User Interfaces Flow: Modeling practical cases using User Interface Transition Diagrams

Flujo entre Interfaces de Usuario: Modelado de casos prácticos utilizando Diagramas de Transiciones entre de Interfaces de Usuario

Jorge Cervantes-Ojeda y María del Carmen Gómez-Fuentes*

Departamento de Matemáticas Aplicadas y Sistemas
Universidad Autónoma Metropolitana, Unidad Cuajimalpa
Ciudad de México, México
*mgomez@cua.uam.mx

KEYWORDS:

User Interfaces flow specification, UITD, Requirements elicitation

ABSTRACT

Here, we explore the application of the User Interface Transition Diagram (UITD) for modeling the flow between user interfaces in interactive software systems. While UITDs are known tools in requirements elicitation, our work addresses the gap in showing their effectiveness in modeling three common user interaction scenarios that are frequently encountered in practice: (i) actions repeated across multiple interfaces, (ii) actions available only to users with extended privileges, and (iii) dynamically enabled or disabled buttons based on conditional logic. These solutions not only make it easier for customers to understand the model with the user actions and system responses but also facilitate a smooth transition from requirements specification to design phases by precisely defining user interactions. Despite the uniqueness of each software system, many share common traits that can be effectively modeled using UITDs. By offering generic solutions for these scenarios, this work aims to enhance the modeling capabilities of UITDs, promoting their broader adoption in the software industry.

PALABRAS CLAVE:

Especificación del flujo entre interfaces de usuario, DTIU, Extracción de Requerimientos

RESUMEN

En este artículo, exploramos la aplicación del diagrama de transición de interfaz de usuario (UITD) para modelar el flujo entre interfaces de usuario en sistemas de software interactivos. Si bien los UITD son herramientas conocidas en la obtención de requisitos, nuestro trabajo aborda la brecha en la demostración de su eficacia en el modelado de tres escenarios comunes de interacción de usuario que se encuentran con frecuencia en la práctica: (i) acciones repetidas en múltiples interfaces, (ii) acciones disponibles solo para usuarios con privilegios extendidos y (iii) botones habilitados o deshabilitados dinámicamente según la lógica condicional. Estas soluciones no solo facilitan que los clientes comprendan el modelo con las acciones del usuario y las respuestas del sistema, sino que también facilitan una transición fluida de las fases de especificación de requerimientos a las de diseño, al definir con precisión las interacciones del usuario. A pesar de la singularidad de cada sistema de software, muchos comparten rasgos comunes que se pueden modelar de manera efectiva utilizando UITD. Al ofrecer soluciones genéricas para estos escenarios, este trabajo tiene como objetivo mejorar las capacidades de modelado de los UITD, promoviendo su adopción más amplia en la industria del software.

• **Recibido: 20 de octubre de 2024** • **Aceptado: 15 de abril de 2025** • **Publicado en línea: 1 de octubre de 2025**

DOI: 10.30973/progmat/2025.17.3/1

1. INTRODUCTION

The success of software development projects depends on the ability to meet user needs, a process significantly enhanced by effective communication between developers and customers [1]. However, communication challenges between developers and customers remain a significant issue [2], and clear specifications of system-user interactions and interfaces are crucial for this purpose. The User Interface Transition Diagram (UITD) [3] has emerged as a valuable tool in this domain, offering a simplified modeling language that aids in describing the flow between different user interfaces in a software system. While the basic utility of UITDs in requirements elicitation and design phases is well-documented [4-8], there remains a need to address how UITDs can be adapted to handle specific, recurring scenarios in interactive systems.

This paper aims to bridge this gap by presenting solutions for modeling three common user interaction scenarios that are frequently encountered in practice: (i) user actions repeated across multiple interfaces, (ii) actions available exclusively to users with extended privileges, and (iii) dynamic button visibility based on conditional logic. These scenarios are frequently encountered in the design of interactive software systems but are not comprehensively addressed in existing UITD literature. Our contribution lies in showing how these scenarios can be effectively modeled using UITDs, thereby enhancing the utility and flexibility of this modeling language. Through practical examples and detailed explanations, we show the advantages of our approach.

The structure of this paper is as follows: Section II discusses the motivation and advantages of UITDs. Section III summarizes related work on modeling UI flows in interactive systems. Section IV explains the symbols used in UITDs. Section V describes our solution for modeling user actions repeated across multiple UIs. Section VI explains the modeling of UIs where buttons are conditionally enabled or disabled. Section VII presents our solution for modeling user actions reserved for users with additional privileges. Finally, Section VIII contains the discussion, and Section IX offers the conclusions.

2. MOTIVATION

User Interfaces (UIs) within a software system can be represented using various tools, which are not mutually exclusive. These tools can range from functional software to graphical notations. Among functional software, there are applications like InVisionApp [9] and Sketch [10], designed primarily for designing the visual aspects of UI prototypes and simulating transitions. However, they lack the precision and completeness required to specify all the requirements of transition triggers between UIs. Additionally, there are available graphic notations, which can be informal or formal. Informal options include Storyboards and Navigation Maps [11]. Formal graphical notations like the Interaction Flow Modeling Language (IFML) [12] can be employed for a comprehensive approach. Other formal graphic notations, whose use is not so widespread, include Coloured Petri Nets (CPN) [13], Behaviour Trees (BT) [14] [15], and State Machines (SM) [16].

IFML, endorsed by the Object Management Group (OMG), serves as the standard for fully modeling the abstract descriptions of UIs. However, comprehending a formal modeling language can be challenging, as many are tailored for software developers rather than non-experts. Van der Linden et al. [17] note that simplicity is essential for non-experts when working with modeling notations. It is expected that if a simple modeling language is used, stakeholders will be able to understand it, thus facilitating the recollection of feedback for system design improvement.

The challenges identified in utilizing IFML symbols for front-end interface modeling, as highlighted in a study by Randerson et al. [18], revealed instances of improper utilization of certain IFML elements. So, rather than a language with too many symbols, like IFML, it is desirable to have a modeling language that is technically accurate and accessible to non-experts. Therefore, a formal modeling language like UITD, tailored for defining UI-user interactions, offers essential symbols for clear communication and effective modeling, without overwhelming stakeholders with unnecessary complexity. UITDs also provide unique constructs for modeling UI transitions, such as nested interfaces, which are not available in other languages like IFML. This makes UITD a suitable choice for scenarios requiring nested interfaces without needing additional notation redefinition or extension.

3. RELATED WORKS

According to Ogunyemi et al. [19], few works report advances related to navigation design. Within this type of work, we find that of Reggio et al. [20], who acknowledge the importance of a well-structured approach to requirements elicitation, particularly in defining user interface transitions. One of their proposals is that UI screen mock-ups should be associated with use cases to enhance understanding of functional requirements. The UITD integrates all use cases related to user interaction with the system, effectively modeling the flow of the user interfaces. This makes it a valuable tool during the requirements elicitation phase. Furthermore, [21] demonstrated the utility of UI mockups in understanding functional requirements, and the UITD is precisely intended to be used alongside mockups, as explained in [4]and [5].

While UITDs can be created using general-purpose graphic tools such as draw.io, Lucidchart, etc., the UITD-Editor [22] is specifically designed for drawing up UITDs. It offers specialized functionalities to facilitate the editing of UITD properties. This editor is free and accessible at:

http://148.206.168.145/EditorUITDEnglish/examples/indexF.html

A study in [23] demonstrated that the UITD editor outperforms draw.io in terms of efficiency and accuracy when drawing User Interface Transition Diagrams.

A comparative analysis with modeling languages such as UML, IFML, and others is available in [3] and [6], where the advantages and limitations of the UITD are thoroughly discussed. Examples of systems that have utilized UITDs in their development can be found in [4-8]. This work advances the modeling of interactive systems with UITD by introducing unpublished methods for modeling menus and navigation bars, buttons that are enabled or disabled based on specific conditions, and user interfaces that offer additional actions or privileges to certain users. In the next section, we will explore the concept of UITD and its fundamental components.

4. THE USER INTERFACE TRANSITION DIAGRAM (UITD)

The UITD is a formal modeling notation that expresses the requirements of a system regarding its User Interface (UI) transition triggers from a source UI to a destination UI [3]. It is designed to define completely the user-system interactions and, simultaneously, to be simple enough so that any non-technical stakeholder can understand it [6]. A UITD is a directed graph with nodes representing user interfaces and edges representing transitions between user interfaces. Each user interface has a name and a unique number that allows easy identification during the design and subsequent stages of the project (see Fig. 1).

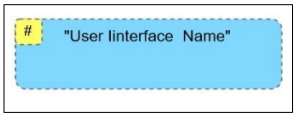


Fig. 1. A User Interface contains two indicators: a number and a name.

Transitions have an origin UI, a destination UI, and a label. Labels in transitions are composed of exactly one mandatory user action on the origin UI and, if necessary, a trigger condition. Fig. 2 depicts transitions from a *source* "User Interface 1" to a *destination* "User Interface 2". The transition label specifies that the user performed a certain action on the source UI, which triggers the display of the destination UI. As seen in Fig. 2, a UI can be both the source and destination of a transition, as is the case with the transition for the user action "save".

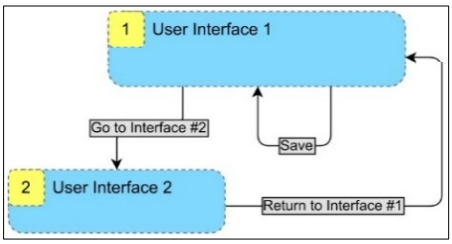


Fig. 2. Transition from a source UI to a destination UI is an arrow with a label.

When two transitions originating from the same UI are triggered by the same user action then, the label includes additional details about the specific conditions needed to activate each transition. So, the *condition* is necessary only when, with a single user action, two or more transitions are triggered from the same UI (see Fig. 3).

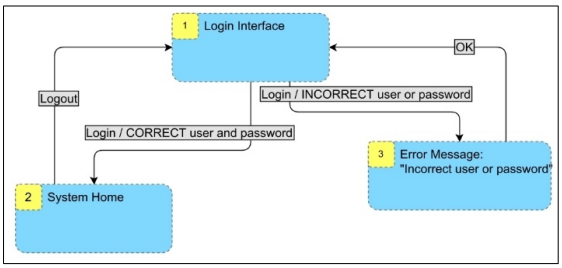


Fig. 3. If two transitions have the same user action, the label contains information about the conditions to trigger the transition.

The UITD graphical symbols are summarized in Table 1. This table is an improvement to the one published in [6].

Table 1. UITD graphical symbols

Symbol	Meaning	UITD Notation
Single UI	A state of the UI displayed to the user	# User Interface Name
Transition	A change in the state of the UI from a user action (and sometimes a condition)	User action and condition →
Nested UI	Transitions with origin in the *contained* interface are also available from the *containing* interface and not vice versa	1 Containing Interface 2 Contained Interface
Continuous and dashed UI	A continuous UI border means that all the possible transitions to/from this UI are visible in the current UITD fragment or sub-diagram.	# Continuous Borders # Dashed borders

Now we briefly explain the two key features when modeling with UITD: "nested UI" and the interfaces with "continuous and dashed borders".

4.1. Nested UI (Containing and Contained UI)

One notable feature of UITD is its capability to depict a UI nested within another UI (see Fig. 4). This implies that all transitions originating from the contained UI are also accessible from the containing UI (but not the other way around). The transition triggers available from the contained UI represent a subset of those available from the containing UI. In other words, the containing UI extends the transitions set of the contained UI with some extra transitions. A containing interface can contain one or more UIs. Next, we define these two types of user interfaces.

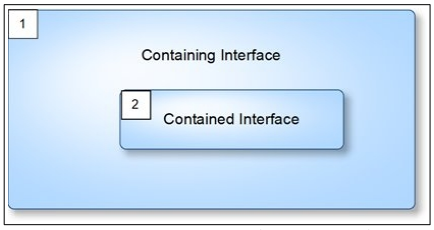


Fig. 4. A Containing and Contained UI.

Containing UI: The containing UI serves as the overarching interface that encapsulates one or more other UIs, known as contained UIs. This means the containing UI provides a broader context or environment within which the contained UIs operate. One key characteristic of a containing UI is that it extends the set of available transitions beyond those provided by the contained UIs. In other words, users interacting with the containing UI have access to all the transitions that the contained UI has, plus additional transitions that are unique to the containing UI. Essentially, the containing UI provides a superset of functionalities, enabling users to choose among the user actions within the nested interfaces and additional options in the containing interface.

Contained UI: The contained UI is a subordinate interface nested within a larger, containing UI. It represents a more focused or specific set of functionalities within the overall system. The transitions available from the contained UI are a subset of those available from the containing UI, meaning that while the contained UI allows certain interactions and navigations, it is limited to a specific scope or context. The user actions in the contained UI are also accessible when interacting with the containing UI. However, the reverse is not true; when a user interacts with the contained UI, he/she does not have access to the broader set of transitions offered by the containing UI.

For example, the UITD in Fig. 5 illustrates the UIs involved in the shopping process of an online store. When the user is on UI #3 "Product List" and he/she is *logged out*, they can select a product from the list to view its details in UI #8 "Product Details"; however, they do not have access to the shopping option. When he/she is *logged in*, they have access to UI #7 "Order Request", where the option to buy the product or add it to the cart is present. In this scenario, UI #7 serves as the ***containing*** interface, while UI #8 functions as the ***contained*** interface. It's crucial to highlight that the user actions found in UI #8 are also accessible within UI #7, in this case, the option to return to UI #3. The remaining sections will explain the parts of the UITD that depict the online store system.

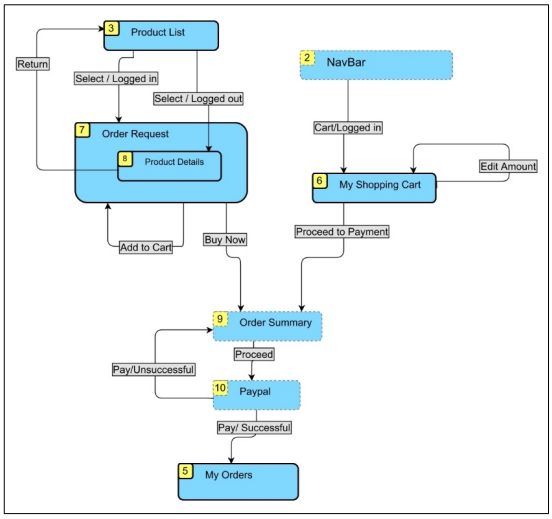


Fig. 5. UITD for a shopping process.

4.2. Continuous/dashed border User Interfaces

The purpose of the continuous/dashed border is to divide the UITD into multiple fragments. When a specific UI has its border continuous, it indicates completeness. This implies that all transitions connected to/from this UI are documented within the fragment. Conversely, a UI with a dashed border means that not all associated transitions are included within the fragment. Therefore, to review all transitions, another fragment must be consulted. A UI may appear in multiple fragments, and it is recommended that all UIs appear at least once with a continuous border in some fragments. Below, we define the

meaning of continuous/dashed borders in the UITD.

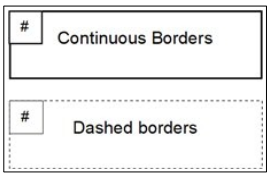


Fig. 6. Continuous and dashed borders.

Continuous Borders: A continuous border around a UI in a User Interface Transition Diagram (UITD) signifies that the representation of this UI within the current fragment is complete. This means all transitions coming to and from this UI are fully documented within the same diagram fragment. When a UI has a continuous border, users can be confident they are viewing a comprehensive view of the UI's possible interactions, as all relevant transitions are included. This completeness is crucial for understanding the full scope of interactions associated with that UI within the context of the specific fragment. Ensuring that all UIs appear at least once with a continuous border in some fragment of the UITD, guarantees that a complete set of transitions for each UI is documented and accessible within the overall diagram.

Dashed Borders: A dashed border around a UI in a UITD indicates that the depiction of this UI within the current fragment is partial or incomplete. Specifically, not all transitions connected to or from this UI are shown in the fragment where it appears with a dashed border. This partial depiction means that to gain a complete understanding of the UI's transitions, one must refer to other fragments of the UITD where the UI might appear, potentially with a continuous border, indicating completeness. The dashed border allows the diagram to be broken down into manageable fragments without overcrowding any fragment with too much information. However, it also signals to users that they need to look at other UITD fragments to fully comprehend the UI's transition possibilities, ensuring they don't overlook any interactions that are not immediately visible in the current view.

Please note that in previous works [3-6], [22], [23], bold and non-bold UI borders were used. However, through practical application, we found that using dashed/continuous outlines enhances their visibility. In addition, our findings revealed that the sixth symbol included in previous works: "sub-diagram" is unnecessary in practice, resulting in Table 1 containing only 5 symbols.

As an example, see that in the UITD depicted in Fig. 5, when users opt to purchase a product, they are directed to UI #9 "Order Summary" which has a dashed border, indicating that not all user actions possible in UI #9 are outlined in this diagram fragment. However, upon selecting "proceed to payment," users are directed to UI #10 "PayPal", also distinguished by dashed borders due to having an incomplete specification of all possible actions. If the purchase is successful, UI #5 "My Orders" is presented, with a continuous border; otherwise, UI #9 is displayed again. Also, in Fig. 5 there is UI #6 "My Shopping Cart", with a continuous border because all the transitions to/from this UI are visible in this fragment.

5. REUSING A UI: USER ACTIONS REPEATED ACROSS MULTIPLE INTERFACES

In [6], it was found that the concept requiring the most explanation for users to understand UITD is that transitions triggered from a contained interface can also be initiated from the containing interface. Therefore, we believe that modelers should further study this concept. In this section, we demonstrate its practical application.

During the design of UI mockups, it is common to have a section of user actions that repeats across different interfaces, such as a menu, a navigation bar, a footer, or a banner. Here, we illustrate how a UI contained within multiple UIs, being the same

in each of them, can be detailed only once, thus simplifying the diagram. Fig. 7 models the case of a navigation bar repeated across multiple UIs. As observed, by defining the navigation bar only once outside of all the UIs that contain it, we prevent the cluttering of transition arrows. Furthermore, these actions are accessible in all the UIs containing the navigation bar. For instance, UI #2, the "navigation bar," is included in the containing interfaces #1, #3, #5, and #6 with dashed outlines, signifying that UI #2 is present in those UIs, however, the user actions associated with UI #2 are not specified there but elsewhere.

The initial UI showcased is UI#1 "Welcome," which includes a navigation bar allowing the user to access UI#3 "Product List," UI#5 "My Orders," and UI#6 "My Shopping Cart." Moreover, users can log in by navigating to UI #4.

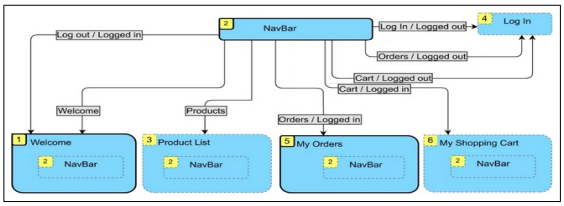


Fig. 7. Modeling a navigation bar present in various Uis.

The UITD also shows that when users are logged in, they can navigate to UI#4 "My Orders" or UI#5 "My Shopping Cart". However, when logged out, they are initially directed to UI#4 "Log in" to access these interfaces. Additionally, in Fig. 7, we note that the borders of UI #2 "NavBar" at the top, are continuous. This means that all user actions available in this interface are specified. UIs #1 and #5 serve as query interfaces and have no other transitions than those shown, so they also have continuous borders. The actions that users can perform in these interfaces are through the navigation bar.

Fig. 8 shows a UITD depicting the log-in-related interactions. Upon entering the correct account and password, users are directed to UI #1 "Welcome"; otherwise, an error message is displayed.

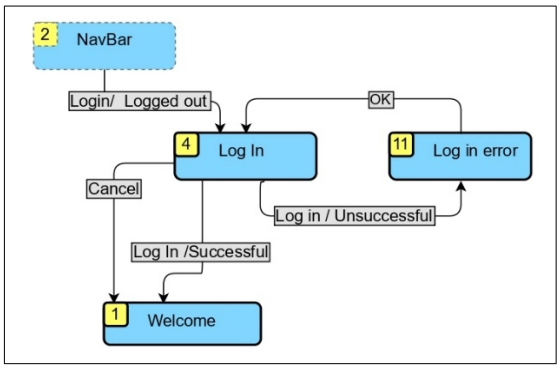


Fig. 8. Log-in-related interactions.

In the following section, we explain how to model in a UITD the transitions triggered by user actions that can be enabled or disabled on the user interface.

6. BUTTONS THAT ARE ENABLED OR DISABLED DEPENDING ON A CONDITION

In specific cases where the same action can trigger different transitions, it is easy to understand the conditions that trigger each transition. For example, for the same "login" action in Fig. 8, two transitions depend on whether the login was successful or not. However, it requires practice to clearly define the conditions determining each transition when modeling with a UITD. Next, we study the case where a UI enables possible user actions (buttons), depending on some of the states in the system.

It is a common practice in certain web development frameworks, like React, to present a list of non-homogeneous objects in the user interface, for example, a view of a product with buttons that sometimes are enabled and sometimes are not. An example scenario is seen in online stores.

When a product is out of stock, it is not logical to present a "Buy" button, and it makes sense to display a "Request Order" button instead. Consequently, we can model the

condition for enabling a user action within the UI by documenting the criteria for activating the button that initiates the transition in the condition of the transition arrow. Fig. 9 displays a variant of the UITD depicted in Fig. 5. Now, two scenarios are addressed within UI #7 “Order Request”. In the first scenario, when the product is in stock, the "buy" button is active and the “waitlist” button is disabled, while in the second scenario, when the product is out of stock, the "buy" button is disabled, and the "waitlist" button is enabled. In other words, the condition for the button's availability is the same as that stated in the transition arrow. Note that for the user actions "buy" and "waitlist", there is no transition when the condition is not met, as in this case, the button is disabled and can’t be clicked.

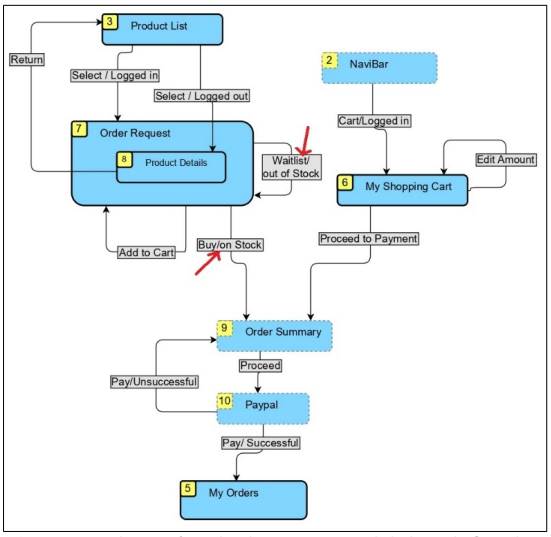


Fig. 9. Condition for the button's availability defined in the transition arrow.

In the forthcoming section, we introduce the concept of the extended interface, which has proven to be practical for modeling a common scenario: users with higher privileges.

7. EXTENDED INTERFACES: GRANT ADDITIONAL PRIVILEGES

Here, we analyze a scenario where a contained UI is extended by a containing UI. This occurs when, for instance, special users need to access more actions, and so the basic set of actions is "extended" by a containing UI. The typical scenario involves a super-user who needs to perform more actions than a regular user, like modifying products in an online store. In such cases, conceptualizing an interface that extends the original one is simpler than dealing with a contained and a contained UI.

Continuing with the online store, Fig. 10 models the scenario for the administrator user, who can modify the store's products. In Fig. 10, we can observe that UI #13 “Product List Admin” is an extension of UI #3 “Product List”, offering the user additional actions, which are "edit," "add," and "delete" a product.

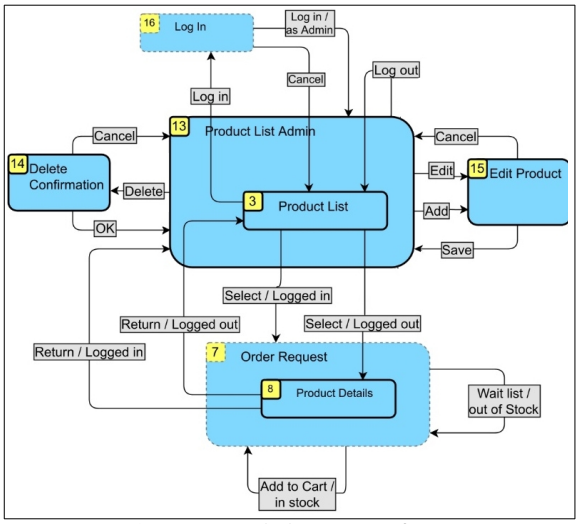


Fig. 10. Extended User Interface

A key feature of extended interfaces is the presence of transitions to the containing UI and the contained UI. For instance, in Fig. 10, when the administrator logs out from UI #13, a transition occurs, causing UI #13, "Product List Admin," to disappear, leaving only the options of UI #3, "Product List," visible. In UIs #7 and #8 “Product Details” in Fig. 9, one can see that there are transitions to both of them too meaning that sometimes the user sees UI #8 only and sometimes UI #8 and UI #7 together depending on the condition “Logged in” or “Logged out”.

Additionally, when dealing with nested interfaces, it's crucial to document the condition in the return action, as it dictates which UI will be presented to the user. For example, when the administrator is logged in, the return action from UI #8 “Product Details” leads to UI #13 “Product List Admin”; however, when logged out, the interface that appears upon return is UI #3 “Product List”.

8. DISCUSSION

The User Interface Transition Diagram (UITD) plays a crucial role in the requirements elicitation phase by effectively mapping out the flow of user interfaces. Integrating UITDs with UI mockups facilitates the transition from specification to design phases by precisely defining user navigation within the system. This integration not only aids in understanding functional requirements but also provides stakeholders with a clear visualization of interface flows, thus enhancing their comprehension and improving the effectiveness of requirements elicitation processes. Our proposed workflow is as follows:

A. Requirements phase
   1. Create the domain model
   2. Define system requirements
   3. Sketch the User Interfaces
   4. Create the UITD (User Interface Transition Diagram)

B. Design phase
   1. System architecture
   2. Database design
   3. Detailed Sequence Diagrams (DSD) (during the design phase, a DSD is designed for each transition in the UITD built during the requirements phase [8])

C. Coding
D. Testing
E. Maintenance

The integration of UITDs into the software development workflow has been possible in the academic field. So far, we have guided over 17 students in developing their final projects, where UITDs have proven to be an effective tool for defining user-system interactions. While creating the UITD, combined with user interface mockups, the student and professor have worked closely together to design and refine the user-system interactions. Moreover, once the User Interface mockups and the UITD were clearly defined, the transition from the requirements to the design phase became easier for novice developers.

Moreover, non-technical stakeholders often prefer simpler notations that facilitate a clearer understanding of software system models [17]. The more graphical symbols a modeling language contains, the more challenging it becomes for users to learn and apply the notation correctly. UITD, with its five graphical symbols, maintains expressiveness without overwhelming users. In contrast, other formal modeling notations are more complex: a) behavior trees (BT) [15] have 12 symbols; b) UML state machines [16] have 15 symbols; c) Interaction Flow Modeling Language (IFML) [12] has 26 symbols; and d) colored Petri nets (CPN) [13] include a 9-tuple with a non-empty color set of size n. Given empirical findings that UITD is relatively easy to understand [6], it is reasonable to expect that software development organizations would prefer to adopt UITD notation.

On the other hand, the adoption of UITD is still missing in the software industry, where deadlines and product delivery are the predominant factors. Adopting new tools requires an investment in training, and not everyone is willing to commit to it. However, we are confident that showing the clarity with which the UITD defines user interactions within the system will generate interest in adopting our proposal.

9. CONCLUSIONS

User Interface Transition Diagrams (UITD) provide a precise and comprehensive method for specifying transition triggers between User Interfaces (UIs), thus facilitating clear communication and effective modeling of user interactions within software systems. This study has presented new solutions for effectively modeling three common features of interactive software systems using UITDs. The first involves modeling user actions repeated across multiple interfaces, such as navigation bars or menus. The second solution focuses on modeling buttons within a UITD that are dynamically enabled or disabled in a UI based on specific conditions. Finally, we demonstrate how to model user interfaces that offer additional actions or privileges to certain users, such as those assigned to an administrator with more privileges than normal users.

Empirical research findings [6] indicate that UITD is readily understandable by individuals regardless of their level of programming expertise. This underscores its accessibility and suggests its potential for widespread acceptance among stakeholders involved in software development processes. Furthermore, the availability of a specialized tool designed specifically for UITD construction [22], which has documented advantages over draw.io for drawing UITDs [23], streamlines the modeling process. This work represents a step forward in facilitating the modeling of interactive software systems using UITD.

By addressing these specific scenarios and showcasing the enhanced capabilities of UITDs, we aim to contribute to the field of software engineering, particularly in the modeling of interactive systems. This study provides practical solutions and promotes the adoption of UITDs within the software industry. We aim to improve the effectiveness of software development practices while promoting clearer communication and stronger collaboration among stakeholders by using simple language to model the flows between user interfaces. This research shows that UITDs can be adapted to address common modeling needs, making them a valuable tool for modern software engineering.

REFERENCES

[1] Bano M, Zowghi D. A systematic review on the relationship between user involvement and system success. Information and Software Technology. 2015;58:148-169. doi: 10.1016/j.infsof.2014.06.011

[2] Mendez Fernandez D, Wagner S, Kalinowski M, Felderer M, Mafra P, Vetro A, et al. Naming the pain in requirements engineering. Empirical Software Engineering. 2017;22(5):2298-2338. doi: 10.1007/s10664-016-9451-7

[3] Gómez MC, Cervantes J. User Interface Transition Diagrams for Customer-Developer Communication Improvement in Software Development Projects. Journal of Systems and Software. 2013;86(9):2394-2410. doi: 10.1016/j.jss.2013.04.022

[4] Gómez-Fuentes M, Cervantes-Ojeda J. Application of User Interface Transition Diagrams in the Construction of a Software System. En: 7th International Conference in Software Engineering Research and Innovation (CONISOFT); 2019 Oct 23-25; Mexico City, Mexico. IEEE; 2019. 123-131. doi: 10.1109/CONISOFT.2019.00026

[5] Ramírez-Viveros LG, Gómez-Fuentes MC, Cervantes-Ojeda J. Modelado de una tienda virtual mediante Diagramas de Transición entre Interfaces de Usuario y Diagramas de Secuencia Detallados: Un caso de éxito. Programación matemática y software. 14(1):53-65. doi: 10.30973/progmat/2022.14.1/6

[6] Cervantes-Ojeda J, Gómez-Fuentes MC, Chacón-Acosta G. Can Non-Developers Quickly Learn a Simplified Modeling Notation? Journal of Software: Evolution and Process. 2022;34(8):e2481. doi: 10.1002/smr.2481

[7] González-Pérez PP, Gómez-Fuentes MC, Velázquez JH. A Hybrid Expert System for the Estimation of the Environmental Impact of Urban Development. British Journal of Mathematics & Computer Science. 2015;7(1):1-17. doi: 10.9734/BJMCS/2015/15239

[8] Gómez-Fuentes MC, Cervantes-Ojeda J. Sequence Diagrams Tailored for Software Design Used to Build a Carpooling Management System. En: 7th International Conference in Software Engineering Research and Innovation (CONISOFT); 2019 Oct 23-25; Mexico City, Mexico. IEEE; 2019. 116-122. doi: 10.1109/CONISOFT.2019.00025

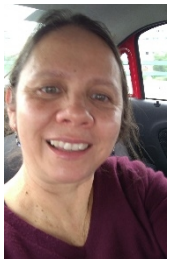
[9] InVisionApp, accessed 6 April 2024, available at https://www.invisionapp.com

[10] Sketch, accessed 6 April 2024, available at https://www.sketch.com

[11] Constantine LL, Lockwood LAD. Software for use: a practical guide to the models and methods of usage-centered design. Boston: Pearson Education; 1999.
[12] IFML: Interaction Flow Modeling Language, accedes April 2024, available at https://www.ifml.org
[13] Kristensen LM, Christensen S, Jensen K. The practitioner's guide to colored Petri nets. International Journal on Software Tools for Technology Transfer. 1998;2(2):98-132. doi: 10.1007/s10009-007-0038-x
[14] Colvin R, Hayes IJ. A semantics for Behavior Trees [technical report]. Brisbane: The University of Queensland, School of Information Technology and Electrical Engineering; 2010 May. Report No.: SSE-2010-03. Available from: http://espace.library.uq.edu.au/view/UQ:204809
[15] Dromey RG. Formalizing the transition from requirements to design, mathematical frameworks for component software - models for analysis and synthesis. In: He J, Liu Z, editors. Mathematical frameworks for component software: models for analysis and synthesis. Singapore: World Scientific Publishing; 2006. p. 156-187. doi: 10.1142/9789812772831_0006
[16] OMG: Object Management Group, UML Standard, The current UML specification, accessed April 3, 2024, available at http://www.uml.org/#UML2.0.
[17] Van der Linden D, Hadar I, Zamansky A. What practitioners want: requirements for visual notations in conceptual modeling. Software and Systems Modeling. 2019;18(3):1813-1831. doi: 10.1007/s10270-018-0667-4
[18] Queiroz TC, Marques AB, Conte T. Using IFML for user interface modeling: an empirical study. In: Proceedings of the 30th International Conference on Software Engineering & Knowledge Engineering (SEKE 2018); July 1-3, 2018; Redwood City, CA, USA. Knowledge Systems Institute Graduate School; 2018. doi: 10.18293/seke2018-103
[19] Ogunyemi AA, Lamas D, Lárusdóttir MK, Loizides F. A Systematic Mapping Study of HCI Practice Research. International Journal of Human-Computer Interaction. 2018;35(16):1461-1486. doi: 10.1080/10447318.2018.1541544
[20] Reggio G, Leotta M, Ricca F, Clerissi D. DUSM: a method for requirements specification and refinement based on disciplined use cases and screen mockups. Journal of Computer Science and Technology. 2018;33(5):918-939. doi: 10.1007/s11390-018-1866-8
[21] Ricca F, Scanniello G, Torchiano M, Reggio G, Astesiano E. Assessing the effect of screen mockups on the comprehension of functional requirements. ACM Transactions on Software Engineering and Methodology. 2014;24(1):1-38. doi: 10.1145/2629457
[22] Cervantes-Ojeda J, Badillo-Salas A, Gómez-Fuentes MC. Specialized Tool for Editing User Interface Transitions Diagrams (UITD). En: 9th International Conference in Software Engineering Research and Innovation (CONISOFT); 2021 Oct 25-29; San Diego, USA. IEEE; 2021. 10-16. doi: 10.1109/CONISOFT52520.2021.00014
[23] Gómez-Fuentes MC, Cervantes-Ojeda J, Badillo-Salas A. The User Interfaces Transition Diagram-Editor: A Tool to Simplify User-System Interaction Modeling. Journal of Software Engineering and Applications. 2023;16(9):483-495. doi: 10.4236/jsea.2023.169023

ACERCA DE LOS AUTORES

Dr. Jorge Cervantes Ojeda, recibió una maestría y un doctorado en Ciencias de la Computación de la Universidad Nacional Autónoma de México en 2004 y 2007 respectivamente. Actualmente es profesor asociado de tiempo completo en la Universidad Autónoma. Metropolitana desde 2007. Trabajó como Ingeniero de Software y Líder de Inspecciones en Alcatel (Bélgica y México) de 1991 a 2000. Sus intereses de investigación son la ingeniería de software, la computación evolutiva y las redes neuronales artificiales.



Dra. María del Carmen Gómez Fuentes, es Doctora en Ciencias (Computación) por la Universidad Nacional Autónoma de México. Estudió Ingeniería Electrónica en la Universidad Autónoma Metropolitana. Actualmente es Profesor Asociado de tiempo completo en la UAM Unidad Cuajimalpa, en el Departamento de Matemáticas Aplicadas y Sistemas. Trabajó 8 años como ingeniero de software en la empresa Alcatel participando durante tres años y medio en el departamento de Diseño y Desarrollo de Software en Alcatel Bell, Bélgica. Su principal área de interés es la Ingeniería de Software, particularmente la Ingeniería de Requerimientos, el modelado y el desarrollo de sistemas. También trabaja en la aplicación de los sistemas computacionales para resolver problemas de teoría de gráficas.