model and handles all user requests. It consists of a `servlet` and a Java class. Servlet receives all the user requests sent from the web browser and calls the appropriate methods of the Java class where the actual data conversion happens, i.e., conversion of user data to bx tool model specific data before sending the them to the bx tool for further processing and conversion of bx tool model specific data to user data after receiving the processed data from the bx tool.

The `model` component consists of a bx tool i.e., `eMoflon`, and a few Java classes encapsulating the implementation of the bx tool. It keeps all the models and transformation rules, processes the data sent by the controller, and manages all the tasks related to business logic, business rules, data, meta-models, state of meta-models etc.

Request-Response Cycle Figure 16 and sequence diagram shown in Figure 17 illustrates how the components interact with each other. Both the diagrams shows a high-level overview of the communication process between the components.

1. User interacts with the application on a browser and user actions are sent to the controller along with a unique user id (GUID) as a request in the form of JSON data.
2. After accepting the request, controller checks the existence of the GUID. Then, it decides on how to handle the request and send it to the controller helper for further processing. Controller helper converts the user data to bx tool model specific data.

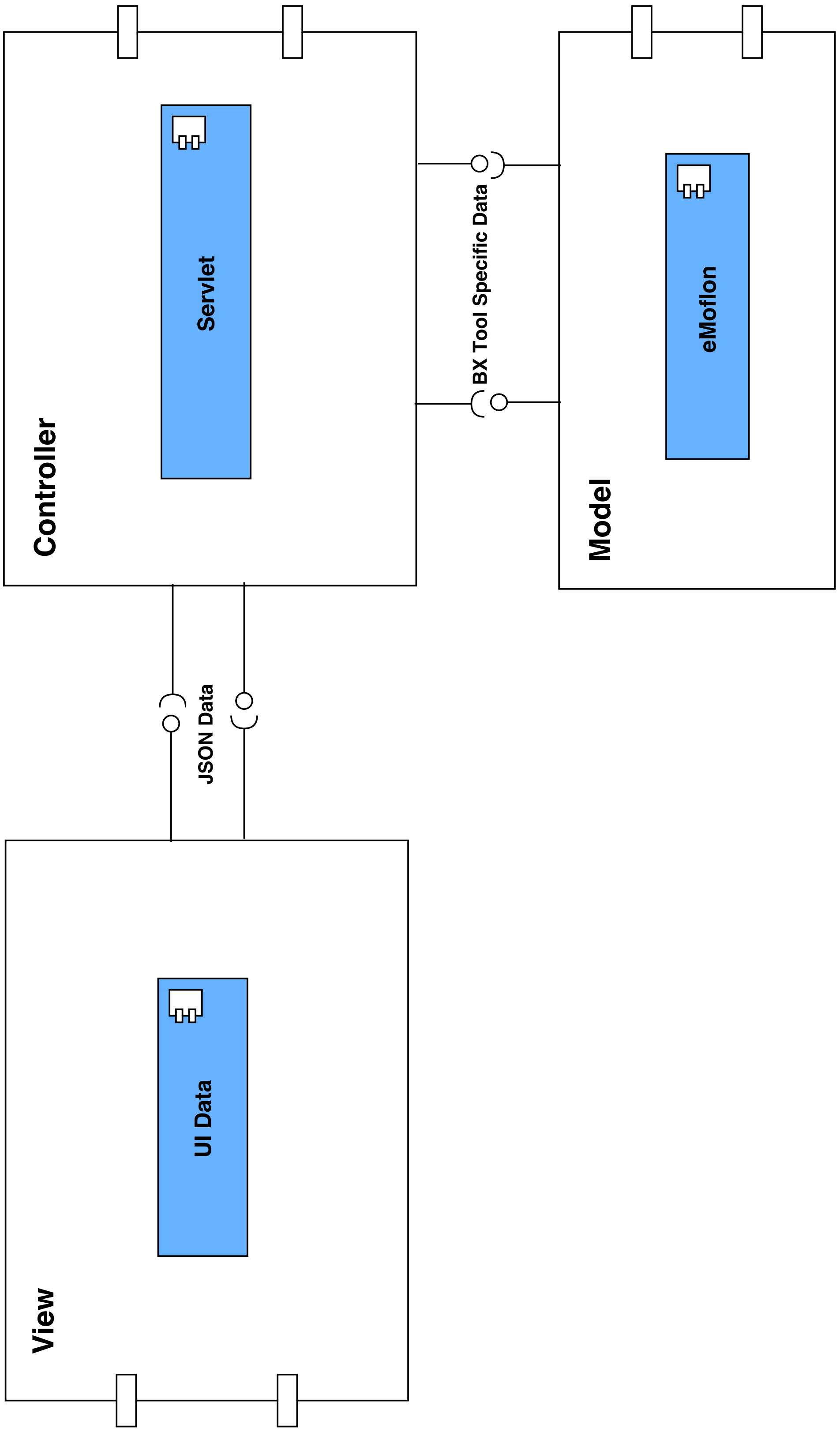


Figure 15: Component Diagram of Demon-BX Tool

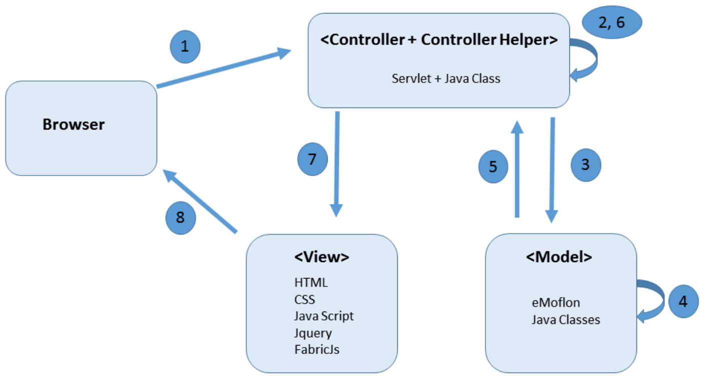


Figure 16: Request - Response cycle between Components

3. Then, controller helper sends the data to the Model for further processing.
4. The model analyzes and processes the request.
5. The model generates the response, which is sent back to the controller helper.
6. After receiving the response, controller helper converts the bx tool model specific data to user data and sends it to the controller.
7. The controller sends the response to the view in the form of JSON data.
8. View processes the data and finally, the final response is generated and loaded into the browser.

5.4.2 Model

Model is mainly responsible for encapsulating the access of data and handling the business logic of the application [28] [26]. It ensures the data abstraction and provides methods to access it, due to which the complexity of writing the code on developers' part is highly reduced [27].

Sub-Components Figure 18 describes the architecture of the model in the form of a component diagram. In my case, the model contains two sub-components i.e., `wrapper classes` and `eMoflon` tool. The wrapper classes consist of interfaces and its concrete implementation designed to access the eMoflon tool on the top of the framework provided by the `Benchmarx`.

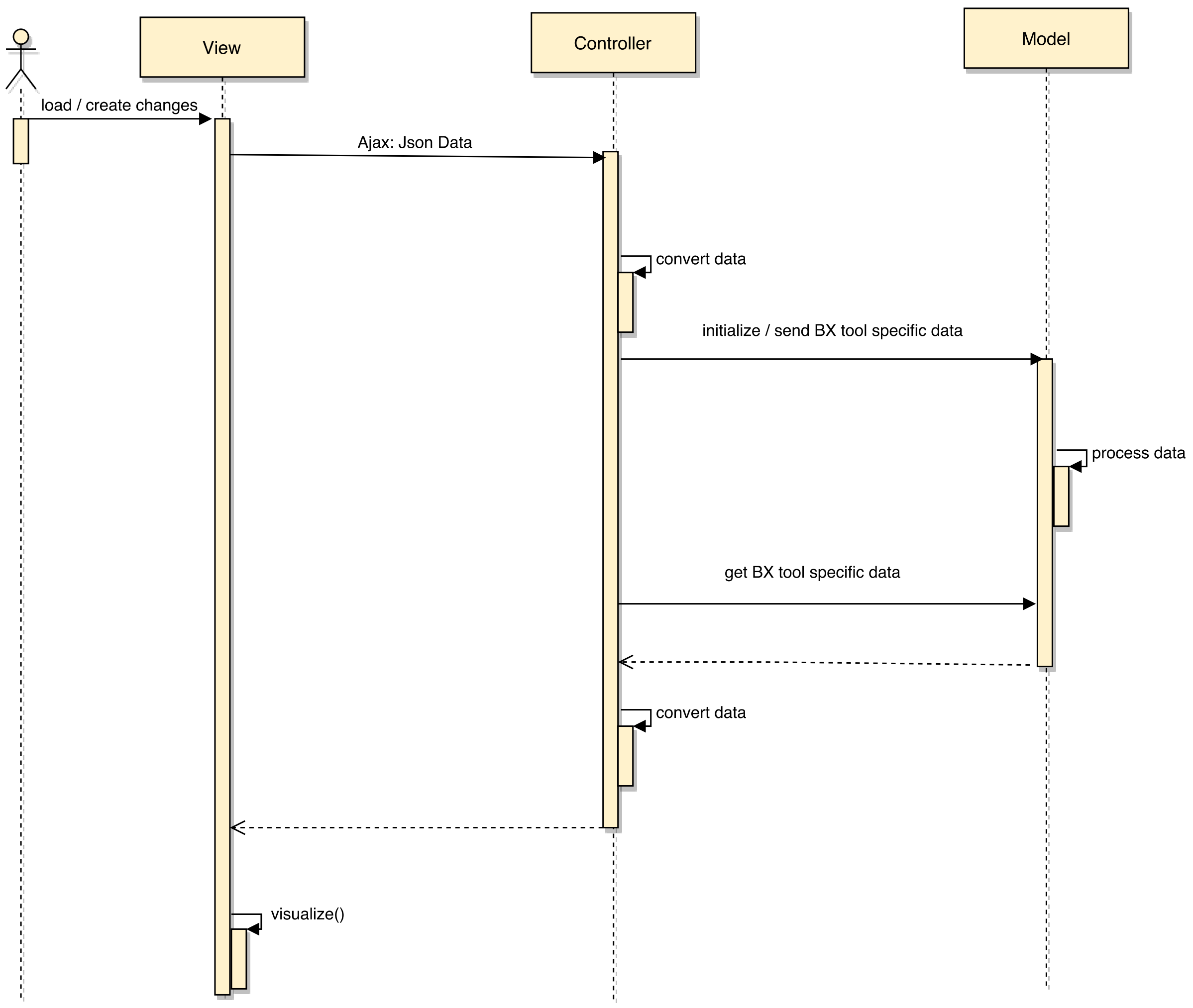


Figure 17: High-Level Sequence Diagram of Demon-BX Tool

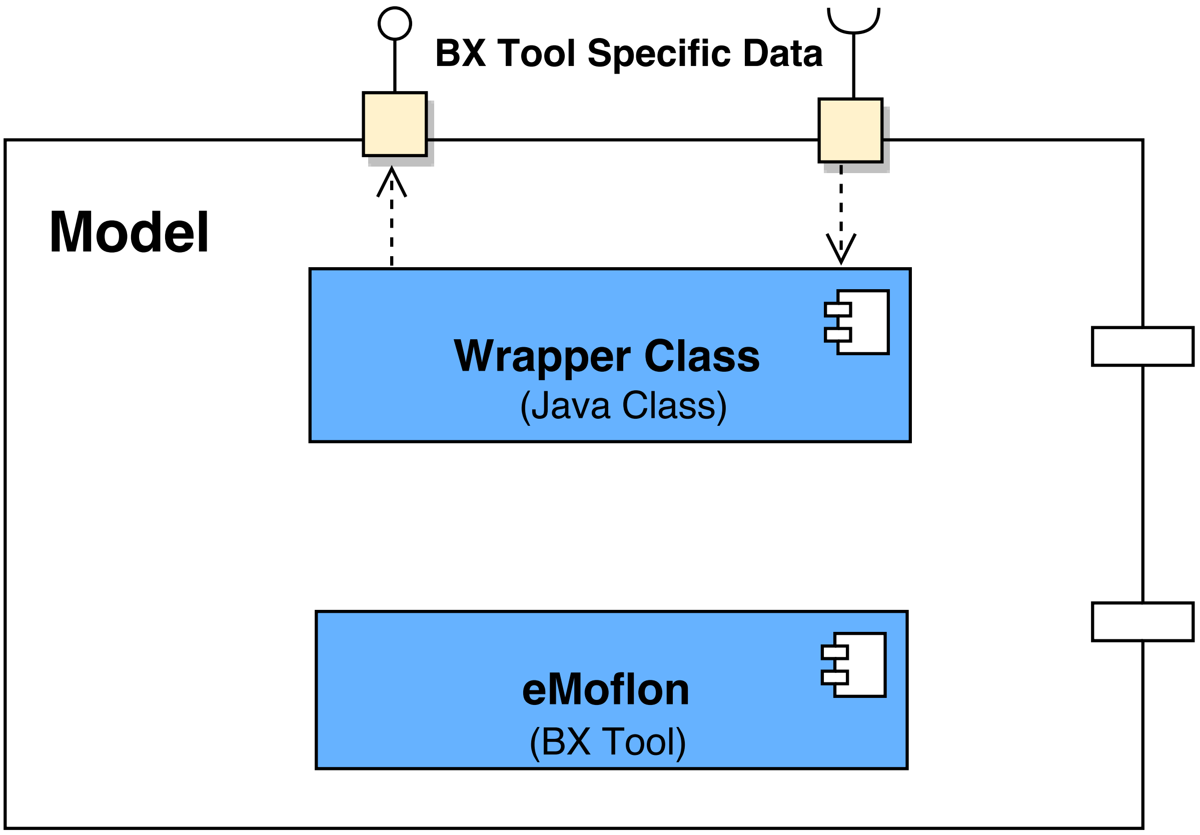


Figure 18: Component Diagram of Model

The eMoflon tool contains all the meta-models, state of the meta-models and the associated transformation rules.

For implementing the example *Arranging a Kitchen* through the *eMoflon* tool, the first task was to create related models inside the tool. The bx tool will keep these models, create and manage instances of these models by applying associated transformation rules as per the requirement. As the example has two different views, it requires two different model in the tool as well i.e., `KitchenLanguage` to represent high-level view and `GridLanguage` to represent low-level.

Figure 5 describes the structure of the `KitchenLanguage` model in a class diagram from an object-oriented point of view. It contains three classes i.e., `Kitchen`, `ItemSocket`, and `Item`. `Kitchen` class has the attributes *xSize* and *ySize* to describe its size and contains zero or more itemsockets. `ItemSocket` class has the attribute *id* and contains exactly one item. `Item` class has the attributes *xIndex* and *yIndex* to describe its position. `Sink`, `Table`, and `Fridge` are different types of items.

Figure 4 describes the structure of the `GridLanguage` model in a class diagram from an object-oriented point of view. It contains three classes i.e., `Grid`, `Group`, and `Block`. `Grid` class has the attribute *blockSize* to describe the size of each block contained inside it, contains zero or more itemsockets, and zero or more blocks. Each `Group` class occupies one or more blocks. `Block` class has the attributes *xIndex* and *yIndex* to describe its position and is being occupied by one group.

Workflow Figure 19 and Figure 20 illustrate the workflow of the model in the form of a sequence diagram.

1. **Load**: During first time load of the application, the controller forwards the request to the controller helper after receiving the initialization command for the bx tool from the view. After receiving the request, controller helper instantiates all the wrapper classes. Afterward, the wrapper class sends the initialize signal to the eMoflon tool and the tool gets initialized by creating the instances of the `Source` and `Target` models. Then, controller helper requests for the newly generated models through the wrapper class. Wrapper class receives the tool specific models and send it back to the controller helper. The entire process is shown in Figure 19.

2. **Create Deltas**: After converting the deltas send from the view into eMoflon specific model data, controller helper calls the appropriate method of the wrapper class e.g., either to propagate deltas into source model or target model. Accordingly, wrapper class initiate forward synchronization or backward synchronization and send the deltas to the tool. The tool processes the deltas taking into account all the associated transformation rules. During processing of the data, the tool checks whether any of the deltas cause more than one transformation rule to invoke. If not, processing is completed and controller helper

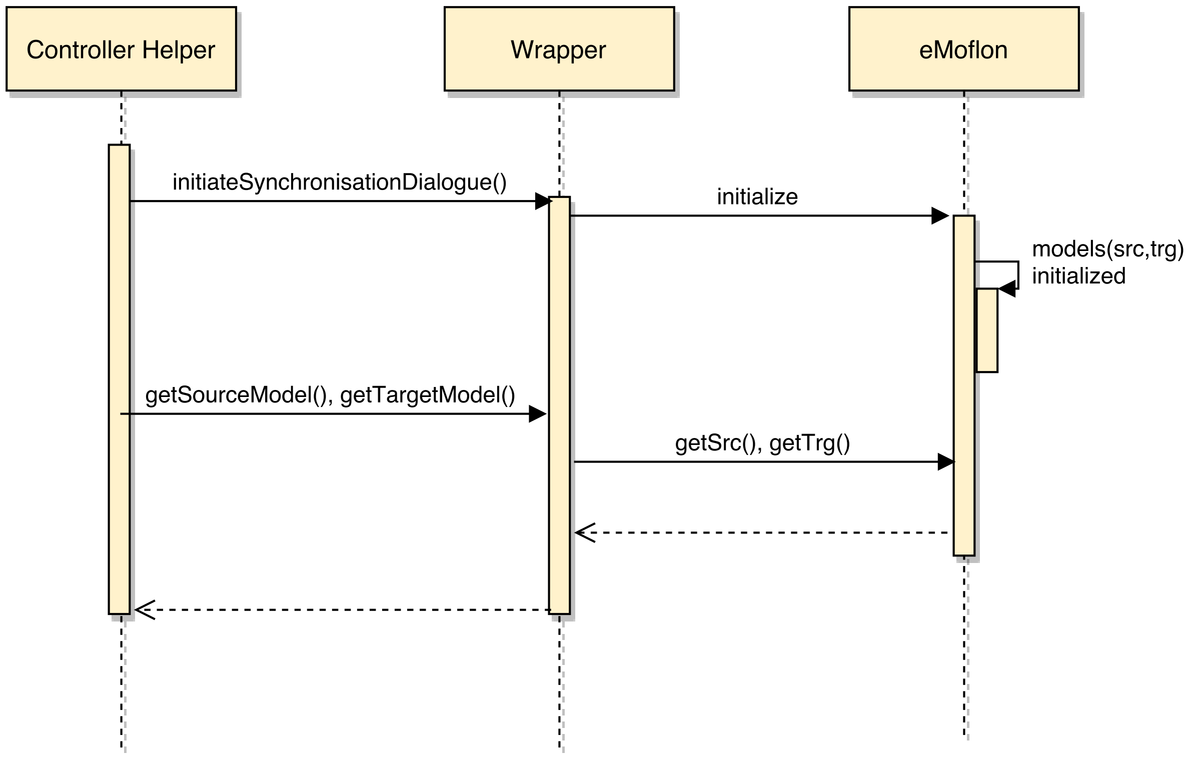

Controller Helper
Wrapper
eMoflon
initiateSynchronisationDialogue()
initialize
models(src,trg) initialized
getSourceModel(), getTargetModel()
getSrc(), getTrg()

Figure 19: Detail Sequence Diagram of Model: initialization

requests for the updated models through the wrapper class. Wrapper class receives the updated models and send it back to the controller helper. The entire process is shown in the upper part of the conditional case in Figure 20.

3. **User Choice**: During processing of the data, if the tool gets to know that one of the deltas cause more than one transformation rule to invoke, it pauses the data processing and sends the matching rules back to the view asking for user's decision through controller helper. On receiving the user's decision, controller helper sends resume data processing signal along with the user's decision to the tool through wrapper class. Afterward, tool completes the data processing with the user's decision. Then, controller helper requests for the updated models through the wrapper class. Wrapper class receives the updated models and send it back to the controller helper. The entire process is shown in the lower part of the conditional case in Figure 20.

5.4.3 View

The view handles the graphical user interface part of the application. Hence, it contains all the graphic elements and all other HTML elements of the application. The view separates the

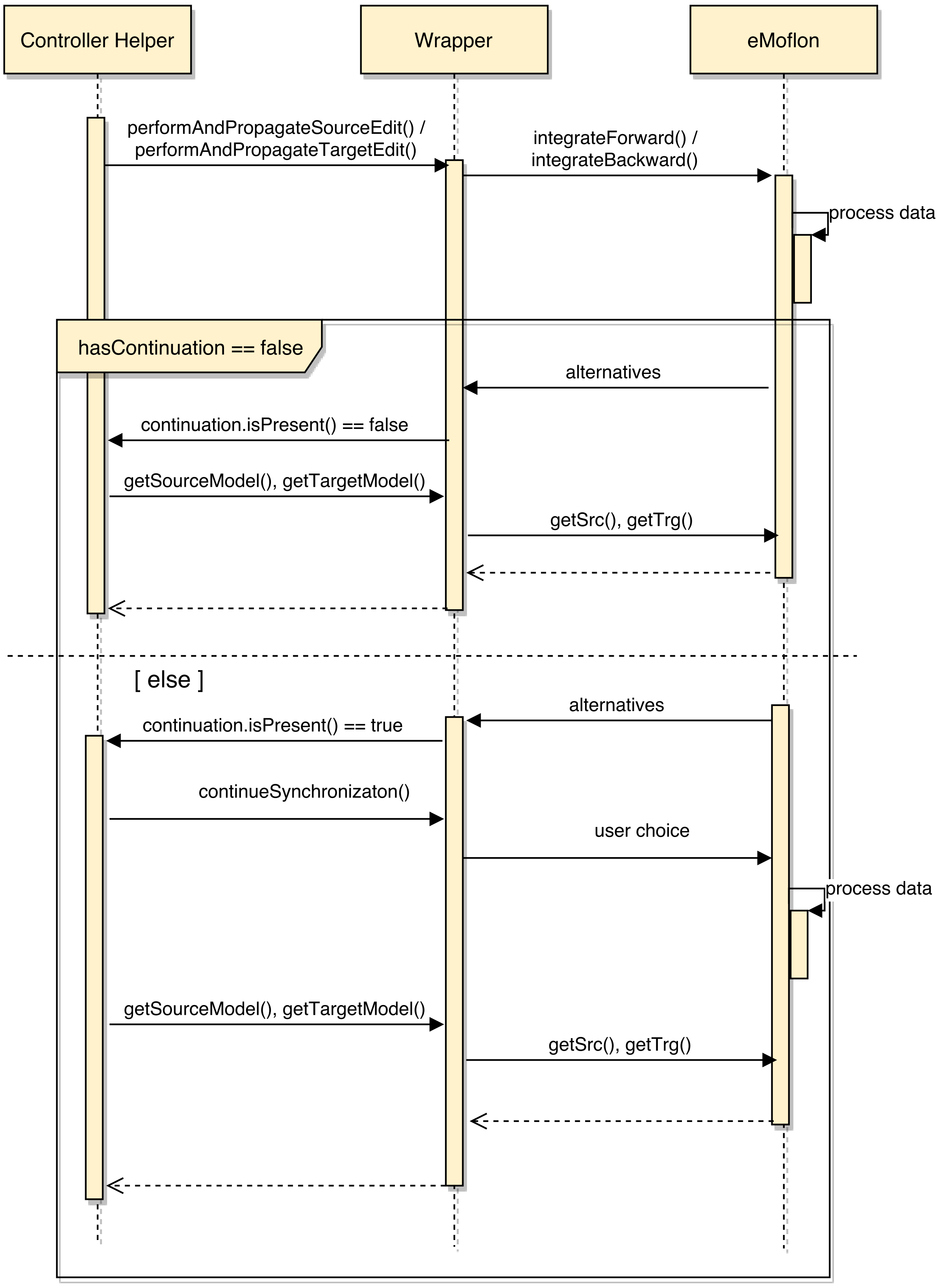


Figure 20: Detail Sequence Diagram of Model: delta propagation

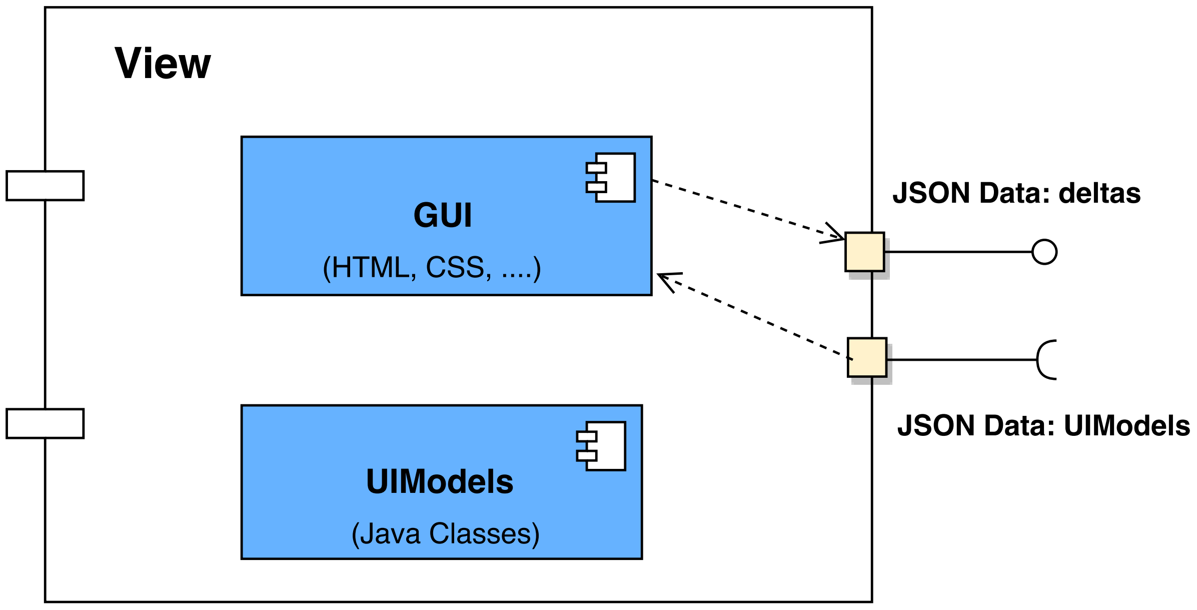


Figure 21: Component Diagram of View

design of the application from the logic of the application due to which the front end designer and the back end developer can work separately without thinking about the errors which could have shown up in case of an overlapping [28] [26]. The view controls how the data is being displayed, how the user interacts with it and provides ways for gathering the data from the users.

Sub-Components Figure 21 describes the architecture of the view in the form of a component diagram. View basically contains two sub-components i.e., `GUI` and `UIModels`. `GUI` consist of the technologies like HTML, CSS, JavaScript, Jquery, and FabricJs, which combinedly create a user interface for the user to interact. Whereas, `UIModels` consists of Java classes created for the user interface to handle the user actions, visualize the bx tool specific models, and act a connecting bridge between them.

Figure 22 shows the structure of the `JSON` data that is being exchanged between view and controller in the form of a class diagram.

View always receives the response from the controller in the form of a `UIModels` and visualize them. UIModels consist of classes like `Canvas`, `Layout`, `Workspace`, `UIModels`, `UIGroup`, `Element`, `Rectangle`, and `Change`. `Canvas` is the parent class for `Layout` and `Workspace`. It has attributes *height* and *width* to describe its size representing both the views. The `Workspace` represents the symbolic view in the user interface and contains zero or more objects (elements) and an objectTypeList. It represents the eMoflon specific model `KitchenLanguage` on the UI side. The `Layout` represents the layout view in the user

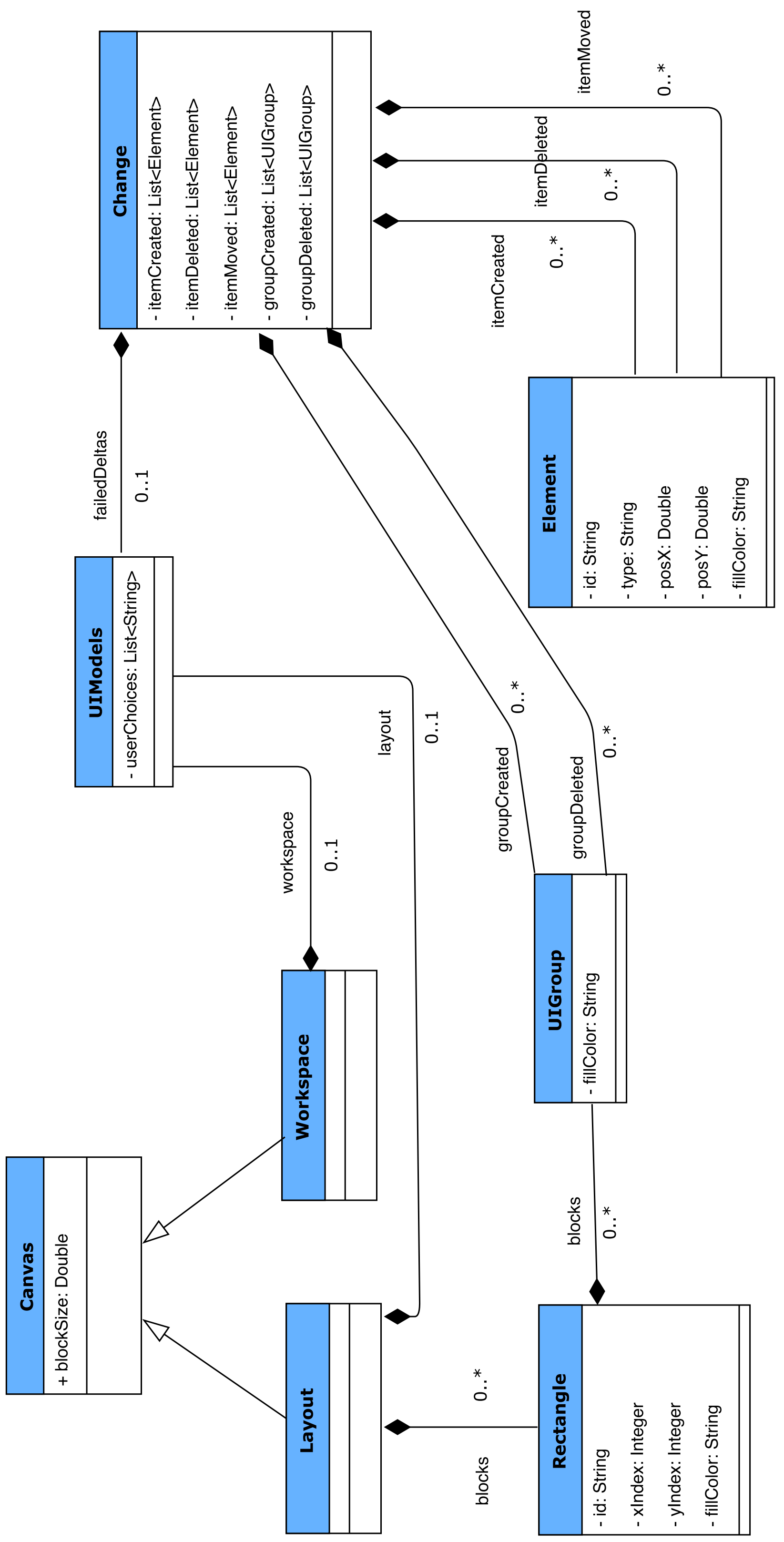


Figure 22: Structure and Relationship between UI Models

interface and contains zero or more blocks and groups. It represents the eMoflon specific model `gridLanguage` on the UI side. `UIGroup` has attribute *fillColor* to uniquely identified as a new group on UI and contains zero or more blocks. It represents the eMoflon specific model `Group` on the UI side. `Rectangle` has attributes *id*, *xIndex*, *yIndex*, and *fillColor* to deal with the UI related actions. It represents the eMoflon specific model `Block` on the UI side. The `Element` has attributes *id*, *type*, *xPos*, *yPos*, and *fillColor*. It represents the eMoflon specific model `Item` on the UI side. `UIModels` is the class which carries all the information related to UI. After receiving the response from the bx tool, it is converted into `UIModels` and send it to view for visualization. It has an attribute userChoices to deal with the user options and contains a layout, a workspace, and a set of failedDeltas wrapped inside a `Change` class.

While user interacts with the demonstrator, actions i.e., changes performed by the user in the UI are captured in the form of `deltas` and packaged in a `Change` object. The kitchen allows three actions i.e., creation of a new item, deletion of an item and movement of an item. Whereas, the layout allows two actions i.e., creation of a new group and deletion of a group. Hence, `Change` class track all of these five actions separately. `Change` class shown in Figure 22 contains itemCreated, itemDeleted and ItemMoved as a list of `Elements` to capture creation, deletion, and movement of an item respectively in the kitchen. It also contains groupCreated and groupDeleted as a list of `UIGroups` to capture creation and deletion of a group respectively in the layout.

Workflow Figure 23 illustrates the workflow of the view in the form of a sequence diagram.

1. **Load**: To load the application, the user enters the complete URL on a web browser. As soon as the web page is loaded, view generates a unique user id (GUID) for the current user and sends an initialization command for the bx tool i.e., eMoflon along with the GUID in the form of `JSON` data to the controller wrapped inside an `Ajax` call. In return, the view gets the `UIModels` generated by the controller. After getting the models, view visualizes them in two HTML canvas elements i.e., `Layout` and `Kitchen`.

2. **Create Deltas**: A user is allowed to create deltas in one of the HTML canvas elements i.e., `Layout` and `Kitchen`. To propagate the deltas to the other view, the user presses the synchronization button. View sends all the deltas created along with the GUID generated earlier for the user in the form of `JSON` data to the controller wrapped inside an `Ajax` call. In return, the view gets the `UIModels` generated by the controller. After getting the models, view visualizes them in two HTML canvas elements i.e., `Layout` and `Kitchen`. This process is shown in the upper part of the conditional case in Figure 23.

3. **User Choice**: During processing of the data, if the bx tool requires a user input, the request is being notified to the view through the controller and view prompts the user for the input. After the input is given by the user, view sends it along with the GUID generated earlier for the user in the form of `JSON` data to the controller wrapped inside an `Ajax` call. In return, the view gets the `UIModels` generated by the controller. After

getting the models, view visualizes them in two HTML canvas elements i.e., `Layout` and `Kitchen`. This process is shown in the lower part of the conditional case in Figure 23.

External Design For the visualization of the example *Arranging a Kitchen* as explained in Section 5.1, the first task was to design the layout view and symbolic view along with its functionalities.

Both the views represent a kitchen area and its certain behavior. The symbolic view has more functionalities and flexibility in usage than the layout view. As both the views are independent of each other and resonates a confined area in which certain task related to animation/graphics has to be performed, I chose *Canvas* [29] HTML element as a container for my views. *Canvas* was the best fit for my views as it provides great support and application for creating animation and drawing graphics on the web.

For the symbolic view, I kept the *Canvas* clean to represent an empty kitchen space where addition and manipulation of different objects such as sink, table, fridge etc. can be done. To make the kitchen space more realistic, I have even added water outlet on the western wall and electrical fittings on the northen wall so that the user can relate to it. For layout view, I have filled the *Canvas* with grids/blocks. The layout view represents exact kitchen space as shown in the symbolic view but, divided into blocks which restrict certain flexibility and functionality. Figure 24 shows a sample of the symbolic view (Kitchen) and layout view (Layout).

Next step was to handle the user interactions in the process of performing various task on both the views. In web development, javascript is the most used language for handling the user interactions and programming the behavior of web pages [33]. Hence, I have analyzed a few canvas libraries available in market e.g., Fabric.js [30], Processing.js [31], Pixi.js [32] by working on a few Proof of Concepts (PoC). The main idea was to check the feasibility and support for interactivity to perform different user defined tasks on the *Canvas*. Finally, I chose Fabric.js as my javascript library for handling the user interactions because of below factors [30]:

- It is good at displaying a large number of objects on the canvas.
- It handles object manipulation such as, moving, rotating, resizing for any kind of object.
- It has a great support for rendering and displaying object of any kind.

Internal Design Second task in the process of visualization was defining the user actions with respect to both the views, capturing them, and displaying the views after transformation is done.

In the kitchen, a user can perform addition, removal, and movement of the kitchen objects as it is done in everyone's house. A new object can be added and an already existing object can be moved or removed within the empty space available with the different mouse events. To make

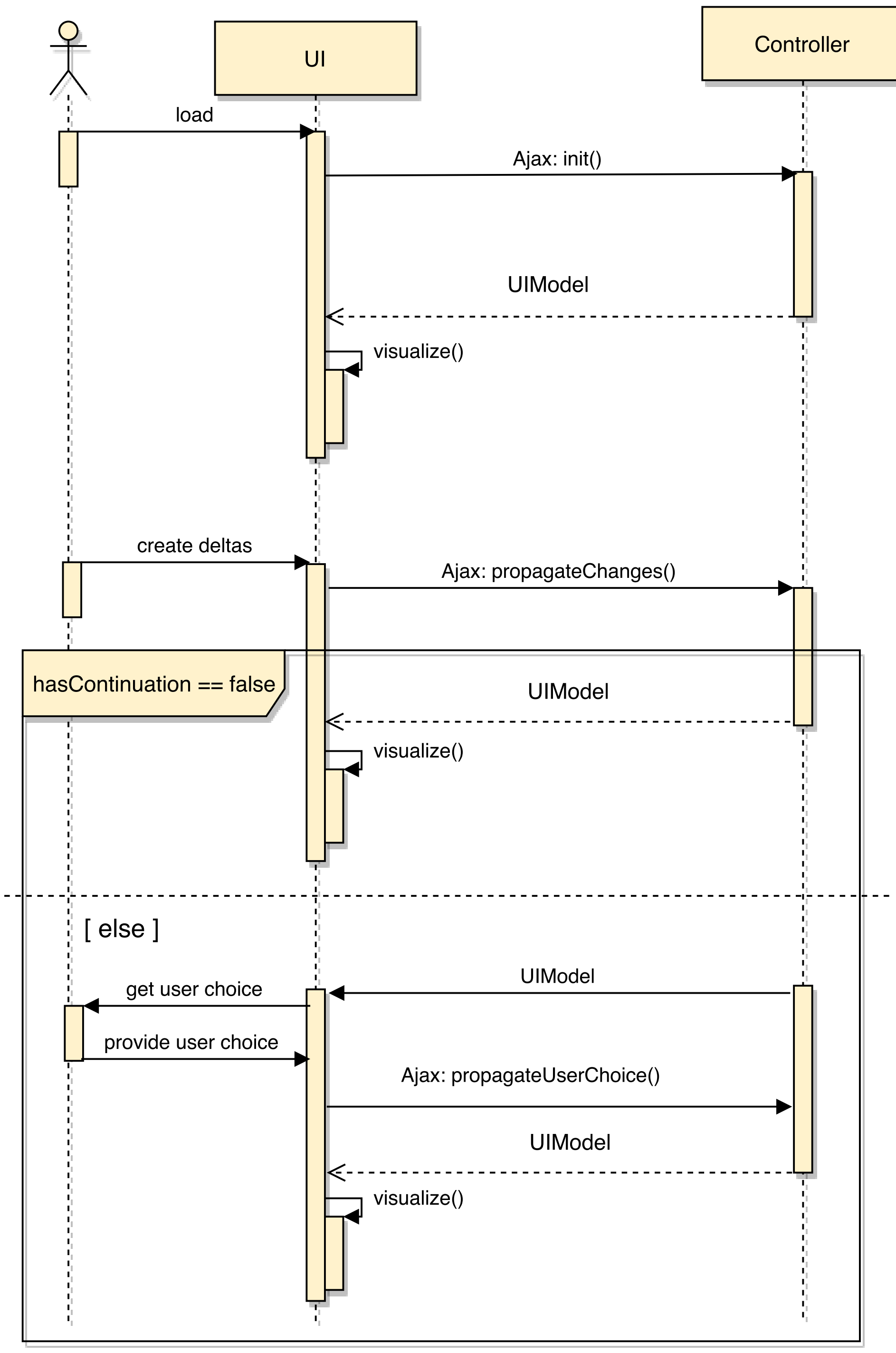


Figure 23: Detail Sequence Diagram of View

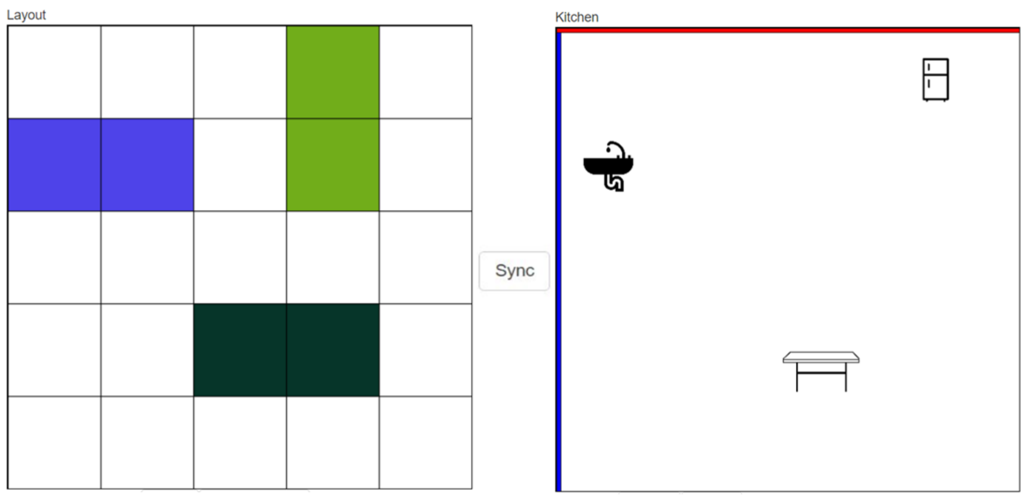


Figure 24: Layout and Kitchen

the example more realistic, I have used similar images of the kitchen objects as shown in the right side canvas (kitchen) in figure 24. Every object is tracked on the basis of its position in the view and every change i.e., creation, deletion or movement of objects is captured inside a `delta` by the kitchen and send it to the controller for further processing.

As layout offers fewer functionalities than the kitchen, a user can perform only addition and removal of the kitchen objects. The layout represents the kitchen space divided into blocks. Hence, each kitchen object is represented in the form of a group, combining any number of block(s) arranged in vertical or in horizontal direction filled with a unique color everytime a new object is added. For example, the sink is represented by two horizontal blocks attached to one another and filled with blue color as shown in the left side canvas (layout) in figure 24. Every group is tracked on the basis of the block's position that it is consist of along with its color and every change i.e., addition or removal is captured inside a `delta` by the layout and send it to the controller for further processing.

After the changes are done on either view, to see the effect on the other, the user can click on the synchronization button and both the views will be updated.

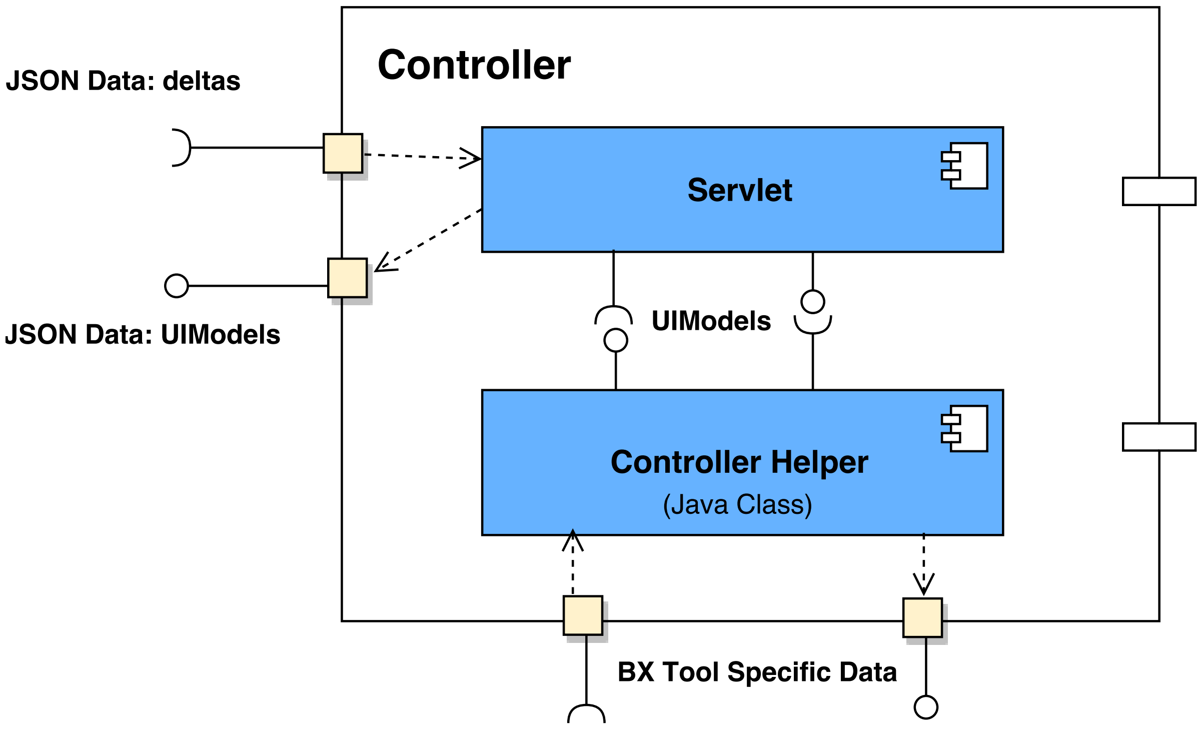


Figure 25: Component Diagram of Controller

5.4.4 Controller

The controller is mainly responsible for event/action handling and manages the relationship between a view and a model [27]. These actions are triggered while a user interacts with the application on a web browser. It accepts the user requests, interacts with the model, and generates the view from the response.

Sub-Components Figure 25 describes the architecture of the controller in the form of a component diagram. I am using a thin controller consists of two sub-components i.e., `Servlet` and a `Controller Helper` which is a java class.

Servlet is a technology which provides a component-based, platform-independent method for building Web-based applications [36]. Servlet is built on Java platform to extend the capabilities of a web server which makes it robust and scalable. It resides inside a web server to generate dynamic content.

Servlet is capable of handling all kinds of client-server protocol but popularly and mostly used with the HTTP, the HyperText Transfer Protocol. A web container is essentially required to run a servlet. A web container is a component of the web server that interacts with the servlets and

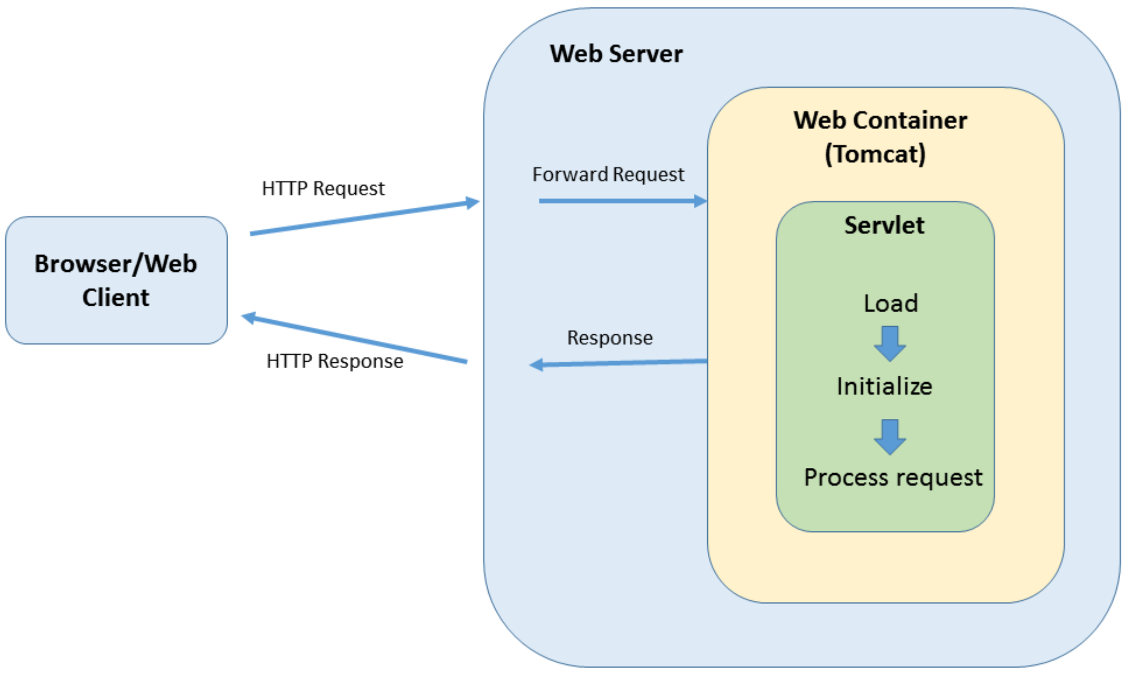


Figure 26: Servlet Lifecycle

manages the lifecycle of servlets. In my case, I am using *Apache Tomcat* as my web container.

Figure 26 describes the complete lifecycle of a servlet w.r.t. a web server and a web container. The lifecycle steps are described as follows [36]:

1. The web server receives the `HTTP request` from the client interacting through a browser.
2. After accepting the request, web server forwards the request to the web container i.e., tomcat.
3. Web container sends the request to the `Servlet class`.
4. If an instance of the servlet does not exist, the web container

   loads the servlet class then creates an instance of the servlet class and initializes the servlet instance by calling the `init` method.
5. After that, web container invokes the `service` methods (normally HTTP methods i.e., `get` , `post` , `put` , `delete`) of the servlet class by passing the request and response objects and the actual processing of the request is done and the response is generated.

6. Web container sends the response to the web server. Afterward, web server creates the HTTP response and send it back to the client.

In my application, the servlet is strongly coupled with a controller helper i.e., a Java class. Controller helper class is responsible for the conversion between UI data and bx tool specific data.

Workflow Figure 27 and Figure 28 illustrate the workflow of the controller in the form of sequence diagrams. The main task of the controller is to act as a communication bridge between view and model. In my case, this task is shared between servlet and controller helper.

1. **Load**: To load the application, the user enters the complete URL on a web browser. As soon as the web page is loaded, view generates an unique user id (GUID) for the current user and sends an initialization command for the bx tool i.e., eMoflon along with the GUID in the form of `JSON` data to the controller wrapped inside an `Ajax` call. After receiving the request from the view, the controller creates a new instance of the controller helper class and store it inside a `map` pointing to the GUID sent in the request. Then, controller forwards the initialization request to the controller helper class and eMoflon tool gets instantiated. Afterward, controller requests for the newly generated models through the controller helper. Controller helper receives the tool specific models, convert it to the UI specific models and send it back to the controller. After receiving the response, controller forwards it to the view for visualization. The entire process is shown in Figure 27.

2. **Create Deltas**: After the synchronization button is pressed, view sends all the deltas created along with the GUID generated earlier for the user in the form of `JSON` data to the controller wrapped inside an `Ajax` call. After receiving the request from the view, the controller extracts the GUID and checks its existence inside the `map`. If the GUID exits, controller picks the corresponding controller helper's instance and forward it all the deltas wrapped inside a `Change` class. If the GUID is not found, controller creates a new instance of the controller helper class and store it inside the `map` pointing to the guid. After receiving the `Change` object, controller helper converts it to the tool specific model data and forward it to the tool for processing. After processing is done, controller helper receives the tool specific models, convert it to the UI specific models and send it back to the controller. After receiving the response, the controller forwards it to the view for visualization. The entire process is shown in the upper part of the conditional case in Figure 28.

3. **User Choice**: During processing of the data, if the tool requires a user input, user choices are sent to the controller helper. Controller helper converts it to the UI specific models and sends it back to the view through the controller. Controller receives the user's input along with the GUID generated earlier for the user in the form of `JSON` data to the controller wrapped inside an `Ajax` call. After receiving the request from

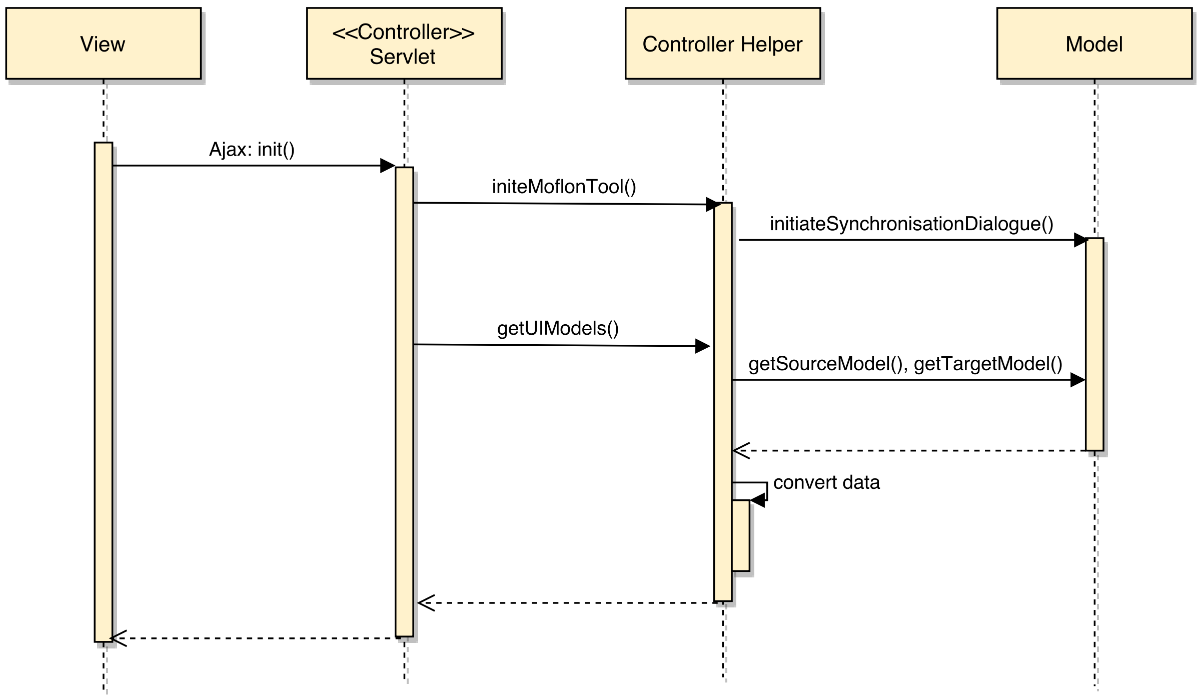


Figure 27: Detail Sequence Diagram of Controller: initialization

the view, controller extracts the GUID and checks its existence inside the `map`. If the GUID exists, controller picks the corresponding controller helper's instance and forward it the user's input. Controller helper forwards it to the tool for further processing. After processing is done, controller helper receives the tool specific models, convert it to the UI specific models and send it back to the controller. After receiving the response, controller forwards it to the view for visualization. The entire process is shown in the lower part of the conditional case in Figure 28.

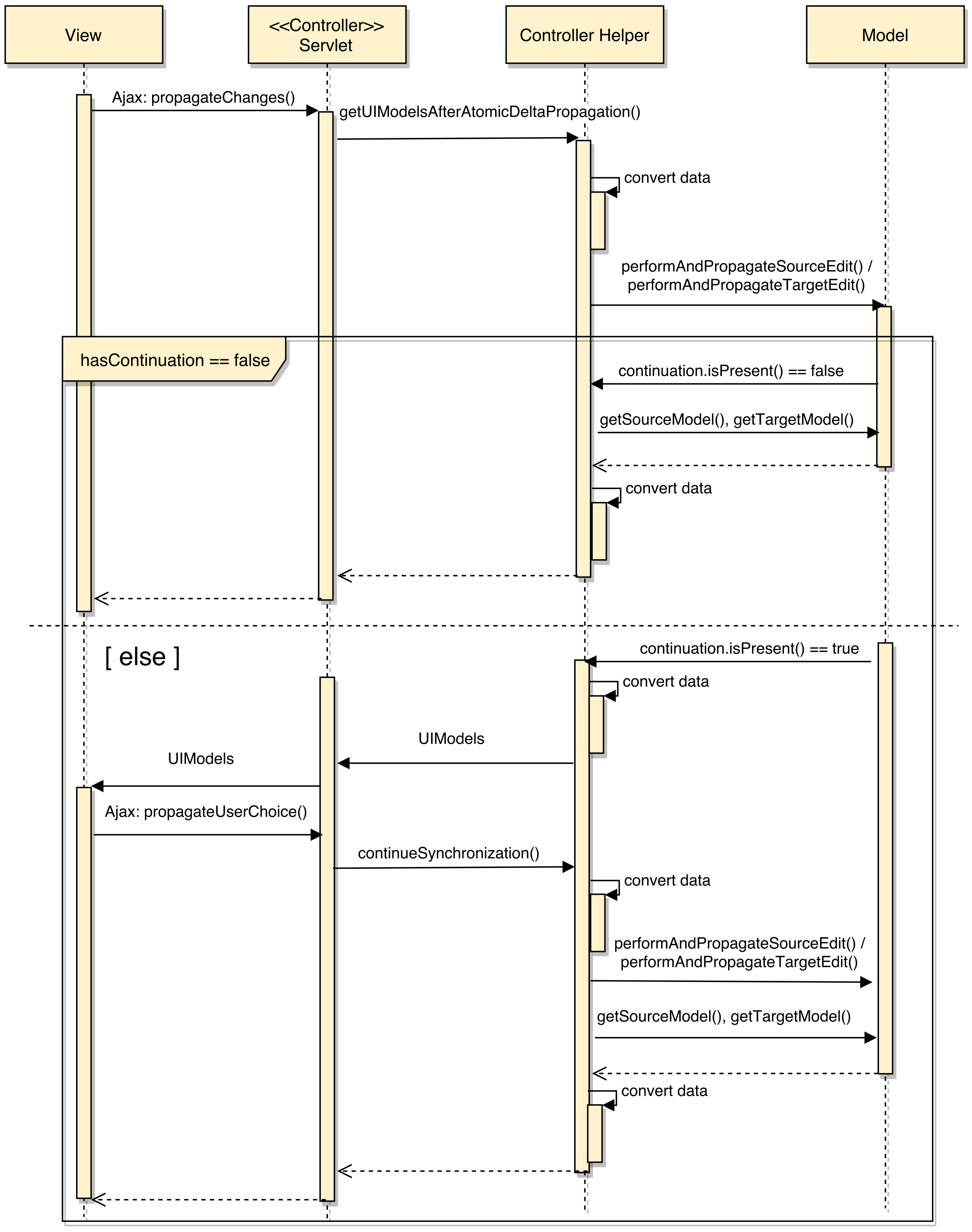

View
<<Controller>> Servlet
Controller Helper
Model
Ajax: propagateChanges()
getUIModelsAfterAtomicDeltaPropagation()
convert data
performAndPropagateSourceEdit() / performAndPropagateTargetEdit()
hasContinuation == false
continuation.isPresent() == false
getSourceModel(), getTargetModel()
convert data
[else]
continuation.isPresent() == true
convert data
UIModels
UIModels
Ajax: propagateUserChoice()
continueSynchronization()
convert data
performAndPropagateSourceEdit() / performAndPropagateTargetEdit()
getSourceModel(), getTargetModel()
convert data

Figure 28: Detail Sequence Diagram of Controller: delta propagation

5.5 Challenges

During the entire designing and implementation process as explained in previous sections of this chapter, I came across a few challenges. This section describes them in detail.

UI interaction and capturing deltas From UI design and interaction point of view, after designing the views i.e., an empty canvas(Kitchen) to represent the symbolic view and a canvas filled with blocks(Layout) to represent the layout view, the first challenge was to conceptualizing the `deltas` and capturing them.

As mentioned earlier, symbolic view has more functionalities than layout view in example *Arranging a Kitchen*. Firstly, I fixed the deltas that can be performed on either view. For example, three types of changes are allowed in kitchen i.e., creation of a new item, deletion of an item, and movement of an item. Whereas, two types of changes are allowed in layout i.e., creation of a new group and deletion of a group. Implementing the user interactions in the kitchen was relatively easy as the user have to deal with the creation, deletion, and movement of objects with actions such as mouse movements and button clicks. But, the real challenge was to handle the user interactions in the block structure of the layout. After having a few discussions, I and my supervisor decided to go ahead with colored blocks to show the occupancy of groups in the layout. Each group will be shown with a set of unique colored blocks. A user can generate different colors by multiple mouse clicks to fill an empty/non-occupied block while creating a new group and same colored blocks will be considered as one group. For deletion of a group, a user can click on any one of the same colored blocks (blocks belong to the same group) and blur its color. Figure 29 shows a sample of the layout with a different set of groups filled with different colors.

Along with creating deltas, another challenge was to uniquely identify different deltas in the process of capturing. Capturing every change that occurs on the views and sending them to the bx tool for processing is error-prone, complex, and sometimes even not required for the tool to handle. For example, creating an item in the kitchen, then deleting it and sending both of the changes to the bx tool do not make sense as practically the item does not exist anymore. Hence, rather capturing every change that occurs on the views, I have implemented a few rules to uniquely identify different deltas while capturing. Figure 29 describes all such rules in detail.

Example: A user creates a table in the kitchen and then move it to a new location. In this situation, instead of creating two different deltas, I capture only one i.e., item created. Because creating a new item and then moving it to a new location is equivalent to creating a new item at the new location.

Processing deltas during synchronization One of the challenges during the implementation phase was how and in which order the deltas are supposed to be fed to the eMoflon tool for processing. The problem was when a series of deltas are given to the eMoflon tool for

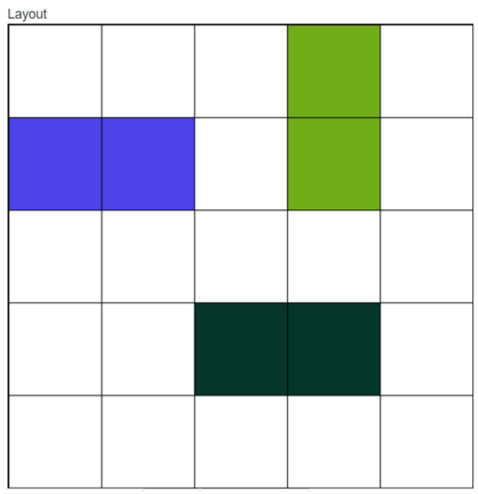
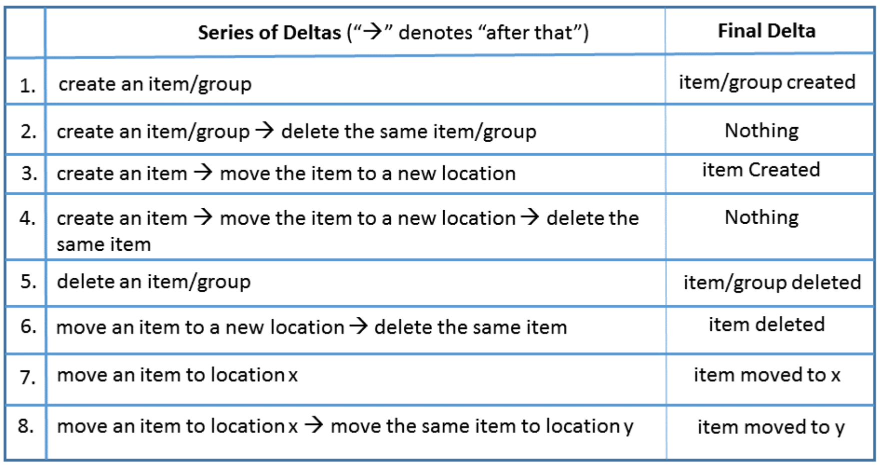


Figure 29: Layout View



	Series of Deltas ("→" denotes "after that")	Final Delta
1.	create an item/group	item/group created
2.	create an item/group → delete the same item/group	Nothing
3.	create an item → move the item to a new location	item Created
4.	create an item → move the item to a new location → delete the same item	Nothing
5.	delete an item/group	item/group deleted
6.	move an item to a new location → delete the same item	item deleted
7.	move an item to location x	item moved to x
8.	move an item to location x → move the same item to location y	item moved to y

Figure 30: Rules for selecting final delta

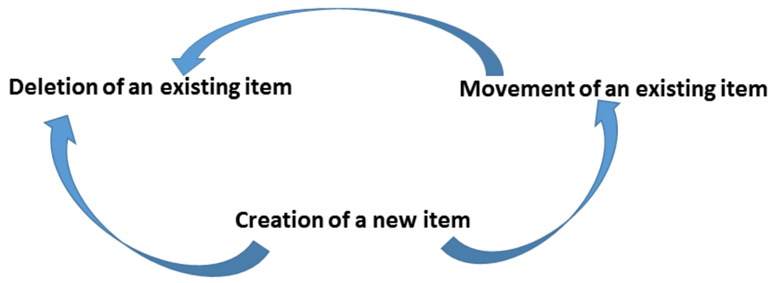


Figure 31: Dependency of states with each other

processing, not all of them are accepted and processed but some of them are rejected as well according to the associated transformation rules. So, if the eMoflon tool is fed with a series of deltas at one go, after processing only the accepted deltas will be contained in the updated models and it will not be possible to track the rejected deltas.

To solve this problem, I decided to implement *atomic delta* method. In this method, one delta is given to the eMoflon tool for processing one after another from the pool of the collected deltas so that the *failed delta*s can be tracked and sent to the view bundled inside the `UIModel` for visualization. If any of the deltas is rejected during processing, eMoflon restores the state of the meta-models to the previous consistent state and resume processing the remaining deltas. Also, it is important to consider the correct order of processing the deltas as a wrong order can create conflict while processing. For example, a user deletes an item x from a location and then move another item y to the same location. While processing, delta for movement will be rejected by the eMoflon tool if it is processed first. Because the tool will not move the item y to the location where item x is still present. So the correct order is deletion then movement and creation at the end. This is because of the fact that creation of a new item depends on the movement and deletion of an already existing item and movement of an item depends on the deletion of an already existing item as shown in Figure 31. Hence, in my implementation, all the deltas related to deletion are processed first. Then, all the deltas related to movement are processed and at the end, all the deltas related to creation are processed.

Example: A user creates three deltas in the kitchen i.e., creating a new sink close to the wall with water outlet(valid delta), moving an already existing fridge away from the wall with electrical fittings (invalid delta), deleting an existing table (valid delta) and presses the synchronization button. All three deltas are sent to the eMoflon tool but fed one by one in the order deletion of the table then the movement of the fridge and creation of a new sink at the end. Processing of deletion of the table will be accepted by the tool and meta-models will be updated as it is a valid delta. Then, processing of movement of the fridge will be rejected by the tool and meta-models will be restored to the previous consistent state as it is a valid delta. At last, Processing of creation of a new sink will be accepted by the tool and meta-models will be updated as it is a valid delta.

Handling UI interaction during synchronization

Handling multiple users accessing the application

5.6 Choices and Threats

During the design and implementation process of the final prototype, I had to make decisions out of the available choices. With every decision I took, there might be some threats associated with it and can effect the final result. For example,

- selection of the most suitable example to implement has impact on user's interactivity and ease of usage of the demonstrator.
- selection of the bx tool has an impact on deciding the functionalities and usefulness of the demonstrator.
- selection 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.

6 Evaluation

In this chapter, I am going to evaluate the outcome of my thesis i.e. research and implementation work. Figure 32 shows the phases of the entire evaluation process. Section 6.1 describes the research and case study phase. Section 6.2 illustrates the evaluation design method that I have used for conducting the experiment. This is then followed by a brief description of the planning phase in Section 6.3. Afterward, Section **??** deals with the formulas to investigate the goals of the evaluation process. Then, Section 6.4 describes the preparation and test execution phase in detail. Section 6.5 explains the associated threats with the evaluation process and their mitigation criteria. At last, Section 6.6 discusses the result generated from the gathered data.

The aim of this chapter is as follows:

"Analyse the outcome of the usage of an interactive demonstrator for the purpose of evaluation with respect to the user from the point of view of the researcher in the context of spreading the basic concepts of bidirectional transformation (bx) and making them understandable and accessible."

6.1 Goals

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 [20]. Hence, for any evaluation process, most important thing are selecting cases that are most relevant to the research and narrowing down the research questions only associated with the exact problems in hand.

Research Questions Our experience based on teaching and cooperation with industry has led us to suspect that people often draw their intuition for desired synchronization behavior directly from the special case of bijections. This can be problematic and leads to statements such as: “Why do I need a *bidirectional* transformation language if the transformation at hand is not *bijective*?” Ironically, bidirectional transformation languages are often especially helpful when a transformation is *not* bijective. As a sub-question of the research question **RQ 3** described earlier in Section 1.2, I propose to investigate if such misconceptions are really widespread or not:

RQ 3.1: Do people tend to derive their (in general wrong) intuition for synchronization scenarios from the special case of bijections?

To impart and train a more general intuition for synchronization scenarios I have implemented `Demon-BX`, an online demonstrator for bidirectional transformations, as a platform for easily creating synchronization scenarios to help achieve corresponding learning goals. As a proof-of-concept, I have formulated five concrete learning goals and designed corresponding scenarios

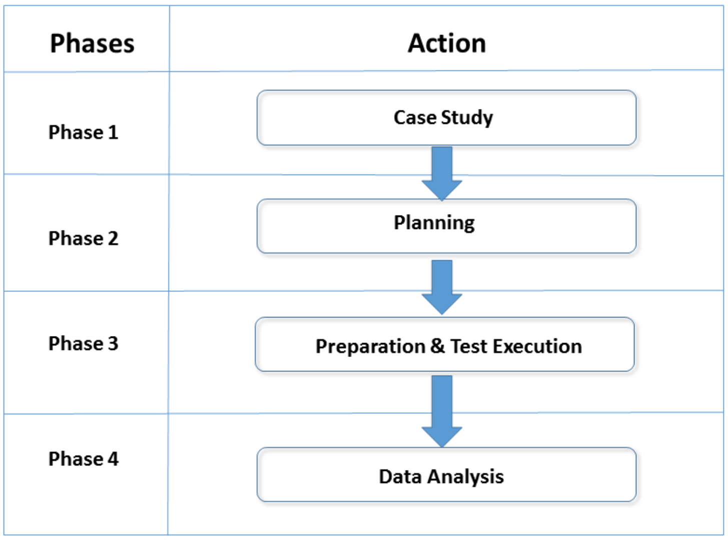


Figure 32: Evaluation Phases

based on a simple example. We believe (i) that example-based demonstrators are an effective way of achieving my (and similar) learning goals, and (ii) that a demonstrator is only useful in combination with carefully designed scenarios. As a sub-question of the research question **RQ 4** described earlier in Section 1.2, I propose to investigate these conjectures with the following two research questions:

RQ 4.1: Does demon-bx support achieving corresponding learning goals?

RQ 4.2: How much does this support depend on the scenarios? Would just playing with the demonstrator and the example already have an equal or comparable (positive) effect?

Purpose The purpose of the experiment is to evaluate whether it is possible to teach and enhance the understanding of the basic concepts of bx through the demonstrator.

Perspective The perspective is from the point of view of the researcher, i.e. the researcher would like to know if the usage of the demonstrator enhances the understanding of bx concepts of a user.

Context This experiment is on a bx tool demonstrator which falls under an educational environment and specifically under computer science branch. Hence, this experiment is mainly

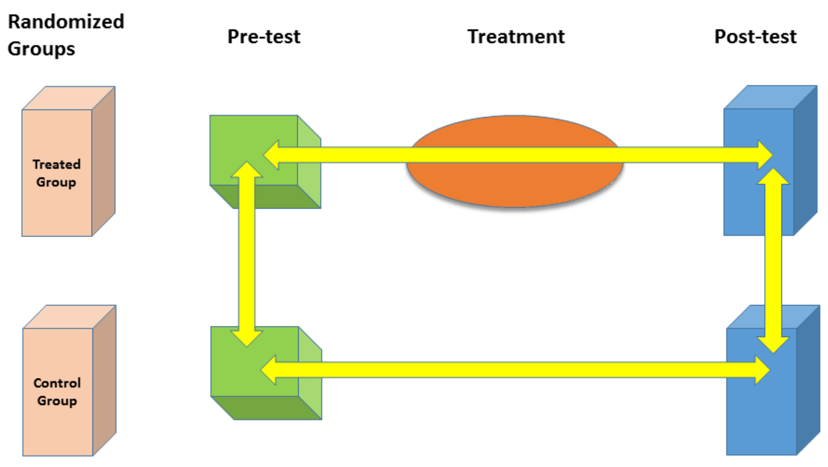


Figure 33: Pretest-Posttest Control Group Design

designed for the group of students/teachers/researchers from computer science area with or without prior knowledge of model driven software development field.

6.2 Design Method

To investigate my research questions, I have used the Pretest-Posttest design method [38], a paired data analysis method in which the same experimental object is measured on some variables on two different occasions under different testing conditions. Here, I am using an extension of the Pretest-Posttest design method, called Pretest-Posttest control group design [39]. It is a highly prestigious and one of the most popular research designs in use. The design principle is relatively simple, which involves two groups, a test group and a control group. First, the groups are pre-tested and then the test group is given the treatment. Afterward, both the groups are post-tested and data is collected from both occasions. Then, the analysis is done by comparing pretest and posttest results collected from both the groups as explained in Figure 33.

Selection of the Pretest-Posttest control group design method for the experiment is driven by the following reasons [40]:

- It provides control over threats to internal validity.
- This allows the researchers to collect and compare posttest result from two groups, which

give them an idea about the effectiveness of the treatment.

- The researcher can see how the groups have performed from pretest to posttest, whether one, both or neither improved over time.

In my case, I have designed test questions, one for each learning goal (brief description in Section 6.3.4), to check if a participant has attained the learning goals or not. We are also interested in the participants' subjective level of certainty for every given answer. The experiment is to be conducted as follows:

1. All participants are divided randomly into two groups of equal size: the treated group and the control group.
2. All participants take the pretest (answer all test questions).
3. Both groups are allowed to use demon-bx for the same amount of time and are provided with an introduction and overview of the concrete example used in the demonstrator. The treated group is additionally provided with carefully chosen scenarios to work through, while the control group is not.
4. When the time is up, all participants take the posttest (answer the same test questions again).

6.3 Planning

This section explains the entire planning phase in detail. In this phase, I have investigated and finalized all the factors required to evaluate the research questions as well as the execution of the experiment. Following sub-sections describe the factors one by one.

6.3.1 Participants

As the context of the experiment is mainly focussed only on the computer science area, it will be conducted on a group of masters' student of computer science branch at Paderborn University. The participants are chosen based on convenience, i.e. the participants are the students taking a similar course.

6.3.2 Hypotheses

Formulating hypotheses formally state that what is going to be evaluated in the experiment. I have constructed my hypotheses focussing on the research questions **RQ 3.1**, **RQ 4.1**, and **RQ 4.2** as described in Section 6.1. Following are the hypotheses I have chosen to focus in my experiment:

Operational Hypothesis

H$_{OP1}$: Students derive their (in general wrong) intuition for and expectations of synchronization scenarios from the special case of bijections.

H$_{OP2}$: The use of demon-bx has a positive effect on the achievement of corresponding learning goals.

H$_{OP2}$: The use of demon-bx in combination with suitable scenarios has a positive effect on the achievement of corresponding learning goals.

To evaluate the above stated hypotheses, the corresponding null hypotheses are stated below:

Null Hypothesis

H$_{NU1}$: Students do not derive their (in general wrong) intuition for the synchronization scenarios.

H$_{NU2}$: There is no significant improvement in the learning outcome of the students after using the demon-bx.

H$_{NU3}$: There is no significant improvement in the learning outcome of the treated group compared to the control group.

6.3.3 Experimental Variables

Hypothesis described above helped in deciding the variables related to the experiment. Following paragraphs describe and Table 5 summarizes all of them .

Independent Variables The independent variables, which the experimenter purposely changes during the experiment are listed below:

Does a group get scenarios to work through or not: nominal (yes or no)

Controlled Variables The controlled variables, which are kept the same throughout the experiment are listed below:

Demonstrator platform: Demon-BX
Concrete example: Arranging a Kitchen
Group participants are chosen from: {master students, PhD students, bx researchers, ... }

	Name	**Possible Values**
Independent Variable	Group gets scenarios to work	nominal (yes or no)
Controlled Variables	Demonstrator Platform	Demon-BX
	Concrete Example	Arranging a Kitchen
	Group participants	{master students, PhD students, . . . }
	Test questions	5
	Learning goals	5
	Max. time interval	55 min
Dependent Variables	Correctness score of pretest	ordinal
	Correctness score of posttest	ordinal
	Level of certainty of pretest	ordinal
	Level of certainty of posttest	ordinal

Table 5: Experimental Variables

Test questions: 5
Learning goals: 5
Max. time interval to answer questions: 55 min

Dependent Variables The dependent variables, which are likely to be changed in response to the independent variable are listed below:

Correctness score of pretest: ordinal
Correctness score of posttest: ordinal
Level of certainty of pretest: ordinal
Level of certainty of posttest: ordinal

6.3.4 Learning Goals & Questions

To investigate the research questions **RQ 3** and its sub-question **RQ 3.1**, I carefully chose five bx concepts to evaluate and imply them as the learning goals (referred to as **LG** from now on) for the experiment. These concepts are related to the basic fundamentals of bx described as follows:

LG 1: It is possible to avoid/minimize unnecessary information loss in bx.

LG 2: Not all possible changes done in one model can be translated/synchronized into another model.

LG 3: Synchronisation is interactive. User interaction (or some other, possibly automated means) can be used to decide between multiple equally consistent results (to handle non-determinism).

LG 4: Undoing changes in one model to revert to a previous state does not necessarily imply that this can be reflected analogously in the other model.

LG 5: Bx frameworks are not always state based but also can be delta based. The actual change performed can have an effect on synchronization results, even if the final result might appear to be exactly the same in both cases.

To evaluate these learning goals, I prepared five questions. Each question referred to a concept of bx which also corresponds to a learning goal.

The questions are designed to have two sections for answers. One section contains multiple choice answers and the other contains certainty scale ranging from "I just guessed" to "I am certain". Correctness will be decided from the selection of answer in multiple choice section and certainty will be decided from the certainty parameters. Hence, combining the answer and the certainty parameter I can differentiate between *correct answer*, *absolute correct answer*, *wrong answer*, and *absolute wrong answer*.

6.4 Execution

This section explains the entire experiment execution process in detail along with the preparation steps in the following subsections.

6.4.1 Preparation

To handle two separate groups i.e., control and treated and to conduct two separate tests i.e., pre-test and post-test on them, I had to prepare well before the experiment date.

First, I prepared a two-minute video tutorial explaining some of the basic concepts of bx. Then, I prepared a set of questions relating each question to one learning goals and put them into two different forms i.e., pretest and posttest prepared with google forms to make them available online for the sake of easy sharing. Then to investigate the research questions **RQ 4** and its sub-question **RQ 4.1**, I prepared the scenarios with the demonstrator to explain the learning goals in a simple way but with a concrete example. Finally, I prepared two different instruction sheets for both the groups explaining the steps they need to follow during the experiment.

6.4.2 Test Execution

The experiment was performed on a group of Master students attending a computer science lecture at Paderborn University. They were informed in advance that such experiment will be conducted on a particular date and that their participation in the experiment is completely voluntary with no consequences what so ever. Participants were requested to bring their laptops to the lecture on the day of the experiment.

Confidentiality The participants were informed regarding the confidentiality and anonymity of data. The purpose of evaluating the tool was stated, but not the hypotheses of the experiment.

Randomization To perform the experiment, the students were given the instruction sheet prepared earlier for either the control or the treated group randomly while entering the room. The participants were not informed about which group they are in and received suitable and separate instructions for each group.

No Interference After that, they were asked to sit in two different areas according to their group allotment. The groups were spatially separated so that discussion between groups was almost impossible.

These information sheets had all the appropriate links to the corresponding questionnaires with questions for the pre-test and post-test, demonstrator links with prepared scenarios for the treated group, and a different demonstrator link without scenarios for the control group.

Firstly, all participants went through a two-minute video tutorial to establish basic concepts and notation used in the test questions. Then all participants were asked to take the pre-test. After finishing the pre-test, the students were asked to use the demonstrator links given to them and to work with it. The link given to the control group only had information on how to operate the demonstrator, while the treated group had access to scenarios chosen to support my learning goals.

After playing with the demonstrator, both groups were asked to take the post-test. Post-test questions were exactly the same as pre-test questions but include some extra questions to get additional qualitative feedback about the demonstrator. The experiment was stopped exactly after 55 minutes.

6.4.3 Data Validation

Data was collected from 40 students. I stopped taking responses on Google forms i.e. pre-test and post-test forms from students exactly after 55 minutes. After the experiment, I checked the data entered by students in both the forms. Data from one student was removed, due to the fact that the data was regarded as invalid as the student could not finish both the test in the given time limit.

Hence, after removing one student out of the 40, I had data from 39 students for statistical analysis and interpretation of the results.

Finally, based on unique identifiers (identifying the group and participant uniquely) derived for each participant, answers from pre-test and post-test were evaluated to get the results.

6.5 Threats to Validity and Mitigation

A good research work depends highly on its validity and quality of work involved and results. So, it is very important to consider, analyze and mitigate the validity threats to the research work and the results. In my evaluation, I have focused on the validity analysis and threats given by Wohlin et al [42] and further explained by Feldt et al [41]. Wohlin et al discuss four main types of validity threats: internal, external, construct and conclusion.

This section describes all the threats that are associated with the entire evaluation process i.e., planning, preparation, and execution in the following subsections.

6.5.1 Internal Validity

Did the treatment/change I introduced cause the effect on the outcome? Can other factors also have had an effect?

To measure improvement I am forced to perform arithmetic with ordinal values. This might be problematic as the difficulty of the test questions might not be equal (although I have tried to ensure this). If, for example, one of the test questions is much more difficult than all the rest, then subtracting test scores and comparing improvements between groups is questionable. Hence to make the questions/concepts more understandable, I have provided some explanations in the post-test as a video tutorial showing the relation between the abstract example and a concrete example.

6.5.2 External Validity

Is the cause and effect relationship I have shown valid in other situations?

Our results are only valid for the choice of control variables. This can be problematic as, for example, my test questions might not actually be suitable for checking if my learning goals have been reached.

6.5.3 Construct Validity

Do the groups involved in the experiment equally balanced to have a positive effect on the outcome I measure?

It might be possible that students of equal caliber are sitting together or might have a discussion between the test to have a negetive effect on the results. Hence to avoid that, I have randomly selected students for control and treated group by giving them instruction sheets randomly while entering the class and making them sit in two separate groups apart from each other.

6.5.4 Conclusion Validity

Does the treatment/change I introduced have a statistically significant effect on the outcome I measure? Can I draw conclusions based on the data?

A problem could be that participants just guess the answers wildly. So it might possible that the data could be faked or incorrect due to mistakes. To avoid faking of data and to add more authenticity to the experiment, I have added certainty values for each question so that I will get to know whether the participant is just guessing the answer or certain about it. Also, I have scrambled the order of the scenarios given to the treated group to play with and the questions asked so that participants will not get a clue about the relation between them.

6.6 Data Analysis, Results, and Discussion

To get the results for my experiment, I have investigated on each hypothesis with the data collected separately. Analysis of the data, descriptive statistics are used to visualize the data collected and described in the following subsections.

6.6.1 Analyzing Hypothesis 1

Hypothesis 1 i.e., $\mathbf{H}_{OP1}$ and $\mathbf{H}_{NU1}$ is designed to investigate research question **RQ 3.1**.

Formula Data for **RQ 3.1** can be collected from all pretest results i.e., combining the pretest results from both control and treated group. As five questions were asked, result can be calculated for each question from the pretest data.

Calculation of the result for each question involves the correctness and certainty of the answers given by the students. For ith Question Q_i,

$CO_{pre(i)}$ ε {-1, 1}, correctness of the answer to the ith pretest question is either right (1) or wrong (-1).

$CE_{pre(i)}$ ε [0, 1], certainty of the answer to the ith pretest question is between 0 ("I just guessed") to 1 ("I am certain").

$RES_{pre(i)} := CO_{pre(i)} * CE_{pre(i)}$ ε [-1, 1], result for the ith pretest question is between -1 to 1.

For the final calculation, following result will be considered:
$R_{pre(i)}$:= Set containing all the $RES_{pre(i)}$ calculated from the answers to the ith pretest question.

Null Hypothesis, H_{NU1}: Mean ($R_{pre(i)}$) <= 0

Operational Hypothesis, H_{OP1}: Mean ($R_{pre(i)}$) > 0

Data Analysis

Discussion

6.6.2 Analyzing Hypothesis 2

Hypothesis 2 i.e., $\mathbf{H}_{OP2}$ and $\mathbf{H}_{NU2}$ is designed to investigate research question **RQ 4.1**.

Formula Data for **RQ 4.1** can be collected from all pretest and posttest results i.e., combining the pretest results from both control and treated group and combining the posttest results from both control and treated group. As five questions were asked, result can be calculated for each question from the pretest and posttest data.

Calculation of the result for each question involves calculation of improvement in correctness and certainty of the answers given by the students in posttest than pretest. For ith Question Q_i,

$CO_{pre(i)}$ ε {-1, 1}, correctness of the answer to the ith pretest question is either right (1) or wrong (-1).

$CE_{pre(i)}$ ε [0, 1], certainty of the answer to the ith pretest question is between 0 ("I just guessed") to 1 ("I am certain").

$RES_{pre(i)} := CO_{pre(i)} * CE_{pre(i)}$ ε [-1, 1], result for the ith pretest question is between -1 to 1.

$CO_{post(i)}$ ε {-1, 1}, correctness of the answer to the ith posttest question is either right (1) or wrong (-1).

$CE_{post(i)}$ ε [0, 1], certainty of the answer to the ith posttest question is between 0 ("I just guessed") to 1 ("I am certain").

$RES_{post(i)} := CO_{post(i)} * CE_{post(i)}$ ε [-1, 1], result for the ith posttest question is between -1 to 1.

$IMP_{(i)} := (RES_{post(i)} - RES_{pre(i)})/2$ ε [-1, 1], improvement for the ith question from pretest to posttest is between -1 to 1.

For the final calculation, following result will be considered:
$I_{(i)}$:= Set containing all the $IMP_{(i)}$ calculated from the answers.

Null Hypothesis, H_{NU1}: Mean ($I_{(i)}$) <= 0

Operational Hypothesis, H_{OP1}: Mean ($I_{(i)}$) > 0

Data Analysis

Discussion

6.6.3 Analyzing Hypothesis 3

Hypothesis 3 i.e., $\mathbf{H}_{OP3}$ and $\mathbf{H}_{NU3}$ is designed to investigate research question **RQ 4.2**.

Formula Data for **RQ 4.2** can be collected from all pretest and posttest results separately from control and treated group. As five questions were asked, result can be calculated for each question from the pretest and posttest data separately for control and treated group.

Calculation of the result for each question involves calculation of improvement in correctness and certainty of the answers given by the students in posttest than pretest separately for control and treated group. For ith Question Q_i,

Parameters used in the below formulas e.g., $RES_{post(ci)}$, $RES_{pre(ci)}$, $RES_{post(ti)}$, $RES_{pre(ti)}$ are calculated similar to the formulas as described in Section 6.6.2 but separately for control and treated group.

$IMP_{(ci)}$:= ($RES_{post(ci)}$ - $RES_{pre(ci)}$)/2 ε [-1, 1], improvement for the control group for the ith question from pretest to posttest is between -1 to 1.

$IMP_{(ti)}$:= ($RES_{post(ti)}$ - $RES_{pre(ti)}$)/2 ε [-1, 1], improvement for the treated group for the ith question from pretest to posttest is between -1 to 1.

For the final calculation, following result will be considered:
$I_{(ci)}$:= Set containing all the $IMP_{(ci)}$ calculated from the answers of the control group.

$I_{(ti)}$:= Set containing all the $IMP_{(ti)}$ calculated from the answers of the treated group.

Null Hypothesis, H_{NU1}: Mean ($I_{(ci)}$) - Mean ($I_{(ti)}$) < 0

Operational Hypothesis, H_{OP1}: Mean ($I_{(ci)}$) - Mean ($I_{(ti)}$) > 0

Data Analysis

Discussion

7 Summary and Future Work

7.1 Conclusion

Finding the limitations of the tool, benchmarx.

7.2 Future Work

In this thesis, I have designed an application framework for the bx tool demonstrator, implemented the demonstrator based on a concrete example, and further evaluated the demonstrator to check its effectiveness in order to spread the bx concepts to a wide spread audience to the best of my knowledge and belief. However, I could not realize a few other features due to time constraints. In the following paragraphs, I discuss some enhancements that I would like to apply to my work in the future.

New Rulesets

New Examples I have implemented one example in the demonstrator. However, it would be fascinating to see the implementation of a few more examples on the same or different UI platform but on the top of the same application framework.

Another BX tool

BX Tool as a webservice

New experiment only showing a video to the control grp

8 Appendix

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

List of Figures

List of Tables

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, 17

[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, 17

[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. 1, 2, 26

[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. 26

[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. 7, 10, 26, 27

[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. 19

[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. 19

[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, 19

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

[11] BX Community, [Online]. Available: http://bx-community.wikidot.com/examples:home 19, 24

[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. 2, 17, 19

[14] Z. Hu and H.-Shang Ko, *Principle and Practice of Bidirectional Programming in BiGUL*, Tutorial [Online]. Available: http://www.prg.nii.ac.jp/project/bigul/tutorial.pdf 19

[15] *BiYacc, tool designed to ease the work of writing parsers and printers*, [Online]. Available: http://biyacc.yozora.moe/ 17

[16] *SHARE - Sharing Hosted Autonomous Research Environments*, [Online]. Available: http://is.ieis.tue.nl/staff/pvgorp/share/ 17

[17] *VM Virtual Box*, [Online]. Available: https://www.virtualbox.org/ 17

[18] A. Bucaioni and R. Eramo, *Understanding bidirectional transformations with TGGs and JTL*, In Proc. of the Second International Workshop on BX, 2013. 12

[19] F. Hermann et al., *Model synchronization based on triple graph grammars: correctness, completeness and invertibility*, Software and Systems Modeling, vol. 14, pages 241-269, 2013. 10, 12

[20] 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 4, 51

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

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

[23] 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

[24] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns*, Addison-Wesley, Boston Massachusetts USA, vol. 47, 1995. 26

[25] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Abstraction and Reuse of Object-Oriented Design*, In Nierstrasz O.M. (eds) ECOOP 1993 - Object-Oriented Programming. Lecture Notes in Computer Science, vol 707. Springer, Berlin, Heidelberg. 26, 27

[26] J. Deacon, *Model-view-controller (mvc) architecture*, Computer Systems Development, pages 1-6, 2009. 30, 36

[27] D. Distante, P. Pedone, G. Rossi and G. Canfora, *Model-driven development of Web applications with UWA, MVC and JavaServer faces*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 4607, pages 457-472, 2007. 30, 42

[28] E. Freeman, E. Robson, K. Sierra and B. Bates, *Head First Design Patterns*, O'Reilly, USA, pages 536-566, 2004. 27, 30, 36

[29] *MDN, "Canvas API",* Retrieved May, 2017, from https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API. 39

[30] *Fabric.js,* [Online]. Available: http://fabricjs.com/. 39

[31] *Processing.js,* [Online]. Available: http://processingjs.org/. 39

[32] *Pixi.js,* [Online]. Available: http://www.pixijs.com/. 39

[33] D. Goodman and M. Morrison, *Javascript Bible*, Wiley Publishing Inc., 6th Edition, pages 1-8, Indianapolis, Indiana, 2007. 39

[34] J. Tidwell, *Designing interfaces*, O'Reilly, vol. XXXIII, Pages 81-87, 2012.

[35] K. Kusano, M. Nakatani and T. Ohno, *Scenario-based interactive UI design*, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13, 391, 2013.

[36] J. Hunter and W. Crawford, *Java servlet programming*, vol. 37, 2001. 42, 43

[37] P. Sharma and S. Dhir, *Functional & non-functional requirement elicitation and risk assessment for agile processes*. International Journal of Control Theory and Applications, vol. 9, pages 9005-9010, 2016. 14, 15

[38] P.L. Bonate, *Analysis of pretest posttest designs*, FL: Chapman & Hall/CRC, pages 1-5, 2000. 53

[39] D.T. Campbell & J.C. Stanley, *Experimental and quasi-experimental designs for research* , In N. L. Gage (Ed.), Handbook of research on teaching. Chicago: Rand McNally, 1963. 53

[40] S.W. Huck & R.A. McLean, *Using a Repeated Measures ANOVA to Analyze the Data from a Pretest-Posttest Design: A Potentially Confusing Task* , Psychological Bulletin, vol. 82, No. 4, pages 511-518, 1975. 53

[41] R. Feldt & A. Magazinius, *Validity Threats in Empirical Software Engineering Research - An Initial Survey* , In Proc. of the Int'l Conf. on Software Engineering and Knowledge Engineering, issue March, pages 374-379, 2010. 59

[42] C. Wohlin, M. HÃ¶st, P. Runeson, M. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in software engineering: an introduction* , Kluwer Academic Publishers, 2000. 59