IEEE Copyright Notice

Published in: ***Proceedings of ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS'19)*** September 2019

A Focus+Context Approach to Alleviate Cognitive Challenges of Editing and Debugging UML Models

Cite as:

P. Pourali and J. M. Atlee, "A Focus+Context Approach to Alleviate Cognitive Challenges of Editing and Debugging UML Models," *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, Munich, Germany, 2019, pp. 183-193.

BibTex:

```
@INPROCEEDINGS{8906900,
author={P. {Pourali} and J. M. {Atlee}},
booktitle={2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)},
title={A Focus+Context Approach to Alleviate Cognitive Challenges of Editing and Debugging UML Models},
year={2019},
pages={183-193},
```

DOI: https://doi.org/10.1109/MODELS.2019.000-3

A Focus+Context Approach to Alleviate Cognitive Challenges of Editing and Debugging UML Models

Parsa Pourali
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada
ppourali@uwaterloo.ca

Joanne M. Atlee
David Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
jmatlee@uwaterloo.ca

***Abstract*—Model-Driven Engineering has been proposed to increase the productivity of developing a software system. Despite its benefits, it has not been fully adopted in the software industry. Research has shown that modelling tools are amongst the top barriers for the adoption of MDE by industry. Recently, researchers have conducted empirical studies to identify the most-severe cognitive difficulties of modellers when using UML model editors. Their analyses show that users' prominent challenges are in remembering the contextual information when performing a particular modelling task; and locating, understanding, and fixing errors in the models. To alleviate these difficulties, we propose two Focus+Context user interfaces that provide enhanced cognitive support and automation in the user's interaction with a model editor. Moreover, we conducted two empirical studies to assess the effectiveness of our interfaces on human users. Our results reveal that our interfaces help users 1) improve their ability to successfully fulfil their tasks, 2) avoid unnecessary switches among diagrams, 3) produce more error-free models, 4) remember contextual information, and 5) reduce time on tasks.**

***Index Terms*—User-Centric Software Development, Empirical Study, UML, Modelling Tools, Modelling Challenges.**

I. Introduction

Model-Driven Engineering (MDE) is a software design methodology that focuses on improving the productivity of developing software systems by representing and testing the important properties of the system before coding begins. MDE involves several artefacts amongst which models are the core assets. The engineer designs precise models to express the elements of a software system and their properties (by means of graphical or textual notations) in order to enhance the process of automated code generation and improve understanding and reasoning on the system. Despite their benefits, models have not been well-adopted in the industry because of various barriers, amongst which the lack of proper tooling techniques is the most crucial [1] [2].

Model editors allow modellers to edit and debug models. Tool vendors have expended considerable effort to design and develop model editors that are easy to use, but their endeavors have not fully succeeded in overcoming adoption barriers because their approaches are mostly *artefact-centric*. That is, their model-easing features facilitate modelling tasks mostly by taking into account only the state and properties of the model, the meta-model, and the constraints on them. Although artefact-centric approaches offer many different tooling techniques and features to improve the editors' usefulness, their effectiveness is rarely empirically assessed. Tool developers rarely investigate modellers' prominent challenges (by performing a thorough analysis of the modellers and their tasks), and they do not assess the effectiveness of modellers in using their tools. Thus, they miss opportunities to address usability concerns, leading to a chasm between what modellers expect and what the model editors provide.

We believe that a more effective and complementary approach to improve modelling editors' usability is to employ *user-centric* techniques, which focus on understanding users' difficulties and augmenting users' cognitive abilities such as visual capabilities, working memory, and task-specific design comprehension. Accordingly, our approach, which we call **U**ser-**C**entric and **A**rtefact-Ce**n**tric **D**evelopment **o**f **Models** (*UCAnDoModels*), employs both user-centric and artefact-centric strategies to enhance the MDE editors' usability. Our approach consists of five important steps: 1) understanding the foremost difficulties experienced by model editor users, 2) correlating each of the difficulties to human cognitive challenges, 3) proposing tooling solutions that address the cognitive challenges, 4) assessing the effectiveness of the proposed solutions on human users, and 5) iterating over the solution to optimize it based on feedback from users.

Based on the results of our previous empirical study [3], users' most-severe difficulties are: (1) *Context: remembering contextual information* and (2) *Debugging: locating, understanding and fixing errors and inconsistencies in models*. To overcome these difficulties, we propose two *Focus+Context* [4] interfaces that aim to reduce cognitive challenges of developing models by providing users with the information (*Context*) that are relevant to performing a particular task (*Focus*). We implemented our proposed interfaces in a UML model editor that we developed using Eclipse-based modelling technologies. Finally, we conducted a user study to evaluate the effectiveness of our proposed Focus+Context interfaces.

The rest of this paper is organized as follows. In Section II, we discuss the background knowledge relevant to understanding our work. In Section III, we present an architectural overview of our model editor. In Sections IV and V, we present more detailed descriptions of our Focus+Context user interfaces for the *Context* and *Debugging* challenges, respectively.

In Section VI, we present the results of an empirical evaluation of our techniques followed by the threats to the validity in Section VII. Section VIII discusses the related works. We conclude in Section IX.

II. Background

The Unified Modelling Language (UML) offers *static* and *dynamic* types of diagrams to model different aspects of a system: static diagrams illustrate the structure of the system, whereas dynamic diagrams model the behaviour of the system. The most-prominent examples of static and dynamic diagrams are the Class diagram and the State-Machine diagram.

A Class diagram represents a system's entities and their properties (e.g., attributes and operation), as well as the relationships among the entities. Fig. 1 shows an example Class diagram consisting of a few classes. It shows the attributes and operations of the classes (if any), and illustrates how these classes are related to each other through edges. An edge can represent an association (e.g., a *Transponder* can be sensed by a *Sensor*), a composition (e.g., a *Gate* includes *Sensors*), or a Generalization (e.g., *GateA* is a sub-type of *Gate*).

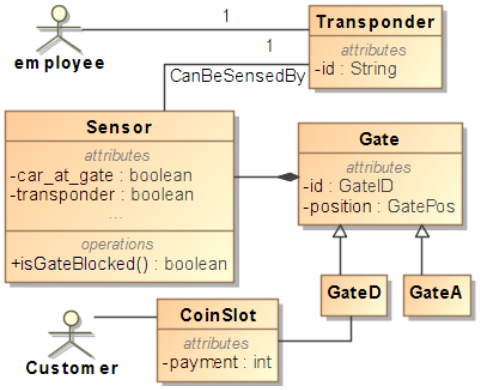

Fig. 1: An example of a Class diagram

For each class in the Class diagram, there can be a State-Machine diagram that represents its behaviour. Fig. 2 shows an example State-Machine diagram for the class *GateA*. It consists of an initial pseudo-state and two normal states (*Closed* and *Open*). These states are connected by *transition* links. A *transition* is labelled by an *expression* that shows the *event* that triggers the *transition*, the conditions (referred to as a *guard*) must hold for the transition to execute, and the effects of the transition (referred to as an *action*) when it executes.

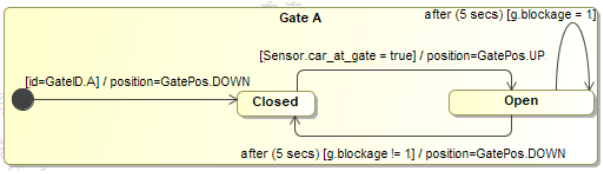

Fig. 2: An example of a State-Machine diagram

In spite of the invaluable benefits offered by the UML, researchers have shown that users' challenges with UML tools are one of the major obstacles to the adoption of MDE by industry [1] [2]. We [3] investigated further and conducted a formative user study to identify and understand the difficulties of editing and debugging UML Class and State-Machine diagrams, representing the static and dynamic diagrams[1] used most permanently in model analysis and code generation, respectively. Their study revealed that users' most-severe cognitive challenges are 1) *Context: remembering contextual information needed to write correct, complete, and precise transition expressions in State-Machine diagrams*, and 2) *Debugging: locating and fixing errors in the model*. Ghazi and Glinz [5] partially confirms these same challenges, in the context of tools to develop requirements artefacts.

These challenges stem primarily from the separation of concerns that are inherent in the distinct views captured in distinct UML diagrams. UML modelling tools offer diagram-specific editors that do little to help users fetch and understand the inter-related information that is expressed separately in other related diagrams [6]. For example, developing a correct expression for a state-transition guard in a State-Machine diagram can be cumbersome for the modeller because they need to refer to precise details about names, types, attributes, parameters, and associations of model elements that are defined in a separate Class diagram.

III. Tool Overview

We propose UCAnDoModels, which employs both user-centric and artefact-centric strategies to overcome users' foremost challenges with UML model editors and we built a model editor that incorporates the UCAnDoModels approach. This section presents some of the underlying capabilities of our editor. The editor's architecture is shown in Fig. 3. The **Graphical and/or Textual Editors (Editors)** embody the primary components that allow users to edit a model; their default diagramming features were built using Eclipse's Ecore Modelling Framework (EMF) [7], the Graphical Modeling Framework (GMF) [8], and the Xtext Textual Editor [9].

Within the Editors component, our primary contributions are **Task-Oriented Model Editors**, which aim to alleviate the *Context* challenge. They are composed of the following modules:

Distance-Oriented Objects Indexer: responsible for collecting the information about relationships among all of the model elements declared in the Class diagram. Its main task is to *order* the model elements by the $Distance$ value between each element and the specific element that is the current focus of an editing task. For example, if we are writing a transition expression in a State-Machine diagram for class C1, the Distance-Oriented Objects Indexer categorizes the model's elements into five categories as follows:

Category A: comprises the values or literals of the basic types (e.g., Boolean, Integer, enumerations). For instance, if the model element of current focus is of type Boolean, then

[1]Hereafter, the term *modelling* refers to editing and debugging Class and State-Machine diagrams.

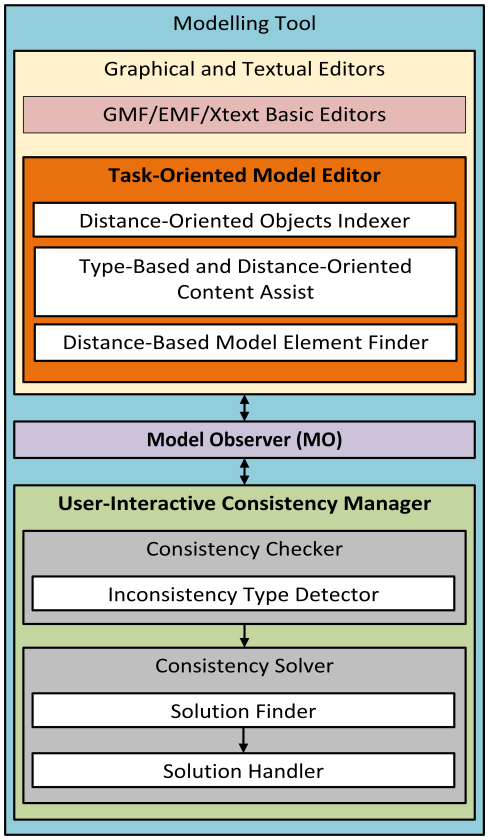

Fig. 3: The overall architecture of our UCAnDoModels editor

values of $True$ and $False$ values are included in this category. Basic-type values or literals have a $Distance$ of 0.

Category B: comprises the object members (e.g., operations or attributes) that are defined in C1. The distance for an element in Category B is one ($Distance = 1$).

Category C: comprises the model elements that are defined in the ancestor classes of C1. The elements in this category have a $Distance$ of 2.

Category D: comprises the model elements that are defined in the classes with which C1 has a direct relationship (association, composition or aggregation). The $Distance$ for a class related by composition or aggregation is equal to 3 and the $Distance$ for a class related by an association is 4. Thus, compositions have priority over associations.

Category E: comprises the model elements that are defined within the classes that are indirectly related to C1. The $Distance$ for an element in this category depends on the number of links (edges) between the element and C1, that is $Distance = 4 + (NumberOfLinks - 1)$. Thus, the further the element, the less related it is.

After categorizing the elements, the Indexer ranks them based on their respective $Distance$ values and alphabetically sorts the elements that have the same $Distance$.

Type-Based and Distance-Oriented Content-Assist: Content-Assist in Java [10] (or Intellisense in Visual Studio [11]) is a tooling feature that enables the developer to view a list of the next available commands to write (at a specific cursor position) when writing Java code. The list is populated by the IDE and is based on the types of the elements declared in a program or on the programming language's grammar. Model editors are starting to borrow such techniques and allow modellers to use Content-Assist when writing model expressions. However, the main challenge of designing a Content-Assist is proposing relevant and valid options to the user [12]. The existing editors retrieve the list of available options from the meta-model and its constraints. They essentially populate the Content-Assist list based on the language syntax, rather than analysing the model elements and filtering out the ones that are not semantically sound. In contrast, our editor provides **Type-Based and Distance-Oriented** Content-Assist to propose values or model elements that are most likely to be the intended next clause in the expression currently under edit, thereby reducing the effort of editing the expression. Our Content-Assist uses the *Distance-Oriented Objects Indexer* component described above and embeds a *Type-Based Classifier* that performs lightweight type checking to ensure that variable assignments and conditional expressions (e.g., guard conditions) are semantically sound by type. For instance, if an operand in an assignment is an attribute of type T1, then the other operand should be limited to attributes or operations with the return type of T1 or sub-type of T1. After filtering out the elements that are not sound-by-type, the Distance-Oriented Objects Indexer ranks and orders the elements.

With our Content-Assist, the user will be given only options that are syntactically correct and semantically sound. Such assistance is expected to improve model correctness and reduce modelling efforts, because the modeller is offered a refined list of valid options rather than an exhaustive list of all syntactically-related model elements, which can be lengthy and difficult to explore.

Distance-Based Model Element Finder: This module allows users to search for an intended model element among all the existing model elements. It first attempts to find the elements whose names exactly match what the user typed. However, it is not always the case that the user remembers the exact name of an element; users usually remember a part of a name or tries to "guess" a similar name. Therefore, if no element with the exact name is found, this module uses the Levenshtein edit distance [13] algorithm, an approximate string-matching algorithm, to look for elements whose names are similar to what the user typed, and lists them in decreasing order of the computed edit distance. If multiple elements with the same edit distance are found, the module uses the Distance-Oriented Object Indexer to rank and order them before displaying them to the user.

The **Model Observer (MO)** component continuously listens to model edits and checks that each edit satisfies the editor's consistency rules. Our model editor adapts nine types of *Well-formedness* rules and five types of *Consistency* rules proposed by Lange et al. [14]. The *Well-formedness* rules prevent the modeller from using incorrect UML syntax or writing an ill-formed expression. The *Consistency* rules check for more severe types of errors such as using an undefined element, producing a semantically wrong expression, or introducing

TABLE I: List of Recall and Modify actions, and the relevant sections of the Focus+Context Transition Editor

ID	Type	Description	Section(s)
Rec.1	Recall	Remembering the elements in the Class diagram (i.e., Classes, Attributes, Operations, Types)	A,B,C
Rec.2	Recall	Remembering the relationships amongst the entities in the Class diagram	A
Rec.3	Recall	Remembering/reusing expressions (i.e., event, guard, actions) that are used in other transitions	D
Rec.4	Recall	Looking for an intended element based on the modeller's recollection of its name	C
Mod.1	Modify	Creating an intended element (e.g., classes, attributes, operations) in the Class diagram	A,B,C
Mod.2	Modify	Creating a relationship between entities in the Class diagram	A
Mod.3	Modify	Changing the type of an element (e.g., attribute) in the Class diagram	B

type-mismatches in a condition. If a Well-formedness or a Consistency rule is violated as a result of a modification or a deletion in the model (e.g., using an element in a transition expression that is undefined in the Class diagram), the MO conveys the inconsistency to the User-Interactive Consistency Manager module, described below.

The User-Interactive Consistency Manager component consists of two main modules: a Consistency Checker and a Consistency Solver. The Consistency Checker detects the *type* of inconsistency based on Lange's definition [14], and finds a proper solution, using the Solution Finder and Solution Handler modules (embodied in the Consistency Solver), respectively. A solution can be Auto-Fix, Quick-Fix, or Interactive-Fix. Auto-Fix applies if the Solution Finder suggests that there is only one possible valid fix for an inconsistency (e.g., renaming a model element should rename all its uses in the model); the Solution Handler asks the user to confirm the fix before automatically fixing the model. Quick-Fix applies if the Solution Finder identifies multiple valid fixes for an inconsistency. The Solution Handler then proposes the list of possible fixes and allows the user to choose one from the list. For example, if there is an initial pseudo-state in a region that is not connected to any other states, the Solution Handler provides the user with a list of possible regular states to which the initial pseudo-state can be connected. If the Solution Finder decides that neither Auto-Fix nor Quick-Fix techniques are useful, it displays our Interactive-Fix dialogue and asks the user to intervene to resolve the issue before proceeding with subsequent model edits.

These modules provide fundamental capabilities in our Task-Oriented Model Editor, described in Section IV, and our User-Interactive Consistency Manager, described in Section V. Please note that, the capabilities of the interfaces are the main focus of this paper and not the visual design.

IV. Focus+Context Transition Editor

We believe that the *Context* challenge is mostly due to two difficulties: 1) *Recall*: consulting multiple relevant diagrams to find contextual information (e.g., a modeller may need to switch back to the Class diagram to recall an intended element that he/she wants when writing a state-transition expression in a State-Machine diagram), and 2) *Modify*: performing the prerequisite steps to some modelling task (e.g., defining a model element in the Class diagram before using it in a state-transition expression).

Focus+Context editors [4] have been proposed to reduce users' cognitive load, by allowing them to integrate various

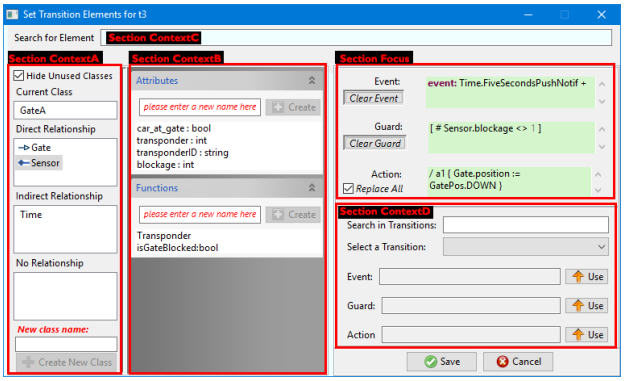

Fig. 4: Our Focus+Context Transition Editor

diagrams (i.e., Context) that are relevant to their current task (i.e., Focus). Accordingly, our model editor employs the **Focus+Context Transition Editor** (see Fig. 4) to alleviate the task of editing transition expressions. The Focus section pertains to the editing task; in this case, the task of writing a state-transition expression in a State-Machine diagram. Sections A, B, C, and D are used to alleviate the challenges of *Recall* and *Modify*. Table I lists all the user's possible contextual actions (i.e., *Recall* and *Modify* activities) related to the editing of a transition expression, and shows which aspects of the editor's interface is designed to tackle each of the contextual actions (see the *Section* column). In the following, we introduce each section of the interface and explain how it can help alleviate the difficulties of *Recall* and *Modify*.

- **Section ContextA:** A major difficulty that a user may face when editing a transition label is knowing about the classes, attributes, and operations declared in the Class diagram (*Rec.1*). It is also important to understand the relationships between the current class (whose State-Machine diagram is currently being edited) and other classes in the Class diagram (*Rec.2*). A modeller is more likely to refer to an attribute or operation of the current class or of an associated class, than to an attribute or operation of an unrelated class. The purpose of *ContextA* section is to display the relationships among the classes, based on the data collected from the Distance-Oriented Object Indexer (Section III). Specifically, modellers are informed about the current class, the classes that have a *Direct Relationship* with the current class (the type of relationship is illustrated by an icon), the classes that have an *Indirect Relationship* (i.e., connecting through another class), and the classes that have *No Relationship* with the current class. This categorisation of the classes allows the modeller to focus on the classes to which they are most

likely need to refer.

Furthermore, a user can create an association between the classes (*Mod.2*), by dragging the class from the list of *Indirect Relationship* or *No Relationship* classes and dropping it into the list of *Direct Relationship* classes. The tool then creates an association between the current class and the dragged class, but the user can always right click on the newly associated class and modify the type of the relationship from association to composition, aggregation, or generalization. Alternatively, the user can create a new class (*Mod.1*) by typing the name of the class in the text field located at the bottom of the section.

- **Section ContextB:** A modeller may continuously refer to the Class diagram to look for the attributes and operations of classes (*Rec.1*) as these elements are the most-likely candidates to be referenced in a transition label. Section *ContextB*, lists the attributes and operations from the class selected from the lists shown in Section *ContextA*. In Fig. 4, *ContextB* lists the attributes and operations of the *Sensor* class selected in *ContextA*. If the intended attribute or operation does not exist in the selected class, the user can simply create them using the interface (*Mod.1*). Moreover, the user can right-click on an attribute or operation to change its type (*Mod.3*).
- **Section ContextC:** To help users overcome the challenges of remembering an intended element, section *ContextC* enables the modeller to look for an element based on what they can remember of the element's name. As the user types into the search text field, the Distance-Based Model Element Finder (Section III) displays in real-time the list of elements whose names match exactly to the searched word followed by the model elements whose names most-closely match to the typed name (*Mod.1*).
- **Section ContextD:** The results of prior user studies suggest that users often look for a similar transition to reuse or learn from when editing another transition's expression. This, however, can be cumbersome as it may not be easy to identify a specific transition in a large and complex model. In Section *ContextD*, we allow modellers to search among the State-Machine diagrams in the model for other transitions by their name or elements in their labels (*Rec.3*), and reuse their event, guard, or action.
- **Section Focus:** This section provides aids that improve the modeller's *Focus* on the current editing task. It divides the task into three text fields which correspond to the event, guard, and action segments of the transition expression that is being edited; and it supports two different ways of setting these segments: *Feature-Rich Text Fields* and *Drag-n-Drop*.

  Feature-Rich Text Fields provide various capabilities to the users such as syntax-highlighting, displaying errors and warnings, and Content-Assist that together nudge the modeller towards a correct model and warn the modeller when they violate a consistency rule. The Content-Assist is user-activated and uses *Type-Based and Distance-Oriented Ranking* module to list related model elements in order of hypothesized relevance.

  Drag-n-Drop: In addition to writing in the text fields to set a transition's event, guard, or action, modellers can drag any element from the list of attributes and operations (*ContextB*) and drop it to the corresponding event. guard or action text fields. The tool will then automatically set the event, guard, or action expressions for the user.

We hypothesize that: *Providing editing facilitators for the **Recall** and **Modify** activities can reduce the efforts related to the **Focus** (i.e., editing a transition).* We derive two research questions from the hypothesis:

- *RQ1: Does using our Focus+Context Transition Editor improve the effectiveness (i.e., task success) and efficiency (i.e., time on tasks and number of diagram views) of users when developing transitions versus using other editors?*
- *RQ2: How well does our Focus+Context Transition Editor alleviate Context-related challenges?*

V. User-Interactive Consistency Management

A typical large-scale system can be composed of hundreds of classes and corresponding State-Machine diagrams [6] [15]. Maintaining consistency in such large-scale systems requires a lot of effort due to the number of inter-related elements among the different diagrams. In our opinion, for a consistency management approach to be successful, it should take the following human-centric concerns into the account: Recognizing the sources of inconsistency, alerting the user of an inconsistency without being intrusive, ensuring that the user can easily access to all relevant information needed to resolve the inconsistency (Context), and supporting the users in their plans to resolve an inconsistency.

Recognition: The first step to resolve an inconsistency is to recognize when a change to the model (i.e., editing the Focus) has introduced an inconsistency across the model. The recognition process can be influenced by two memory-related issues: 1) *Transience:* a measure of how easily information can be retrieved from memory and the degree to which memory deteriorates over time [16]; and 2) *Absentmindedness*: lapses in paying attention [16]. The process of recognizing an occurred inconsistency can fail because the modeller decides to resolve the inconsistency at a later time (and then forgets), or he/she does not even notice the inconsistency. To avoid failure to address inconsistencies, our tool attempts to locate and resolve inconsistencies while they are being made.

We are convinced that it is easier and faster to recognize and correct errors while they are being made because the error-inducing part of the model is still fresh in the modeller's mind. Our editor includes three error detection and resolution strategies (Auto-Fix, Quick-Fix, and Interactive-Fix) that attract the user's attention to a new error. Auto-Fix and Quick-Fix approaches were described in Section III. Our Interactive-Fix interface, shown in Fig. 5, pops up a dialogue box during model-editing whenever a detected error is complex enough to require greater user input.

Intrusiveness: It is often argued that resolving inconsistencies on-the-fly during modelling is disruptive and counter-

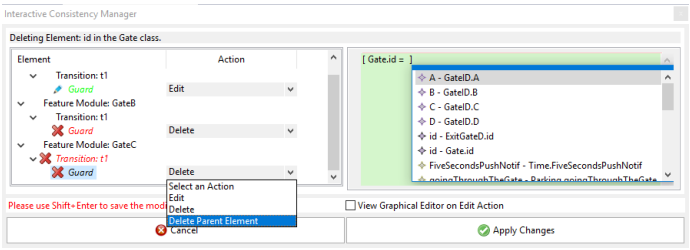

Fig. 5: Our Interactive-Fix Interface

productive to users because of the memory overload imposed by the context switching among the different diagrams to locate an error [17] [18]; and because the interruption distracts the user from their original train of thought [19]–[22]. In contrast, we argue that recovering errors in real-time need not adversely affect a user's performance and satisfaction. We devise a model editor that mitigates the disruptive impacts of interruptions while enabling on-the-fly error resolution.

Make the Invisible Visible: One of the cognitive factors involved in the process of maintaining model consistency is *Association*, which refers to cross-linking a piece of information to other related pieces of information. That is, the ability to remember and list the cross-linked and out-of-sight (or out-of-mind) model elements that may become inconsistent as a result of an edit.

To promote the *Association* (cognitive) ability, our User-Interactive-Fix interface (see Fig. 5) provides a hierarchical view of all of the inconsistent elements in the model that are affected by an original edit. The root node in the hierarchy is the diagram that contains the erroneous element(s). For example, Fig. 5 shows that the *id* attribute of the *Gate* class has been deleted from the Class diagram, which introduces errors in different guard expressions in the model. Accordingly, the root nodes are the containing State-Machine diagrams with child nodes representing the parent transitions of the guards which themselves have child nodes corresponding to the erroneous guards (e.g., guard expression of transitions *t1* in the *FeatureModule: GateB* diagram).

Action Planning: An important step in problem-solving and decision-making is to choose from the possible alternative actions. Most editors do not assist the modeller in fixing an inconsistency when it occurs; a few editors suggest that the modeller undo the edit or delete the inconsistent element. Our tool allows the modeller to edit or delete (using an **Actions-To-Do** menu) the inconsistent element, or its parent element. For example, if a newly inconsistent element is a guard, then the user can select to edit the guard expression (using the embedded textual editor) or delete it, or just delete the parent transition. Note that, the edit action is not limited to the guard expression: the user can easily access and edit the event or actions of the parent transition, or other ancestor elements.

Context: Recalling the *Context* [23] is another critical cognitive factor influencing the process of resolving inconsistencies. It refers to remembering and integrating the essential pieces of information that help the user to resolve an inconsistency. These information may be spread into other diagrams than the *Focus* diagram; thus requiring the user to switch amongst different diagrams, which can be cumbersome.

To support the Context ability, our Interactive-Fix interface embeds a *feature-rich textual editor* which presents the textual representation of a selected inconsistent element and allows the modeller to edit the element. The textual editor uses our *Type-Based and Distance-Oriented Content-Assist* module to facilitate the task of editing by prioritizing and proposing relevant model elements. Alternatively, our tool provides a graphical editor for viewing and editing the containing diagram.

In summary, we hypothesize that: *Our User-Interactive Consistency Management interface improves the users' effectiveness in creating more correct models and reduces users' feelings of being disrupted when fixing an inconsistency, by taking the involving human-cognition factors.* Based on this hypothesis, we investigate the following research questions:

- *RQ3: Do modellers who use our Interactive Consistency Management interface create more correct models than modellers who use other existing editors?*
- *RQ4: Do modellers who use our Interactive Consistency Management interface judge the interface as being too intrusive and disruptive to them when working with the editor versus the level of correctness that it provides?*

VI. Empirical Evaluation

We conducted two user studies[2] (Context study followed by Debugging study) to evaluate the effectiveness of our tooling advancements. The Context user study aimed to assess the effectiveness of our Focus+Context Transition Editor as an example of our Focus+Context editors, in alleviating the challenges in remembering modelling Context; whereas the Debugging study was aimed to evaluate our User-Interactive Consistency Management interface in reducing the efforts of Debugging. We designed the experiments according to the guidelines from Tullis and Albert [24] and Pietron et al. [25], such as number of subjects and data-collection techniques.

A. Experimental Design

Task Design: We designed nine tasks for the Context user study and eight tasks for the Debugging user study. Each participant was given structured descriptions of the tasks and was asked to perform the tasks accordingly. The Context-related tasks included editing the Class and State-Machine diagrams of a Parking Lot system. For example:
Open the State Machine for Gate D. Find the transition that is labelled as Tran3. Set the Event, Guard, or Action for Tran3 based on the following description.

- *Event: After five seconds (recall that the time object notifies the system every five seconds).*
- *Guard: The gate's sensor does NOT sense blockage.*
- *Action: The gate should become closed; that is, the gate's position should be set to down.*

The Debugging-related tasks involved editing parts of the model and resolving any inconsistencies that may occur as a

[2]More information regarding the study materials can be found at: https://github.com/ppourali/UserStudy-on-Tooling-Advances.git

result of that edit. For example:
The Sensor class has a Blockage attribute with the type of int. Change the Blockage attribute's type from int to bool, and fix any inconsistencies that edit may cause.

Recruitment: We recruited participants by sending an email message to CS and SE students at our institution who were expected to have sufficient knowledge of UML tools as well as UML Class and State-Machine diagrams. The following two steps were taken to ensure that our subjects are as representative as possible of the target population of UML modellers: 1) We asked prospective subjects about their competency in creating UML Class and State-Machine diagrams; 2) We asked subjects to answer 10 UML-specific questions which we incorporated in our recruitment letter. During six months of advertisement, we recruited 18 eligible subjects. All the 18 subjects participated in both Context and Debugging studies declared to have experience with modelling tools.

Application Domain: We chose a fairly simple Gated Parking Lot system as the application domain in order to mitigate the effects of domain knowledge on the participants' performance. Moreover, the participants were given the Class diagram of the system in advance, and were asked to study a textual description of the Parking Lot domain as well as the classes and their properties (i.e., attributes and operations) to become familiar with the system before the studies began. They could always refer back to the description if needed. Also, they had access to the Class diagram in their tool.

Treatment Allocation: We employed a randomized Between-Subjects strategy to assign tools to participants. Each participant was randomly assigned to work with either our tool or the modelling tool of their own choice. We decided to allow subjects choose their favorite modelling tool in part because there is no best competing tool (e.g., while one tool has features to help users write guard expressions, another tool might be better at showing and highlighting errors); and in part to mitigate against the threat that a subject in the control group performed poorly due to the unfamiliarity with the tool.

All the subjects went through a 10-minutes "warm-up" phase before starting the tasks, in which the researcher and the subjects walked through the tool (being our tool or the tool of their choice) and practiced developing an example transition. This helped subjects recall the relevant tool's features and become ready to use the tool. Table II shows the distribution of tools used by subjects. Tool determination made in Context study applied also to the Debugging study.

TABLE II: Number of participants per tool

Our tool	Capella	VisualParadigm	MagicDraw	Papyrus
9	3	2	2	2

Data Collection and Metrics: We collected several metrics to answer our research questions about the subjects' effectiveness and efficiency in using the tools: *1) success score, 2) number of errors, 3) time spent on each task, 4) lostness score, and 5) self-reported metrics.*

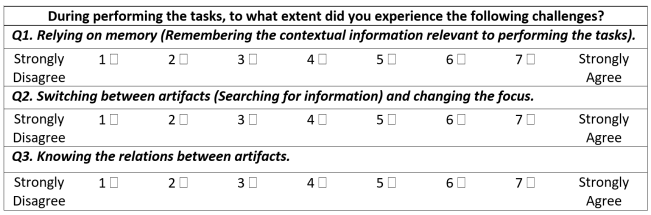

Fig. 6: Post-Session rating of experienced Context-related challenges

1) Success Score: A subject earned a success score of 1.0 when they completed a tasks successfully on their own, a score of 0.5 (partial success) when they sought help (e.g., *'What was the class's name?'*) and subsequently completed the task successfully, and a score of 0.0 when they failed to meet the goal of the task.
2) Number of Errors: We counted the number of errors that each participant made per task. Specifically, we counted errors that fall into the error taxonomy proposed by Lange et al. [15] (i.e, various *well-formedness* and *consistency* types of error).
3) Time Spent on Each Task: We measured the time that each subject took to perform each task, as a measure of the subjects efficiency in using a tool.
4) Lostness Score: As a second measure of the subjects' efficiency with the tools, we computed a *Lostness Score* (ranging from *Zero* to *One*), which represents how *'lost'* a subject was when performing a particular task. Lostness (L) [24] is computed using three inputs:
   N = The number of different (unique) diagrams viewed when performing a task,
   S = The total number of diagrams opened during the task (included repeated openings), and
   R = The optimum number of diagrams that should have been viewed to fulfill the task.
   A Lostness Score is calculated as:
   $$L = \sqrt{(\frac{N}{S} - 1)^2 + (\frac{R}{N} - 1)^2} \quad (1)$$
   The higher the Lostness score, the more often the subject viewed unnecessary diagrams during a task. According to Smith [26], a Lostness Score of greater than 0.5 is deemed to be high.
5) Self-Reported Metrics: We asked the subjects about their opinion of their anticipated and (perceived) actual performance with their respective modelling tool. Specifically, we asked the subjects to rate the following experiences based on the Likert scale ranging from 1 (Strongly Disagree) to 7 (Strongly Agree) with a neutral value of 4:
   - Context Challenge Rating: At the end of each session of the Context study, we asked the subjects to what extent they experienced Context-related challenges when performing tasks(See Fig. 6).
   - Intrusiveness Rating: To measure how intrusive our Consistency Management interface is, we asked the subjects who used our tool, before the Debugging sessions, to rank on a Likert scale (from 1=Strongly

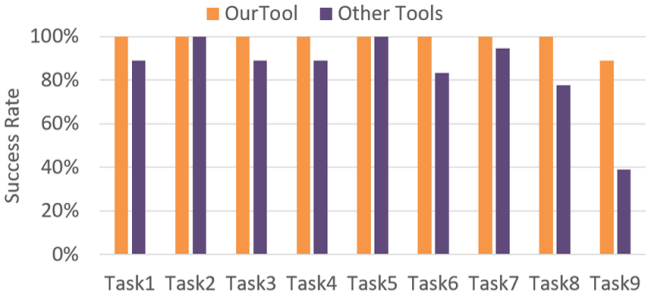

Fig. 7: Context Study: Average success rate per task for users of our editor vs. other editors

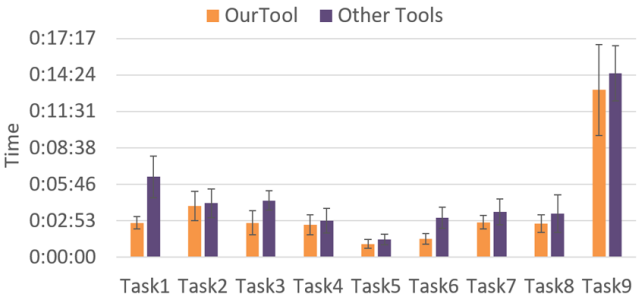

Fig. 8: Context Study: Average time per task for our editor vs. other editors

Disagree to 7=Strongly Agree) the extent to which they agreed that maintaining model consistency at all times is counter-productive (disrupting the modelling tasks). At the end of the Debugging study, we asked subjects to rank the extent to which they felt that the interruptions by the Consistency Management editor were disruptive.

B. Results of the Context User Study

For each of 18 subjects, we conducted one session lasting 60 to 90 minutes, during which our Focus+Context Transition Editor was evaluated for its efficiency and effectiveness.

RQ1: Does using our Focus+Context Transition Editor improve the effectiveness (i.e., task success) and efficiency (i.e., time on tasks and number of diagram views) of users when developing transitions versus using other editors?

Our results indicate that subjects' effectiveness and efficiency increased significantly when using our editor.

Task Success: We added the subjects' success scores and computed a single score (referred to as *General Success*) for the subjects. The average *General Success* ratio for all the subjects, shown in Fig. 7, illustrates a raise in the success rate of the users of our editors versus other editors for all the tasks.

Time on Task: Fig. 8 shows the mean time on each task for our editor versus other editors. For a more meaningful result, we included the average time only for the successful tasks. As shown, the average time per task is shorter for the users of our editor compared to the users of the other editors.

Lostness: Fig. 9 shows the average of the subjects' Lostness scores broken down by task. The participants who used other editors were *"lost"* in all of the tasks, whereas the users of

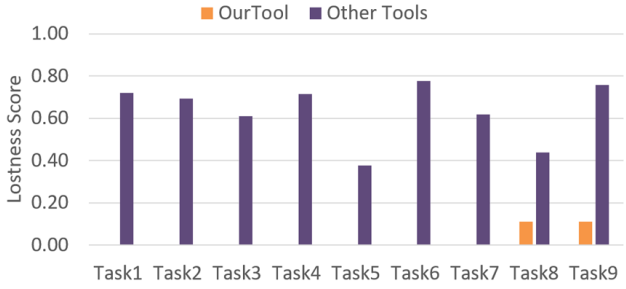

Fig. 9: Context Study: Average Lostness score per task for users of our editor vs. other editors

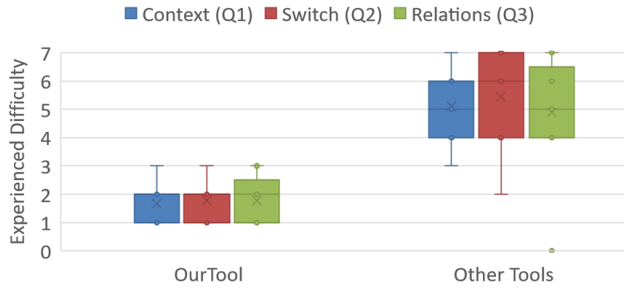

Fig. 10: Context Study: Context-related experienced challenges experienced on average by subjects who used editor vs. other editors

our editor showed almost no sign of being lost (except for the tasks 8 and 9).

These results strongly suggest that modellers benefit from a sliced view (from related diagrams) of the context of their modelling task.

RQ2: How does our focus+context transition editor perform in alleviating the Context-related challenges?

We used self-reported metrics to assess the Context-related challenges that the subjects faced when using the model editors. As mentioned, we used three questions to measure the extent to which each subject experienced Context-related challenges (see Fig. 6). Fig. 10 shows the average of the subjects' answers to each question for our editor versus their answers for other editors. As can be seen, Context-related challenges were deemed to be experienced more than twice as much in other editors than in our editor. This significant gap suggests that Focus+Context editors can play a critical role in alleviating users' cognitive load- especially in reducing the challenges of remembering and editing contextual information.

C. Results of the Debugging User Study

For each of 18 subjects, we conducted one session lasting 60-90 minutes, during which we evaluated the effectiveness of our User-Interactive Consistency Manager in reducing the number of errors in a model and mitigating the disruptive impacts of real-time error resolution.

RQ3: Do modellers who use our Interactive Consistency Management interface create more correct models than modellers who use other existing editors?

The main research question in developing the User-Interactive Consistency Management was to assess its effectiveness in reducing the number of errors. For this purpose, the task success scores and the number of errors per tasks were

collected and the results of subjects using our editor were compared against the results of subjects using other editors.

Task Success: As shown in Fig. 11, participants who used our editor were more successful in locating and resolving inconsistencies than the participants who used other editors.

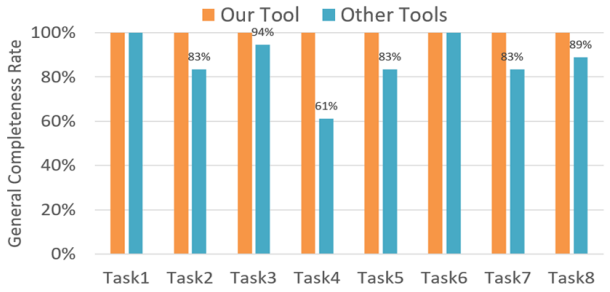

Fig. 11: Debugging Study: Percentage of tasks that were successfully completed by subjects who used our editor vs. other editors

Errors on Tasks: As shown in Fig. 12, subjects who used our editor made on average no errors in all tasks, which is a significant improvement over the performance of the subjects who used other tools.

Time on Tasks: It is not enough to be effective, if users are not also efficient. As shown in Fig. 13, subjects who used our editor completed all tasks, except Task5, in less time on average than the subjects who used other editors. After reviewing the recorded video and audio files of the sessions, we noticed that the time on Task5 was lengthened mostly because the users found it necessary to pause and point out a feedback on how to improve the scalability of our interface.

RQ4: Do modellers who use our Interactive Consistency Management interface judge the interface as being too intrusive and disruptive to them when working with the editor versus the level of correctness that it provides?

One of the main hypotheses underlying the design of our User-Interactive Consistency Management interface is that managing consistencies in real-time while editing a model does not have to be intrusive to the modeller (in contrast to what researchers believe) if tool developers take human cognitive factors into account when designing interfaces. For the users of our editor, the result of their self-reported Expected Disruption asked Pre-Session and their self-reported Experienced Disruption asked Post-Session is depicted in the box plot shown in Fig. 14. It shows that the subjects strongly

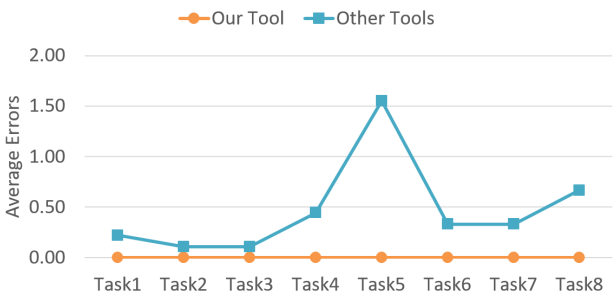

Fig. 12: Debugging Study: Average number of errors per task for user of our editor vs. other editors

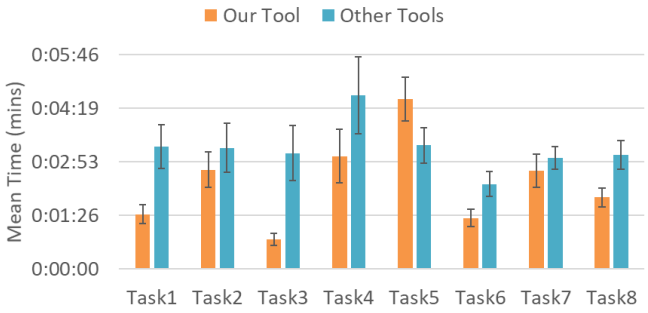

Fig. 13: Debugging Study: Average time per task for subjects using our editor vs. other editors

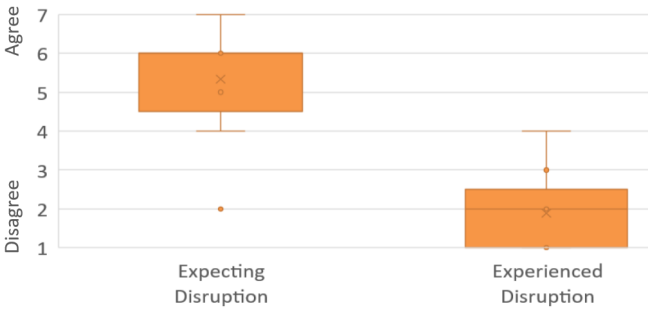

Fig. 14: Debugging Study: Subjects' assessment of the intrusiveness of our Interactive Consistency Manager

expected that maintaining model consistency at all times would be intrusive. However, after using our editor, their experience ratings showed the opposite. That is, their mean experience rating indicates that they barely felt disrupted when asked by our editor to debug inconsistencies during model editing.

VII. Threats to the Validity

The main threat to the validity is the participants' competency with the UML and UML editors. We tried to mitigate against this threat by asking the subjects to self-declare their familiarity with the UML and editors. In addition, we embedded a UML exercise in our recruitment questionnaire and only recruited those subjects who could pass the exercise. One might argue that recruiting participants during a six-month period may result in variability in their experience with modelling tools. However, we applied the same screening procedure for all participants and the participants were not actively learning about tools during this time frame. We do not believe that the recruiting period introduced variability in participants' background.

Another threat to external validity is the participants' familiarity with the application domain and the system's complexity. To cope with this threat, we chose a simple and familiar application domain. In addition, we designed the task to be simple enough so that the subjects could perform their tasks without a complete understanding of the system. Moreover, we designed Task9 in the Context study and Tasks 4 and 5 in the Debugging study to be more difficult than other tasks, so that we can observe how the users' performance would be affected in case of more difficult tasks. The results show that there is still an improvement for the subjects who used our tool

versus the subjects who used other tools. However, the size of the model in our studies is not comparable to models of large-scale complex systems (e.g., see [15]), and it is possible that the performance improvements that the subjects demonstrated in the studies would not scale to much larger models.

The relatively small number of subjects may be another threat to the validity, but the number is recognized as adequate by various sources [24] [27].

VIII. Related Work

Many artefact-centric techniques are proposed to improve the usefulness of model editors. However, little work has been done on applying human-factors theory to enhance the usability of these editors. We are convinced that the state-of-the-art still lacks a systematic consideration of humanr-centric solutions when designing modelling tools [28]. As far as this paper is concerned, the below related work are divided into the solutions that alleviate the *Context* and *Debugging* challenges.

A few features in model editors [29]–[34] reduce (directly or indirectly) the *Context*-related challenges of editing UML models by employing different usability techniques. For example, MagicDraw [31], VisualParadigm [35], and ArgoUML [33] are industrial modelling tools that help reduce the efforts of modelling by offering several tooling features such as well-designed UIs, navigability and zooming features, wizards, and search capabilities. These tools however are more concerned with the general usability of their editors rather than targeting specific challenges that users experience. Recently, a few Eclipse-based model editors (e.g., Capella [34], Papyrus [30], Yakindu [32]) have augmented their capabilities to offer more content-specific features such as Content-Assist. Although Content-Assist alleviates some *Context*-related challenges, their approach still suffers from information overload and the lack of *filtering* and *ranking* algorithms that can effectively reduce the amount of contextual information that the modeller typically reads before finding what they are looking for.

Task-Oriented Interface [6] [36]–[38] can be seen as a type of Focus+Context [4] solution that alleviates the users' load by enabling the user to view the contextual information that are relevant to the Focus task. For example, FlexView [37] provides features that enables the user to easily divide the screen into different regions in order to view different requirements artefacts. However, their tool does not aid users in recognizing the contextual relationships, as ours does.

Code and model completion techniques [39]–[43] are also proposed in the literature. While these techniques, in general, can be employed in different domains, tools and stages of coding or modelling, they mostly differ in their approach to propose more exact and useful completions. For instance, Steimann and Ulke [41] look for ways to complete the specification of an incomplete model element based on the meta-model's well-formedness rules. We, however, employ Content-Assist to help modellers make their next step (e.g., setting a property value or changing it) based on more than just the well-formedness rules (e.g., semantic rules).

With respect to the model *Debugging* challenge, the literature is rich when it comes to managing consistency in UML models [17] [44] [45] [46] [47] [48]. Many approaches are Proactive (Consistent-by-Construction) in that they ensure that a model is consistent and correct at any point in time [17] [48]. However, most proactive consistency-management approaches barely take into account human factors, and as a result, their implementations are deemed intrusive to the user [18], leading many researchers believe that "maintaining consistency at all time is counterproductive" [49]. In our work, we have taken a human cognitive-based approach that aims to provide modellers with an easy-to-use supportive (e.g., Content-Assist) interface that helps modellers resolve inconsistencies in real-time, while their intent is still fresh in their mind.

The above approaches provide some tooling features that improve understandability and navigability of the artefacts; but without any consideration of the semantics of the artefacts or the relations among them [38]. Moreover, most modelling editors have not been evaluated with respect to whether they enhance users' effectiveness in editing and debugging models. Consequently, there is a chasm between what users expect in the way of tool support and what the tools actually provide [50]. In contrast, our work focuses on 1) taking into account users' cognitive factors to improve their interactions with model editors to edit and debug Class and State-Machine diagrams, and 2) assessing the effectiveness of our proposed techniques on human users and gaining feedback.

IX. Conclusion

Research has shown that tools are amongst the top barriers to adopt MDE in industry [1] [2]. UML modellers face unnecessary burdens to the modelling tasks, leading practitioners to be reluctant to employ MDE [2]. The main reason is that tool vendors do not take into account human-cognition factors when designing their tools.

To address challenges of using UML editors, we identified the most-relevant human-cognition factors and devised tool advancements based on the factors. More specifically, we employed two Focus+Context user interfaces that effectively reduce efforts of *Context* and *Debugging* challenges when designing UML Class and State-Machine diagrams. We subsequently conducted an empirical user study to assess the effectiveness of our user interfaces on human users.

The results of our studies indicate that employing a Focus+Context approach can significantly improve users' satisfaction of using model editors and can increase their effectiveness and efficiency in editing and debugging models. Our results have implications for tool vendors to enhance and improve the quality of UML model editors, which is crucial for a greater adoption of MDE by industry.

References

[1] R. Jolak, T. Ho-Quang, M. R. Chaudron, and R. R. Schiffelers, "Model-based software engineering: A multiple-case study on challenges and development efforts," in *Proceedings of the 21th ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems*. ACM, 2018, pp. 213–223.

[2] G. Mussbacher, D. Amyot, R. Breu, J.-M. Bruel, B. H. Cheng, P. Collet, B. Combemale, R. B. France, R. Heldal, J. Hill *et al.*, "The relevance of model-driven engineering thirty years from now," pp. 183–200, 2014.

[3] P. Pourali and J. M. Atlee, "An empirical investigation to understand the difficulties and challenges of software modellers when using modelling tools," in *Proceedings of the 21th ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems*. ACM, 2018, pp. 224–234.

[4] A. Cockburn, A. Karlson, and B. B. Bederson, "A review of overview+ detail, zooming, and focus+ context interfaces," *ACM Computing Surveys (CSUR)*, vol. 41, no. 1, p. 2, 2009.

[5] P. Ghazi and M. Glinz, "An exploratory study on user interaction challenges when handling interconnected requirements artifacts of various sizes," in *24th International Requirements Engineering Conference (RE)*. IEEE, 2016, pp. 76–85.

[6] H. Kagdi and J. I. Maletic, "Onion graphs for focus+ context views of uml class diagrams," in *4th International Workshop on Visualizing Software for Understanding and Analysis*. IEEE, 2007, pp. 80–87.

[7] S. Berlik, *Eclipse Modeling Framework*. Addison-Wesley, 2007.

[8] M. Herrmannsdoerfer, D. Ratiu, and G. Wachsmuth, "Language evolution in practice: The history of gmf," in *Int. Conf. on Software Language Engineering*. Springer, 2009, pp. 3–22.

[9] S. Efftinge and M. Völter, "oaw xtext: A framework for textual dsls," in *Workshop on Modeling Symposium at Eclipse Summit*, vol. 32, 2006, p. 118.

[10] M. Scheidgen, "Integrating Content Assist into Textual Modelling Editors," *Modellierung 2008, 12.-14. März 2008, Berlin*, vol. 127, pp. 121–131, 2008.

[11] M. Press, *Microsoft Visual InterDev 6.0 Programmer's Guide*. Microsoft Press, 1998.

[12] T. Pati, S. Kolli, and J. H. Hill, "Proactive modeling: a new model intelligence technique," *Software & Systems Modeling*, vol. 16, no. 2, pp. 499–521, 2017.

[13] V. Levenshtein, "Binary codes capable of correcting spurious insertions and deletions of ones," *Problems of Information Transmission*, vol. 1, pp. 8–17, 1965.

[14] C. F. J. Lange and M. R. V. Chaudron, "Effects of Defects in UML Models: An Experimental Investigation," in *Proceedings of the 28th Int. Conf. on Software Engineering*, 2006, pp. 401–411.

[15] C. F. J. Lange, M. R. V. Chaudron, and J. Muskens, "In practice: Uml software architecture and design description," *IEEE software*, vol. 23, no. 2, pp. 40–46, 2006.

[16] D. Schacter, D. T. Gilbert, and D. M. Wegner, *Psychology (2nd Edition)*. New York: Worth, 2011.

[17] A. Egyed, "Instant consistency checking for the uml," in *Proceedings of the 28th Int. Conf. on Software engineering*. ACM, 2006, pp. 381–390.

[18] I. Hadar and A. Zamansky, "Cognitive factors in inconsistency management," in *23rd Int. Conf. on Requirements Engineering(RE'15)*. IEEE, 2015, pp. 226–229.

[19] P. D. Adamczyk and B. P. Bailey, "If not now, when?: the effects of interruption at different moments within task execution," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2004, pp. 271–278.

[20] A. Baethge and T. Rigotti, "Interruptions to workflow: Their relationship with irritation and satisfaction with performance, and the mediating roles of time pressure and mental demands," *Work & Stress*, vol. 27, no. 1, pp. 43–63, 2013.

[21] B. P. Bailey and J. A. Konstan, "On the need for attention-aware systems: Measuring effects of interruption on task performance, error rate, and affective state," *Computers in human behavior*, vol. 22, no. 4, pp. 685–708, 2006.

[22] D. A. Boehm-Davis and R. Remington, "Reducing the disruptive effects of interruption: A cognitive framework for analysing the costs and benefits of intervention strategies," *Accident Analysis & Prevention*, vol. 41, no. 5, pp. 1124–1129, 2009.

[23] R. Budiu, "Memory recognition and recall in user interfaces," *Nielsen Norman Group*, 2014.

[24] W. Albert and T. Tullis, *Measuring the user experience: collecting, analyzing, and presenting usability metrics*. Newnes, 2013.

[25] J. Pietron, A. Raschke, M. Stegmaier, M. Tichy, and E. Rukzio, "A Study Design Template for Identifying Usability Issues in Graphical Modeling Tools," Tech. Rep.

[26] P. A. Smith, "Towards a practical measure of hypertext usability," *Interacting with computers*, vol. 8, no. 4, pp. 365–381, 1996.

[27] J. Sauro and J. R. Lewis, *Quantifying the user experience: Practical statistics for user research*. Morgan Kaufmann, 2016.

[28] S. Abrahão, F. Bourdeleau, B. Cheng, S. Kokaly, R. Paige, H. Stöerrle, and J. Whittle, "User experience for model-driven engineering: Challenges and future directions," in *20th Int. Conf. on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2017, pp. 229–236.

[29] M. a. Garzon, H. Aljamaan, and T. C. Lethbridge, "Umple: A framework for Model Driven Development of Object-Oriented Systems," in *2015 IEEE 22nd Int. Conf. on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 494–498.

[30] S. Gérard, C. Dumoulin, P. Tessier, and B. Selic, "Papyrus: A UML2 tool for domain-specific language modeling," in *Proceedings of the International Dagstuhl Conference on Model-Based Engineering of Embedded Real-Time Systems (MBEERTS'07)*, 2007, pp. 361–368.

[31] N. M. Inc, "Magicdraw, uml," 2013.

[32] A. Muelder, "Yakindu Statechart Modeling Tools," 2011.

[33] J. E. Robbins and D. F. Redmiles, "Cognitive support, uml adherence, and xmi interchange in argo/uml," *Information and Software Technology*, vol. 42, no. 2, pp. 79–89, 2000.

[34] P. Roques, "MBSE with the ARCADIA Method and the Capella Tool," in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, Toulouse, France, Jan. 2016.

[35] V. Paradigm, "Visual paradigm for uml," *Visual Paradigm for UML-UML tool for software application development*, p. 72, 2013.

[36] M. Glinz, S. Berner, and S. Joos, "Object-oriented modeling with adora," vol. 27, no. 6. Elsevier, 2002, pp. 425–444.

[37] P. Ghazi, N. Seyff, and M. Glinz, "FlexiView: A Magnet-Based Approach for Visualizing Requirements Artifacts," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*, ser. Lecture Notes in Computer Science (LNCS), 2015, vol. 9013, pp. 262–269.

[38] M. Kersten, "Focusing knowledge work with task context." Ph.D. dissertation, University of British Columbia, 2007.

[39] S. Das and C. Shah, "Contextual code completion using machine learning," 2015.

[40] A. Dyck, A. Ganser, and H. Lichter, "Model recommenders for command-enabled editors," in *International Workshop on Model-driven Engineering By Example (MDEBE)*, 2013, pp. 12–21.

[41] F. Steimann and B. Ulke, "Generic model assist," in *Int. Conf. on Model Driven Engineering Languages and Systems (MODELS)*, ser. Lecture Notes in Computer Science (LNCS), A. Moreira, B. Schätz, J. Gray, A. Vallecillo, and P. Clarke, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, vol. 8107, pp. 18–34.

[42] T. Pati, D. C. Feiock, and J. H. Hill, "Proactive modeling: autogenerating models from their semantics and constraints," in *Proceedings of the 2012 workshop on Domain-specific modeling*. ACM, 2012, pp. 7–12.

[43] V. Raychev, M. Vechev, and E. Yahav, "Code completion with statistical language models," *ACM SIGPLAN Notices*, vol. 49, no. 6, pp. 419–428, 2014.

[44] A. A. Alshazly, A. M. Elfatatry, and M. S. Abougabal, "Detecting defects in software requirements specification," *Alexandria Engineering Journal*, vol. 53, no. 3, pp. 513–527, 2014.

[45] X. Blanc, I. Mounier, A. Mougenot, and T. Mens, "Detecting model inconsistency through operation-based model construction," in *30th Int. Conf. on Software Engineering*. IEEE, 2008, pp. 511–520.

[46] C. Nentwich, W. Emmerich, A. Finkelstein, and E. Ellmer, "Flexible consistency checking," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 12, no. 1, pp. 28–63, jan 2003.

[47] M. Sabetzadeh, S. Nejati, S. Liaskos, S. Easterbrook, and M. Chechik, "Consistency checking of conceptual models via model merging," in *15th IEEE International Requirements Engineering Conference (RE 2007)*, Oct 2007, pp. 221–230.

[48] M. Snoeck, C. Michiels, and G. Dedene, "Consistency by construction: the case of merode," in *Int. Conf. on Conceptual Modeling*. Springer, 2003, pp. 105–117.

[49] B. Nuseibeh, S. Easterbrook, and A. Russo, "Making inconsistency respectable in software development," *Journal of Systems and Software*, vol. 58, no. 2, pp. 171–180, 2001.

[50] S. Lahtinen and J. Peltonen, "Adding speech recognition support to uml tools," *Journal of Visual Languages & Computing*, vol. 16, no. 1, pp. 85 – 118, 2005, 2003 IEEE Symposium on Human Centric Computing Languages and Environments.