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 Software Engineering Research Group
in Partial Fulfillment of the Requirements for the
Degree of

Master of Science

by
ARJYA SHANKAR MISHRA
Alois Fuchs Weg 7
33098 Paderborn

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

Paderborn, July 21, 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 faster, less complex, and more human-friendly.

One of the approaches for reducing the complexity of software development is abstraction and separation of concerns [24]. 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 [24]. It reduces development costs and increases the reusability and maintainability of software. The objective of MDSD [24] 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 [23].

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 of 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, I 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 the 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 teaching 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: To what extent is such a bx demonstrator reusable?

RQ 3: Is there a need 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?

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
- Teach bx concepts to a wide audience and make them accessible and understandable

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

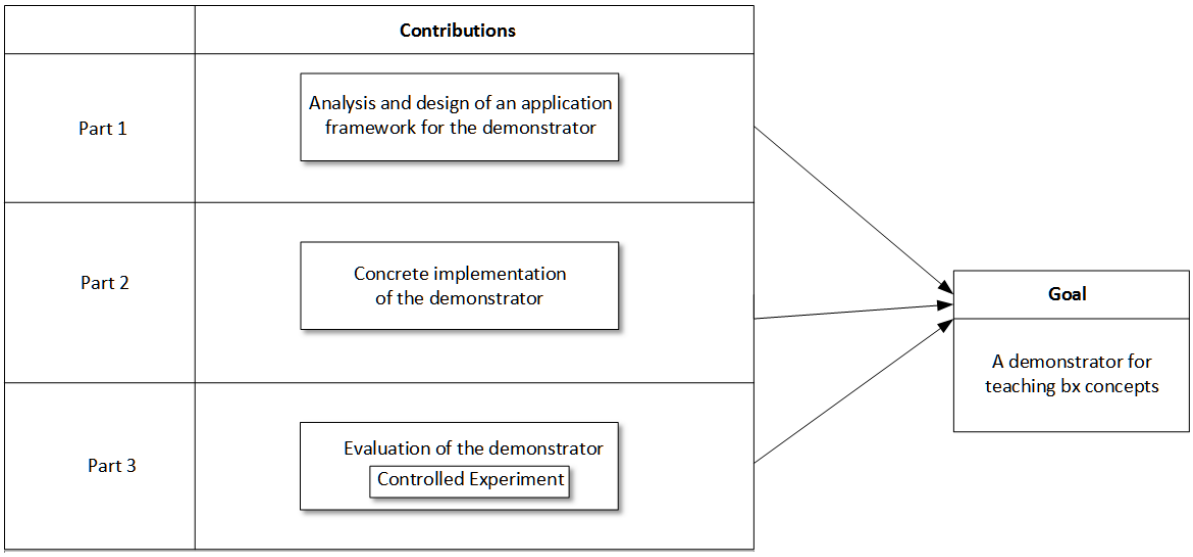


Figure 1: Contributions

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 an 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 was needed. Hence, 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 bx tools with concrete examples.

Along with this achievement, a concrete implementation of a demonstrator to check the feasibility and validity of the application framework was needed. Hence, I have implemented a fully functional online demonstrator (*Demon-BX*[1]) 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 was needed. Hence, I have evaluated the demonstrator based on the research method called controlled experiments [21] for the accomplishment of point (3). This experiment was based on three hypotheses, which helped in deciding which variables, i.e., dependent, independent, and controlled to include and how to measure them. A well-designed experiment was performed with two different groups of participants and collected data were analyzed to measure the outcome.

All of my contributions help to achieve the goal of this thesis, i.e., a demonstrator for teaching 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 the last few years and their related problems. Chapter 5 describes all the high-level design and implementation details for the demonstrator along with UML diagrams. Chapter 6 contains the 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.

[1] *Short for a demonstrator for bidirectional transformation*

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 fundamentals of bx for a 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 relationship between the *Kitchen Model* denoted by *Kitchen* (right-hand side canvas) and the *Grid Model* denoted by *Layout* (left-hand side canvas).

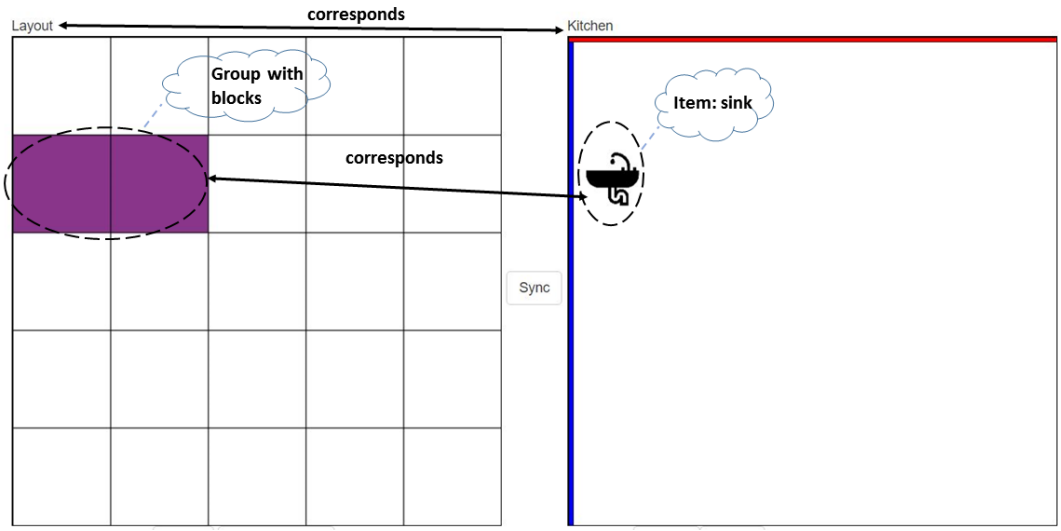


Figure 2: Running Example: Layout and Kitchen

The *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 colored groups of blocks organized in a grid. Here, the *Kitchen* corresponds to the

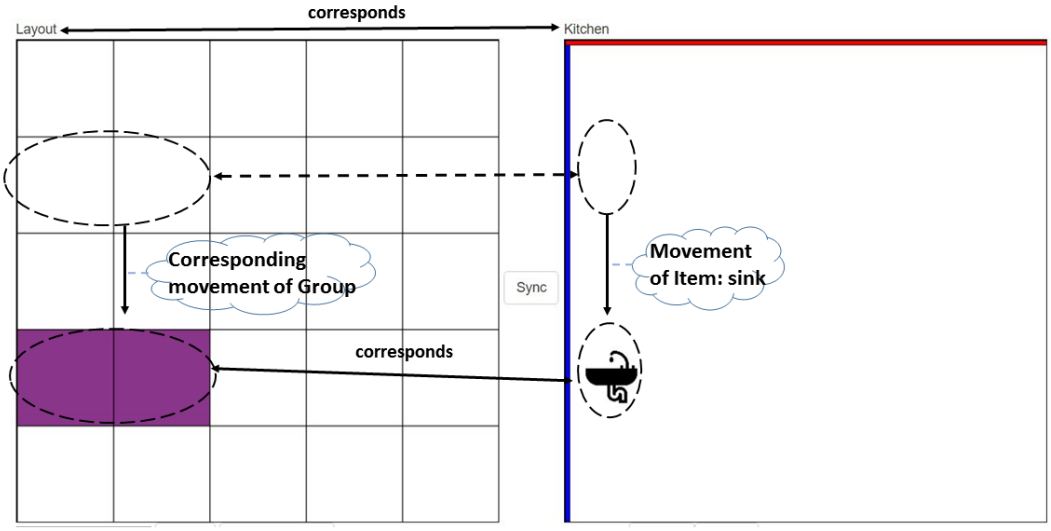


Figure 3: Running Example: Consistency Preservation

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 [22] [23]. Model creation helps in keeping a clear focus on selected concepts and rules relevant for a particular concern and omitting irrelevant details.

A meta-model describes a set of models, i.e., defines a modeling language.

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

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

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.

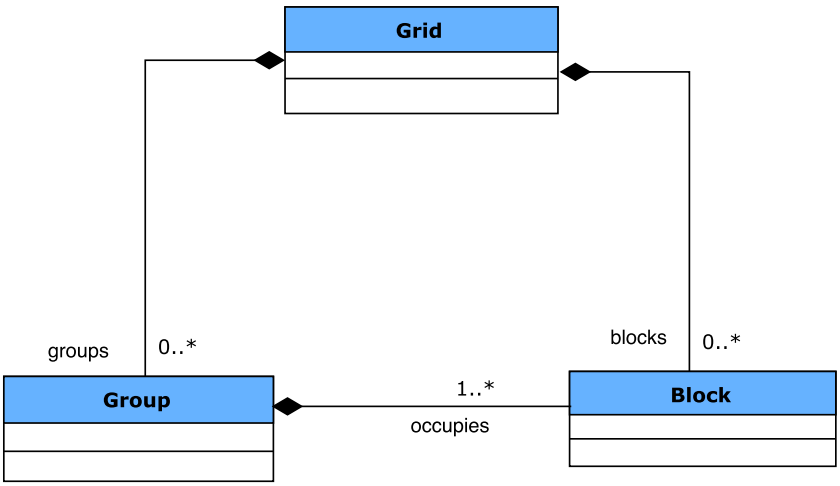


Figure 4: Grid Meta-Model

Figure 6 illustrates an abstract and a 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.

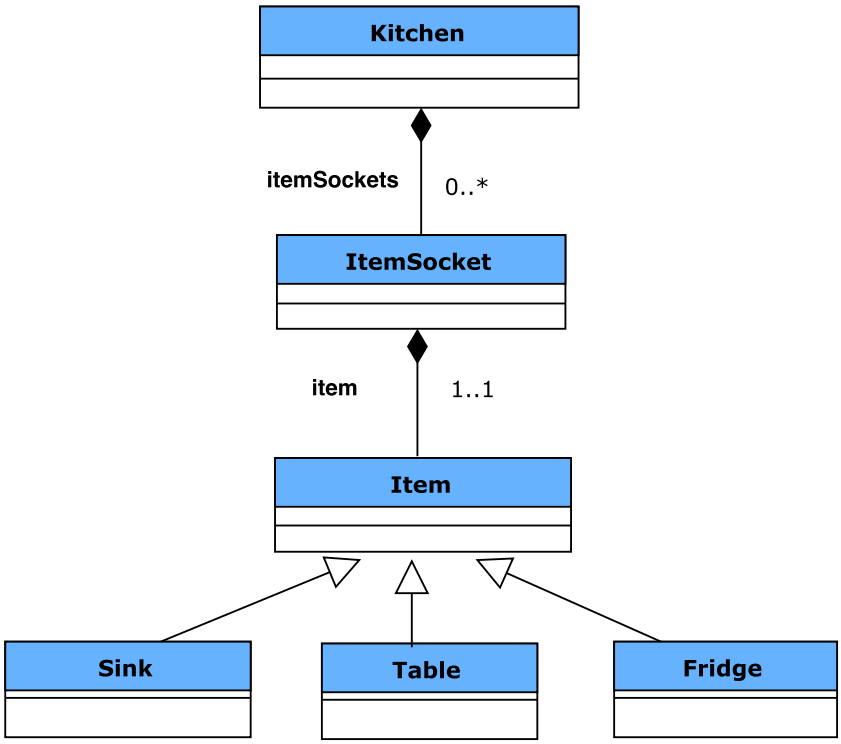


Figure 5: Kitchen Meta-Model

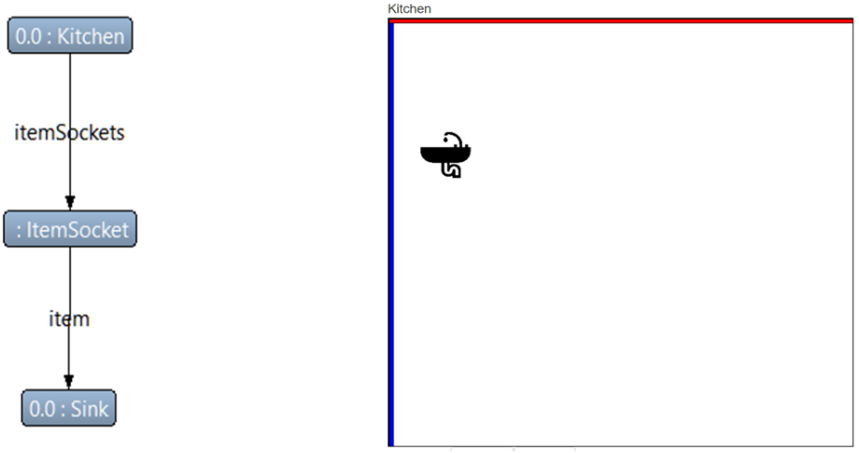


Figure 6: Kitchen Model (Abstract & Concrete)

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

Table 2: Examples of Delta in Kitchen Model

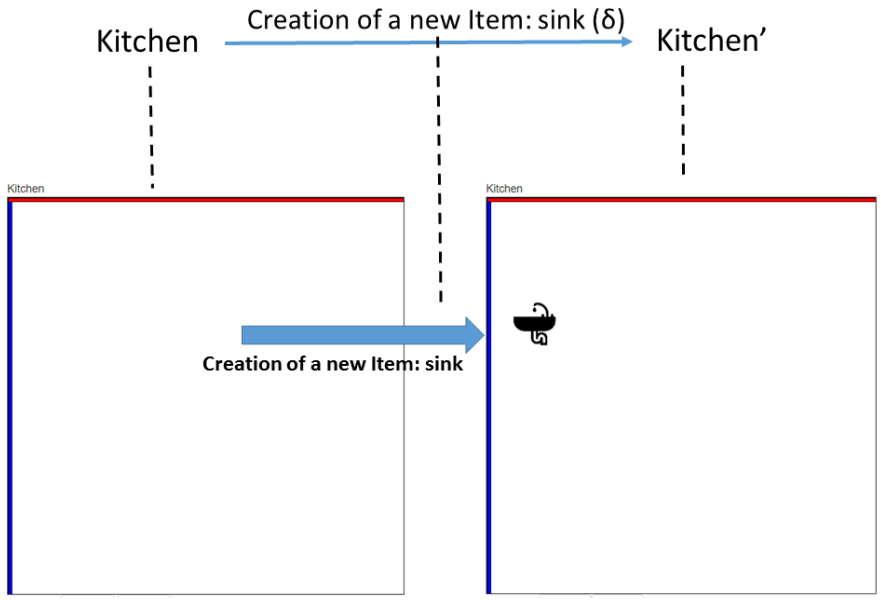


Figure 7: Delta Propagation

Definition 2 *(Delta)*
A delta is a change applied to one or more properties of an artifact. 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 artifact (models of the same type) and all the deltas which lead from one model to another.

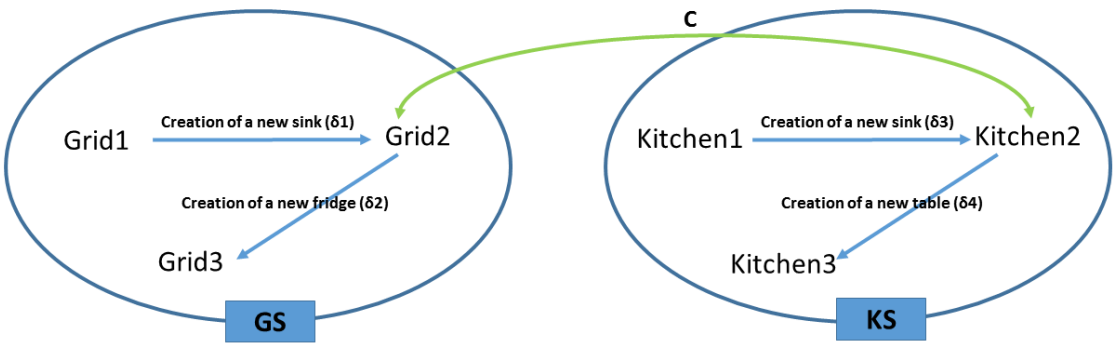


Figure 8: Model Space

Example: In my demonstrator example, a concrete example of a subset of a model space is shown in Figure 8. GS contains the states Grid1, Grid2, and Grid3 of the *Grid* Model along with deltas $\delta 1$ and $\delta 2$. Whereas, KS contains the states Kitchen1, Kitchen2, and Kitchen3 of the *Kitchen* Model along with deltas $\delta 3$ and $\delta 4$.

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 [20]. 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 example of consistency preservation where a change in the *Kitchen* (Kitchen model) i.e.,

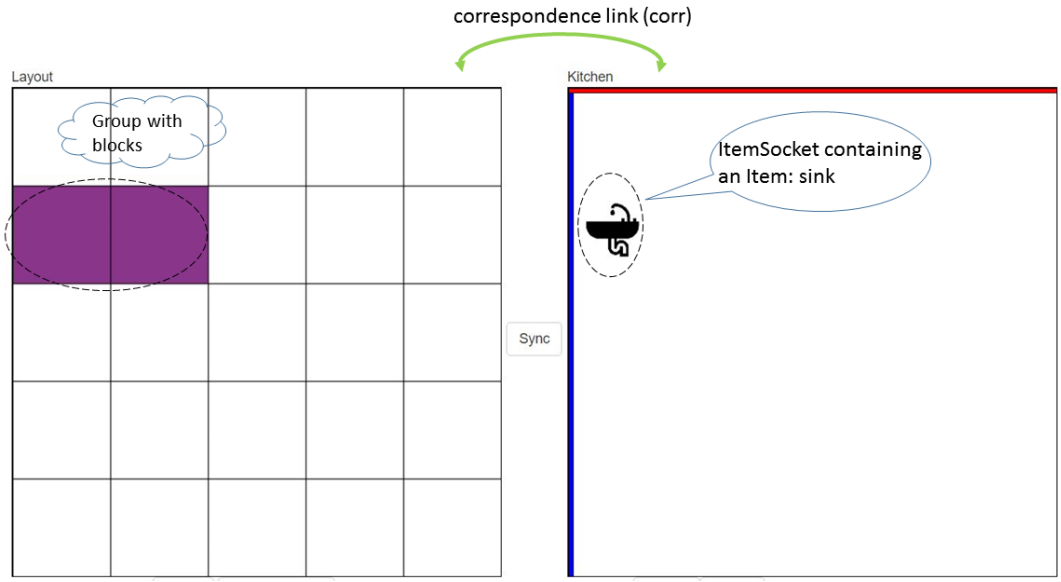


Figure 9: Correspondence Links

movement of the sink ensures 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 the source and target model is always ensured as model transformation ensures that for each consistent source model there exist a consistent target model [20].

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 [19].

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".

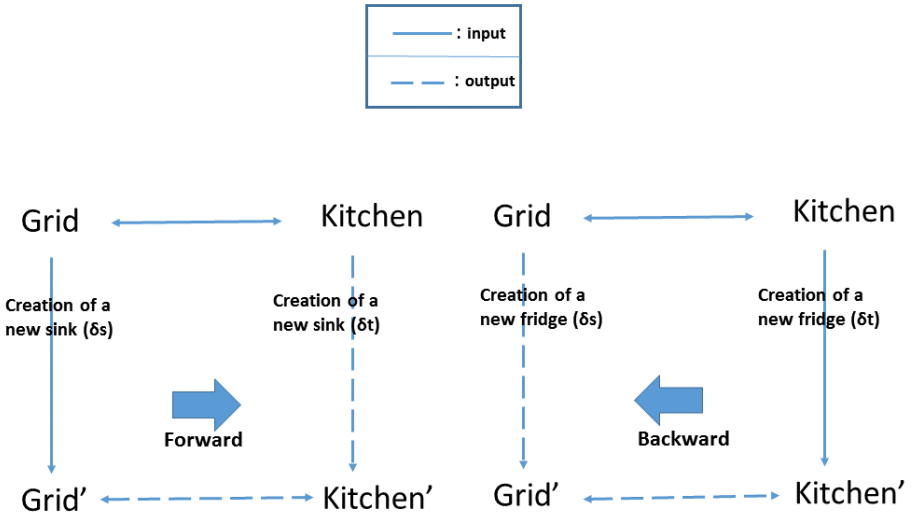


Figure 10: Bidirectional Transformation

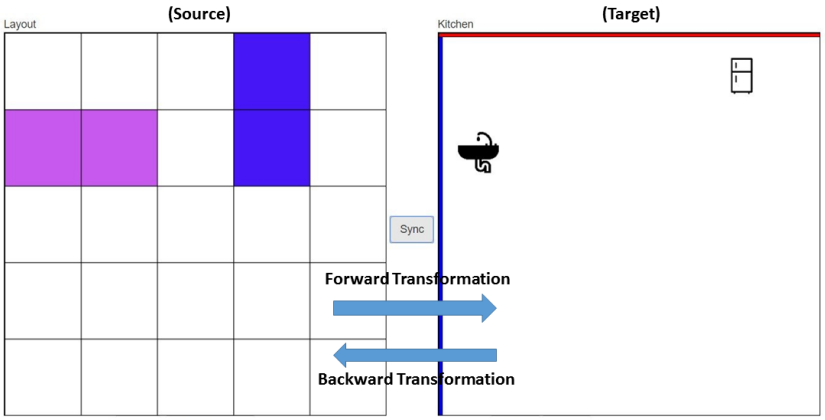


Figure 11: Transformation (Concrete Diagram)

Figure 10 shows an abstract example of the "BX" describing both forward and backward 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 [36] 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 defined separately in the corresponding views.

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 synchronization 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 [36].

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

NR1: Minimal installation time

The demonstrator should not take much time to install and should be up and running very fast. It should involve simple and easy steps so that it can be carried out by anyone.

NR2: Fun to play

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: Simple example

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

NR4: User guidance

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. The demonstrator should guide the user through all the aspects that are intended to be taught/learned with it.

NR5: Informative

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

NR6: Public access

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

NR7: Robust

The application should be platform independent, i.e., it should run on any operating system and all kinds of browser. Different environment on the user's machine should not affect the application.

NR8: Minimal requirements on user's environment

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

NR9: Easily deployable project

The 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: Extendable

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. Also, 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.

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. Finally, Section 4.5 explains the related problems and their comparison with my demonstrator i.e., Demon-BX.

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. The 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 transformation 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. Bx community has

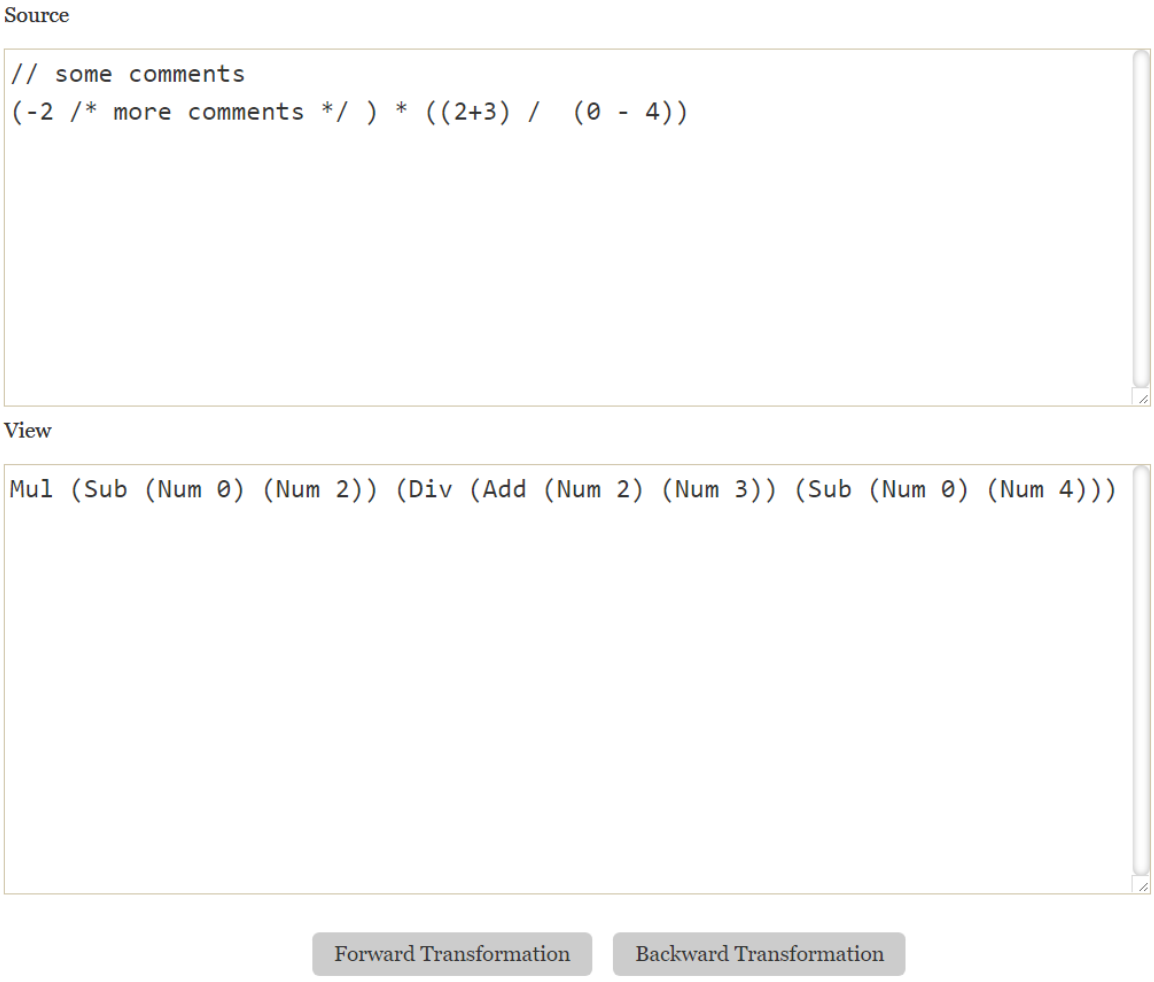


Figure 12: Web GUI for Demonstrator: Biyacc

prepared a collection of virtual machines (e.g., for various TGG tools) hosted on SHARE and VM Virtual Box.

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 simplifies the access process for end-users. To access the shared material, virtual machine either has to be 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[2], 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.

I 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 website (`http://bx-community.wikidot.com/`). A user can find relevant information about the examples on the respective web pages. Some examples are very well documented along with class diagrams, activity diagrams, object diagrams, etc. and source code of a few examples is available as well.

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

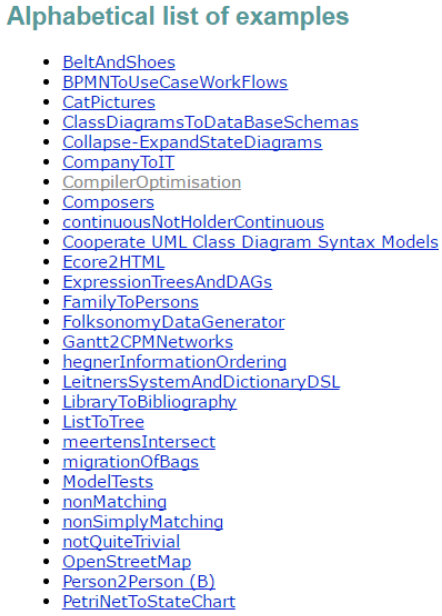


Figure 13: BX Example Repositories

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 needs 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, the installation process is time consuming and requires technical expertise 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 while forming the requirements for my demonstrator. To overcome these problems, I took the shortcomings from each related work and designed the functional and non-functional requirements.

Based on the requirements described in chapter 3 and considering the problems as discussed in Section 4.5.1, Table 3 illustrates a comparison between all the related work discussed above with the demon-bx tool. "✓" denotes the fulfillment of the requirement, "✗" denotes the non-fulfillment of the requirement, and "?" denotes insufficient information about the requirement. This comparison table clearly shows that the demon-bx tool overcomes all the problems associated with these related work.

Related Work	Requirements													
	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 steps taken in selecting the bx-tool in Section 5.2. Section 5.3 deals with the decisions taken for finalizing the application architecture design. Section 5.4 describes the application framework in detail along with its components. Finally, Section 5.5 describes the challenges that I faced during the design and implementation of the entire framework.

My demonstrator is called *Demon-BX*. Demon-BX is designed in such a way that the implementation is extensible for a family of (similar) examples, and for more than one bx tool. Due to time limitations and scope of this thesis, I chose one of each for the implementation process.

5.1 Choosing an Example

To solve the problems 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 as candidates for the implementation as follows:

Task Management This idea is influenced from the project management tool such as *Trello* (`https://trello.com/`), a web-based project management tool for organizing and managing project related work. This prototype can be used for allocating tasks in a team. It contains two views, a supervisor's view and an employee's view. A supervisor can allocate tasks to their subordinates. An employee can view only 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. The 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. A bx controls 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.

This example is well-known among the developer's community. But, the example contains a set of rules related to project management which is complex to operate on and user interaction does not come naturally with it. In this example, a user can relate to the bx concepts with both

forward and backward synchronization, but it involves texts, tables etc. which is not intuitive for a user.

Quiz This prototype can be used for an online quiz game. It contains two views, an administrator's view and a 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 view 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 participants can see questions along with its options. As soon as the participant chooses the answer to any question, a bx controls how updates are handled and states are reflected in the different views of the project.

Most of the students and computer game lovers are familiar with this example. But, the example contains a set of rules which a user first needs to understand before starting the game and user interaction does not come naturally with it. In this example, a user can relate to the bx concepts with both forward and backward synchronization, but it involves texts, menus, tables etc. which is not intuitive for a user.

Playing with Shapes This prototype contains two views, a low-level view (depicts a UI[3] for a low-level language, i.e., a UI with less functionality) and a high-level view (depicts a UI for a high-level language, i.e., a UI with more functionality). The user will sketch a geometric shape, i.e., triangle/square/rectangle/circle using the low-level view. The high-level view tries to recognize the geometric shape and draws it precisely. A bx controls how updates are handled and states are reflected in the different views of the project. In the high-level view, more functionality will be available, e.g.,, moving one shape from one place to another, creating a clone of an existing shape, rotations, etc. which is not possible in low-level view.

This example is based on a new concept and hence contains rules which are completely new to the users. But user interaction is an integral part of this example. In this example, a user can relate to the bx concepts with both forward and backward synchronization and involves shapes, mouse movements etc. which are very interesting for a user.

Planning a Kitchen This prototype contains two views, layout view (depicts a grid structure containing blocks) and a symbolic view (empty space which depicts the 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 the layout view. The user will create/delete/move a

[3] *Short for User Interface*

kitchen item, i.e., sink/table in the symbolic view and if changes are compatible with a bx then the items will be reflected in the layout view with colored blocks and vice-versa.

Any user can relate to this example very well as everybody is familiar with a kitchen and its environment. This example involves no technical details and simple rules as the environment is a part of the day to day life and hence users will be able to relate to the learning concepts easily. As an integral part of this example, user interactivity involves shapes, colors, drag, and drop, etc. which creates interest for a user.

Families to Persons The concept of this example is taken from the example *FamilyToPersons* located in the bx examples repository [11]. It contains two views, a Families view and a Persons view. The Families view contains many families and each family consist of members. The Persons view contains persons (corresponding to the members of each family). The addition of a new person to the Persons view will be reflected in the Families view and vice-versa. Surnames don't have to be unique. If a new person could be placed in multiple families the synchronizer can simply ask which one is to be chosen (user interaction).

The potential users are not familiar with this example. However, this example is very simple to understand as everybody is familiar with the concept of a family and hence users will be able to relate to the learning concepts easily. But user interactivity does not involve any colors, shapes, mouse movements, etc. to attract a user.

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 selection of the example and its implementation will directly affect the research question *RQ 4* specified in Section 1.1 and requirements *NR2* (fun to play), *NR3* (simple example) described in Section 3.2.

Hence, by taking into account all these factors, I and my supervisor constructed a few evaluation criteria (referred to as **EC**), as due to time constraint, we could not ask potential users or conduct an experiment to find out what people will prefer. Then, I evaluated each example on the basis of the evaluation criteria for the selection of the most suitable example. The evaluation criteria that I have considered are listed below:

EC 1: User should be familiar with the example.

EC 2: Example should be simple and intuitive with a minimum of technical details.

EC 3: Interactivity should be a natural part of the example.

EC 4: Both the forward and backward direction of consistency restoration should make sense for the example.

EC 5: Example should be visual and involve shapes, colors and drag and drop (as opposed to text, menus, or tables).

	Evaluation Criteria				
Examples	**EC1**	**EC2**	**EC3**	**EC4**	**EC5**
Task Management	✓	✗	✗	✓	✗
Quiz	✓	✗	✗	✓	✗
Playing with Shapes	✗	✗	✓	✓	✓
Person and Family	✗	✓	✓	✓	✗
Planning a Kitchen	✓	✓	✓	✓	✓

Table 4: Comparison of examples based on evaluation criteria

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 *Planning a Kitchen* is the most suitable example as it fulfills all the evaluation criteria. 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 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 bx tools like eMoflon, Echo, BIGUL. Also, I came across the benchmark [5] [6], the first non-trivial benchmark where Anjorin et al. provide 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 fairly generic 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 having good support. 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 e.g., BiGUL (Haskell-based), or NMF synchronization (C# based).
- the technologies that I planned to use (JavaScript, Tomcat, servlets) during my implementation is also rather Java 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* (minimal installation effort), *NR6* (public access), *NR7* (extendable), and *NR8* (minimal requirements on user's environment) described in Section 3.2.

Web application development has evolved from single page design to very complex layering structures since the beginning of World Wide Web. Many design patterns [25] [26] consisting of different technologies and programming languages were proposed, adopted and implemented to address the demands of customers and users on the web. With the rapid changes occurring on World Wide Web, some technologies are becoming obsolete and losing demand. Nowadays, the main trend appears to be improving user interaction and allowing developers to build powerful web applications rapidly.

I analyzed some design patterns and commonly used architecture designs 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. The main idea was thus to check the feasibility of the architecture and the data flow within its components on top of the interface provided by the *Benchmarx* framework (proposed by Anjorin et al. [6]) to access the bx-tool. While working on the PoCs, I encountered 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 one variant of the Model-View-Controller (referred as **MVC** from now on) architecture for my application framework. Following were the driving factors for the selection of the MVC architecture [26] [29]:

- it differentiates the layers of a model, view, and controller for the re-usability and easy maintenance of code.
- it splits responsibilities into three main roles which allow 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 number of views. Enhancements in views and support of new technologies for building the view can be implemented easily.
- a person working on the 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, Model, View and Controller to signify the MVC pattern. Figure 14 describes the high-level architecture of the demon-bx tool in the form of a component diagram.

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

The `view` component 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 is responsible for capturing all user-related actions.

The `controller` component acts as a bridge between view and model and handles all user requests. It consists of a few Java classes. The controller receives all the user requests sent from the web browser and calls the appropriate methods where the actual data conversion happens, i.e., conversion of user data to example specific data before sending them to the bx tool for

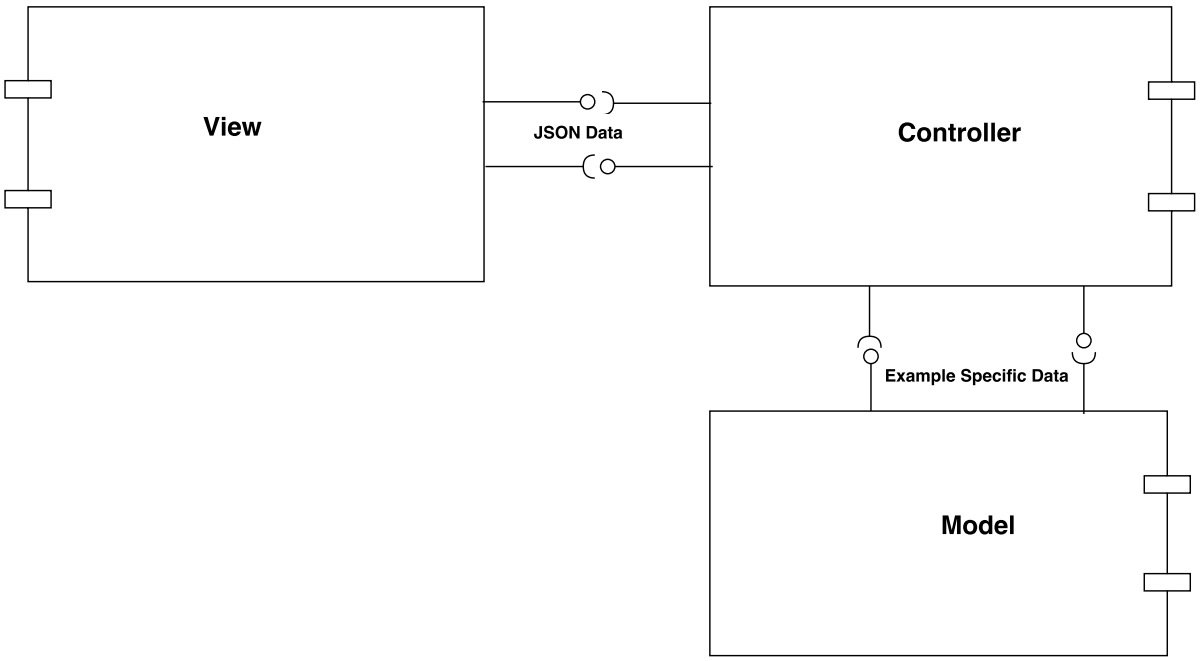


Figure 14: Component Diagram of Demon-BX Tool

further processing, and conversion of example specific data to user data after receiving the processed data from the bx tool.

Request-Response Cycle The sequence diagrams shown in Figure 15, Figure 16, and Figure 17 illustrate a high-level overview of the communication process between all three components in different stages.

1. **Load**: The user loads the application on a browser. The view generates a unique user id (GUID) and sends it to the controller as a request in the form of JSON data. After accepting the request, the controller stores the GUID and sends the initialize signal to the model for initializing the bx-tool. After tool gets initialized, the controller requests for the newly generated example specific models, converts it to the user data, and sends the response back to the view in the form of JSON data. The view processes the data and the final response is loaded into the browser. This entire process is shown in Figure 15.
2. **Delta Propagation (without continuation)**: After the application is loaded on the browser, the user creates a delta in one of the views and sends it to the controller along with the GUID generated earlier for the user as a request in the form of JSON data. After accepting the request, controller checks the existence of the GUID, converts the user data to the example specific data and sends it to the model for processing. During processing, the controller checks if satisfiability of more than one transformation rule exists for the processed data i.e., continuation. If continuation does not exist, the controller requests for

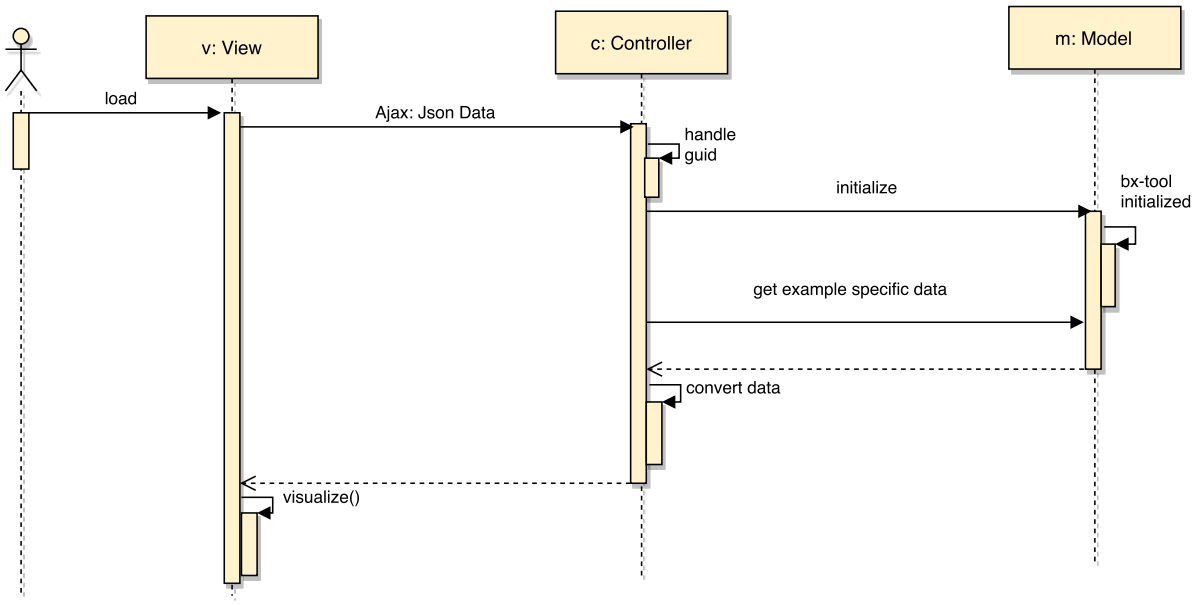


Figure 15: Sequence Diagram of Demon-BX Tool: initialization

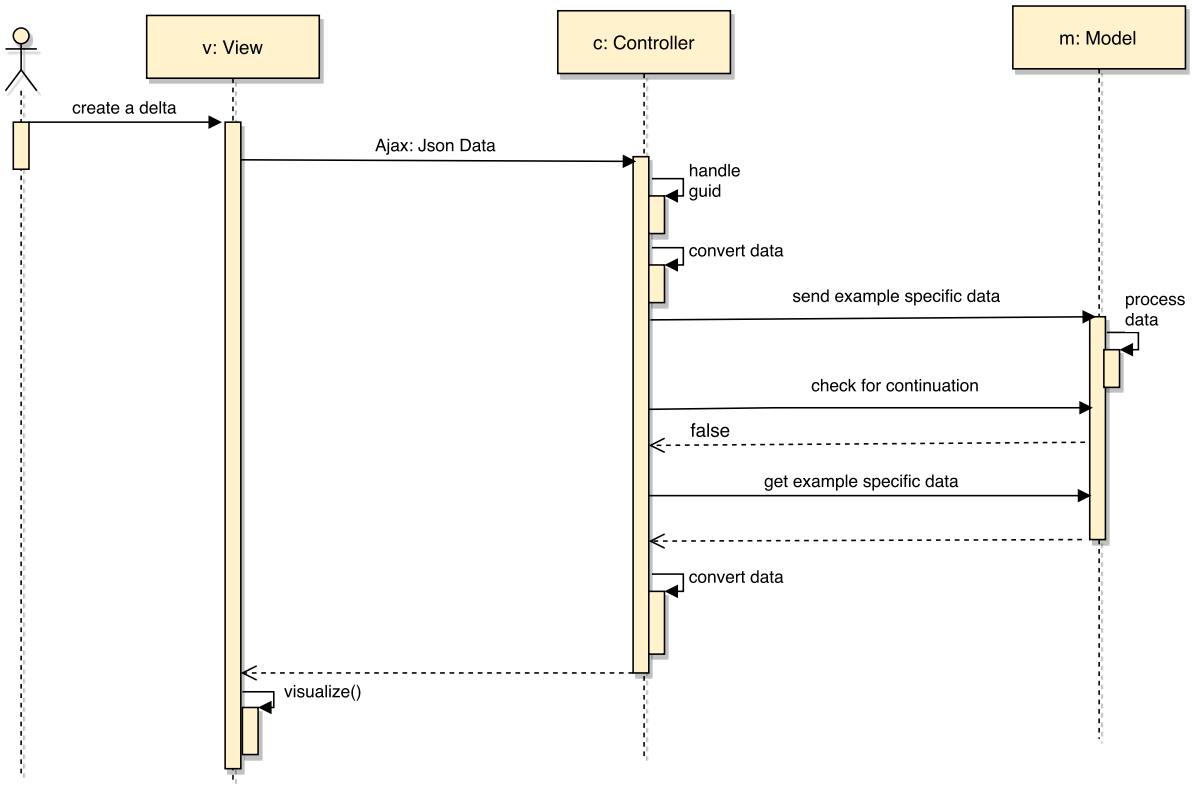


Figure 16: Sequence Diagram of Demon-BX Tool: delta propagation without continuation

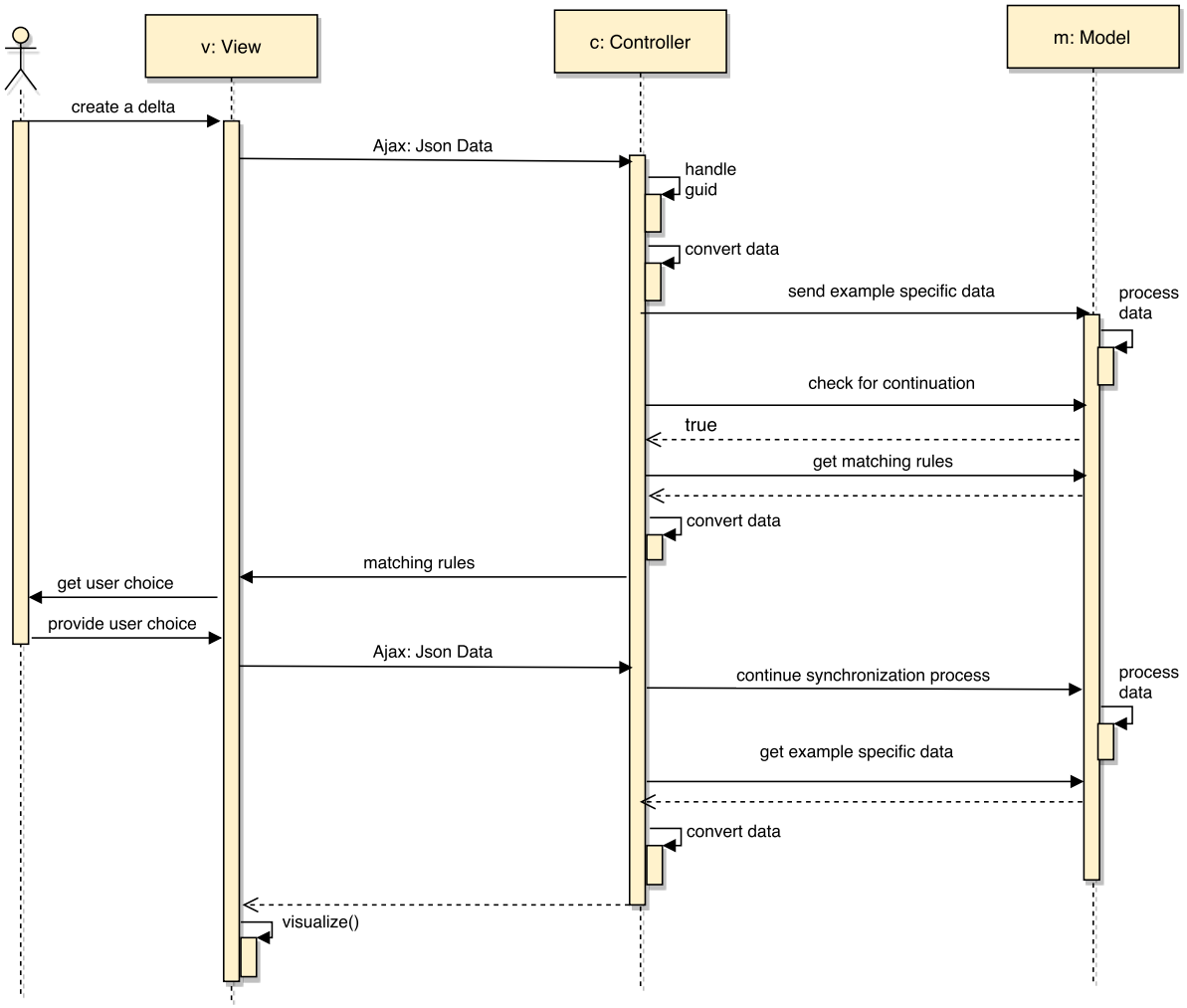


Figure 17: Sequence Diagram of Demon-BX Tool: delta propagation with continuation

the updated example specific models, converts it to the user data, and sends the response back to the view in the form of JSON data. The view processes the data and the final response is loaded into the browser. This entire process is shown in Figure 16.

3. **Delta Propagation (with continuation)**: While the bx tool processes the delta, the controller checks if satisfiability of more than one transformation rule exists for the processed data i.e., continuation. If continuation exists, the controller gets the matching rules from the model, converts it to the user data, and sends it back to the view. The view asks the user for a choice. After the input is given by the user, the view sends it to the controller in the form of `JSON` data. The controller forwards the user's choice to the model and bx tool resumes processing of the data after getting it. After data is processed, the controller requests for the updated example specific models, converts it to the user data, and sends the response back to the view in the form of JSON data. The view processes the data and the final response is loaded into the browser. This entire process is shown in Figure 17.

For better understanding and to avoid complexity, here I have described the delta propagation process for only a single delta. However, the user can generate multiple deltas as well. In that case, view collects all the deltas and sends them to the controller. After receiving the deltas, the controller handles them atomically (one by one).

5.4.2 Model

The model is mainly responsible for encapsulating the access of data and handling the business logic of the application [29] [27]. 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 [28].

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, `BX Tool Wrapper` and bx tool i.e., `eMoflon`.

The bx tool wrapper consists of interfaces and its concrete implementation designed to access the bx tool on the top of the framework provided by the `Benchmarx` [6], a common design space to access different bx tools. With the help of this interface, the controller communicates with the bx tool i.e., eMoflon. However, the interface presented to the controller is not eMoflon specific. The bx tool wrapper for eMoflon translates this generic bx tool interface to eMoflon's interface. The eMoflon tool contains all the meta-models, state of the meta-models and the associated transformation rules.

For implementing the example *Planning 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 the example has two

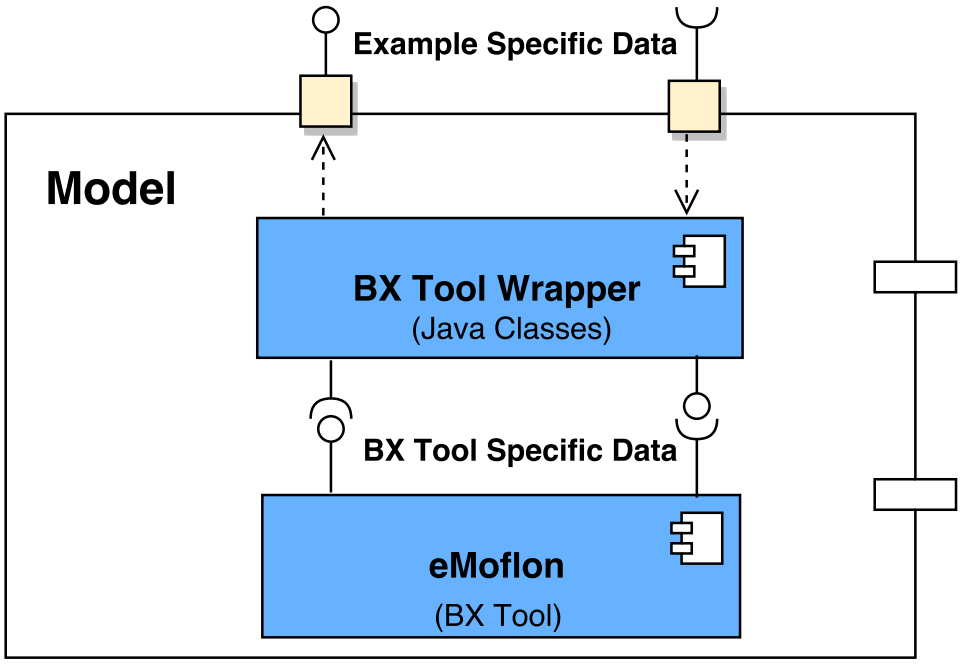


Figure 18: Component Diagram of Model

different views, it requires two different models in the tool as well i.e., `KitchenLanguage` to represent the symbolic view and `GridLanguage` to represent the layout view.

Figure 5 describes the structure of the `KitchenLanguage` as 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` as 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 groups, 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 The sequence diagrams shown in Figure 19, Figure 20, and Figure 21 illustrate the workflow of the demon-bx tool from the point of view of the model component in different stages. As the model component only communicates with the controller, these sequence diagrams show the communication process between the controller and the sub-components of

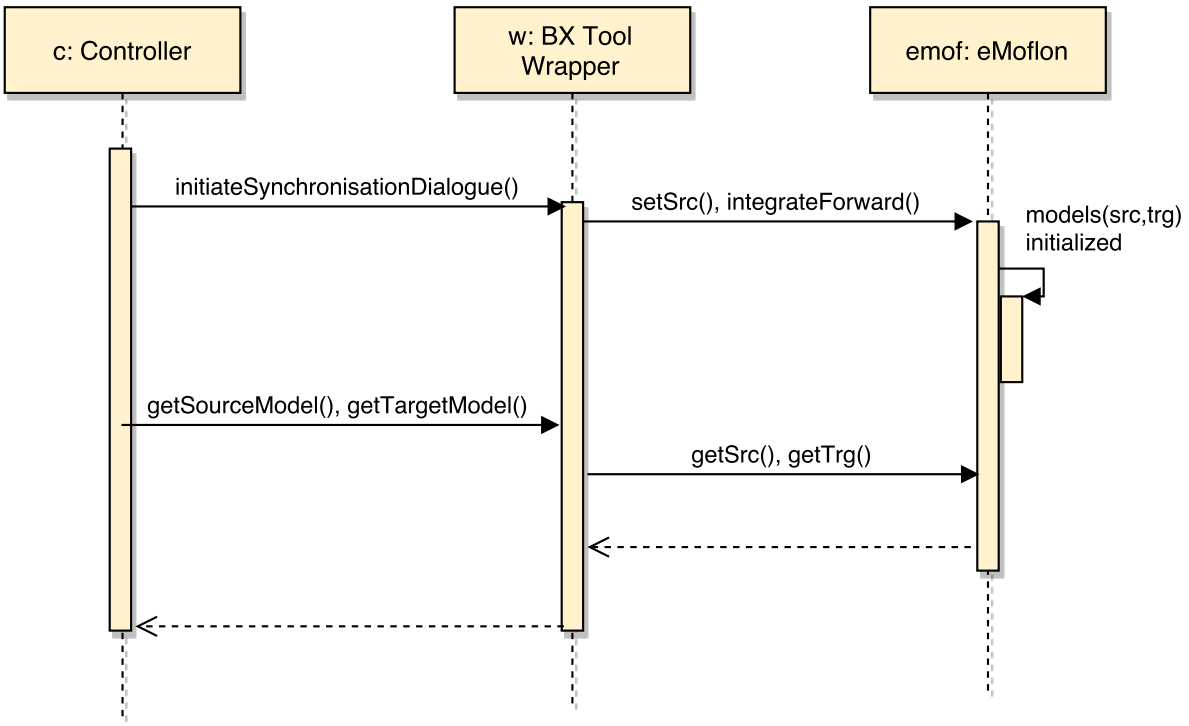


Figure 19: Sequence Diagram of Model: initialization

the model, `BX Tool Wrapper` and the bx tool i.e., `eMoflon`.

1. **Load**: During first time load of the application, the controller instantiates all the wrapper classes and calls the appropriate method (`initiateSynchronisationDialogue()`) to initialize the bx tool after receiving the initialization command from the view. After receiving the request, bx tool wrapper class sends the initialization signal (`setSrc()`, `integrateForward()`) to the bx tool i.e., eMoflon and the tool gets initialized by creating the instances of the `Source` and `Target` models. Then, the controller requests (`getSourceModel()`, `getTargetModel()`) for the newly generated models through the bx tool wrapper class. The wrapper class requests (`getSrc()`, `getTrg()`) for the tool specific models from the bx tool, receives it, and send it back to the controller. This entire process is shown in Figure 19.

2. **Delta Propagation (without continuation)**: After converting the delta sent from the view into example specific data, the controller calls the appropriate method (`performAndPropagateTargetEdit(SrcData)`) of the wrapper class e.g., propagate delta into source model. Accordingly, wrapper class initiate forward synchronization (`setChangeSrc(SrcData)`, `integrateForward()`) and sends the delta to the bx tool i.e., eMoflon. Here I have described only the forward synchronization. However, the process for the backward synchronization goes analogously. The bx tool processes the delta

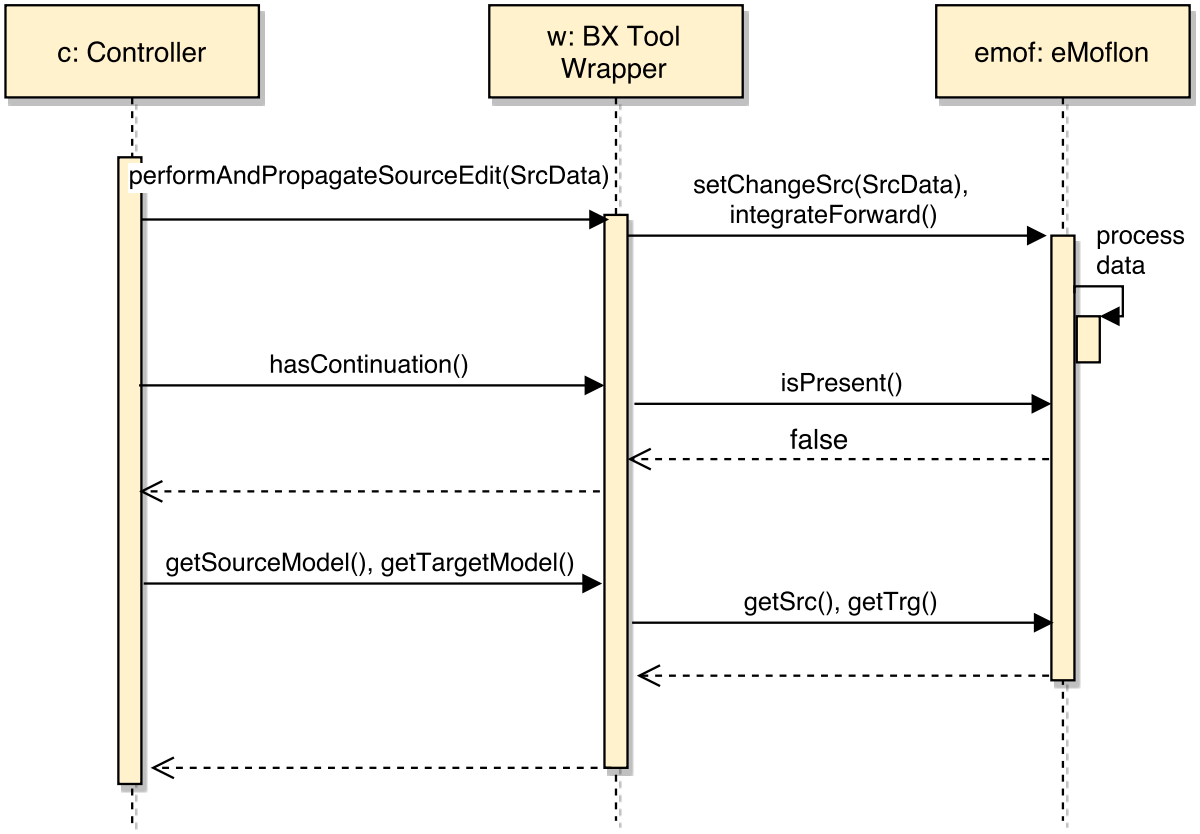


Figure 20: Sequence Diagram of Model: delta propagation without continuation

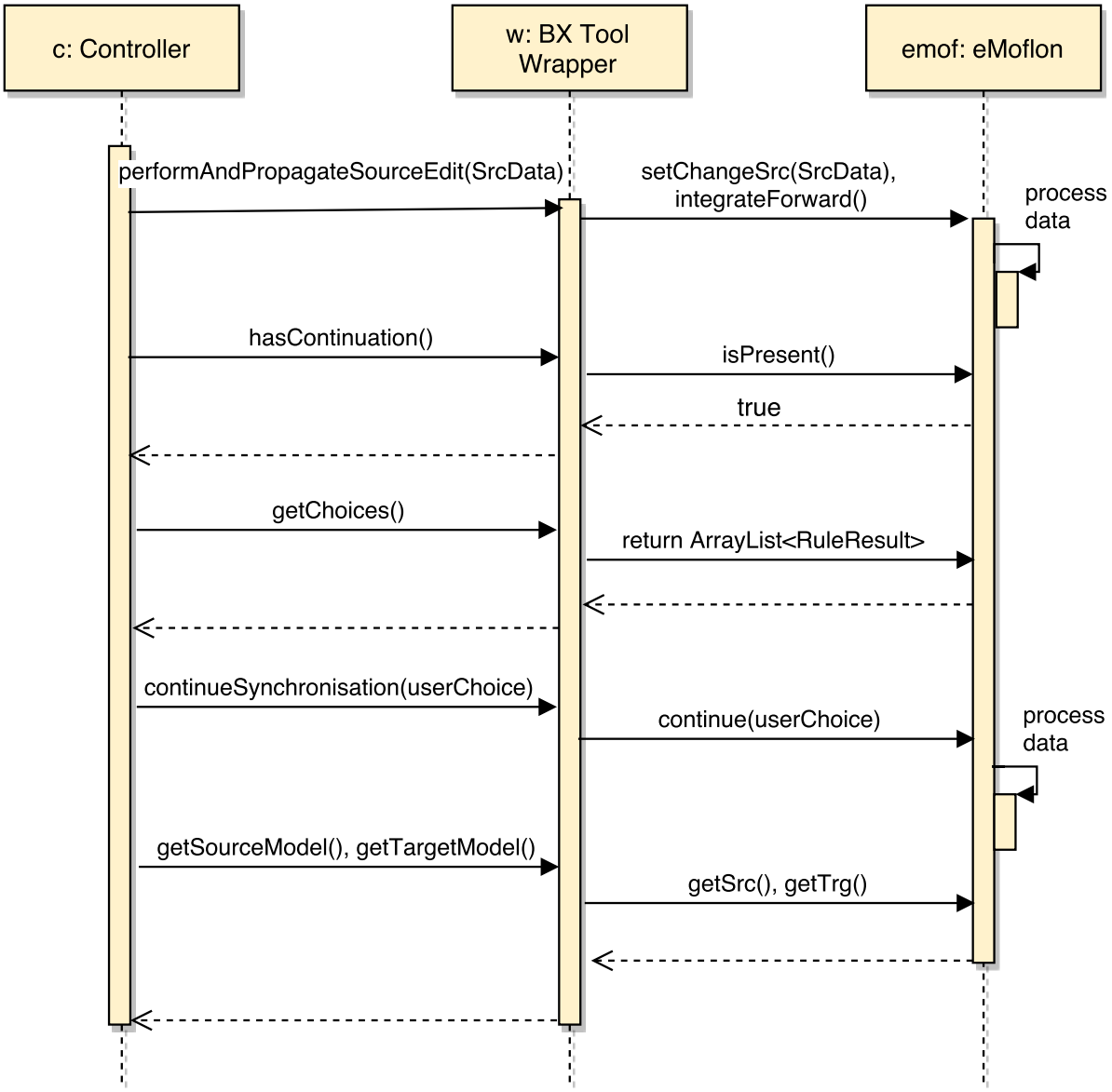


Figure 21: Sequence Diagram of Model: delta propagation with continuation

taking into account all the associated transformation rules. During processing of the data, the controller checks if satisfiability of more than one transformation rule exists (`hasContinuation()`) for the processed data i.e., continuation. If continuation does not exist, processing is completed and the controller requests (`getSourceModel()`, `getTargetModel()`) for the newly generated models through the bx tool wrapper class. The wrapper class requests (`getSrc()`, `getTrg()`) for the tool specific models from the bx tool, receives it, and send it back to the controller. This entire process is shown in Figure 20.

3. **Delta Propagation (with continuation)**: While the bx tool i.e., eMoflon processes the delta, the controller checks if satisfiability of more than one transformation rule exists (`hasContinuation()`) for the processed data i.e., continuation. If continuation exists, the controller requests (`getChoices()`) for the matching rules from the bx tool i.e., eMoflon through the bx tool wrapper class. The wrapper class requests, receives the matching rules from the bx tool, and send it back to the controller. On receiving the user's decision, the controller sends resume data processing signal (`continueSynchronisation(userChoice)`) along with the user's decision to the bx tool through the wrapper class. Afterward, the bx tool completes the data processing with the user's decision. Then, the controller requests (`getSourceModel()`, `getTargetModel()`) for the newly generated models through the bx tool wrapper class. The wrapper class requests (`getSrc()`, `getTrg()`) for the tool specific models from the bx tool, receives it, and send it back to the controller. This entire process is shown in Figure 21.

For better understanding and to avoid complexity, here I have described the delta propagation process for only a single delta. However, the user can generate multiple deltas as well. In that case, after receiving all the deltas from the view, the controller converts them to example specific data. Then, the controller handles them atomically (one by one). Hence, the above described delta propagation process (with or without continuation) is executed for each delta until the data processing is complete by the bx tool.

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 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 the case of an overlapping [29] [27]. 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.

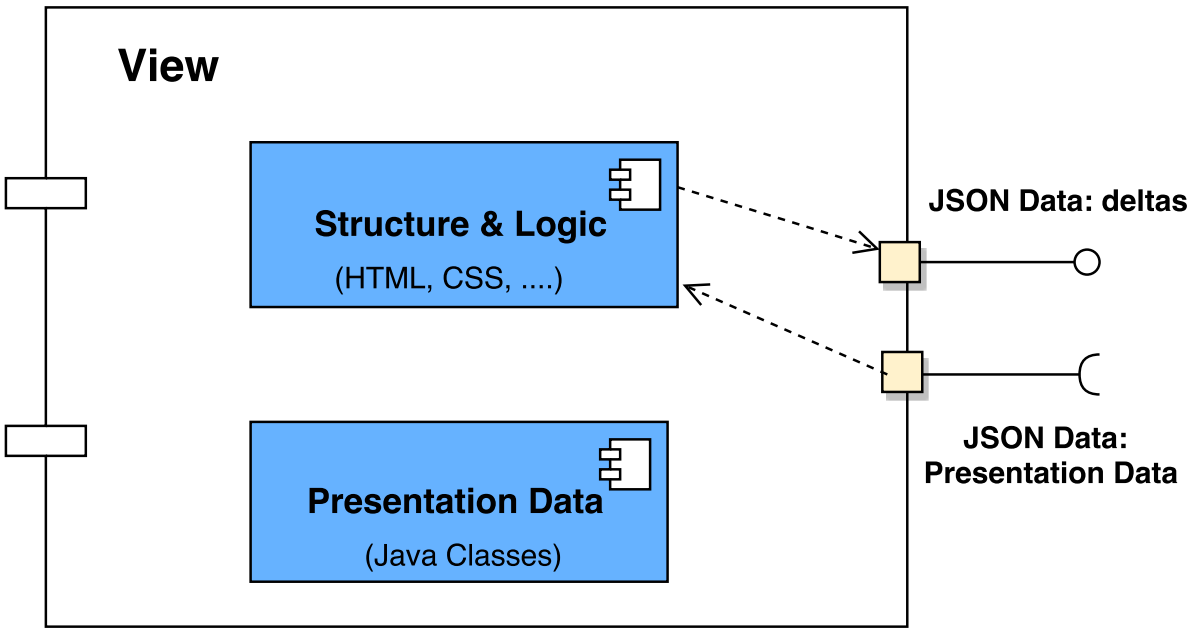


Figure 22: Component Diagram of View

Sub-Components Figure 22 describes the architecture of the view in the form of a component diagram. View basically contains two sub-components, `Structure & Logic` and `Presentation Data`.

`Structure & Logic` makes use of technologies like HTML, CSS, JavaScript, Jquery, and FabricJs, which combinedly create a user interface for the user to interact and the logic to handle the user actions.

`Presentation Data` 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. The view always receives the presentation data as a response from the controller in the form of JSON data and visualize them.

Figure 23 shows the structure of the presentation data in the form of a class diagram. It consists of classes like `Canvas`, `Layout`, `Workspace`, `PresentationData`, `UIGroup`, `Element`, `Rectangle`, and `ChangeData`. `Canvas` is the parent class for `Layout` and `Workspace`. It has attributes *height* and *width* to describe its size representing the views. The `Workspace` represents the symbolic view in the user interface and contains zero or more objects (elements). The `Layout` represents the layout view in the user interface and contains zero or more blocks and groups. `UIGroup` has attribute *fillColor* to uniquely identified as a new group on UI and contains zero or more blocks. `Rectangle` has attributes *id*, *xIndex*, *yIndex*, and *fillColor* to deal with the UI related actions. The `Element` has attributes *id*, *type*, *xPos*, *yPos*, and *fillColor*. `PresentationData` is the class which carries all the information

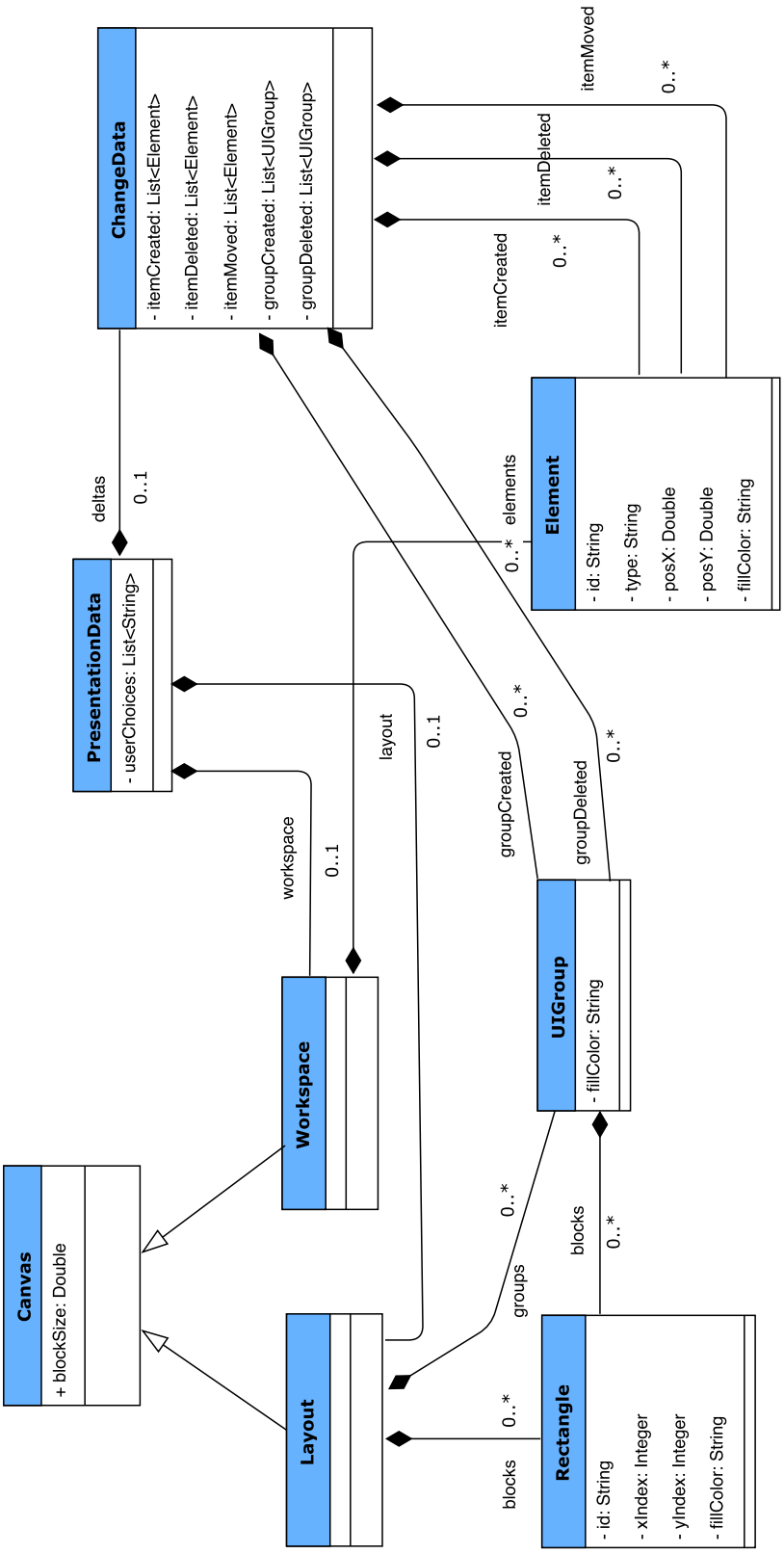


Figure 23: Class Diagram of Presentation Data

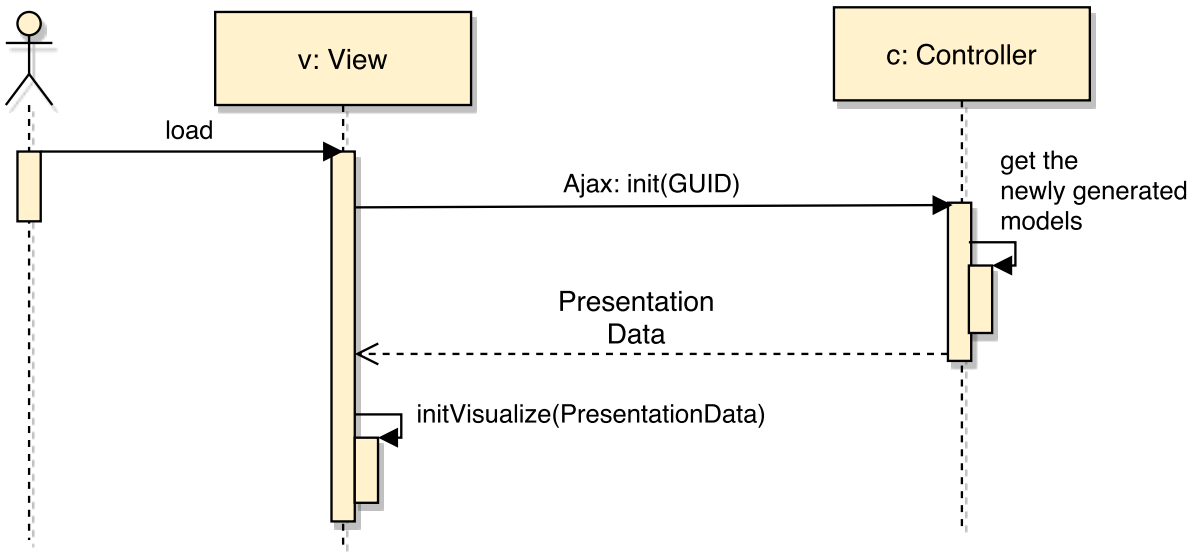


Figure 24: Sequence Diagram of View: initialization

related to UI. After receiving the response from the bx tool, the controller converts the bx tool specific models into `PresentationData` and send it to view for visualization. It has an attribute userChoices and contains a layout, a workspace, and a set of deltas wrapped inside a `ChangeData` 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 `ChangeData` object. The symbolic view allows three actions i.e., creation of a new item, deletion of an item, and movement of an item. Whereas, the layout view allows two actions i.e., creation of a new group and deletion of a group. Hence, `ChangeData` class tracks all of these five actions separately. `ChangeData` class shown in Figure 23 contains itemCreated, itemDeleted, and ItemMoved as a list of `Elements` to capture creation, deletion, and movement of an item respectively in the symbolic view. It also contains groupCreated and groupDeleted as a list of `UIGroups` to capture creation and deletion of a group respectively in the layout view. Also, these `deltas` are captured atomically so that they can be executed independently by the bx tool.

Workflow The sequence diagrams shown in Figure 24, Figure 25, and Figure 26 illustrate the workflow of the demon-bx tool from the point of view of the view component in different stages. As the view component only communicates with the controller, these sequence diagrams show the communication process between the controller and the view only.

1. **Load**: To load the application, the user enters the complete URL on a web browser. As soon as the web page is loaded, the view generates a unique user id (GUID) for the current user and sends an initialization command (`init(GUID)`) for the bx tool

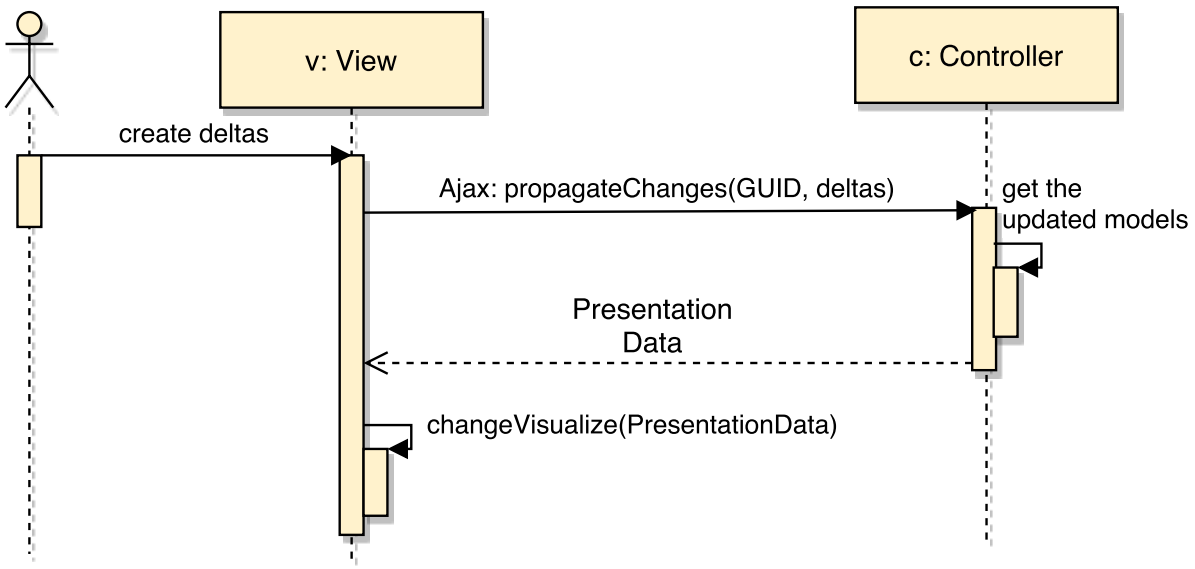


Figure 25: Sequence Diagram of View: delta propagation without continuation

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 `PresentationData` from the controller after the initialization of the bx tool. After getting the data, the view visualizes (`changeVisualize(PresentationData)`) them in two HTML canvas elements i.e., `Layout` and `Kitchen`. This entire process is shown in Figure 24.

2. **Delta Propagation (without continuation)**: A user is allowed to create one or more 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. Then the view sends (`propagateChanges(GUID, deltas)`) 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. If continuation (satisfiability of more than one transformation rule) does not exist, the bx tool completes the processing and the controller gets the updated models. In return, the view gets the `PresentationData` from the controller. After getting the data, the view visualizes (`changeVisualize(PresentationData)`) them in two HTML canvas elements i.e., `Layout` and `Kitchen`. This entire process is shown in Figure 25.

3. **Delta Propagation (with continuation)**: During processing of the data, If continuation (satisfiability of more than one transformation rule) exists, the controller gets the matching rules from the bx tool. In return, the view gets the `Presentation-Data` with user choices from the controller. After getting the data, the view prompts (`userChoiceVisualize(PresentationData)`) the choices to the user and gets a decision in return. View sends (`propagateUserChoices(GUID, user-`

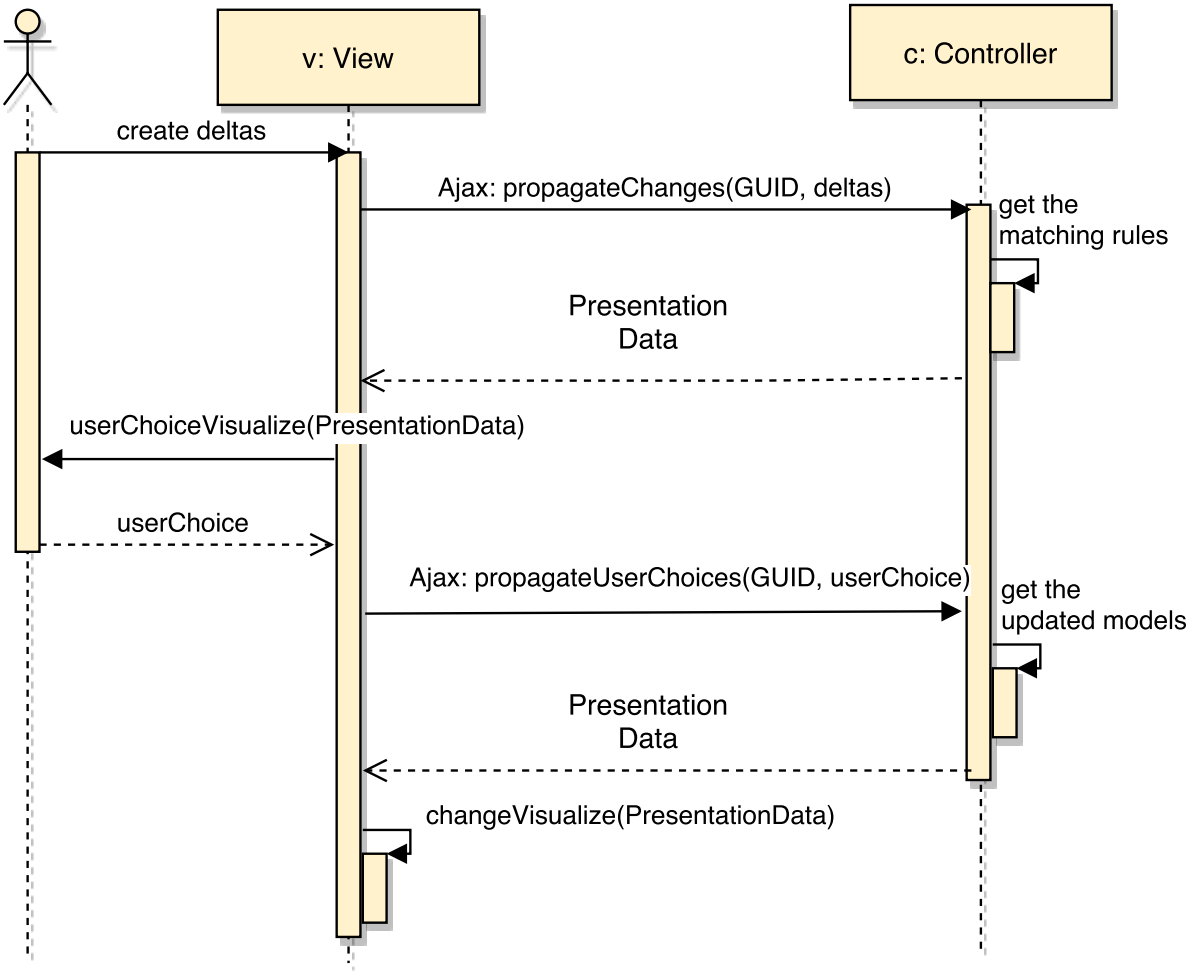


Figure 26: Sequence Diagram of View: delta propagation with continuation

`Choice`)) the user's choice along with the GUID generated earlier for the user in the form of `JSON` data to the controller wrapped inside an `Ajax` call. Then, the bx tool completes the processing and controller gets the updated models. In return, the view gets the `PresentationData` from the controller. After getting the data, the view visualizes (`changeVisualize(PresentationData)`) them in two HTML canvas elements i.e., `Layout` and `Kitchen`. This entire process is shown in Figure 26.

External Design For the visualization of the chosen example *Planning a Kitchen* as explained in Section 5.1, the first task was to design the layout view and the symbolic view along with its functionalities. However, the user interface is independent of any specific example and can be extended to implement a family of (similar) examples.

Both the views represent a workspace 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* [30] 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.

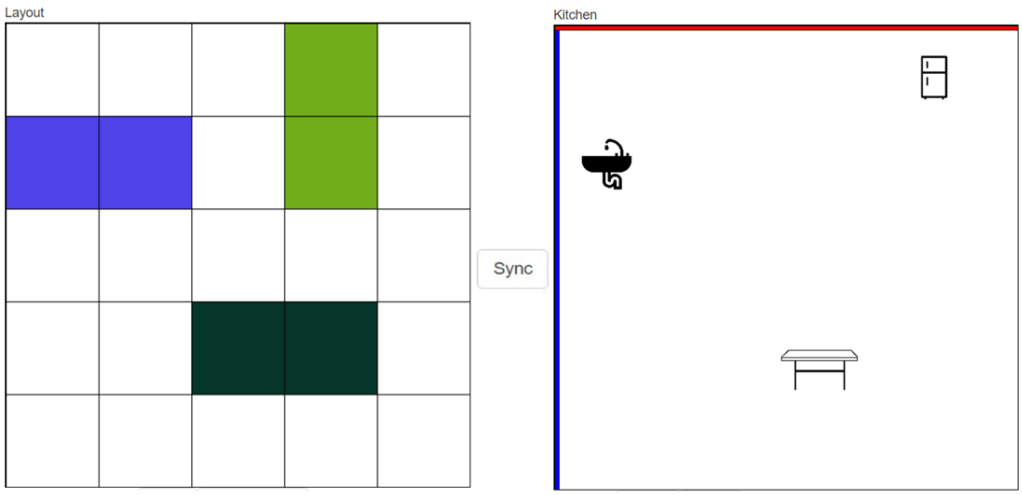


Figure 27: Layout and Kitchen

For the symbolic view, I kept the *Canvas* clean to represent an empty space where addition and manipulation of different objects can be done. As the user interface is meant to be example

independent, the four sides of the *Canvas* are "special" and can be interpreted in some way for a certain example to make things more interesting. For example, for the implementation of the example *Planning a Kitchen*, the four sides represent four walls of a kitchen space inside which different kitchen objects such as sink, table, fridge etc. can be added. Also, to make the kitchen space more realistic, I have even added water outlets on the western wall and electrical fittings on the northern 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 workspace space as shown in the symbolic view but, divided into blocks which restrict certain flexibility and functionality. Figure 27 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 [34]. Hence, I have analyzed a few canvas libraries available in market e.g., Fabric.js [31], Processing.js [32], Pixi.js [33] 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 [31]:

- 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 symbolic view, a user can perform addition, removal, and movement of any objects. 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. 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 symbolic view and send it to the controller for further processing. For example, for the implementation of the example *Planning a Kitchen*, a user can create, delete or move different kitchen objects such as sink, table, fridge etc. Also, to make the example more realistic, I have used similar images of the kitchen objects as shown in the right side canvas (kitchen) in Figure 27.

As the layout view is divided into blocks and offers fewer functionalities than the symbolic view, a user can perform only addition and removal of the objects. Hence, each 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 every time a new object is added. 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 view and sent to the controller for further processing. For example, for the implementation of the example *Planning a Kitchen*, 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 27.

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.

5.4.4 Controller

The controller is mainly responsible for event/action handling and manages the relationship between a view and a model [28]. 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.

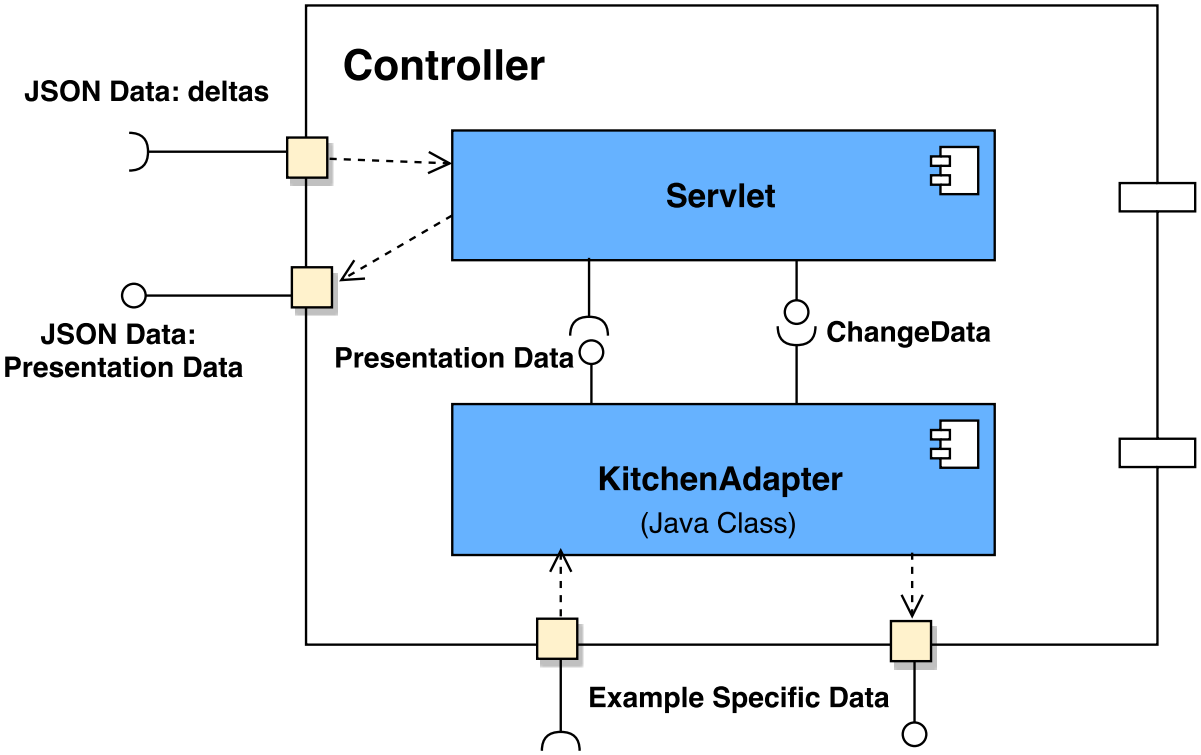


Figure 28: Component Diagram of Controller

Sub-Components Figure 28 describes the architecture of the controller in the form of a component diagram. In my case, controller consists of two sub-components, `Servlet` and an example specific adapter i.e., `KitchenAdapter`.

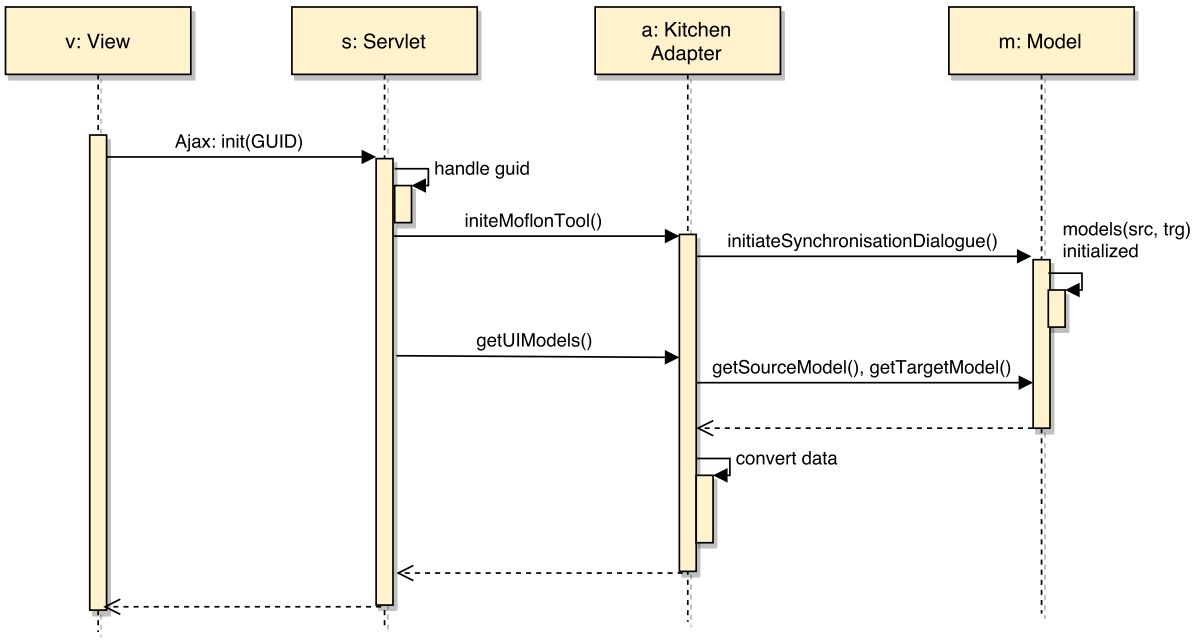


Figure 29: Sequence Diagram of Controller: initialization

Servlet is a technology which provides a component-based, platform-independent method for building Web-based applications [35]. 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 manages the lifecycle of servlets. In my case, I am using *Apache Tomcat* as my web container. Please refer Appendix for more information about servlet lifecycle.

In my application, the servlet is strongly coupled with an example specific adapter (kitchenadapter) i.e., a Java class. The adapter is responsible for the conversion between UI data and example specific data.

Workflow Figure 29, Figure 30, and Figure 31 illustrate the workflow of the demon-bx tool from the point of view of the controller component in different stages. As the controller component communicates with both the view and model, these sequence diagrams show the communication process between the view, model and the sub-components of the controller, `Servlet` and the `KitchenAdapter`.

1. **Load**: During first time load of the application, the servlet receives an initialization command (`init(GUID)`) from the view along with the newly generated GUID for the user

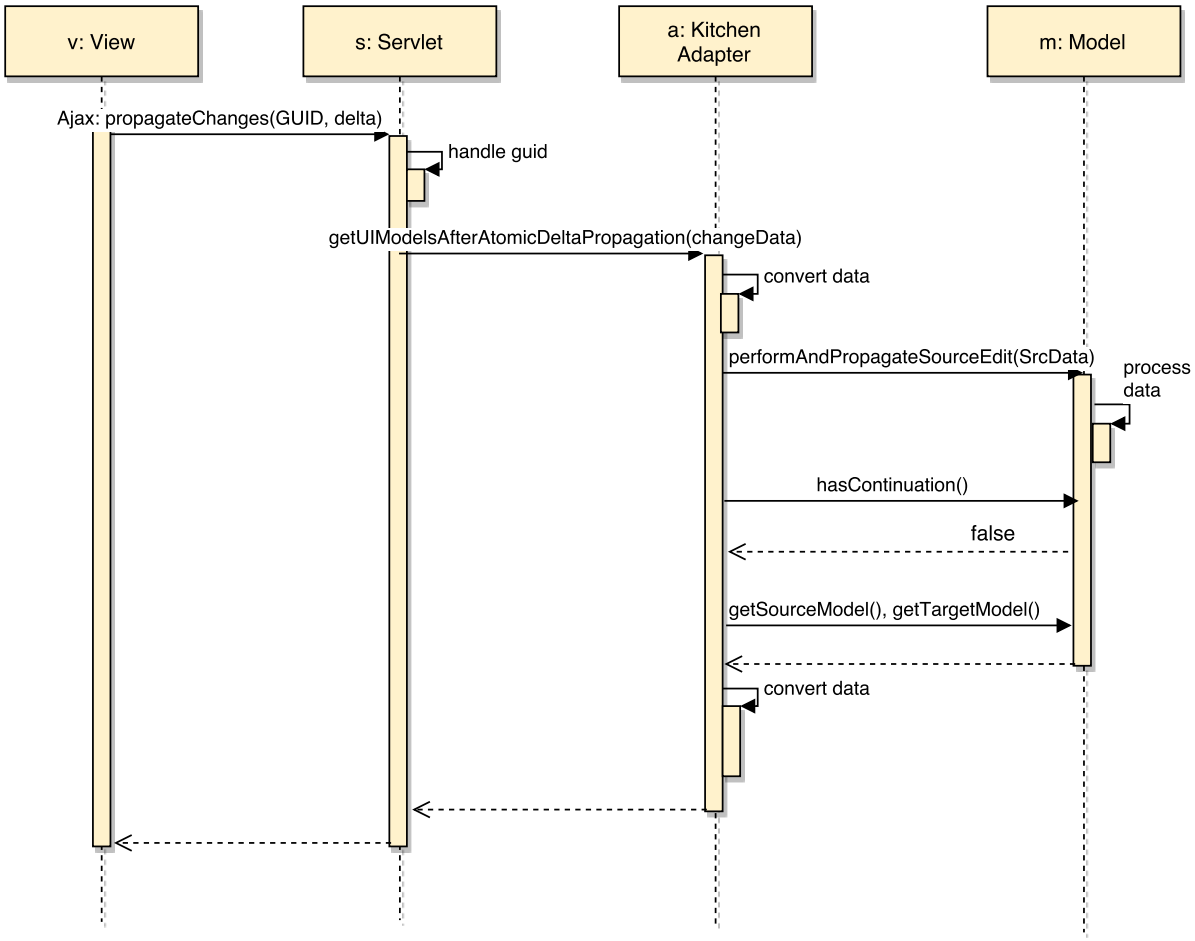


Figure 30: Sequence Diagram of Controller: delta propagation without continuation

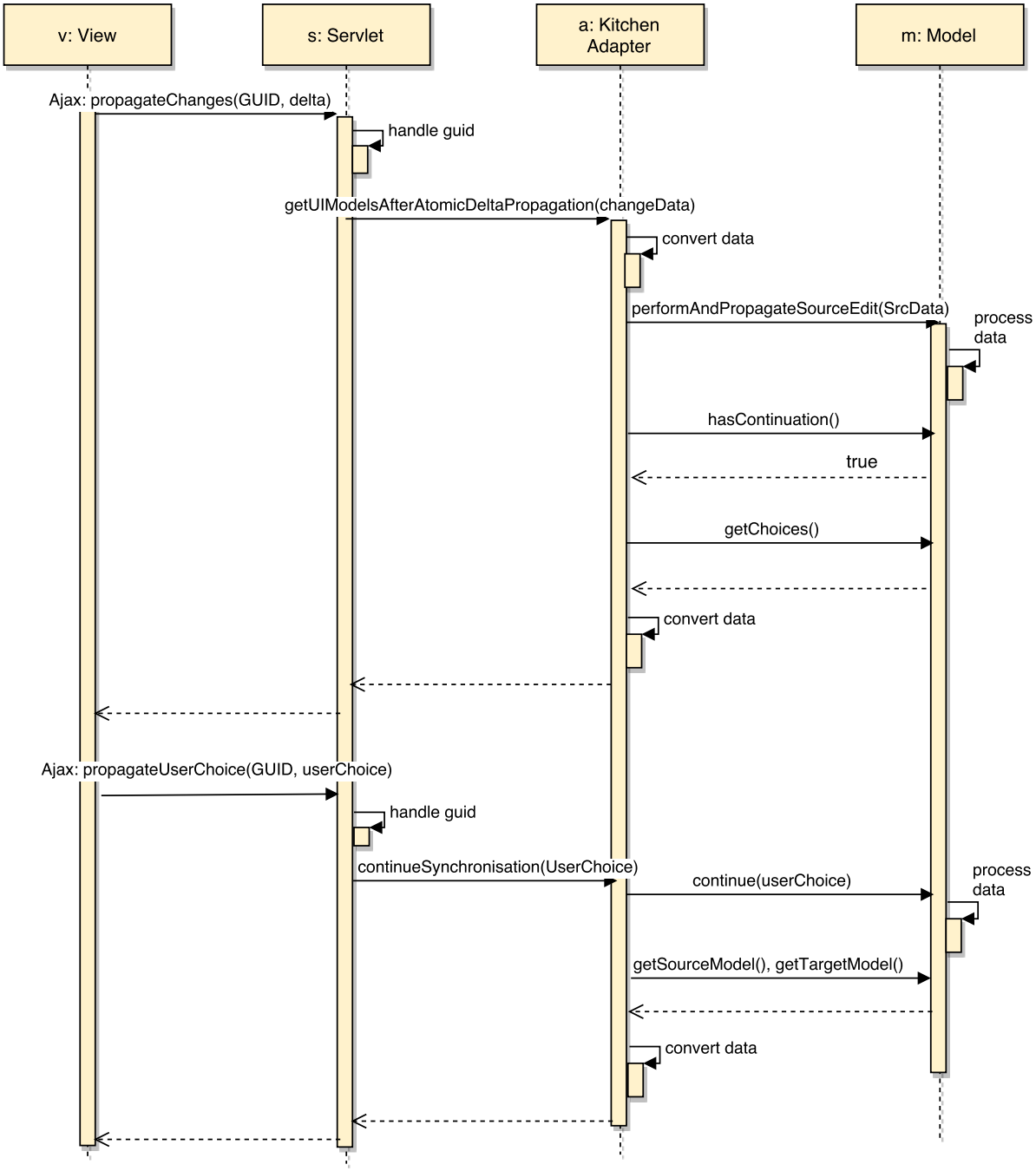


Figure 31: Sequence Diagram of Controller: delta propagation with continuation

in the form of `JSON` data wrapped inside an `Ajax` call. After receiving the request, the servlet creates a new instance of the adapter and stores the GUID. Then, the servlet forwards the initialization request (`initeMoflonTool()`) to the adapter. The adapter instantiates the bx tool i.e., eMoflon tool (`initiateSynchronisationDialogue()`) and the tool gets initialized by creating the instances of the `Source` and `Target` models. Afterward, servlet requests (`getUImodels()`) for the newly generated models through the adapter. The adapter requests (`getSourceModel(), getTargetModel()`) for the example specific models from the model, receives it, converts it to the UI specific data, and send it back to the servlet. After receiving the response, the servlet forwards it to the view for visualization. This entire process is shown in Figure 29.

2. **Delta Propagation (without continuation)**: After the synchronization button is pressed, view sends (`propagateChanges(GUID, delta)`) the delta created along with the GUID generated earlier for the user in the form of `JSON` data to the servlet wrapped inside an `Ajax` call. After receiving the request from the view, the servlet handles the GUID (checks its existence) and forwards (`getUIModelsAfterAtomicDelta Propagation(changeData)`) the delta wrapped inside a `ChangeData` class to the adapter. After converting the delta into example specific data, adapter calls the appropriate method (`performAndPropagateSourceEdit(SrcData)`) and sends the delta to the bx tool i.e., eMoflon. Here I have described only the forward synchronization. However, the process for the backward synchronization goes analogously. The bx tool processes the delta taking into account all the associated transformation rules. During processing of the data, adapter checks if satisfiability of more than one transformation rule exists (`hasContinuation()`) for the processed data i.e., continuation. If continuation does not exist, processing is completed by the bx tool. Then the adapter requests (`getSourceModel(), getTargetModel()`) for the updated example specific models from the model, receives it, converts it to the UI specific data, and send it back to the servlet. After receiving the response, the servlet forwards it to the view for visualization. This entire process is shown in Figure 30.

3. **Delta Propagation (with continuation)**: During processing of the data, If continuation (satisfiability of more than one transformation rule) exists, adapter requests for the matching rules (`getChoices()`) from the bx tool i.e., eMoflon, converts them into the UI specific data, and send it back to the servlet. The servlet forwards it to the view for visualization. After getting the user's choice, view sends (`propagateUserChoice(GUID, userChoice)`) it along with the GUID generated earlier for the user in the form of `JSON` data to the servlet wrapped inside an `Ajax` call. After receiving the request from the view, the servlet handles the GUID (checks its existence) and forwards (`continueSynchronisation(userChoice)`) the user's choice to the adapter. The adapter forwards it to the bx tool for further processing. Then the adapter requests (`getSourceModel(), getTargetModel()`) for the updated example specific models from the model, receives it, converts it to the UI specific data, and send it back to the servlet. After receiving the response, the servlet forwards it to the view for

visualization. This entire process is shown in Figure 31.

For better understanding and to avoid complexity, here I have described the delta propagation process for only a single delta. However, the user can generate multiple deltas as well. In that case, after receiving all the deltas from the view, the controller converts them to example specific data. Then, the controller handles them atomically (one by one). Hence, the above described delta propagation process (with or without continuation) is executed for each delta until the data processing is complete by the bx tool.

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 conceptualizing `deltas` and capturing them.

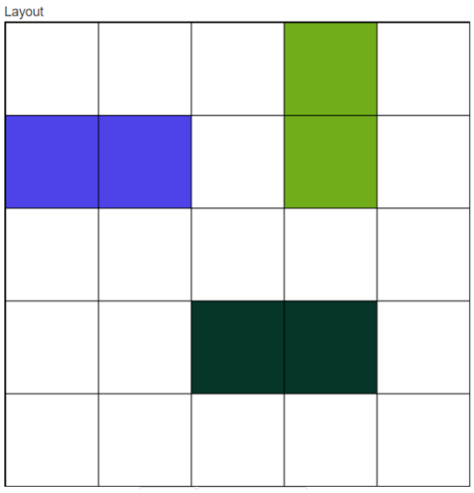


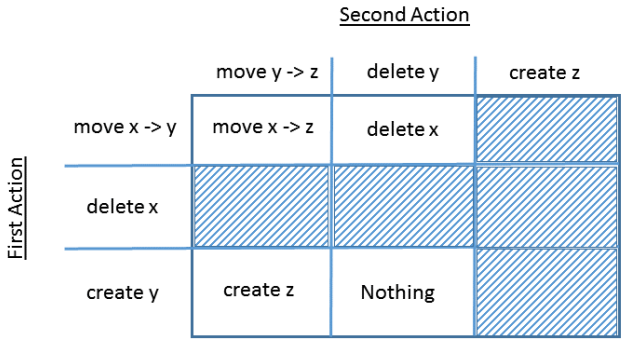
Figure 32: Layout View

As mentioned earlier, the symbolic view has more functionalities than the layout view in example *Planning a Kitchen*. Firstly, I fixed the deltas that can be performed on either view. For

example, three types of changes are allowed in the 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 the 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 has 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 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.

Example: Figure 32 shows a sample of the layout with a different set of groups filled with different colors. The layout contains three groups separated by three unique colors i.e., two horizontally adjacent blocks on the western wall with blue color, two vertically adjacent blocks on the northern wall with dark green color, and two horizontal adjacent blocks in the middle of the layout with dark gray color.

Compressing related deltas Along with creating deltas, another challenge was to compressing different deltas (in some cases) into one in the process of capturing.



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

Figure 33: Rules for compressing deltas

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 the item does not exist anymore. Hence, rather capturing every change that occurs on the views, I have implemented a few rules while capturing them. The handling of a single delta is clear. But, the interesting part is rules for how to compress two deltas into one in some cases. All possible combinations are described in Figure 33.

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 at a new location. Because creating a new item and then moving it to a new location is equivalent to creating a new item at the new location.

Handling failed deltas Another challenge was to handle the rejected/failed deltas during and after the processing by the eMoflon tool. The problem was when a series of deltas are given to the eMoflon tool for 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, it will not process the remaining deltas after encountering the first invalid delta even if all or some of the remaining deltas are valid. Also, 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, deltas are given to the eMoflon tool for processing one after another from the pool of the collected deltas from `Controller`. If any of the deltas is rejected during processing, eMoflon sends it to the `Controller`, restores the state of the models to the previous consistent state, and resume processing the remaining deltas. After the completion of processing, `Controller` gathers all the *failed delta*s as `deltas` bundled inside the `PresentationData` class as shown in Figure 23 and sends it to the view for visualization.

Example: A user creates two deltas in the kitchen i.e., moving an already existing fridge away from the wall with electrical fittings (invalid delta) and creating a new sink close to the wall with water outlet (valid delta). After pressing the synchronization button, both the deltas are sent to the eMoflon tool but fed one by one. As the movement of the fridge is an invalid delta, the tool rejects the delta, sends it to the controller, restores the state of the models to the previous consistent state, and resume processing the remaining deltas. Then, the processing of the delta for the creation of a new sink will be accepted by the tool and models will be updated as it is a valid delta.

Processing deltas in correct order One of the challenges during the implementation phase was in which order the deltas are supposed to be fed to the eMoflon tool for processing. It is important to consider the correct order of processing the deltas as a wrong order create conflict while processing. For example, a user deletes an item x from a location and then move another

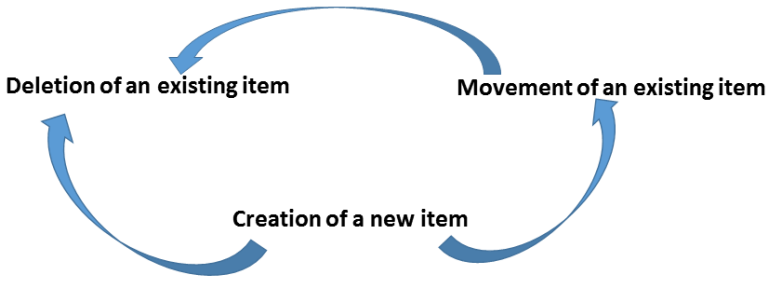


Figure 34: Dependency of states with each other

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 34. 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 two deltas in the kitchen i.e., deleting an already existing sink from the location "x" (valid delta) and moving an already existing table to the location "x" (valid delta). After pressing the synchronization button, both the deltas are sent to the eMoflon tool but fed one by one in the order deletion of the sink then the movement of the table. Processing of the deletion of the sink will be accepted by the tool and models will be updated as it is a valid delta. Then, the processing of movement of the table will also be accepted by the tool and models will be updated as it is a valid delta. But, the important part is if the processing of the delta for movement of the table to the location "x" was processed first, then eMoflon would have rejected it. Because the tool will not move the table to the location "x" where a sink is still present.

Handling multiple users accessing the application Another challenge that I faced during the implementation of the servlet was handling multiple user's requests accessing the application at the same time. The problem was to maintain different states of the bx tool specific models for different users.

I solved this problem by generating a unique user id (GUID) for a user on the first time load of the application and associating all the further requests from that user with the specific GUID generated earlier till the browser is closed. On the first time load of the application, view generates a random sixteen digit unique user id (GUID) for the current user and sends it with

Key (16 digit GUID)	Value (Adapter's instance)
GUID1	a1
GUID2	a2
...	...

Figure 35: Key-Value map for storing GUID

the initialization request in the form of `JSON` data to the servlet wrapped inside an `Ajax` call. As the GUID contains sixteen digits generated randomly from system's time stamp, it is almost impossible that two users will have the same GUID. After receiving the request, the servlet creates a new instance of the adapter class and stores the GUID pointing to the newly created instance of the adapter inside a `key-value map` as shown in Figure 35. Then, the servlet forwards the initialization request to the newly created instance of the adapter class. As explained earlier in Section 5.4.4, the adapter initializes and communicates with the model and hence each instance of the adapter class is associated with a unique instance of the bx tool i.e., eMoflon. Hence, storing a GUID pointing to an instance of the adapter class actually maintains a link between a user and a specific instance of the bx tool i.e., instances of the bx tool specific models. Afterward, view always sends any request to the servlet along with the GUID generated earlier for the user. After receiving the request from the view, the servlet extracts the GUID and checks its existence inside the key-value map. If the GUID exists, the servlet forwards the request to the corresponding adapter's instance. Finally, when the user closes the browser, the servlet deletes the entry of the GUID from the key-value map.

Example: The sequence diagram shown in Figure 36 describes the communication process between the view and the controller involving a GUID for a specific user. On the first time load of the application, view generates unique user id i.e., GUID1 for the current user and sends it with the initialization request to the servlet. After receiving the request, the servlet creates a new instance of the adapter class i.e., a1 and stores GUID1 pointing to a1 inside a `key-value map`. Then, the servlet forwards the initialization request to a1. While sending deltas, view sends GUID1 along with the request to the servlet. After receiving the request from the view, the servlet checks the existence of GUID1 inside the key-value map. If the GUID1 exists, the servlet forwards the request to a1. Finally, when the user closes the browser, the servlet deletes the entry of the GUID1 from the key-value map.

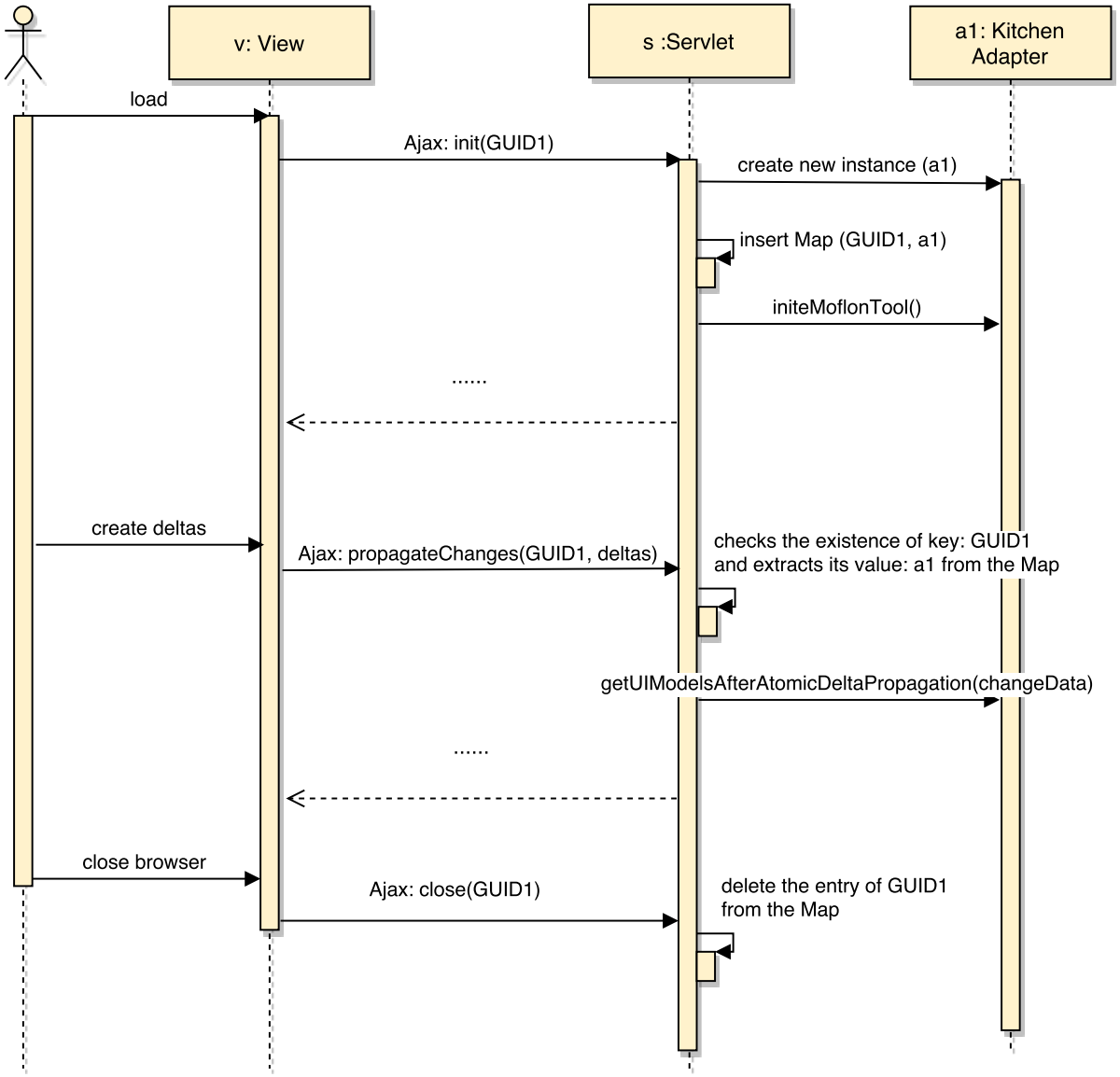


Figure 36: Sequence Diagram showing the flow of the Unique User id (GUID)

6 Evaluation

In this chapter, I am going to evaluate the outcome of my thesis, i.e., research and implementation work. Figure 37 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 6.4 describes the preparation and the test execution phase in detail. Section 6.5 explains the associated threats with the evaluation process and their mitigation criteria. Finally, Section 6.6 discusses the results generated from the gathered data.

The aim of this chapter is as follows:

"Analyze 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 teaching the basic concepts of bidirectional transformation (bx) and making them understandable and accessible."

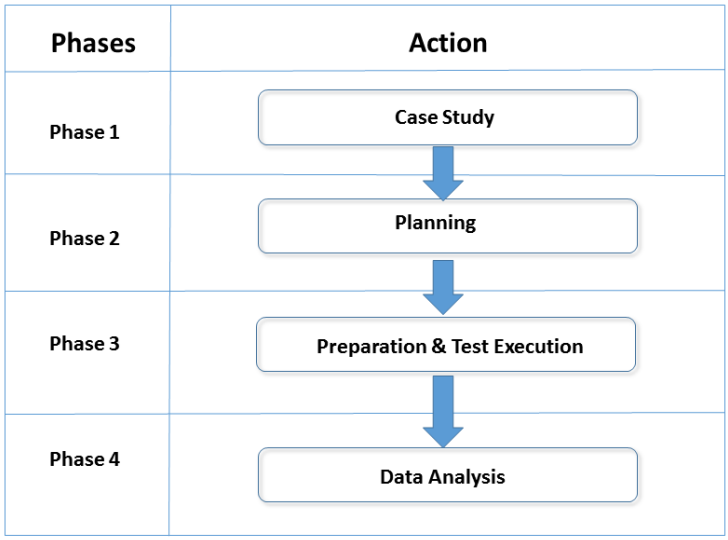


Figure 37: Evaluation Phases

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 [21]. Hence, for any evaluation process, the most important thing is 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 based on a simple example. I 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 the corresponding learning goals?

RQ 4.2: How much does this support depend on supplied 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 the computer science branch. Hence, this experiment is mainly 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 [37], 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 [38]. It is a popular research design. The design principle is relatively simple and 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 are collected from both occasions. Then, the analysis is done by comparing pretest and posttest results collected from both the groups.

Selection of the Pretest-Posttest control group design method for the experiment is driven by the following reasons [39]:

- 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. I am 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. The 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 in 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 on my experiment:

Operational Hypotheses

H_{OP1}: Students have a wrong intuition for and unreasonable expectations of synchronization scenarios (because they derive their intuition from the special case of bijections).

H_{OP2}: The usage of demon-bx has a positive effect on the achievement of corresponding bx-related learning goals.

H_{OP3}: Using demon-bx together with suitable scenarios is more effective than without scenarios.

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

Null Hypotheses

H_{NU1}: Students have a correct intuition for and reasonable expectations for synchronization scenarios.

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

H_{NU3}: There is no significant improvement in the learning outcome of the students using demon-bx with suitable scenarios compared to the students using demon-bx without scenarios.

6.3.3 Experimental Variables

The Hypotheses described above helped in deciding the variables related to the experiment. The 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: Planning a kitchen
- Group participants are chosen from: {master students, PhD students, bx researchers, . . . }
- Test questions: 5
- Learning goals: 5
- Max. time interval to answer questions: 55 min

	Name	**Possible Values**
Independent Variable	Group gets scenarios to work	nominal (yes or no)
Controlled Variables	Demonstrator platform	Demon-BX
	Concrete example	Planning 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	interval
	Level of certainty of posttest	interval

Table 5: Experimental Variables

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: interval
- Level of certainty of posttest: interval

6.3.4 Learning Goals & Questions

To investigate the research questions **RQ 3** and its sub-question **RQ 3.1**, I chose five bx concepts to evaluate the learning goals (referred to as **LG** from now on) for the experiment. These concepts are related to the 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. (Synchronization is not total)

LG 3: Synchronization is interactive. User interaction (or some other, possibly automated means) can be used to decide between multiple equally consistent results (to handle non-determinism: Synchronization is not always functional).

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 teach these learning goals, I carefully prepared five scenarios based on the example *Planning a Kitchen* and added to the demonstrator. Each scenario referred to a learning goal. These scenarios involve guidance steps for the user, e.g., various actions performed on the user interface in a sequential manner. At the end of the steps, the user learns the corresponding learning goal.

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 the 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*. Please refer Appendix for the evaluation test questions.

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., pretest and posttest 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 form [18]. Google form provides a very simple way to prepare a responsive survey form with customized questions which can be shared easily to a mass audience online. Then to investigate the research questions **RQ 4** and its sub-questions **RQ 4.1** and **RQ 4.2**, 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 an experiment will be conducted on a particular date and that their participation in the experiment is completely voluntary with no consequences whatsoever. 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 pretest and posttest, 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 pretest. After finishing the pretest, 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 posttest. Posttest questions were exactly the same as pretest 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 were collected from 40 students. I stopped taking responses on Google forms, i.e., pretest and posttest 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. Out of 39 students, 21 students were from the control group and remaining 18 were from treated group.

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

6.5 Threats to Validity and Mitigation

In my evaluation, I have focused on the validity analysis and threats given by Wohlin et al [41] and further explained by Feldt et al [40].

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. 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. It can be possible that my test questions might not actually be suitable for checking if my learning goals have been reached and thus measuring the output from the test questions does not make any sense. Also, the students being generally unused to and confused by the notation used, it might be possible that they don't even understand the questions and thus cannot improve anything. To avoid these threats, I have carefully prepared the questions corresponding each one of them to a learning goal and tried to ensure that the difficulty level of all the questions is more or less the same. To make the notations more understandable, I have provided extra information about the notation used in the test questions in the two-minute video tutorial shown to all participants before the pretest along with basic bx concepts.

Do the groups (control and treated) 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 negative 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.

Does the formula I used for the result calculation produce statistically significant data? Can I draw conclusions based on the data?

It can be possible that the formula that I have used to calculate the result of the pretest or the posttest taking correctness and certainty into account or the improvement of the posttest compared to the pretest might not produce the exact significant data for further statistical analysis and thus can affect the final outcome. For example, now I am subtracting the pretest result from the posttest result and then dividing it by two for normalization to calculate the improvement. But, there might be other ways to define what improvement is relative to the pretest result. For example, a student might have already improved the maximum he/she could have from pretest to posttest, but dividing it by two for normalization could lower down the actual improvement percentage.

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.

Does the treatment/change I introduced have a statistically significant effect on the outcome I measure?

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 relationship between them.

6.6 Data Analysis, Results, and Discussion

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

In my experiment, I have asked five questions referring to five learning goals. Each question is prepared to test whether a student understood the related bx concept, i.e., learning goal or not. As one of the goals of my thesis is to teach bx concepts through the demonstrator and all of my hypotheses are based on the measurement of correctness and improvement of the answers given by the students, I decided to track these variables question wise rather than evaluating the data collectively for all questions.

The formula for result calculation and analysis of each hypothesis is described in the following subsections.

6.6.1 Formula Used

For statistical analysis of the data collected in two separate groups, i.e., control and treated, the formulas I used for the calculation of results for all hypotheses are described below:

For ith Question Q_i and group X (where X $\in$ {control(c), treated(t), combined(com)}),

$CO_{pre(X_i)} \in$ {-1, 1}, correctness of the answer to the ith pretest question for X group is either 1 ("right") or -1 ("wrong").

$CE_{pre(X_i)} \in [0, 1]$, certainty of the answer to the ith pretest question for X group is between 0 ("I just guessed") to 1 ("I am certain").

$CO_{post(X_i)} \in \{-1, 1\}$, correctness of the answer to the ith posttest question for X group is either 1 ("right") or -1 ("wrong").

$CE_{post(X_i)} \in [0, 1]$, certainty of the answer to the ith posttest question for X group is between 0 ("I just guessed") to 1 ("I am certain").

$RES_{pre(X_i)} := CO_{pre(X_i)} \times CE_{pre(X_i)} \in [-1, 1]$, result for the ith pretest question for X group is between -1 ("absolutely wrong") to 1 ("absolutely right").

$RES_{post(X_i)} := CO_{post(X_i)} \times CE_{post(X_i)} \in [-1, 1]$, result for the ith posttest question for X group is between -1 ("absolutely wrong") to 1 ("absolutely right").

$IMP_{(X_i)} := (RES_{post(X_i)} - RES_{pre(X_i)})/2 \in [-1, 1]$, improvement for the ith question for X group from pretest to posttest is between -1 ("absolute negative learning") to 1 ("absolute positive learning").

It is important to understand the following points from the above formulas:

- while answering any question, being "absolutely wrong" ($RES_{pre/post(X_i)} = -1$) is worse than just being "wrong" ($CO_{pre/post(X_i)} = -1$) as being absolutely certain about a wrong answer is more harmful than just being wrong.
- similarly, while answering any question, being "absolutely right" ($RES_{pre/post(X_i)} = 1$) is better than just being "right" ($CO_{pre/post(X_i)} = 1$) as being absolutely certain about a right answer is more helpful than just being right.

6.6.2 Analyzing Hypothesis 1

Hypothesis 1 i.e., $\mathbf{H}_{OP1}$ and $\mathbf{H}_{NU1}$ is designed to investigate research question **RQ 3.1**. This hypothesis deals with the fact whether students have a wrong intuition for synchronization scenarios. Data for **RQ 3.1** can be collected from all pretest results, i.e., combining the pretest results from both control and treated group ($RES_{pre(com_i)}$). Because at the end of the pretest, both control and treated group have gone through the same two-minute video tutorial to establish basic concepts, answered the same set of questions, and hence are equally weighted in terms of bx knowledge acquired.

As five questions were asked, the result can be calculated for each question from the pretest data.

Data Analysis After calculating the results, i.e., $RES_{pre(com_i)}$ for each question, I used density plots to show their distribution. Figures 38, 39, 40, 41, and 42 show the density plots of the results for each question. In the plots, the x-axis represents the result calculated by combining

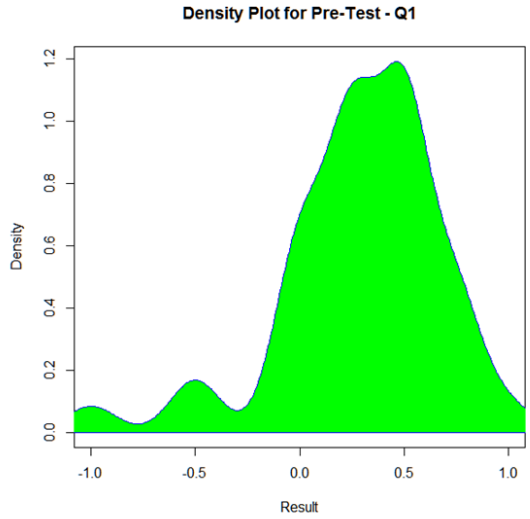


Figure 38: $\text{RES}_{pre(com_1)}$

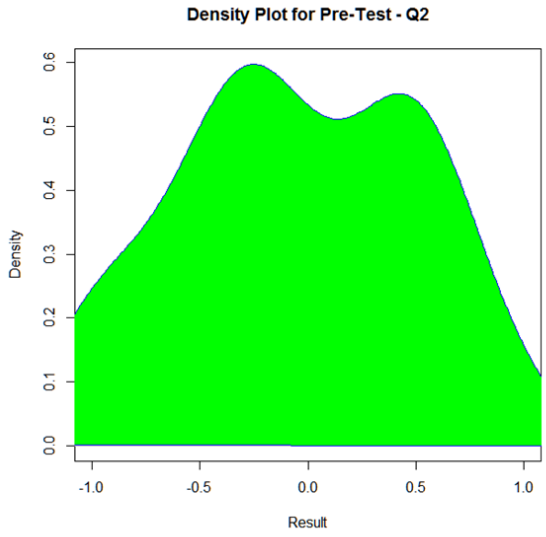


Figure 39: $\text{RES}_{pre(com_2)}$

both correctness and certainty as described in Section 6.6.1 ranging between -1 (absolutely wrong) to 1 (absolutely right). The y-axis represents the density of 39 students lying over the interval. From Figure 38, it is possible to see that the distribution of the results is mainly on the positive side i.e., between 0 to 1 for Q1. But for all other questions, the distribution of the results is rather scattered over the interval and does not give any conclusive result to accept or reject the null hypothesis just by looking at these plots.

To gain a better understanding of the data, I have further analyzed it with a t-test. As this hypothesis is related to only one set of pretest data, I have used the one-sample t-test.

As mentioned earlier, certainty value is also involved in the result calculation. Its value entered by the students ranges from 1 ("I just guessed") to 5 ("I am certain") which corresponds to the interval 0 to 1 considered for calculation. Hence, the result of a correct answer with the lowest non-zero certainty value to any question is 0.25. So this hypothesis can be expressed as:

Null Hypothesis, H_{NU1}: Mean $(RES_{pre(com_i)}) > 0.25$

Hence, I have included the options (*alternative="less", mu=0.25*) during the calculation which signifies that according to the operational hypothesis the mean value has to be less than 0.25. The results from the t-test are shown in Table 6. The 95% confidence interval has been included for this t-test.

Discussion From the t-test results as shown in Table 6, it can be clearly seen that the mean value of $RES_{pre(com_i)}$ for question Q1 is just slightly higher than 0.25 and for questions Q2, Q3,

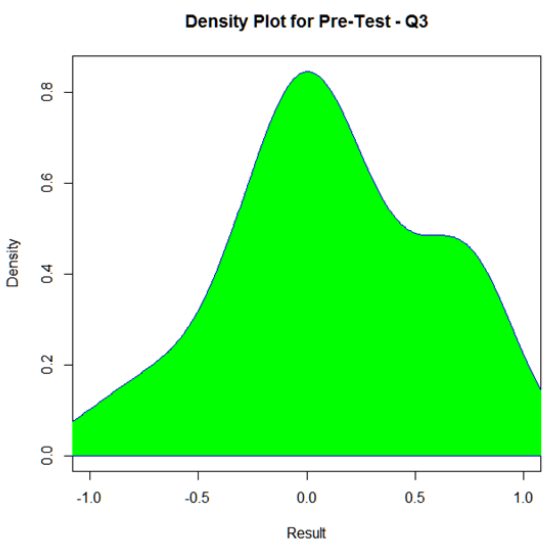


Figure 40: $\text{RES}_{pre(com_3)}$

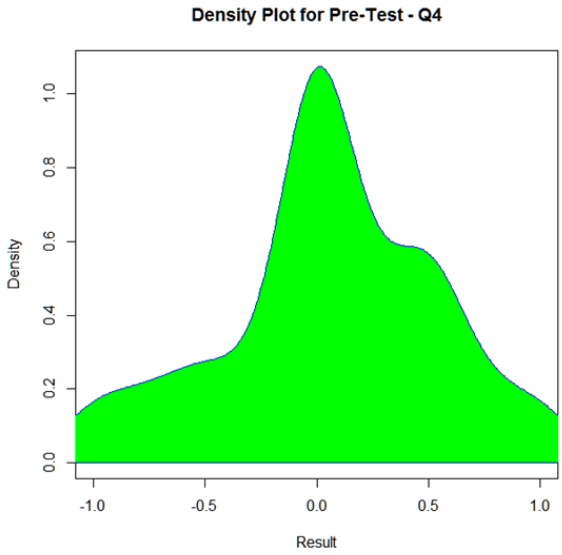


Figure 41: $\text{RES}_{pre(com_4)}$

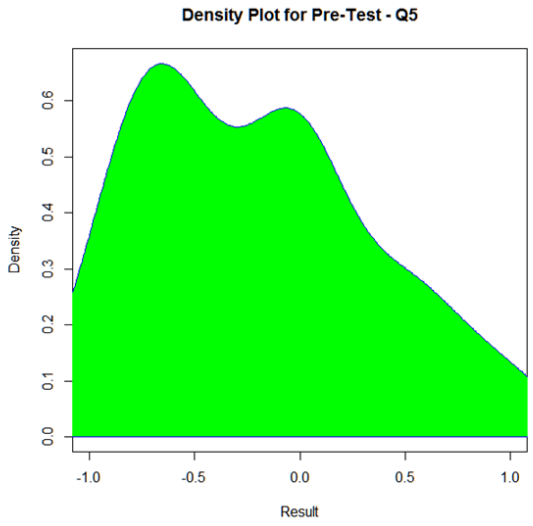


Figure 42: $\text{RES}_{pre(com_5)}$

Q4, and Q5 are much less than 0.25. Also, the p-values in these cases are very low so the results are highly significant. Most students were just guessing in the pretest. A few questions were answered correctly, but considering the very low certainty values, this does not mean much or imply any deep understanding. However, the increase in the mean value for the question Q1 (refer evaluation test questions in Appendix) is most likely due to the fact that this is the easiest question among all and relatively more participants were able to answer it correctly.

These results reject the null hypothesis H_{NU1}. This indicates that the students derive their (in general wrong) intuition for and expectations of synchronization scenarios from the special case of bijections. Hence, there is indeed a need to teach correct expectations for synchronization scenarios and a demonstrator can be potentially very useful in spreading bx concepts.

	Students	**Factor**	**t-value**	**p-value**	**mean**
Q1	39	pre test result	0.7293	0.7649	0.2948
Q2	39	pre test result	-3.3139	0.0010	-0.0448
Q3	39	pre test result	-1.8514	0.0359	0.1089
Q4	39	pre test result	-2.3902	0.0109	0.0641
Q5	39	pre test result	-5.0889	0.000005	-0.1987

Table 6: t-test showing Result for Pre-Test (All Participants)

6.6.3 Analyzing Hypothesis 2

Hypothesis 2 i.e., $\mathbf{H}_{OP2}$ and $\mathbf{H}_{NU2}$ is designed to investigate research question **RQ 4.1**. This hypothesis deals with the fact whether demon-bx tool has a positive effect on the achievement of corresponding bx-related learning goals. Data for **RQ 4.1** can be collected from the improvement calculated for the control group ($IMP_{(c_i)}$) and the treated group ($IMP_{(t_i)}$) separately. Because both control and treated group have used demon-bx tool (with different environments) before posttest and improvement indicates how much a certain group has improved/learned from pretest to posttest.

As five questions were asked, the improvement can be calculated for each question from the pretest and the posttest data separately for control and treated group.

Data Analysis After calculating the improvements, i.e., $IMP_{(c_i)}$ and $IMP_{(t_i)}$ for each question, I used density plots to show their distribution. Figures 43, 44, 45, 46, 47, 48, 49, 50, 51, and 52 show the density plots of the improvements for the control group and the treated group separately for each question. In the plots, the x-axis represents the improvement as described in Section 6.6.1 ranging between -1 ("absolute negative learning") to 1 ("absolute positive learning"). The y-axis represents the density of 21 students in the case of the control group and 18 students in the case of the treated group lying over the interval. These density plots show

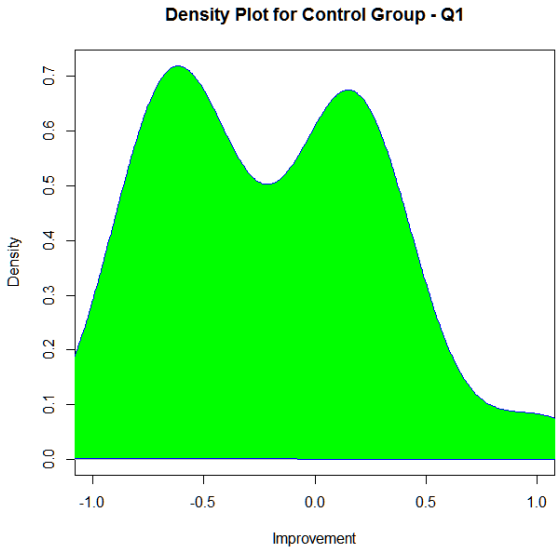


Figure 43: $\text{IMP}_{(c_1)}$

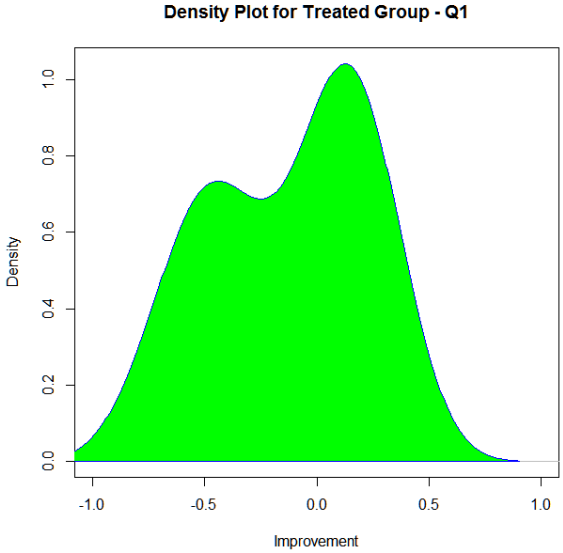


Figure 44: $\text{IMP}_{(t_1)}$

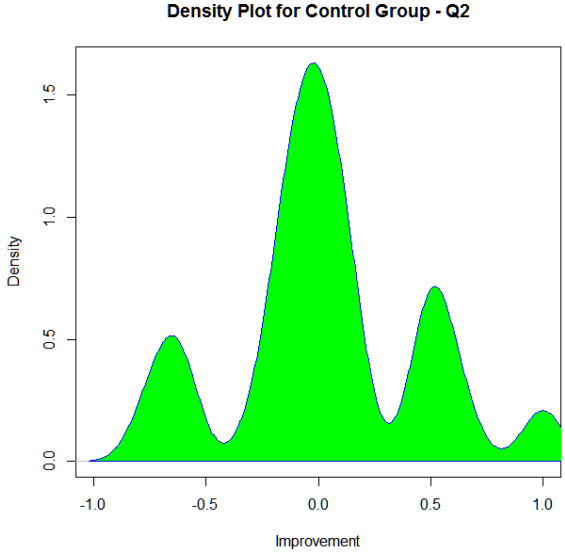


Figure 45: $\text{IMP}_{(c_2)}$

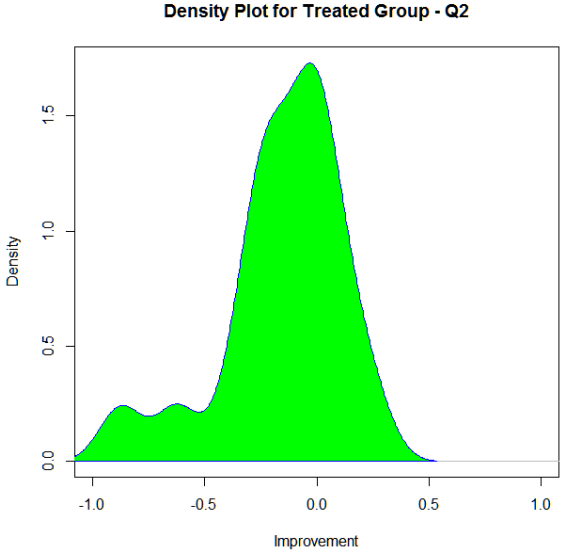


Figure 46: $\text{IMP}_{(t_2)}$

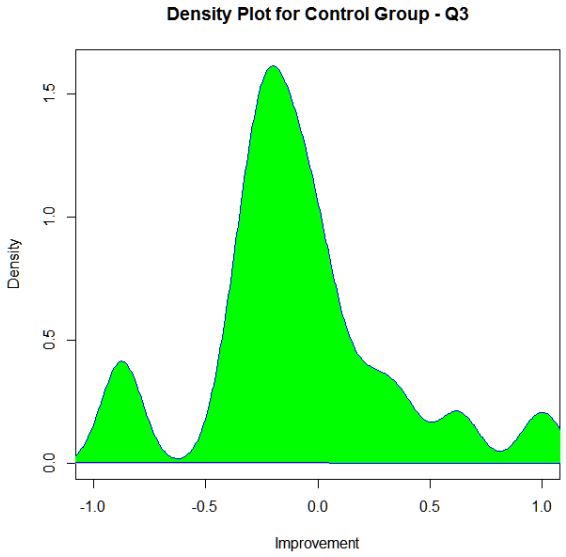


Figure 47: $\text{IMP}_{(c_3)}$

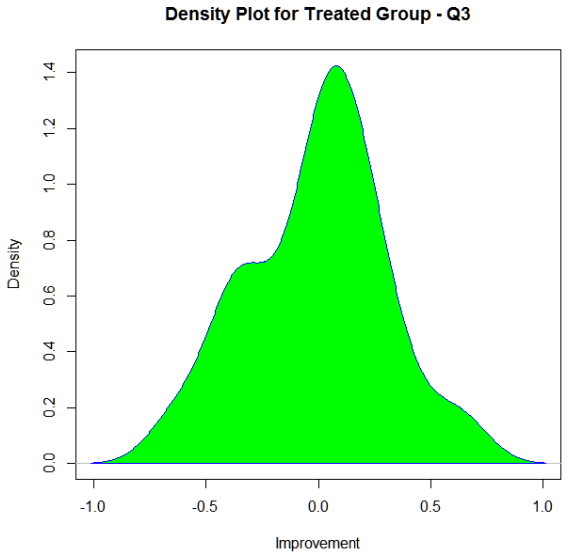


Figure 48: $\text{IMP}_{(t_3)}$

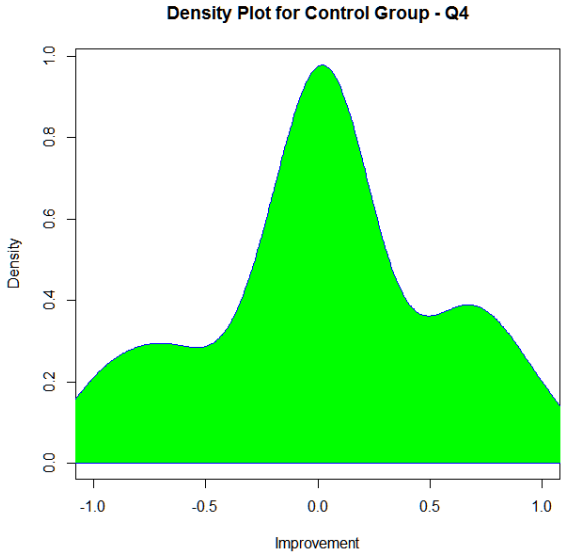


Figure 49: $\text{IMP}_{(c_4)}$

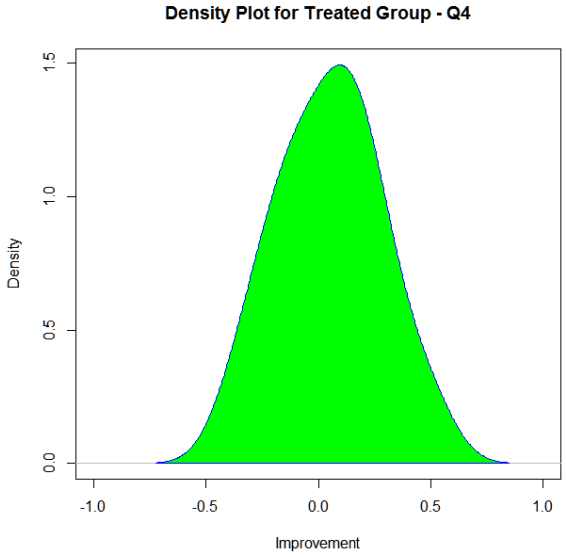


Figure 50: $\text{IMP}_{(t_4)}$

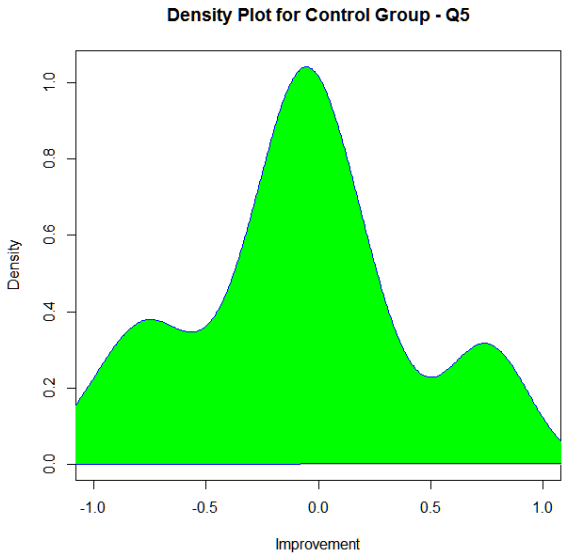


Figure 51: $\text{IMP}_{(c_5)}$

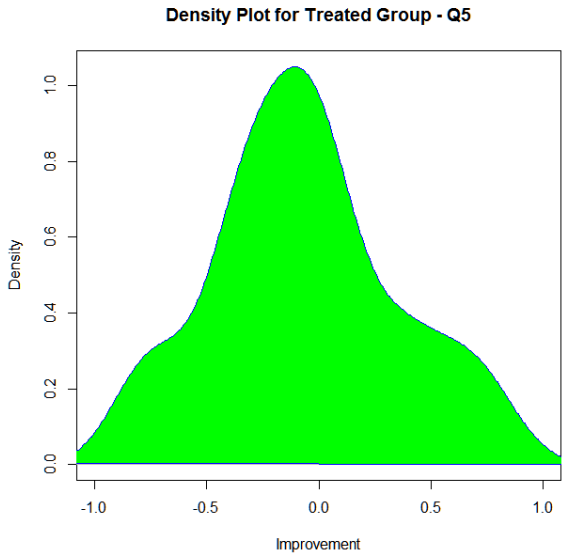


Figure 52: $\text{IMP}_{(t_5)}$

that the improvement values are scattered over the entire interval quite uniformly on both sides of the value 0. But, it is very difficult to say whether the values are more on the positive side or more on the negative side. So it's very hard to accept or reject the null hypothesis just by looking at these plots.

To gain an even better understanding of the data, I have further analyzed it with a t-test. As this hypothesis is related to one set of improvement data at a time, i.e., either the improvement of the control group or the improvement of the treated group, I have used the one-sample t-test.

The improvement is always calculated for posttest compared to the pretest. Here also improvement for control group is calculated subtracting control group pretest ($RES_{pre(c_i)}$) results from control group posttest ($RES_{post(c_i)}$) results and then dividing it by two for normalizing it between -1 to 1. Similarly, improvement for treated group is calculated subtracting treated group pretest ($RES_{pre(t_i)}$) results from treated group posttest ($RES_{post(t_i)}$) results and then dividing it by two for normalizing it between -1 to 1. Hence, the value less than 0 signifies "negative" improvement and the value greater than 0 signifies "positive" improvement. As two independent groups are involved, this hypothesis can be expressed as:

Null Hypothesis, H_{NU2}: Mean ($IMP_{(c_i)}$) < 0 and Mean ($IMP_{(t_i)}$) < 0

Hence, I have included the option (*alternative="greater"*) during the calculation which signifies that according to the operational hypothesis the mean values has to be greater than the default value 0. The results from the t-test are shown in Table 7 and 8 for control and treated group respectively. The 95% confidence interval has been included for this t-test.

Discussion From the t-test results as shown in Table 7, it can be seen that the mean value of $IMP_{(c_i)}$ for all the questions are either less than 0 or close to 0. Similarly, from the t-test results as shown in Table 8, it can be seen that the mean values of $IMP_{(t_i)}$ for all the questions are either less than 0 or close to 0. But, the p-values in these cases are on higher side.

This result indicates that participants somehow learned negatively after using demon-bx tool. Some improvement can be seen for the treated group compared to the control group in terms of mean values, which indicates that demon-bx tool is more harmful without carefully designed scenarios (guidance steps for the user). With these results and higher p-values, the data were not significant enough to reject the null hypothesis H_{NU2}. This shows that for the fact whether demon-bx has a positive effect on the achievement of corresponding learning goals, no conclusive or significant results were obtained.

	Students	**Factor**	**t-value**	**p-value**	**mean**
Q1	21	improvement (control)	-1.7589	0.9531	-0.1904
Q2	21	improvement (control)	0.3841	0.3524	0.0357
Q3	21	improvement (control)	-0.9604	0.8259	-0.8928
Q4	21	improvement (control)	0.2071	0.419	0.0238
Q5	21	improvement (control)	-0.7853	0.7793	-0.8333

Table 7: t-test result showing Improvement (Control Group)

	Students	**Factor**	**t-value**	**p-value**	**mean**
Q1	18	improvement (treated)	-1.5226	0.9269	-0.125
Q2	18	improvement (treated)	-2.3739	0.9852	-0.1527
Q3	18	improvement (treated)	-0.9578	0.5376	-0.0069
Q4	18	improvement (treated)	0.8911	0.1927	0.0486
Q5	18	improvement (treated)	-0.6439	0.7359	-0.0625

Table 8: t-test result showing Improvement (Treated Group)

6.6.4 Analyzing Hypothesis 3

Hypothesis 3 i.e., $\mathbf{H}_{OP3}$ and $\mathbf{H}_{NU3}$ is designed to investigate research question **RQ 4.2**. This hypothesis deals with the fact whether using demon-bx together with suitable scenarios is more effective than without scenarios. Data for **RQ 4.2** can be collected by comparing the improvement of the treated group than the control group. Because before appearing in the posttest, the treated group had used demon-bx with scenarios and the control group had used demon-bx without scenarios.

As five questions were asked, the improvement can be calculated for each question from the pretest and the posttest data separately for control and treated group.

Data Analysis After calculating the improvements for control and treated groups separately, i.e., $IMP_{(c_i)}$ and $IMP_{(t_i)}$ for each question, I used density plots to show their distribution. Figures 43, 44, 45, 46, 47, 48, 49, 50, 51, and 52 show the question wise density plots of the improvements for control and treated group. In the plots, the x-axis represents the improvement as described in Section 6.6.1 ranging between -1 ("absolute negative learning") to 1 ("absolute positive learning"). The y-axis represents the density of 21 students in the case of the control group and 18 students in the case of the treated group lying over the interval. These figures show a comparison between the scattering of the improvement values for control and treated group. From these figures, both uniform and non-uniform distribution of the values can be seen on both sides of the value 0. But, for the naked eye, it is impossible to say whether the improvement in the case of treated group is more or the improvement in the case of the control group is more. So it's very hard to accept or reject the null hypothesis just by comparing these plots.

To gain a better understanding of the data, I have further analyzed it with a t-test. As this hypothesis is related to two sets of improvement data, i.e., control and treated groups, I have used the unpaired two-sample t-test.

As mentioned earlier, improvement is always calculated for posttest compared to the pretest. Here also improvement for the control group is calculated subtracting control group pretest ($RES_{pre(c_i)}$) results from control group posttest ($RES_{post(c_i)}$) results and then dividing it by two for normalizing it between -1 to 1. Similarly, improvement for the treated group is calculated subtracting treated group pretest ($RES_{pre(t_i)}$) results from treated group posttest ($RES_{post(t_i)}$) results and then dividing it by two for normalizing it between -1 to 1. Hence, the value less than 0 signifies "negative" improvement and the value greater than 0 signifies "positive" improvement. As two independent groups are involved, this hypothesis can be expressed as:

Null Hypothesis, H_{NU3}: Mean ($IMP_{(c_i)}$) - Mean ($IMP_{(t_i)}$) > 0

Hence, I have included the option (*alternative="less", var.equal=TRUE*) during the calculation which signifies that according to the operational hypothesis, the mean value of the control group's improvement has to be less than the treated group's improvement assuming that both groups have equal standard deviation. Here I have assumed both groups to be equal as all the students were Master students from the computer science branch with no or very less prior experience in synchronization, bidirectional transformation. The 95% confidence interval has been included for this t-test. The results from the t-test for all the participants are shown in Table 9. For this hypothesis, I have additionally calculated the results with the same t-test only for the good students, i.e., students with some bx background knowledge and more than 90% marks in exams. The results from the t-test for the good students are shown in Table 10.

Discussion From the t-test results (all participants) as shown in Table 9, it can be seen that the mean difference of $IMP_{(c_i)}$ from $IMP_{(t_i)}$ for question Q2 is higher than 0 but for questions Q1, Q3, Q4, and Q5 are less than 0 with higher p-values. From the t-test results (good students)

as shown in Table 10, it can be seen that the mean difference of $IMP_{(c_i)}$ from $IMP_{(t_i)}$ for all questions are either less than 0 or very close to 0 with varying range of p-values.

These results indicate that some positive effect can be seen for the treated group compared to the control group and this improvement in the case of the good students is even better. But, with varying or higher p-values, the data were not significant enough to reject the null hypothesis H_{NU3}. This might be because of the fact that the scenarios are not designed for all types of audience. For example, good students or the students with prior knowledge about bx might comprehend the scenarios better than the average students or the students with no prior knowledge about bx. This shows that with suitable scenarios, some positive learning can be achieved, but this is not evident in all cases and I was not able to conclude this with statistical significance.

	Students	**Factor**	**t-value**	**p-value**	**mean diff.**
Q1	39	Control vs. Treated	-0.8672	0.1957	-0.0654
Q2	39	Control vs. Treated	1.6131	0.9424	0.1884
Q3	39	Control vs. Treated	-0.6813	0.25	-0.0823
Q4	39	Control vs. Treated	-0.1847	0.4272	-0.0248
Q5	39	Control vs. Treated	-0.143	0.4435	-0.0208

Table 9: t-test result showing Improvement (Control vs. Treated Group: All Participants)

	Good Students	**Factor**	**t-value**	**p-value**	**mean diff.**
Q1	12	Control vs. Treated	-2.4058	0.0184	-0.4791
Q2	12	Control vs. Treated	-1.7341	0.0567	-0.2708
Q3	12	Control vs. Treated	-1.2738	0.1158	-0.2916
Q4	12	Control vs. Treated	-0.584	0.2861	-0.1666
Q5	12	Control vs. Treated	0.3274	0.625	0.0833

Table 10: t-test result showing Improvement (Control vs. Treated Group: Good Students)

7 Summary and Future Work

In this chapter, I am going to conclude the entire work done in this thesis. Section 7.1 summarizes the work done in the previous chapters, and Section 7.2 discusses some future enhancements that can be applied to the current work.

7.1 Conclusion

With this thesis, I tried to solve the problem of a missing platform for teaching bx concepts by designing and implementing an interactive demonstrator built on the top of a bx tool. The principal task was to design an application framework to communicate with bx tools and demonstrate their features on an interactive user interface. In the process of solving the problems stated earlier, the following were my contributions:

1. *Analysis and design of an application framework*: I have designed a working, fully functional application framework based on MVC pattern by extending the interface designed for accessing bx tools by Anjorin et al. [6] for implementing a bx tool demonstrator along with an interactive user interface. This framework can be used to implement a demonstrator encapsulating bx tools with concrete examples. Also, it is easy to add additional scenarios to teach bx related concepts, fairly easy to add different versions of the rules used, more or less easy to swap the bx tool, and fairly easy to implement a new example as long as it can work with the user interface.

2. *Concrete implementation of the demonstrator*: After analyzing the existing work on bx and their related problems as described in Section 4, I designed the requirements for my demonstrator to overcome these problems. Afterward, I have implemented a fully functional online demonstrator based on the application framework designed earlier and checked its feasibility and validity. My demonstrator i.e., Demon-BX satisfies all the requirements listed in Section 3 and hence overcomes the identified problems with the existing demonstrator platform for bx. 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.

3. *Evaluation of the demonstrator*: After the implementation, I have evaluated the demonstrator based on the research method called Pretest-Posttest control group design [38] to check its effectiveness and impact. This experiment was driven by three hypotheses and a few experimental variables, i.e., dependent, independent, and controlled. A well-designed experiment was performed with two different groups of participants, data were collected and analyzed to measure the outcome. The results show that basic bx concepts are not well understood by the participants. However, it is challenging to teach bx concepts with a demonstrator but it might be possible to teach bx concepts with a demonstrator along with scenarios (guidance steps for the user) designed for all kinds of participants.

With the above contributions, I was able to build a web platform for a bx tool which is easily accessible to a widespread audience. My demonstrator helps to demonstrate the features of the bx tool, eMoflon along with a few bx concepts. A user can easily try my demonstrator in no time with a few mouse clicks and avoid the complex and time-consuming process which is required in the case of virtual machines, tutorials & handbooks to get a bx tool running.

With these findings, I was able to answer the following research questions with respect to the requirements:

Research Questions	**Solutions**
RQ1 - What are the core requirements for implementing a successful bx demonstrator?	From the work of my thesis, I feel that the most important requirements for implementing a successful bx demonstrator are the functional & non-functional behaviors as described in Section 3. For example, that the user should able to initiate the synchronization process in both directions, the user should able to manipulate the models before synchronization, the example should be simple and fun to play with, the GUI should be interactive, zero setup time for the demonstrator, there is some interaction involved and perhaps decision making while playing with the example, the demonstrator should be accessible worldwide and platform independent, etc.
RQ2 - To what extent is such a bx demonstrator reusable?	A few dimensions involved with the implementation of a bx demonstrator are (i) addition of different synchronization rules, (ii) visualization of different examples, and (iii) the ability to demonstrate the features of different bx tools. From the implementation work of my demonstrator, I can say that my application framework along with the extended interface for accessing the bx tools can be reused/extended to realize these dimensions. It is fairly easy to add different versions of the rules used, more or less easy to swap the bx tool, and fairly easy to implement a new example as long as it can work with the UI.
RQ3 - Is there a need to teach the concepts of bx through a demonstrator?	It is evident from the outcome of the hypothesis H_{OP1} (which corresponds to the research question **RQ3.1**) that the students derive their (in general wrong) intuition for and expectations of synchronization scenarios from the special case of bijections. Hence, there is indeed a need to teach correct expectations for synchronization scenarios and a demonstrator can be potentially very useful in spreading bx concepts.

RQ4 - Does an interactive GUI helps a user to increase his/her understanding related to bx concepts?	To teach bx related concepts through the demonstrator, I have carefully designed the corresponding scenarios by combining them with different parts of the implemented example and allowed the user to play with it in order to learn the concepts. However, this is indeed quite challenging. It is evident from the outcome of the hypotheses H_{OP2} and H_{OP3} (which corresponds to the research question **RQ4.1** and **RQ4.2** respectively) that without carefully designed scenarios, there is no guarantee that the learning goals (**LG**s) are addressed and there might even be negative learning. With scenarios, the situation is a bit better, but I was not able to conclude this with statistical significance.

Table 11: Solutions to the Research Questions

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 teach bx concepts to a widespread audience to the best of my knowledge and belief. However, a few other features that were out-of-scope for this thesis can be added to the current work. In the following paragraphs, I have discussed some enhancements that can be applied to my work in the future.

New Rules My current implementation of the eMoflon tool is based on two meta-models and five transformation rules. However, it will be interesting to add a few more transformation rules to the existing meta-models. It will surely enhance the interactivity of the demonstrator and also test the capability of UI on handling new rules.

New Examples I have implemented one example i.e., *Planning a Kitchen* in the demonstrator. However, it would be fascinating to see the implementation of a few more examples on the same or different UI platform built on the top of the same application framework. Some new examples will definitely test and verify the robustness of the application framework as well as the features of the eMoflon tool further.

Another BX Tool Demon-BX is implemented encapsulating the features of the bx tool `eMoflon`. However, to test the extensibility of the application framework and the interface

built for communicating with bx tools, implementation of another bx tool can be researched and added to the current work.

BX Tool as a Webservice In the current implementation of demon-bx, the `Model` component in the application framework is built as a Java project. Due to a Java project, it has a very thin binding with the `Controller` component. Any changes in model component related to class name, method name and their accessibility will affect the implementation of the example specific adapter class. Hence, to remove this dependency, the model component can be designed and implemented as a web service. After that, the model can be seen as a plug-in component communicating through JSON/XML data, which completely removes its dependency on the controller component. Hence, any researcher can implement the synchronization with their tool and use the web service.

New Experiment As per the research method Pretest-Posttest control group design [38], only treated group should be exposed to the treatment which the researcher wants to test. However, in my experiment, the control group was given the demonstrator without any scenarios and then asked to answer the posttest question. This might affect the final output as somehow control group was exposed to the demonstrator environment and can cause the same effect what treated group experienced with scenarios. Hence, a new experiment can be done by showing the control group only a video about bx concepts in relation with a concrete example instead of exposing them to the demonstrator environment.

8 Appendix

Servlet Lifecycle Figure 53 describes the complete lifecycle of a servlet w.r.t. a web server and a web container. The lifecycle steps are described as follows [35]:

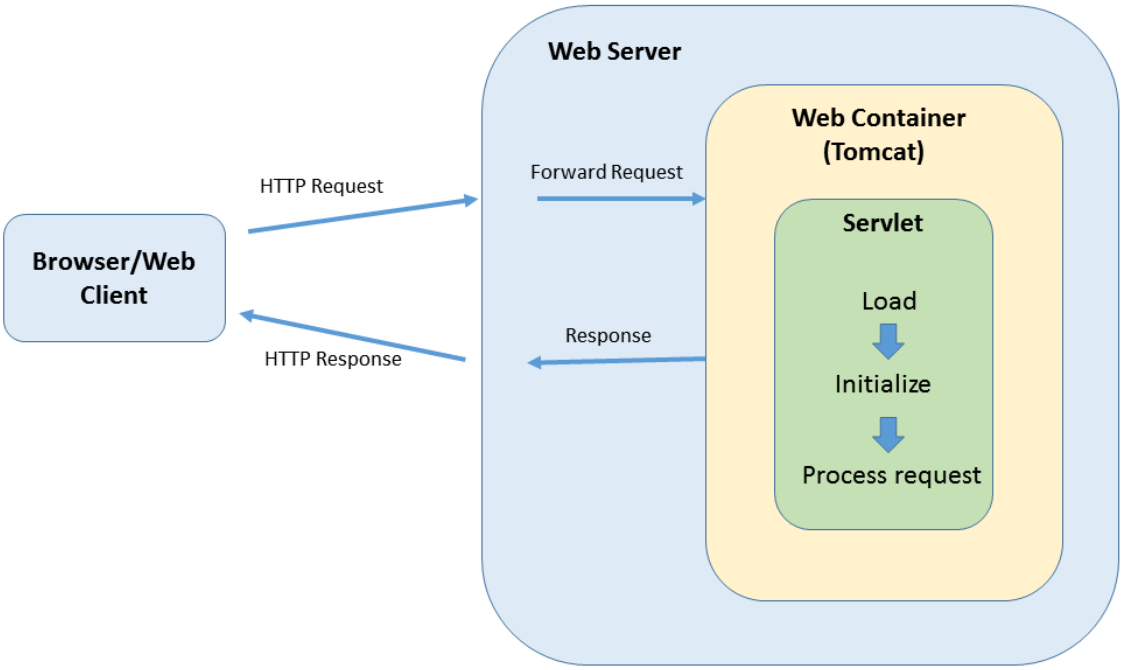


Figure 53: Servlet Lifecycle

1. The web server receives the `HTTP request` from the client interacting through a browser. After accepting the request, web server forwards the request to the web container i.e., Tomcat.
2. Web container sends the request to the `Servlet class`. 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.
3. 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.
4. Web container sends the response to the web server. Afterward, web server creates the HTTP response and send it back to the client.

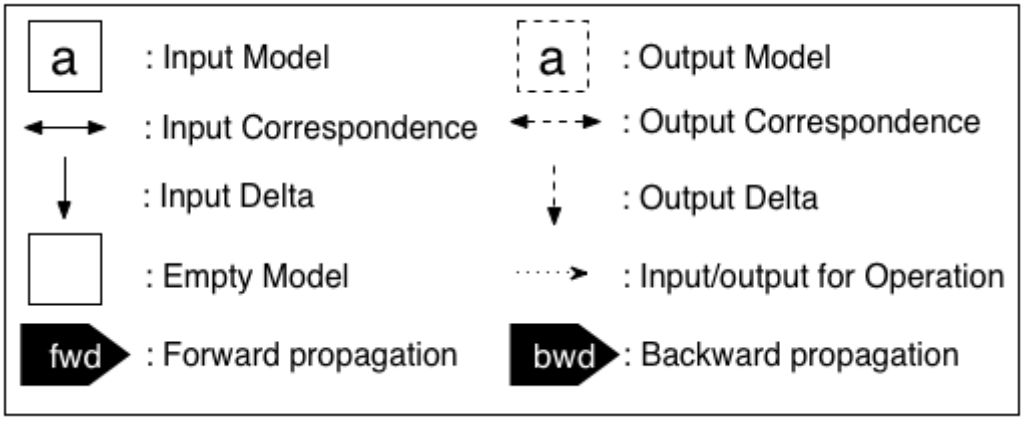
Evaluation Test Questions

* Required

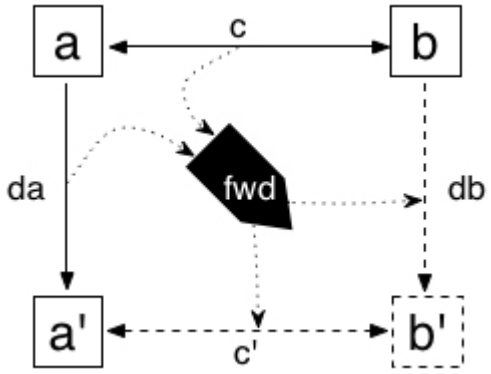
Legend and Assumptions

For easy reference, below is the legend used for all diagrams in the questions.

For all questions we assume a consistency relation R is given, and that the forward and backward propagation operations are correct with respect to R. Models with the same label are assumed to be identical, while models with different labels, e.g. < a > and < a' > are assumed to be different.



1. **Do you think the model b' must be unique?** *



Mark only one oval.

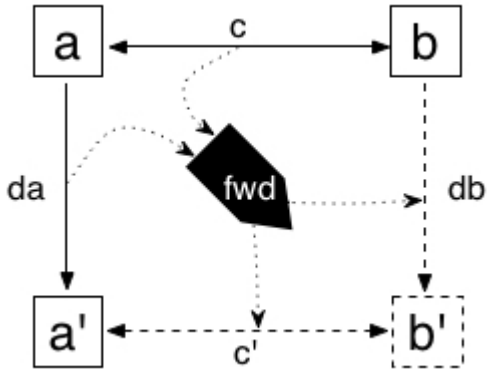
- ◯ Yes, there is always exactly one such model b' that is consistent with a'
- ◯ No, there might be many models b' that are consistent with a'

2. **How sure are you about your answer?** *

Mark only one oval.

	1	2	3	4	5	
I just guessed	◯	◯	◯	◯	◯	I'm certain

3. **Do you think such a model b' must exist?** *



Mark only one oval.

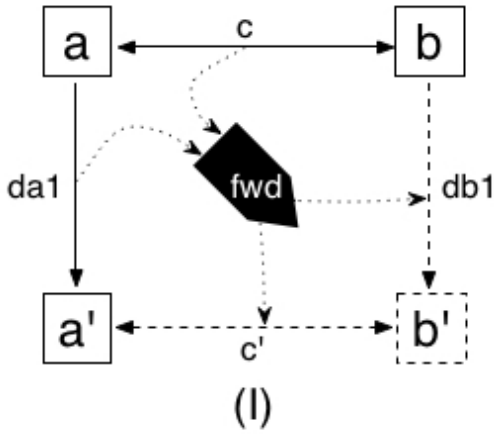
- ◯ Yes, there is always at least one such b'
- ◯ No, such a model b' might not exist at all

4. **How sure are you about your answer?** *

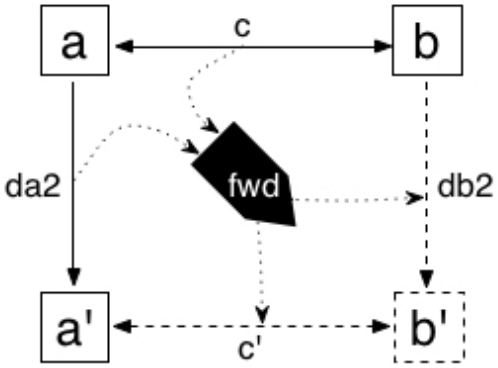
Mark only one oval.

	1	2	3	4	5	
I just guessed	◯	◯	◯	◯	◯	I'm certain

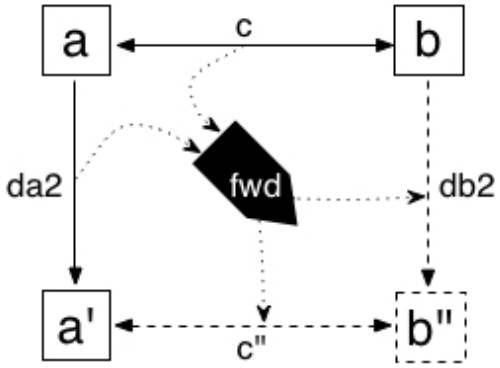
5. **Given the situation depicted in (I), which of the diagrams (II) or (III) is to be expected in general?** *



(I)



(II)



(III)

Mark only one oval.

( ) (II) is to be expected; even though a different delta da2: a -->a' is propagated, it results in the same model a' as da1 so db2 (produced by fwd(da2, c)) must also result in the same model b' as db1

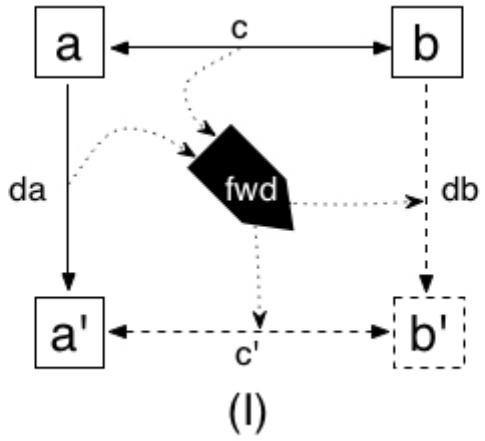
( ) (III) is to be expected; a different delta da2: a -->a' is propagated so the resulting consistent model b'' can be completely different from b'

6. **How sure are you about your answer?** *

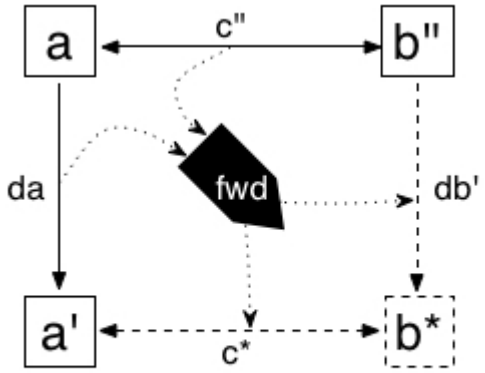
Mark only one oval.

	1	2	3	4	5	
I just guessed	( )	( )	( )	( )	( )	I'm certain

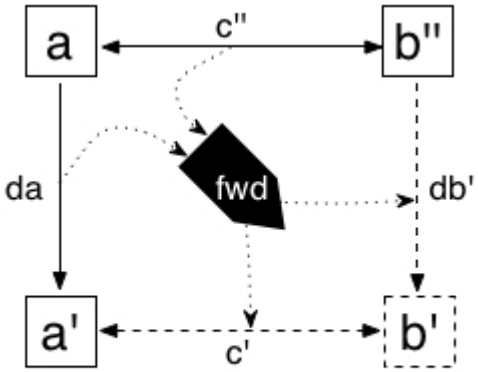
7. **Given the situation depicted in (I), which of the diagrams (II) or (III) is to be expected in general?** *



(I)



(II)



(III)

Mark only one oval.

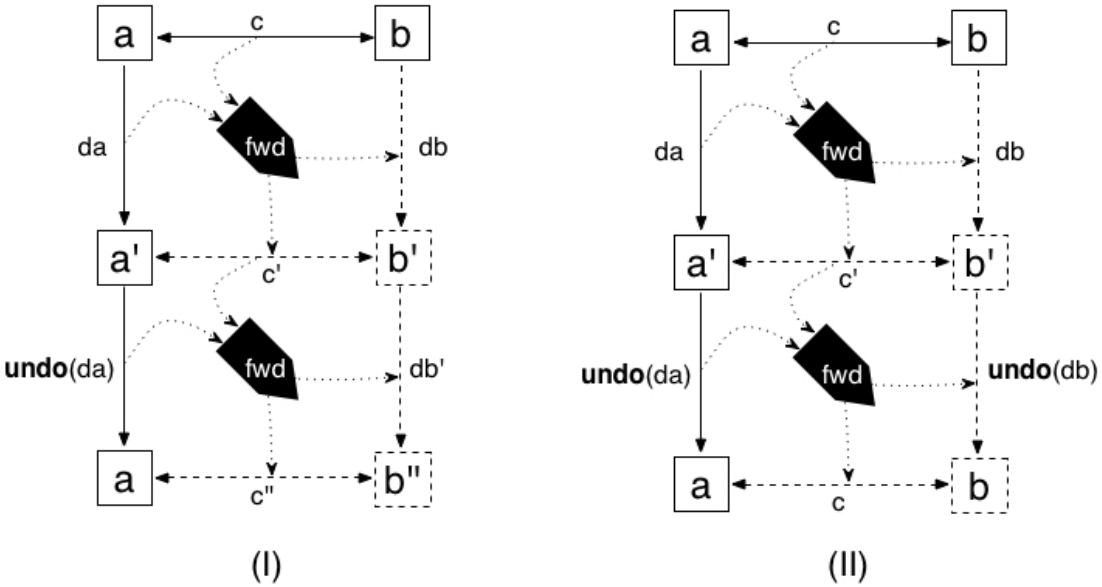
- (II) is to be expected; even though the same delta da is propagated, the result depends on the previous model b''. As b'' is different from b, b* will be in general different from b'
- (III) is to be expected; the same delta da: a --> a' is propagated so this must result in the same model b' as in (I)

8. **How sure are you about your answer?** *

Mark only one oval.

	1	2	3	4	5	
I just guessed						I'm certain

9. **Which diagram (I) or (II) is to be expected in general?** *



Mark only one oval.

(I) is to be expected; undoing da and propagating this delta does not guarantee that the same model b can be recovered

(II) is to be expected; if da can be undone and this change is propagated, then db will also be undone to result in the previous model b

10. **How sure are you about your answer?** *

Mark only one oval.

	1	2	3	4	5	
I just guessed						I'm certain

Transformation Rules In this paragraph, I have described all the transformation rules implemented with eMoflon tool for the source and target models. Source models consist of a single `Grid`, with `Groups` that can occupy multiple `Blocks`. `Blocks` are additionally connected with all their neighboring blocks. Target models consist of a single `Kitchen` with `ItemSockets` as placeholders for exactly one `Item`, i.e., sink, fridge, table, etc.

1. *Kitchen_to_Grid:* Figure 54 describes that if you create a `Grid` in the source, then you should create a corresponding `Kitchen` in the target.

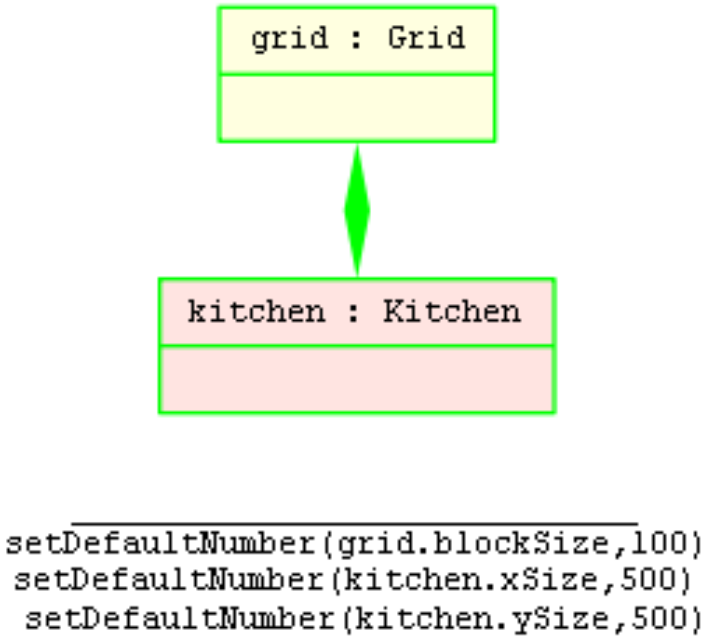


Figure 54: Transformation Rule: Kitchen_to_Grid

2. *ItemSocket_to_Group:* Figure 55 describes that if you add a new `Group` to the grid, then you should add an `ItemSocket` (a placeholder for items) to the kitchen.

3. *Create_a_Sink:* Figure 56 describes that a `Sink` must be created on the western wall and occupies two horizontal blocks.

4. *Create_a_Fridge:* Figure 57 describes that a `Fridge` must be created on the northern wall and occupies two vertical blocks.

5. *Create_a_Horizontal_Table:* Figure 58 describes that if you create a `Horizontal Table`, then its group should be placed on two blocks horizontally.

6. *Create_a_Vertical_Table:* Figure 59 describes that if you create a `Vertical Table`, then its group should be placed on two blocks vertically.

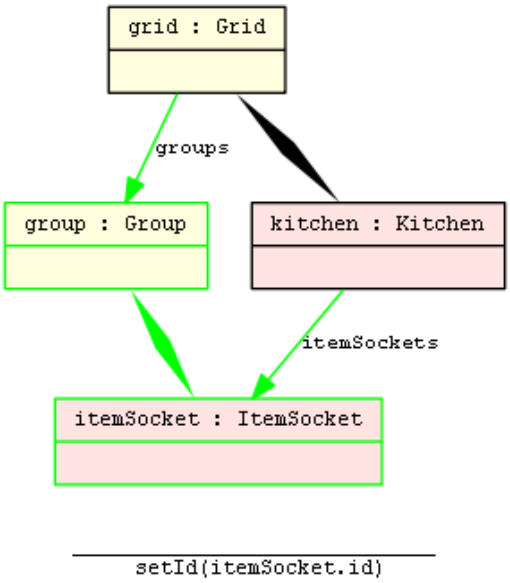


Figure 55: Transformation Rule: ItemSocket_to_Group

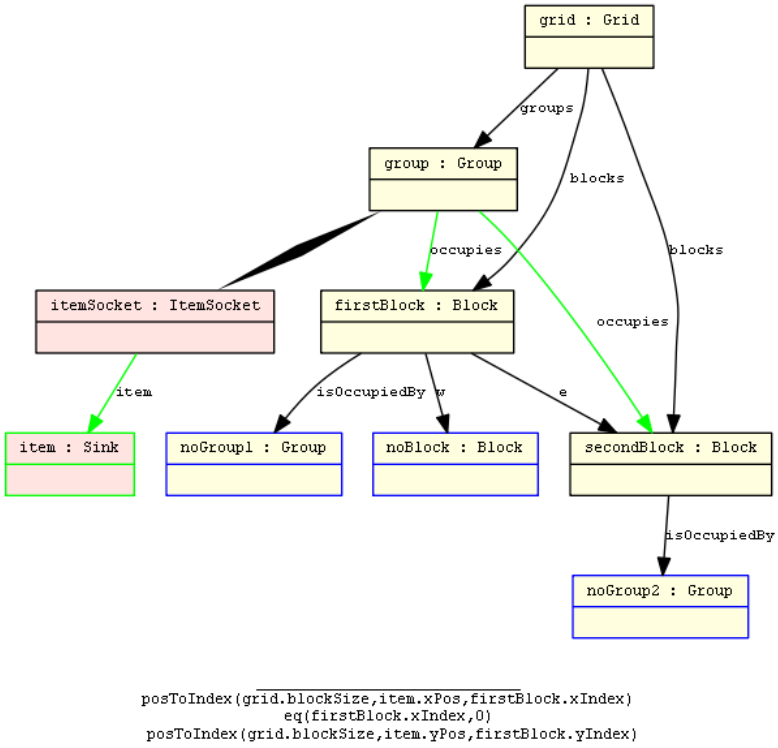


Figure 56: Transformation Rule: Create_a_Sink

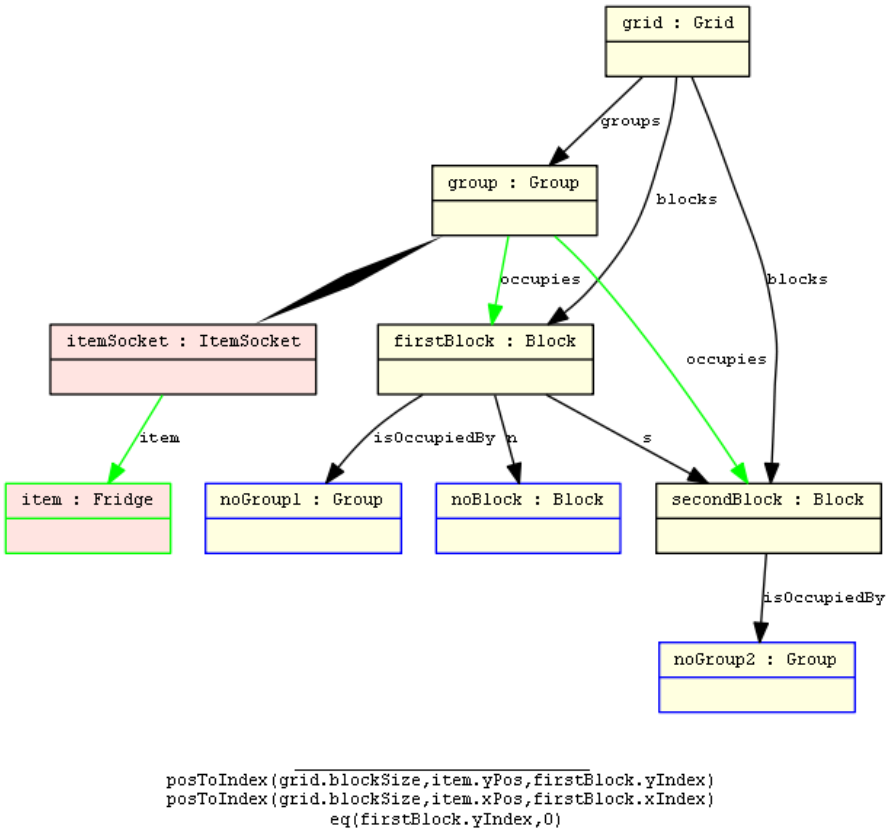


Figure 57: Transformation Rule: Create_a_Fridge

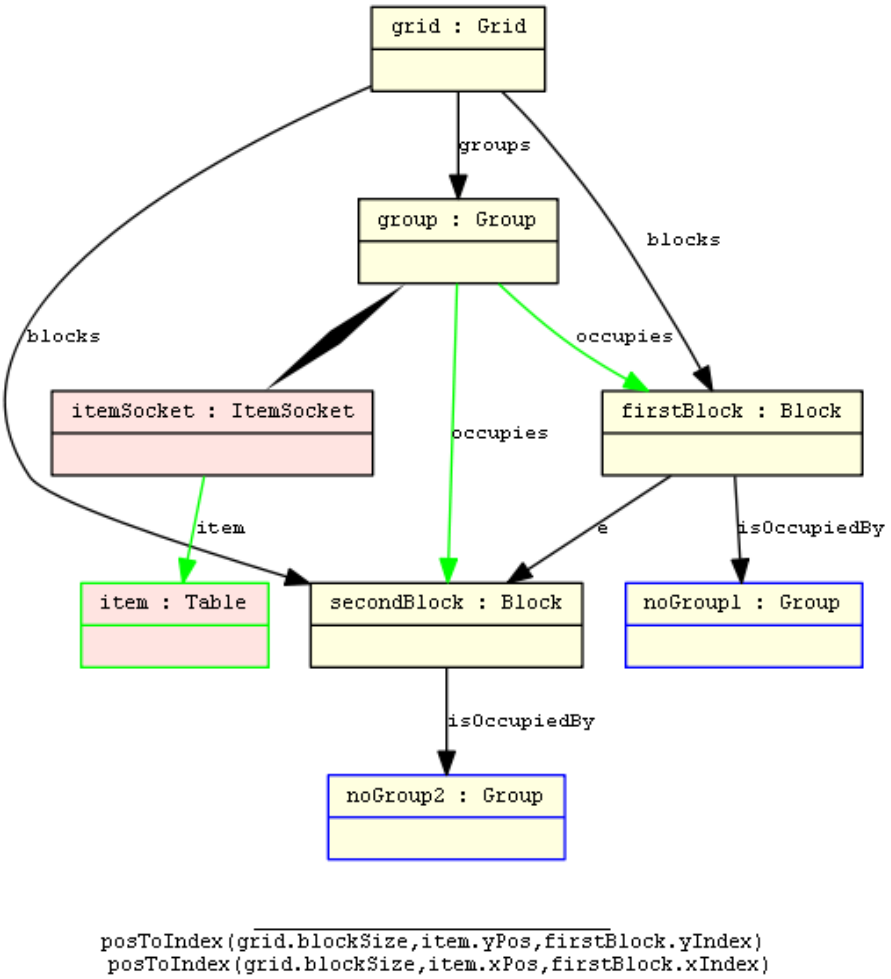


Figure 58: Transformation Rule: Create_a_Horizontal_Table

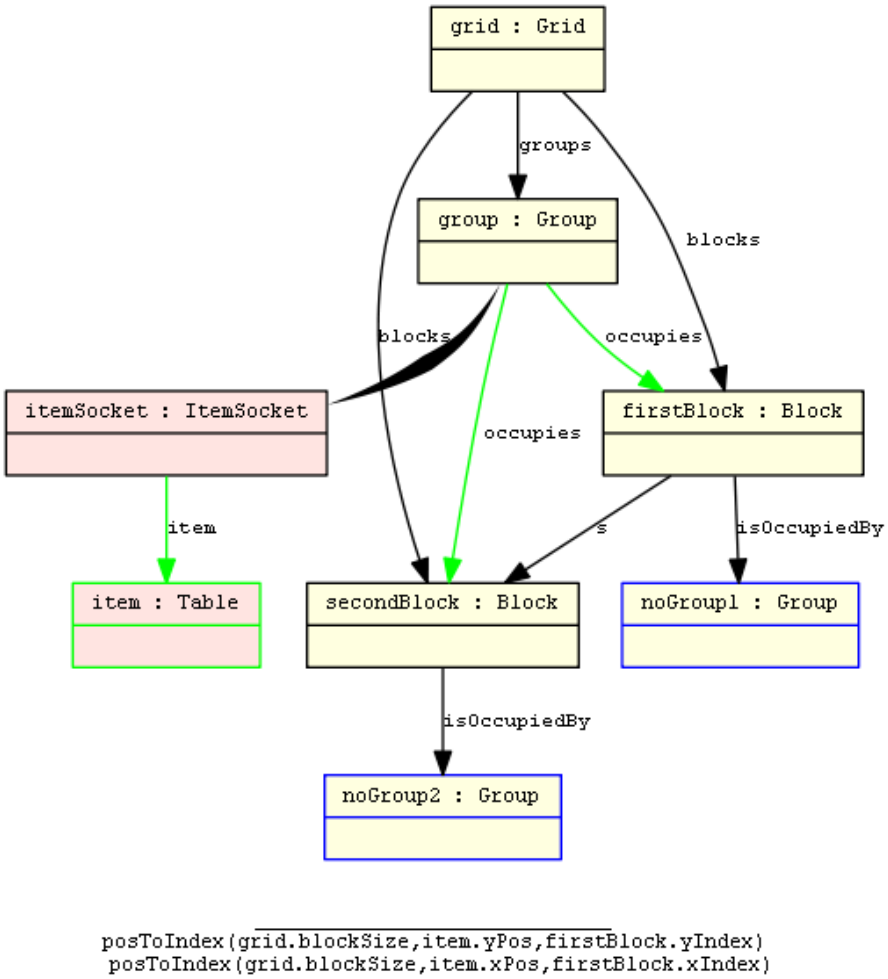


Figure 59: Transformation Rule: Create_a_Vertical_Table

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 on 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, (eds.). In Proc. of the EDBT/ICDT 2014 Joint Conference, CEUR Workshop, pages 82-86, 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.). In Proc. of the Sixth International Workshop on Bidirectional Transformations (BX 2017), Uppsala, Sweden, April 29, 2017, published at http://ceur-ws.org. 9, 10, 26, 27, 32, 76

[7] A. Schürr. *Specification of graph translators with triple graph grammars*, In E. W. Mayr, G. Schmidt, and G. Tinhofer (eds.). In Proc. of the 20th International Workshop, Graph-Theoretic Concepts in Computer Science, WG 94, vol. 903, 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 International Workshop on Bidirectional Transformations (BX 2013), vol. 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 Examples*, [Online]. Available: http://bx-community.wikidot.com/examples:home 19, 25

[12] N. Macedo, T. Guimaräes and A. Cunha, *Model repair and transformation with Echo*, In Proc. of 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 Proc. of Partial Evaluation and Program Manipulation, ACM, pages 61-72, 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] *Google Forms*, [Online]. Available: https://www.google.com/forms/about/ 62

[19] A. Bucaioni and R. Eramo, *Understanding bidirectional transformations with TGGs and JTL*, In Proc. of the Second International Workshop on Bidirectional Transformations (BX 2013). 11

[20] 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, 11

[21] S. Easterbrook, J. Singer, M.-A. Storey, & D. Damian. *Selecting Empirical Methods for Software Engineering Research. Guide to Advanced Empirical Software Engineering*, pages 285-311, 2008. Retrieved from http://doi.org/10.1007/978-1-84800-044-5_11 4, 57

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

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

[24] S. Sendall and W. Kozaczynski, *Model transformation: the heart and soul of model-driven software development*, IEEE, vol. 20, issue 5, pages 42-45, September-October, 2003. 1

[25] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns*, Addison-Wesley, vol. 47, Boston Massachusetts USA, 1995. 27

[26] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Abstraction and Reuse of Object-Oriented Design*, In O.M. Nierstrasz (eds.). ECOOP 1993 - Object-Oriented

Programming, Lecture Notes in Computer Science, Springer, vol. 707, Berlin, Heidelberg. 27, 28

[27] J. Deacon, *Model-view-controller (mvc) architecture*, Computer Systems Development, pages 1-6, 2009. 32, 37

[28] 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), vol. 4607, pages 457-472, 2007. 32, 45

[29] E. Freeman, E. Robson, K. Sierra and B. Bates, *Head First Design Patterns*, O'Reilly, pages 536-566, USA, 2004. 28, 32, 37

[30] *MDN, "Canvas API",* Retrieved May, 2017, from https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API. 43

[31] *Fabric.js*, [Online]. Available: http://fabricjs.com/. 44

[32] *Processing.js*, [Online]. Available: http://processingjs.org/. 44

[33] *Pixi.js*, [Online]. Available: http://www.pixijs.com/. 44

[34] D. Goodman and M. Morrison, *Javascript Bible*, Wiley Publishing Inc., 6th Edition, pages 1-8, Indianapolis, Indiana, 2007. 44

[35] J. Hunter and W. Crawford, *Java servlet programming*, vol. 37, 2001. 46, 80

[36] 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

[37] P.L. Bonate, *Analysis of pretest posttest designs*, FL: Chapman & Hall/CRC, pages 1-5, 2000. 58

[38] D.T. Campbell & J.C. Stanley, *Experimental and quasi-experimental designs for research* , In N. L. Gage (eds.), Handbook of research on teaching, Rand McNally, Chicago, 1963. 58, 76, 79

[39] 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. 58

[40] R. Feldt & A. Magazinius, *Validity Threats in Empirical Software Engineering Research - An Initial Survey* , In Proc. of the International Conference on Software Engineering and Knowledge Engineering, issue March, pages 374-379, 2010. 63

[41] C. Wohlin, M. Höst, P. Runeson, M. Ohlsson, B. Regnell, and A. Wesslen, *Experi-*

mentation in software engineering: an introduction, Kluwer Academic Publishers, 2000. 63